Entrega de Prácticas

Técnicas de Inteligencia Artificial

Andrés Marín Galán

En esta memoria se describe el problema que ha sido resuelto mediante el uso de algoritmos genéticos, la implementación del mismo y diversas pruebas realizadas así como su explicación y evaluación de los parámetros. Realizado en su totalidad en python.

Se ha resuelto el problema de las **capas de aislante** al cual se han introducido tres tipos distintos de restricciones para dificultar aún más el problema:

En primer lugar he decidido rellenar a 0 los blancos de la matriz que se proporciona, cada solución contendrá siempre exactamente 12 capas.

- ☐ Restricción 1: las capas **C**, **F**, **J** no pueden estar en los **extremos**, pues son capas susceptibles a la humedad y si se colocan en los extremos podrían deteriorar las capas interiores. La utilidad de la totalidad de las soluciones que incumplan esta restricción será de **0**.
- ☐ Restricción 2: La unión de capas **G-K** y la **E-C** pasan a ser de utilidad **-10** ya que se ha comprobado que la unión de estas capas debilita la estructura de las restantes.
- ☐ Restricción 3: Las unión triple **F-L-G** pasa a tener una utilidad de **88** que anteriormente sería de 54+18=72 y **J-C-B** pasa a tener una utilidad de **25** que anteriormente sería de 22+31=55. Se sabe que la unión F-LG mejora la impermeabilidad cuando están las 3 capas unidas y a la unión J-C-B le ocurre justo lo contrario.

1.1 Codificación

Se barajaron dos posibles codificaciones:
La primera opción se trata de un vector con **12** elementos en el que cada posición representa una capa de aislante, esta codificación tiene la ventaja de que los cruces se facilitan más que en la segunda codificación con la desventaja de que no se tienen en cuenta las uniones de capas por parejas sino que se hace de manera unitaria.
La segunda opción se basa en un vector de 11 elementos en los que se refleja las uniones de las capas por parejas (A,B,C) -> (A-B,B-C), en esta opción se tiene en cuenta más la tabla por codificarse como parejas, pero dificulta mucho el cruce.
Se escogió la primera codificación para facilitar la implementación del cruce.

1.2 Fitness

Para calcular el valor fitness de una solución se suman todos los valores de la tabla como parejas, pero si se cumple F-L-G o J-C-B sumarlo sin usar la tabla y seguir con la pareja siguiente. Las gráficas mostrarán la mediana de la población en cada iteración.

2.1 Población inicial

Para generar la población inicial generamos para cada individuo un vector del tamaño seleccionado con elementos de 0 a 11, representando de la A a la L. Se indicará en los parámetros el número de individuos de la población inicial.

2.2 Selección

Se seleccionan aleatoriamente un número de individuos para realizar el cruce, este número se indica en los parámetros.

2.3 Cruce

Se seleccionan las parejas de individuos, el primero con el segundo el segundo con el tercero… y por cada valor del vector se escoge con un 50% de probabilidades entre la madre y otro 50% para la madre.
Para la segunda codificación sería complicado pues al codificarse en parejas no pueden existir parejas del tipo (A-B, C-J), pues debería existir la pareja B-C.

2.4 Mutación

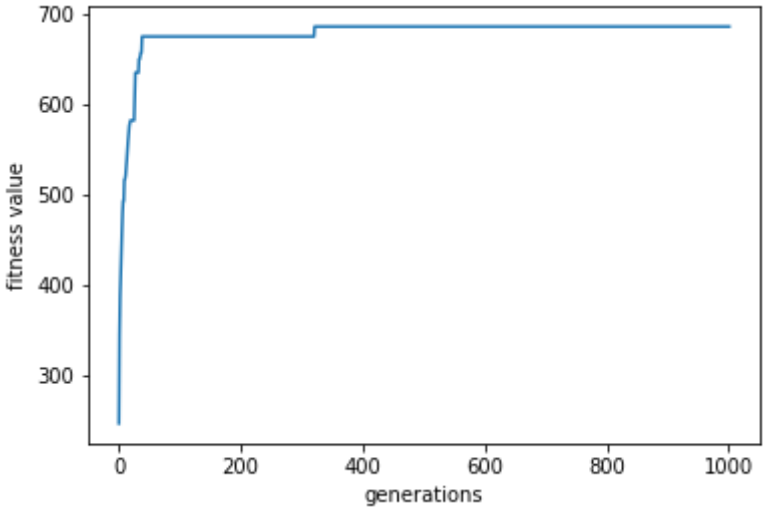
Por cada elemento de la solución se aplica la probabilidad de mutación y si se aplica se cambia dicho elemento por otro aleatoriamente.

