Universidad de Murcia

Sistemas Inteligentes

PCEO

Práctica 1: Algoritmos Genéticos

Autor:
Jose Antonio Lorencio Abril
49196915F
joseantonio.lorencioa@um.es

Profesor:
Manuel Gil Pérez

Diciembre 2020

Índice

Índice de cuadros

1. Introduccion: Algoritmos Genéticos

1.1. La intuición: Teoría de la Evolución de Darwin

Charles Darwin (s. XIX), fue un pionero en el desarrollo de las ideas de la evolución, y, sin duda, uno de sus más influyentes defensores en la época. La teoría de la evolución sostiene que las poblaciones de seres vivos y el medio guardan una especie de equilibrio. Este equilibrio se da entre los cambios que puede haber en un medio determinado, así como las interacciones entre distintos seres vivos, de tal forma que los individuos con características más adecuadas (las diferencias se deben, en última instancia, a las mutaciones genéticas) a una situación dada tienen mayor probabilidad de sobrevivir. Esta probabilidad mayor de supervivencia, a largo plazo y gracias a la reproducción (tranmisión de genes entre generaciones), se traduce en que las características negativas irán desapareciendo, mientras que las positivas prevalecerán. De esta forma, las especies vivas cambian con el paso del tiempo.

Los algoritmos genéticos presentan un enfoque algorítmico de resolución de problemas basado en esta teoría. La idea es que podemos pensar que el problema que queremos resolver es el medio ambiente, y que los individuos mejor adaptados son aquellos que son 'cercanos' a la solución del problema.

Con este enfoque, nuestro objetivo es que haya evolución, de forma que las mejores soluciones prevalezcan frente a las peores, llegando, al final, a la solución del problema, o a un estado lo más cercano posible a la solución como podamos. Por analogía a lo que ocurre en la naturaleza, se trabaja con conceptos como fenotipo, genotipo, mutación, cruce,... que explicaré detalladamente más adelante.

1.2. Explicación técnica de los Algoritmos Genéticos

Poniéndonos técnicos, los algoritmos genéticos (AG) son una variante de los métodos de búsqueda por haz estocástica. Esto quiere decir que son un método de búsqueda local: no suele importar cómo llegar a la solución, tan solo la solución en sí misma; y no se recorre el espacio de soluciones al completo, sino parte de él. Por tanto, no siempre obtendremos una solución óptima, pues podemos obtener óptimos locales. Los AGs constan de las siguientes partes:

- **Población:** Llevaremos la pista de k estados, generados al comienzo aleatoriamente.
- **Individuos o cromosomas:** cada estado de la población.
- **Codificación:** forma de representación de los estados. La codificación será una cadena sobre un alfabeto finito. La codificación puede ser binaria, entera, de orden, o real.
- **Gen:** cada elemento de un estado se denomina gen. Por tanto, un gen será un elemento del alfabeto de codificación. El conjunto de genes de un individuo a veces se denomina genotipo, y a la traducción del significado del genotipo al problema que tratamos de resolver se le denomina fenotipo.
- **Función Fitness:** es la función usada para cuantificar lo bueno que es cada individuo en relación al problema. Esta será la función que buscamos optimizar.
- **Operadores genéticos:** los algoritmos genéticos dan uso a 3 operadores básicos para simular el proceso de evolución:

  -Selección: utilizado para seleccionar qué individuos pasan a reproducirse, y cuales no. Para la selección es muy normal tener en cuenta los valores de fitness de los individuos, ya que es este valor el que indica la calidad de los mismos.

  -Cruce: simula la reproducción de los individuos seleccionados. Define la forma en la que los genes de pares de individuos se entremezclan para dar lugar a nuevos individuos. Por supuesto, los individuos generados mediante cruce deben representar estados válidos del espacio de búsqueda.

  -Mutación: este operador provoca variaciones aleatorias en algunos genes de un individuo. Se utiliza para añadir variabilidad a la población.

