PRACTICA 1
SISTEMAS INTELIGENTES

DAVID MARTINEZ POZUELO 48653244X
GRUPO 3
DAVID.MARTINEZP2@UM.ES

ÍNDICE

Explicación breve y completa del Algoritmo Genético (AG). Debe quedar muy claro cuáles son los elementos y el proceso que sigue dicha técnica. pg3

La tabla que muestre los valores pg4

Análisis de las pruebas de ajuste pg6

El manual de asignación final construido. pg7

Análisis comparativo entre las técnicas Primero en Profundidad y el AG construido. pg8

Explicación breve y completa del Algoritmo Genético (AG).

Son algoritmos de optimización búsqueda y aprendizaje inspirados en los procesos de Evolución Natural y Evolución Genética.
Un algoritmo genético consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuáles de ellos deben generar descendencia (padres) para la nueva generación (hijos).

La terminología que se emplea es: Población, conjunto de posibles soluciones(individuos codificados) para el problema planteado. Cromosoma o individuo, posible solución al problema(individuo codificado). Gen, posición de un elemento del cromosoma, representa las particularidades y variables de cada individuo.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado.

El algoritmo Genético comienza con una población inicial de individuos, estos deben ser codificados en cadenas, estas cadenas pueden ser de enteros, 0s y 1s, reales, etc.

Los individuos son seleccionados acorde a su fitness para la producción de descendientes. Los padres son recombinados para producir dichos descendientes. Todos los descendientes serán mutados con una cierta probabilidad. Entonces, el fitness de los descendientes es calculado. Los descendientes son insertados dentro de la población reemplazando a los padres, produciendo una nueva generación. El ciclo se realizará hasta que se alcance el criterio de optimización.

En la fase de Selección,se identifica cuales individuos de la población serán los que se crucen, para ello se usará el valor fitness previamente calculado obteniendo así los mejores individuos para los cruces. Entre otros métodos, los más usados son:
-Ruleta: Se eligen con probabilidad proporcional a su función de idoneidad.
-Torneo: Se establecen k torneos aleatorios entre parejas de individuos y se eligen los que ganan en cada torneo (mejor función idoneidad).

Una vez terminada la selección se comienza con el Cruce, que produce nuevos individuos combinando la información contenido en los padres ,acoplamiento de la población-padres(los padres son sustituidos por los hijos).
En el cruce de valores binarios hay de dos tipos:

-Cruce en un punto: Se cortan los cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes.

-Cruce en dos puntos: Es una generalización del corte por un punto, en vez de cortar por un punto, ahora realizaremos dos cortes. Se debe tener en cuenta que ninguno de estos segmentos coincida con los extremos de los cromosomas, originando de esta forma 3 segmentos.

Después del cruce, cada descendiente sufre una mutación. Las variables de los descendientes son mutadas por pequeñas perturbaciones (tamaño del paso de la mutación), con baja probabilidad.
-Cambio de un gen aleatorio: esta mutación se produce cuando un gen cambia su valor. (modificación de un alelo).
-Intercambio entre dos genes: dos genes de un individuo intercambian sus valores(intercambio de alelos).

Después de producir descendientes, éstos deben ser insertados en la población. Esto es especialmente importante si se producen menos descendientes que el tamaño original de la población. Otro caso es cuando no todos los descendientes se tengan que usar en cada generación o se crean más descendientes de los que realmente se necesitan.

La tabla que muestre los valores de la función fitness.

Tabla valores obtenidos con AG nreinas

Selector	Cruce	Mutación	Población	pCruce	pMutación	F8	F9
GATournamentSelector	OnePointCrossover	FlipMutator	100	0.8	0,0125	0	1
GATournamentSelector	OnePointCrossover	FlipMutator	100	0.8	0,025	0	0

F10	F11	F12	F13	F14	F15	F16	F17	F18
1	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0

F19	F20	F21	F22	F23	F24	F25	F26	F27
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0

F28	F29	F30	F31	F32	F33	F34
0	0	0	0	0	0	0
0	0	0	0	0	0	0

F35	F36	F37	F38	F39	F40	F41
0	0	0	0	0	0	0
0	0	0	0	0	0	2

F42	F43	F44	F45	F46	F47	F48	F49
0	0	0	0	0	0	0	0
3	3	4	5	3	4	2	4

F50	F55
0	0
3	7

Valores fitness AG nreinas.