2.5 Reemplazo

Se eliminan los peores individuos hasta que la población alcance el tamaño seleccionado en el parámetro de la población a reemplazar.

2.7 Evaluación de Parámetros

```
limit=1000          #Límite de generaciones
initP=200           #Población inicial
selectP=100         #Población a seleccionar para cruzar
probM=0.2           #Probabilidad de mutación sobre todos los alelos
replacN=200         #Población que sobrevive a la siguiente generación
maximo=800          #Valor a buscar como mejor solución
size = 12           #Tamaño de cada individuo
```



Se tiene un buen desempeño para estos valores donde realiza 1000 generaciones sobre una población de 200 con una probabilidad de mutación de 0.2. Se ve como converge sobre las 100 iteraciones y sobre las 350 realiza una leve mejora. La duración de las 1000 generaciones es de 5 segundos un desempeño bastante aceptable frente a la inmensidad de posibilidades que posee el problema. El valor fitness frente a el número de soluciones es de: 686/(1000*(200+99)) = 0.002 o multiplicado por mil 2.29.

Como vemos el valor converge para este tamaño de población por lo que no tiene sentido aumentarla. Si aumentamos la población a seleccionar se obtendrá mayor variabilidad genética.

selectP	Mejor fitness
100	686
150	695
200	698

Bien ahora con selectP:200, trataremos de probar distintas probabilidades de mutación para aumentar la variabilidad genética:

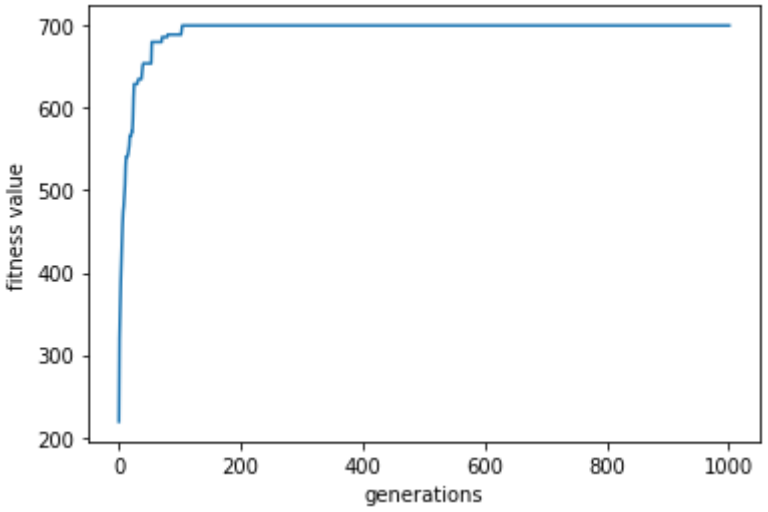
probM	Mejor fitness
0	633
0.2	698
0.4	700
0.6	664
0.8	640
1	598

Como vemos la probabilidad de mutación óptima en este caso es de 0.4 o 40%.
Comparando el número de iteraciones, con los mismos parámetros:

Iteraciones	Mejor fitness
10	533
100	676
1000	700
10000	692

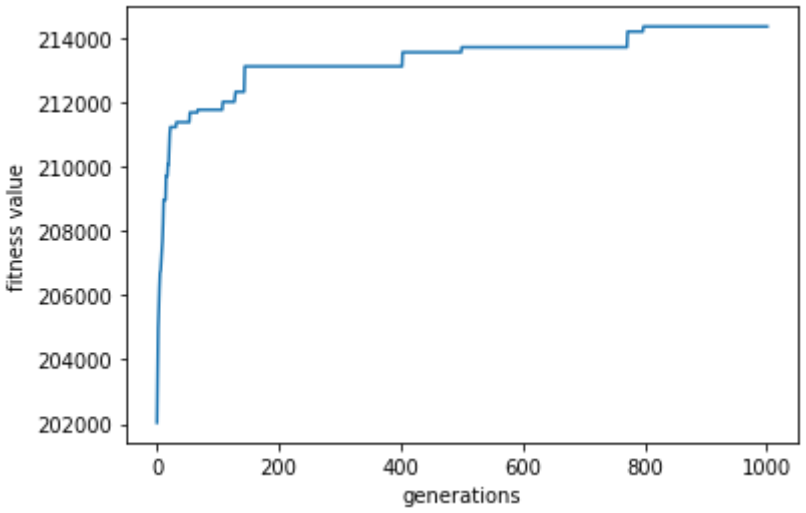
Como vemos la mejor es 1000 iteraciones y aunque 10000 da menor, se trata de la aleatoriedad pero se ve que claramente no mejora, en exceso.

Así con los parametros optimos: iteraciones:1000, initP=200, selectP=200, replaceN=200 y probM = 0.4



Como vemos combinar ambas ideas mejora aún más los resultados a costa por supuesto de 8.7 segundos de ejecución, casi 4 segundos más lento que la ejecución inicial. El valor fitness frente a el número de soluciones será de: 700/(1000*(200+99)) = 0.00234 o (*1000) 2.34. En mi opinión no es necesario aumentar más el tamaño de población por que ya converge con facilidad al valor.

Ahora vamos a aumentar en gran medida la complejidad del problema pasando de una tamaño de individuo de 12 a 7200, emplearemos los valores óptimos, también subimos el máximo a un valor extremadamente alto para que genere las 1000 iteraciones obteniendo:



Con un valor máximo de **214369** y un valor fitness frente a el número de soluciones será de: 214369/(1000*(200+99))=0.71, este valor no es comparable a los anteriores pues ahora la talla del problema es distinta. Esta ejecución ha tardado del orden de 1 hora (58 minutos 41 segundos). Y vemos cómo cada 100 iteraciones más o menos obtenemos un mejor valor. Para saber más o menos si se trata de un buen valor podemos calcular la media de valores en la matriz (28.3) y calcular el producto de la talla por esta media: 28.3*7200=203760 que es inferior a nuestro valor obtenido por tanto podemos concluir que rinde bien, aunque sería mejor para el problema aumentar el número de individuos iniciales de reemplazo y de selección para aumentar ese valor ya que supone un **105%** de la media mientras que para la talla de 12 obtenemos 28.3*12=339.6 y el mejor valor obtenido por nuestro algoritmo es de **707**, que da un **208%** mejor que la media. Si se deja más tiempo (aumentando el límite de iteraciones) es probable que la función no mejore más y que converja a un valor de un porcentaje similar. Para verlo ahora vamos a ajustar un tamaño más bajo, concretamente **120**, ejecutamos este tamaño para los valores iniciales anteriores.
Este nos da un valor de **5223** con un **154%** mejor que la media y un valor de **0.017** del valor fitness frente a el número de soluciones. Tardando 54 segundos.
Se ha aumentado el límite, el tamaño de la población y la probabilidad de mutación pero no se ha obtenido un valor más alto. Pero parece que conforme aumenta la talla del problema la diferencia con la media se reduce debido a que el algoritmo rinde peor por tener mayor cantidad de posibilidades.

Concluimos que para este caso el algoritmo genético funciona muy bien y converge con mucha facilidad encontrando una buena solución en poco tiempo. Aunque no mejora prácticamente nada para iteraciones superiores a 1000 en talla 12. El mejor valor obtenido para la talla 12 del problema ha sido de 707. Se observa que funciona también bien para tallas superiores, pero requiere de mayor tiempo de cómputo para obtener soluciones de alta calidad.

Enfriamiento Simulado

T.I.A.

3.1. Vecinos

Para construir el algoritmo de enfriamiento simulado debemos obtener los vecinos de un individuo, para nuestro problema no hay una estrategia inteligente que pueda dar un mejor individuo pues un cambio de un solo elemento puede convertir el mejor valor de fitness en el peor. Por ello nuestros vecinos serán soluciones aleatorias de la misma manera que se generan en la población inicial del algoritmo genético.