Así, el esquema básico de un AG queda como sigue:

```
funcion AG(poblacion, fitness) devuelve individuo

    entradas: población, función fitness
    repetir
        mi_poblacion := SELECCION(poblacion, fitness)
        para x=1 hasta tamaño(mi_población) hacer
            si(r<pc) entonces
                poblacion_cruzar = poblacion_cruzar + mi_poblacion[x]
        para i=1 hasta tamaño(poblacion_cruzar) hacer
            {x,y} = dos individuos aleatorios de poblacion_cruzar
            h1, h2 = CRUCE(x,y)
            {x,y} = {h1, h2}
        para x=1 hasta tamaño(mi_poblacion)*longitud(individuo) hacer
            si(r<pm) entonces
                gen = MUTAR(gen)
                modificar el gen del individuo correspondiente
        poblacion = mi_poblacion
    hasta condición de fin (fitness mínimo, o suficientemente pequeño, o ha pasado mucho tiempo)
    devolver individuo de poblacion tal que fitness(individuo)=max{fitness(i): i en poblacion}
```

Donde r es un número generado aleatoriamente y p_c es la probabilidad de que un individuo sea seleccionado para cruzarse, p_m es la probabilidad de que se produzca una mutación en un gen determinado, y SELECCION, CRUCE y MUTAR son implementaciones concretas de los operadores que hemos explicado anteriormente. Algunas implementaciones comúnmente utilizadas son:

- Selección:

  -Ruleta: se eligen los individuos de forma directamente proporcional a la función de idoneidad, de modo que individuos con mejos valor fitness tienen más posibilidades de ser elegidos

  -Torneo: se hacen k emparejamientos entre individuos de la población y se eligen los que tienen mejor fitness en cada emparejamiento

- Cruce:

  -Cruce por un punto: se intercambian todos los genes de los dos individuos de un punto en adelante

  -Cruce por dos puntos: se intercambian todos los genes de los dos individuos entre dos puntos seleccionados

- Mutación:

  -Cambio de un gen aleatorio: un gen cambia su valor aleatoriamente

  -Intercambio entre dos genes: dos genes aleatorios intercambian sus valores

1.3. Nuestro caso particular: el problema de las n reinas

Nuestro objetivo ahora es resolver el problema de las n reinas. Sabemos que existe la solución de búsqueda en profundidad, pero esta solución es extremadamente costosa, pues su tiempo de ejecución es de orden n!. Tras conocer los AG, podríamos plantearnos su uso para la resolución de este problema. Para ello, debemos definir todas las partes del AG específicamente para nuestro propósito:

- Individuos: los estados del problema serán n-uplas $x_1, ..., x_n$ donde el gen $x_i \in 1, ..., n$ representa la fila en la que se encuentra la reina de la columna i.
- Codificación: como se deduce del punto anterior, la codificación es entera y el alfabeto será $1, ..., n$.
- Función Fitness: la función Fitness de un individuo consistirá en contar cuántas amenazas hay entre las reinas posicionadas como indica el estado, sin contar repetidos. Por tanto, dado que nuestro objetivo es que no haya ninguna amenaza, buscamos minimizar el valor fitness, idealmente hasta que sea 0.

- Operadores de selección, cruce y mutación: haremos distintas pruebas para conocer cuál es nuestra mejor opción en cada caso, así como para obtener los valores más adecuados de las probabilidades de cruce y mutación.
- Condición de terminación: el algoritmo terminará cuando haya transcurrido una cantidad de generaciones especificada.

2. Ajuste de Parámetros

En clase vimos, tras un ajuste inicial, que, para 10000 generaciones, debemos utilizar los oepradores GATournamentSelector, FlipMutator y OnePointCrossOver, un tamaño de población de 100 y una probabilidad de cruce $p_c = 0{,}8$. Además, parecía intuirse que el valor de la probabilidad de mutación p_m debería ser decreciente al aumentar el tamaño del problema. Esta sección va a estar dedicada a resolver esta cuestión. Para este propósito, he ejecutado el programa proporcionado por el equipo docente, *nreinas.exe* con los parámetros fijos

- nº generaciones: 10000
- método de selección: GATournamentSelector
- operador de cruce: OnePointCrossover
- operador de mutación: FlipMutator
- tamaño de la población: 100
- $p_c = 0{,}8$

Y variando el parámetro de probabilidad de mutación $p_m \in 0{,}0125, 0{,}025$. Además, lo haré para tamaños del problema desde 8 hasta 50. Para ello, creé un sencillo programa consistente en un bucle que ejecuta

$$nReinas.exe\ N\ 100\ 10000\ 0{,}8\ p_m,$$

donde N varía desde 8 hasta 50 y $p_m \in 0{,}0125, 0{,}025$. Los resultados obtenidos pueden verse en el archivo adjunto *salida-AG.txt*, de los cuales he obtenido la tabla resumen que podemos ver en Cuadro 1.