Func

Selector	Cruce	Mutación	Población	pCruce	pMutación	F8	F9
GATournamentSelector	OnePointCrossover	FlipMutator	100	0.8	0,0125	28	35
GATournamentSelector	OnePointCrossover	FlipMutator	100	0.8	0,025	28	36

ión de idoneidad

Función de idoneidad

F10	F11	F12	F13	F14	F15	F16	F17	F18
44	56	66	78	92	106	121	137	154
45	56	67	79	92	106	121	137	154

Función de idoneidad

F19	F20	F21	F22	F23	F24	F25	F26	F27
172	191	211	232	254	276	301	326	352
172	191	211	232	254	277	301	326	352

Función de idoneidad

F28	F29	F30	F31	F32	F33	F34
379	407	436	466	497	529	562
379	407	436	466	497	529	562

Función de idoneidad

F35	F36	F37	F38	F39	F40	F41
596	631	667	704	742	781	821
596	631	667	704	742	781	819

Función de idoneidad

F42	F43	F44	F45	F46	F47	F48
862	904	947	991	1036	1082	1129
859	901	943	986	1033	1078	1127

F49	F50	F55
1177	1226	1486
1173	1223	1479

Búsqueda primero en profundidad

Busqueda primero en profundidad		nreinas5	nreinas6	nreinas7	nreinas8	nreinas9	nreinas10
	Solucion (Si/No)	SI	Si	Si	SI	Si	Si
	Tiempo	0,002	0,657	0,084	0,178	4,761	4.484

nreinas11	nreinas12	nreinas13	nreinas16	nreinas20	nreinas24	nreinas25	nreinas41	nreinas55
Si	Si	Si	Si	Si	Si	Si	Si	Si
43.50	∞	∞	∞	∞	∞	∞	∞	∞

Análisis de las pruebas de ajuste.

En el estudio fundamentos prácticos deducimos.

- GATournamentSelector como FlipMutator son métodos de selección válidos y eficientes para la selección en la población inicial. En este caso usamos GATournamentSelector ya que es un parámetro fijo.
- Usamos el método de cruce OnePointCrossover ya que tiene un comportamiento similar a TwoPointCrossover debido a que el cruce por un punto tiene un menor consumo de recursos hardware, al ser menos complejo en la implementación.
- Respecto a la probabilidad de cruce, obtuvimos que los mejores valores se obtenían en una probabilidad de cruce del 0.8, estableciendo en ese valor como parámetro fijo.
- Número de generaciones 10.000 y población 100 (parámetros fijos)

En el caso de la práctica probabilidad de mutación es un parámetro ajustable y como hemos estudiado en el primer apartado, sabemos cuanto mayor sea el problema (más reinas) menor debe ser la probabilidad de mutación. Según lo dicho anteriormente podemos observar en la tabla como se produce exactamente esto.

Vemos con los resultados como se demuestra lo dicho, CUANDO Pmutacion =0.0125 hay jaque en tamaño de problema(nreinas) bajo y ningun fallo cuando el problema aumenta de tamaño

En pmutacion = 0.025 vemos lo contrario, no ocurre ningún jaque en casos de problema pequeños pero empiezan a haber jaques a partir de un problema de tamaño 41 reinas.

Por tanto obtenemos que, siendo n el tamaño del caso:

Vemos cómo se cumple exactamente lo anterior dicho.
1° parámetro: tamaño del caso(número de reinas, n)
2° parámetro: 100 (valor recomendado)(tamaño población)
3° parámetro: 10000 (valor recomendado)(número de generaciones)
4° parámetro: 0.8 (valor recomendado) (probabilidad de cruce)
5° parámetro:
Si 8<=n<=13 : pM =0.025 (pM= probabilidad de mutación)
Si 13<n<=40: pM= 0.0125-0.025
Si 41<= n <=50: pM= 0.0125

Con los datos vistos antes y con lo dicho anteriormente de la importancia de la probabilidad de mutación(pM) podemos afirmar que cuando n está comprendida entre 8 y 13 nos resultaría interesante seleccionar la probabilidad de 0.025. Cuando n está entre 13 y 40 no nos infiere la elección de nuestro pM (0.0125-0.025) ya que los resultados son los mismos para ambos casos. Pero si avanzamos n, en nuestro caso de 41 a 50 si que nos interesa un pM más bajo (0.0125) ya que no nos ofrece unos resultados óptimos.

Podemos comprobar con las ejecuciones que he realizado del algoritmo desde mi ordenador como según el tamaño de n va creciendo es aconsejable ir bajando el tamaño de la probabilidad de mutación y viceversa, ya que con probabilidades bajas de mutación hemos tenido jaques cuando nuestra n era de tamaño pequeño

Ahora, con estos datos y los obtenidos previamente en fundamentos prácticos podemos construir el manual de asignación para el problema:

El manual de asignación final construido.

Para el mejor resultado a la hora de querer resolver el problema de las n-reinas se recomienda que la entrada de parámetros estén dentro de unos rango establecidos para que sea lo más óptimo posible. Algún valor recomendado depende del valor n (tamaño caso)(1° parámetro).

Comenzaremos definiendo el significado de cada uno de los 5 parámetros que tiene como entrada nuestro algoritmo y el significado de cada una de nuestras nomenclaturas.

- n: es el número de reinas de nuestro programa
- Tamaño poblacion: Este parámetro nos indica el número de cromosomas que tenemos en nuestra población para una generación determinada. En caso de que esta medida sea

insuficiente, el algoritmo genético tiene pocas posibilidades de realizar reproducciones con lo que se realizaría una búsqueda de soluciones escasa y poco óptima. Por otro lado, si la población es excesiva, el algoritmo genético será excesivamente lento. De hecho estudios revelan que hay un límite a partir del cual es ineficiente elevar el tamaño de la población puesto que no se consigue una mayor velocidad en la resolución del problema.