3.2. Solución inicial

Dado que como posteriormente se verá el algoritmo genético ha dado mucho mejor resultado a la hora de obtener la solución correcta, no se ha usado como solución inicial la salida del algoritmo genético pues nunca la mejoraba. Por el contrario se emplea una solución generada de manera aleatoria.

3.3. Temperatura

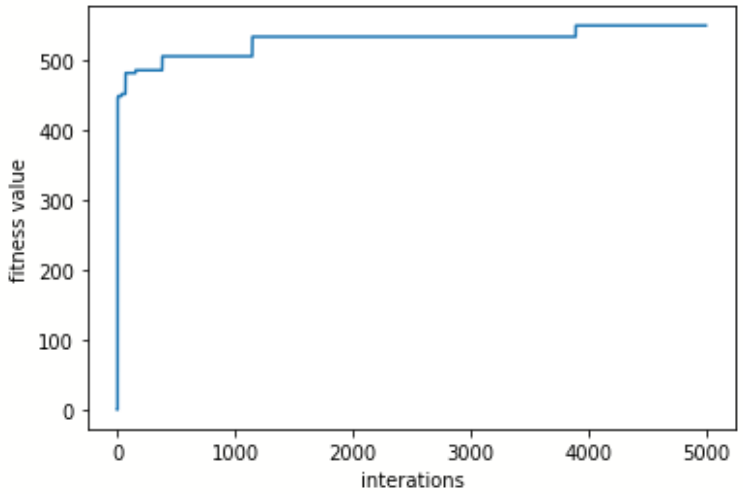
A la hora de considerar la reducción de temperatura se han barajado dos funciones posibles:

T = k*T y T = T/(1+k*T), para ambas se ha obtenido mejores valores para una T = 1000 y para cada función una k=0.85 y k=0.002 respectivamente, para ambas funciones con sus mejores valores de k obtienen rendimientos muy similares.

3.4. Parámetros

```
T = 1000          #Valor inicial de temperatura
nVecinos=5000     #Número de vecinos a explorar
maximo = 800      #Valor máximo a buscar
k = 0.002         #Valor de k para T=T/(1+k*T)
#k=0.85           #Valor de k para T=k*T
size = 12         #Tamaño de cada solución
```

Para todos los valores obtenidos en las gráficas se ha recogido el mejor valor de fitness para cada iteración.



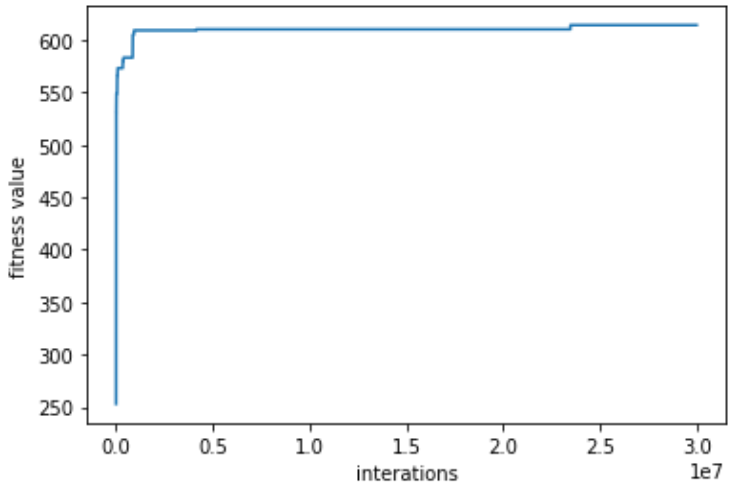
Para k=0.85 con T=k*T como hemos dicho daría una gráfica similar. Para esta ejecución el mejor valor es 550 (550/5000 = 0.11), más bajo que lo que obtuvimos en la primera ejecución del algoritmo genético 686 (686/(1000*(200+99)) = 0.002), una diferencia de 136 (y sobre los individuos generados es bastante más alta que la primera del algoritmo genético). Los valores bajos de la función fitness es debido a que no tenemos una forma inteligente de calcular vecinos más allá de generarlos aleatoriamente. Esta ejecución tardó 120 ms o 0.12s, muy poco por ello vamos a aumentar en gran medida el número de vecinos para ver si alcanzamos al menos el valor inicial obtenido en el Algoritmo genético que es de 686.