Vamos a comentar estos resultados:

- Una primera observación es que el tiempo de ejecución es mejor usando $p_m = 0,025$ en la mayoría de casos hasta llegar a N=38, a partir del cual se obtienen mejores resultados para $p_m = 0,0125$.
- Podemos ver cómo $p_m = 0,025$ arroja un valor fitness óptimo para todos los tamaños inferiores a 41.
- Por otro lado, $p_m = 0{,}0125$ nos da un valor fitness óptimo para todos los tamaños superiores a 24, aunque también lo hace en la mayoría de casos inferiores

Tras estas observaciones, podemos obtener algunas conclusiones. Efectivamente, parece que, tal y como preveíamos en clase, la probabilidad de mutación con mejores resultados iba a decrecer al aumentar el tamaño del problema. Hemos visto como la mayor de ellas proporciona resultados óptimos hasta el caso 42, y la pequeña a partir del caso 24. Entre estos dos valores ambas proporcionan resultados óptimos, y los tiempos de ejecución son más o menos parejos, pero generalmente siendo menores para $p_m = 0{,}025$. Aunque, indudablemente, a partir de tamaño 39, el tiempo ofrecido por $p_m = 0{,}0125$ es considerablemente mejor. Por tanto, parece una decisión razonable utilizar $p_m = 0{,}025$ siempre que $N \leq 38$ y, a partir de aquí, tomar $p_m = 0{,}0125$. Es, además, posible que al continuar aumentando el tamaño del problema la elección óptima de este valor siga disminuyendo.

N	Pm	Valor Fitness	Tiempo
8	0,0125	0	0,004
8	0,025	0	0,002
9	0,0125	1	1,001
9	0,025	0	0,004
10	0,0125	1	1,13
10	0,025	0	0,003
11	0,0125	0	0,1
11	0,025	0	0,004
12	0,0125	1	1,358
12	0,025	0	0,738
13	0,0125	1	1,488
13	0,025	0	0,029
14	0,0125	0	0,009
14	0,025	0	0,084
15	0,0125	0	0,934
15	0,025	0	0,02
16	0,0125	0	0,252
16	0,025	0	0,134
17	0,0125	0	0,017
17	0,025	0	0,427
18	0,0125	0	0,148
18	0,025	0	2,369
19	0,0125	0	0,514
19	0,025	0	0,047
20	0,0125	0	0,015
20	0,025	0	0,668
21	0,0125	0	0,027
21	0,025	0	0,028
22	0,0125	0	1,62
22	0,025	0	0,398
23	0,0125	0	0,645
23	0,025	0	0,043
24	0,0125	1	3,54
24	0,025	0	0,377
25	0,0125	0	0,211
25	0,025	0	0,285
26	0,0125	0	0,043
26	0,025	0	1,186
27	0,0125	0	2,579
27	0,025	0	0,501
28	0,0125	0	0,662
28	0,025	0	1,589
29	0,0125	0	0,138
29	0,025	0	1,3
30	0,0125	0	2,906
30	0,025	0	0,326
31	0,0125	0	0,491
31	0,025	0	0,834
32	0,0125	0	0,885
32	0,025	0	2,258
33	0,0125	0	0,797
33	0,025	0	0,521
34	0,0125	0	0,193
34	0,025	0	0,375
35	0,0125	0	1,182
35	0,025	0	0,408
36	0,0125	0	1,587
36	0,025	0	0,195
37	0,0125	0	1,622
37	0,025	0	0,211
38	0,0125	0	4,581
38	0,025	0	2,395
39	0,0125	0	0,158
39	0,025	0	2,75
40	0,0125	0	0,354
40	0,025	0	2,977
41	0,0125	0	3,085
41	0,025	2	9,693
42	0,0125	0	3,028
42	0,025	3	10,116
43	0,0125	0	1,35
43	0,025	3	10,543
44	0,0125	0	1,404
44	0,025	4	10,976
45	0,0125	0	2,993
45	0,025	5	11,426
46	0,0125	0	1,701
46	0,025	3	11,889
47	0,0125	0	1,871
47	0,025	4	12,357
48	0,0125	0	2,876
48	0,025	2	12,831
49	0,0125	0	8,698
49	0,025	4	13,317
50	0,0125	0	1,384
50	0,025	3	13,807