- Probabilidad de cruce: Indica la frecuencia con la que se producen cruces entre los cromosomas padre es decir, que haya probabilidad de reproducción entre ellos. En caso de que no exista probabilidad de reproducción, los hijos serán copias exactas de los padres. En caso de haberla, los hijos tendrán partes de los cromosomas de los padres. Si la probabilidad de cruce es del 100% el hijo se crea totalmente por cruce, no por partes.
- pM: Nos indica la frecuencia con la que los genes de un cromosoma son mutados. Si no hay mutación, los descendientes son los mismos que había tras la reproducción. En caso de que haya mutaciones, parte del cromosoma descendiente es modificado y si la probabilidad de mutación es del 100%, la totalidad del cromosoma se cambia. En este caso, no se cambian simplemente unos bits del cromosoma sino que se cambian todos, lo que significa que se produce una inversión en el cromosoma y no una mutación por lo que la población degenera muy rápidamente.

1º parámetro: tamaño del caso(número de reinas, n)
2º parámetro: 100 (valor recomendado)(tamaño población)
3º parámetro: 10000 (valor recomendado)(número de generaciones)
4º parámetro: 0.8 (valor recomendado) (probabilidad de cruce)
5º parámetro:
Si 8<=n<=13 : pM =0.025 (pM= probabilidad de mutación)
Si 13<n<=40: pM= 0.0125-0.025
Si 41<= n <=50: pM= 0.0125

Cuando n está comprendida entre 8 y 13 nos resultaría interesante seleccionar la probabilidad de mutacion (pM) a 0.025. Cuando n está entre 13 y 40 no nos infiere la elección de nuestro pM (0.0125-0.025) ya que los resultados son los mismos para ambos casos. Pero si avanzamos n, en nuestro caso de 41 a 50 si que nos interesa un pM más bajo (0.0125) ya que no nos ofrece unos resultados óptimos.

Análisis comparativo entre las técnicas Primero en Profundidad y el AG construido.

Una búsqueda en profundidad es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso

con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.

La búsqueda en profundidad se usa cuando queremos probar si una solución entre varias posibles cumple con ciertos requisitos; como sucede en el problema del camino que debe recorrer un caballo en un tablero de ajedrez para pasar por las 64 casillas del tablero.

El algoritmo de búsqueda en profundidad tiene varias aplicaciones...

En un grafo acíclico dirigido, es decir un conjunto de nodos donde cada nodo tiene una sola dirección que no es consigo mismo, se puede realizar un ordenamiento topológico mediante el algoritmo DFS. Por ejemplo, se puede aplicar para organizar actividades que tienen por lo menos alguna dependencia entre sí, a fin de organizar eficientemente la ejecución de una lista de actividades.

Los puentes en grafos son 2 nodos conectados de tal manera que para llegar a cada uno de sus extremos solo se puede a través de uno de esos nodos. Es decir, si removemos uno de los nodos ya no podremos acceder al otro nodo porque se han desconectado completamente. Esto se puede usar en la priorización de actividades que son representadas por nodos.

Si se quiere resolver un 'puzzle' o un laberinto, se puede resolver a través de DFS; los pasos de la solución pueden ser representados por nodos, donde cada nodo es dependiente del nodo predecesor. Es decir, cada paso de la solución del puzzle depende del anterior paso.

A veces será importante conocer qué tan conectados están ciertas actividades o componentes a fin de disminuir la dependencia de actividades o acoplamiento de componentes. Ésto con el objetivo de organizar en una mejor forma las actividades o agrupar de mejor modo los componentes porque así será más entendible el sistema; esto también se puede resolver a través del algoritmo DFS.

-Completitud: si se impone un límite de profundidad, el algoritmo encuentra una solución sólo si ésta existe dentro de ese límite.

– Optimización: no se garantiza que la solución sea óptima.

– Complejidad temporal: exponencial respecto a la profundidad del límite de exploración O(rm).

– Complejidad memoria: en el caso de no controlar los nodos repetidos el coste es lineal respecto al factor de ramificación y el límite de profundidad O(r m). Si la implementación es recursiva el coste es O(m).

Algoritmo genético.

-Completitud: al no ser un algoritmo completo porque no siempre puede encontrar la solución adecuada y es dependiente de parámetros introducidos

– Optimización: no se garantiza encontrar siempre la mejor solución al problema.

– Complejidad en tiempo: depende directamente de los parámetros introducidos (lo que hace que pueda variar su orden). Aunque ofrece buenos resultados.

– Complejidad memoria: depende de la representación(clase) de los individuos y el tamaño de la población.

Podemos observar en los tiempos de las tablas como el tiempo de la ejecución crece de manera exponencial a partir de una n reina (en mi caso es en la reina 10), llegando incluso a no terminar la ejecución por ejemplo cuando la n>12. Sin embargo, en la ejecución del algoritmo de primero en profundidad podemos observar como todas las n posibles reinas tienen una solución.

Esto es debido a que BPP puede recorrer un camino por el que nunca encontrará una solución al problema.