Considerando la temperatura:

nVecinos	temperatura	Mediana (fitness)
5000	1000	525
5000	100	520
5000	5000	503

Como vemos el valor de temperatura que mejor fitness obtiene es 1000 como se ha usado en las ejecuciones siguientes.
Ahora aumentamos el valor del número de vecinos: nVecinos=30000000 (30 millones)

Obteniendo esta gráfica:



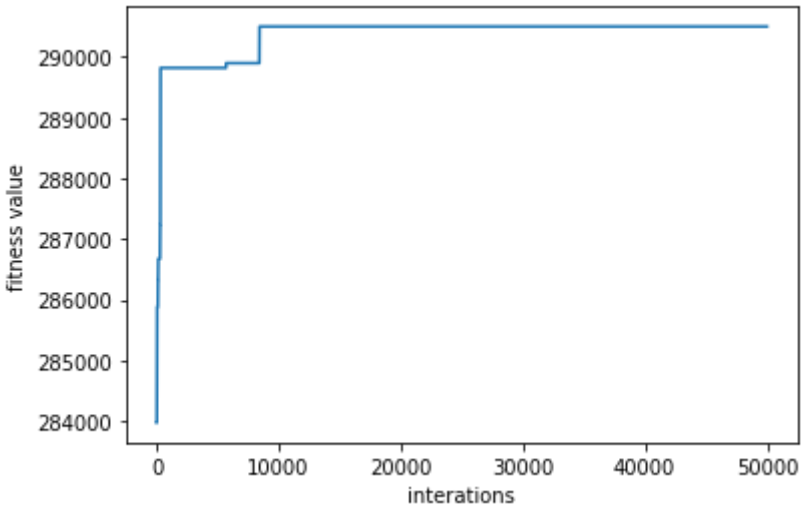
Con un mejor valor de 614 (614/30000000 = 0.00002) mejorando el valor anterior, pero no superando al valor que el algoritmo genético tenía con sus parámetros iniciales 686 (686/(1000*(200+99)) = 0.002) con una diferencia ahora de 72 pero mucha más diferencia como es lógico en el valor fitness frente a el número de soluciones (siendo 100 veces más pequeño que el del genético).

Comparado con la media es de **180%**, un valor razonadamente bueno. Esta ejecución ha tardado 12 minutos y 11 segundos, he observado que el punto que ocupa mayor tiempo de ejecución es precisamente la obtención de los vecinos.

Ahora vamos a aumentar el tamaño del problema como hicimos con el Algoritmo genético, por ahora mantenemos el número de vecinos a 5000, con los demás parámetros iguales al inicial y vamos variando el tamaño.

Primero se ha aumentado el tamaño a 10000 (10k), para este caso ha tardado 1 min y 30s, con un mejor valor de 289487 que comparado a la media es de **102%** un buen valor para una primera ejecución con el número de vecinos inicial. Vamos a aumentar el número de vecinos a 50000 (50k) por ahora.

Obtenemos un valor de 290503 que comparado con la media es de **104%** una mejora muy vaga por ahora. Tardando 14 minutos y 37 segundos.



Por último vamos a ejecutarlo con 100000 (100k) vecinos para ver si mejora algo más el valor fitness, pero quedó un valor muy similar.

Podemos concluir que para este problema el algoritmo de enfriamiento simulado funciona medianamente bien, encontrando una solución razonable en poco tiempo. Aunque no mejora algo para un número de vecinos superior a 1000 en talla 12. El mejor valor obtenido para la talla 12 del problema ha sido de 614. Se observa que funciona también bien para tallas superiores, pero requiere de mayor tiempo de cómputo para obtener soluciones de una calidad razonable, pero menor tiempo que para el algoritmo genético.

4. Conclusión

Podemos asegurar que para este problema propuesto con las restricciones añadidas el algoritmo genético funciona razonadamente mejor que el algoritmo de enfriamiento simulado obteniendo mejores soluciones, aunque el enfriamiento simulado funciona más rápido, pero a costa de un mayor uso de memoria. La razón principal de esto es la incapacidad para obtener vecinos de manera inteligente en el algoritmo de enfriamiento simulado pues se obtienen de manera 100% aleatoria dificultando la obtención de tan buenas soluciones como si ocurre en el algoritmo genético.