Cuadro 1: Tabla de resultados para el análisis de parámetros

3. Manual de Asignación

Al resolver un caso concreto del problema de las n-reinas es recomendable utilizar como parámetros los valores que se detallan a continuación. Algunos de estos valores recomendados dependen del tamaño del problema, n, que se corresponde con el 1º parámetro:

- 2º parámetro: 100 (Valor recomendado)
- 3º parámetro: 10000 (Valor recomendado)
- 4º parámetro: 0.8 (Valor recomendado)
- 5º parámetro:
  - Si $n \leq 38$: 0.025 (Valor recomendado)
  - Si $39 \leq n$: 0.0125 (Valor recomendado

4. Análisis comparativo entre Búsqueda en Profundidad y el Algoritmo Genético

Vamos ahora a comprobar si el esfuerzo de desarrollar un algoritmo genético para resolver este problema realmente merece la pena, o si, por el contrario, la solución clásica nos ofrece mejores resultados. Para ello, vamos a resolver el problema con ambos esquemas, para distintos tamaños de n, y a comparar los resultados para obtener la respuesta.

En las siguientes tablas podemos observar los resultados obtenidos. Para ver la salida completa pueden consultarse los ficheros *salida-AG-Comp.txt* y *salida-BPP-Comp.txt*.

BPP	
N	Tiempo (segs)
5	0,001
6	0,006
7	0,02
8	0,166
9	4,83
10	4,503
11	37,965
12	4908,236

AG			
N	Pm	Fitness	Tiempo(segs)
5	0,025	0	0,003
6	0,025	0	0,003
7	0,025	0	0,001
8	0,025	0	0,002
9	0,025	0	0,005
10	0,025	0	0,004
11	0,025	0	0,004
12	0,025	0	0,736
13	0,025	0	0,029
16	0,025	0	0,134
20	0,025	0	0,665
24	0,025	0	0,376
25	0,025	0	0,284
41	0,0125	0	3,074
55	0,0125	0	6,423

Cuadro 2: Tabla de resultados comparativa

Vemos que BPP solo llega hasta tamaño 12, esto se debe a que, como podemos observar en la tabla, el tiempo de ejecución crece muchísimo al aumentar n. Para tamaño 12 ya tardó un poco más de hora y media, aumentando en un factor de más de 100 respecto al caso anterior. Por tanto, sin siquiera hacer cálculos, podemos esperar que para tamaño 13 tarde 1 día completo o más en ejecutarse. Por otro lado, AG ha sido ejecutado, para cada tamaño, con los valores recomendados en el manual de asignación.

Vamos pues, a comparar los resultados, fijándonos en cuatro aspectos:

1. **Completitud:** BPP es completo, ya que recorre todo el espacio de soluciones (¡que es finito!). Encontrará, en algún momento, la solución. Respecto al AG, en teoría no es completo, ya que es un método de búsqueda local. Pensemos, por ejemplo, qué ocurriría si le decimos que utilice tan solo 2 generaciones. Es altamente improbable que encuentre la solución en ese caso. Al aumentar el número de generaciones utilizadas, aumentamos la probabilidad de que el algoritmo obtenga la solución al problema, pero no podemos hacer que tenga probabilidad 1. Sin embargo, en la práctica observamos que siempre nos devuelve un valor fitness óptimo, gracias a la elección de los parámetros basada en el estudio previo.

2. **Optimalidad:** en este problema se verifica que todas las soluciones son óptimas, porque o bien no hay ninguna amenaza entre las reinas y tenemos solución, o bien hay alguna amenaza, por lo que no hay solución. Por tanto, en este caso $completitud \implies optimalidad$ (y el recíproco siempre se verifica). Esto quiere decir que BPP es óptimo, pero AG, teóricamente no, aunque en la práctica ofrece resultados óptimos para los casos estudiados.

3. **Complejidad en tiempo:** sabemos que BPP tiene un orden de ejecución temporal de b^m, donde b es el factor de ramificación y m la profundidad máxima del orden de búsqueda. En este caso, mirando el pseudocódigo del programa no podemos saber exactamente cómo recorre el espacio de soluciones, suponiendo que lo hace teniendo en cuenta que si hay un elemento en una fila y columna, no puede haber ningún elemento en esa fila ni en esa columna, entonces

las ramificaciones son de tamaño $n, n-1, ..., 2, 1$, obteniendo un orden de ejecución factorial, $O(n!)$. Respecto a AG, observamos que el tiempo de ejecución aumenta al aumentar el tamaño del problema, pero los incrementos no siguen un patrón consistente. Sabemos, eso sí, que, dado que el tamaño de la población y el número de generaciones los hemos mantenido fijos, lo único que va a influir en el tiempo es el tamaño de los individuos, es decir, n, el número de filas del tablero. Es evidente que AG presenta unas resoluciones increíblemente rápidas en comparación con BPP (para n=12 la diferencia es de 3 órdenes de magnitud, y esta diferencia se acentuará al aumentar n).

4. **Complejidad en memoria:** BPP en memoria solo necesita un vector de tamaño n. AG, por otro lado, necesita una cantidad de vectores de tamaño n igual al tamaño de la población, 100 en nuestro caso. Por tanto, ambos presentan un orden lineal de ocupación de memoria $O(n)$, pero la constante es mayor para AG que para BPP. En este apartado es mejor BPP.

Tras todos estos comentarios, parece bastante sensato asegurar que el tiempo invertido en el desarrollo del algoritmo genético ha merecido la pena, ya que, en la práctica, el algoritmo BPP va a ser impracticable para tamaños ni siquiera demasiado grandes del problema, por cuestiones de tiempo. Mientras que nuestro algoritmo genético, si bien ocupa algo más de memoria (tampoco es que vaya a reventar nuestra RAM), ofrece unos resultados rapídisimos y, eligiendo correctamente los parámetros, óptimos.

5. Tiempo de trabajo

Tiempo dedicado a la Práctica	
Preparación de la práctica y ejecución de los programas	5 horas
Comparación de resultados y reflexión sobre los mismos	3 horas
Elaboración del informe	3 horas
Total	11 horas

Cuadro 3: Desglose del tiempo de trabajo

6. Conclusión y valoración personal

Hemos visto, a lo largo de este trabajo, qué son los algoritmos genéticos, así como un caso práctico de su implementación y uso. Hemos podido razonar acerca de su comportamiento en dependencia de las diferentes variantes disponibles, así como distintos valores de los parámetros que pueden tenerse en cuenta. Hemos podido, también, comparar sus resultados con los obtenidos mediante un algoritmo clásico de búsqueda en profundidad, y hemos visto como la solución mediante una técnica que, en un principio puede parecer demasiado enrevesada para dar solución a nuestro problema (no está de más tener presente el principio de la navaja de Ockham), es muchísimo mejor que la solución por búsqueda en profundidad.

Por último, como valoración personal, se me queda un sabor amargo. Siento que lo que hemos tratado es sumamente interesante, pero que al final lo único que he hecho es ejecutar algunos programas y 'ver qué sale'.

Quizás esta parte, en cierto modo pasiva, de la resolución de problemas, sea inevitable en ocasiones. Entiendo la importancia de comparar resultados, valorar qué está sucediendo, tomar decisiones en base a estas reflexiones y trasmitir las conclusiones. No obstante, no creo que esta parte del trabajo sea incompatible con un proyecto en el que el alumno pueda sentirse más involucrado.