Algoritmos Genéticos y Evolutivos

Práctica 1: N Reinas.

Autor:

Daniel Fernández: daniel.f.rico@alumnos.uc3m.es

19 de octubre de 2017

Índice

1. Contexto.

1.1. Problema de las 8 reinas.

El problema de las ocho reinas es un problema clásico en computación y matemáticas, el cual consiste en colocar ocho reinas en un tablero de ajedrez (de 8 casillas) sin que se amenacen entre sí (en ajedrez, las reinas pueden moverse tantas casillas como quieran en cualquier dirección).

El problema original data de 1848 y fue propuesto por el ajedrecista Max Bezzel. Durante años, muchos matemáticos (incluyendo a Gauss), han trabajado en él y lo han generalizado a N-reinas. Las primeras soluciones se plantean en 1850.

Edsger Dijkstra usó este problema en 1972 para ilustrar el poder de la llamada programación estructurada. Publicó una descripción muy detallada del desarrollo del algoritmo de backtracking, "depth-first".

Para nuestro problema, tomaremos un enfoque más abstracto: dado un tablero de dimensiones NxN, queremos colocar N reinas sin que se ataquen entre ellas.

1.2. Problema planteado.

Se plantea resolver el problema de las N-reinas para un valor de N muy inferior a 1000 sin necesidad de hallar todas las soluciones, mediante la utilización de un Algoritmo Genético.

Primero afrontaremos el problema mediante un algoritmo de fuerza bruta con backtracking, para estudiar cuánto tarda en hallar posibles soluciones dentro del espacio de búsqueda.

A continuación, se implementa un Algoritmo Genético con el fin de hallar las soluciones para valores de N computacionalmente intensos.

2. Solucionar las N-reinas empleando fuerza bruta.

2.1. ¿Cuánto se tarda en obtener la primera solución?

2.2. ¿Cuánto se tardan en obtener soluciones subsecuentes?

3. Solucionar las N-reinas con Algoritmos Genéticos.

3.1. Codificación.

La codificación actual para nuestro problema trata de un vector de tamaño N, donde cada valor corresponde a la posición x de una reina sobre la fila y del tablero, siendo y el índice del elemento x en el vector de reinas.

3.1.1. ¿Qué otras alternativas se han barajado y descartado?

En primer lugar, se planteó una codificación del tablero como vector binario de tamaño NxN, de tal modo que cada individuo de la población fuese un vector de 0s y 1s, donde un 1 equivale a una reina en la posición equivalente del tablero, y un 0 la ausencia de ésta.

En segundo lugar, se planteó un vector de reinas de tamaño N, donde sólo se acumulasen las reinas que se encontraban en el tablero. Dicho vector estaría compuesto de tuplas (x, y) correspondientes a la posición de cada reina en el tablero.

Al darnos cuenta de que los números x e y han de ser excluyentes (restricción para evitar ataques en la misma fila y columna), se planteó la codificación actual que hace uso del índice de la lista como posición y de la reina (valor unívoco).

3.1.2. ¿Por qué la codificación actual es mejor que las otras?

Esta codificación es mucho más eficiente que las codificaciones anteriormente propuestas, ya que ocupa mucho menos espacio en memoria (un vector de ints ocupa $K * N$ bits, siendo $K = 4$ para una arquitectura estándar de 32-bit o 64-bit, frente a la peor solución anteriormente propuesta, un vector $N * N$ binario, que ocuparía $N * N$ bits).

Además, esta codificación lleva implícitas las restricciones de fila y columna para cada reina. La restricción fila va implícita en la propia estructura de datos (no puede haber dos elementos en la misma posición de la lista), y la restricción columna va implícita en el hecho de que los valores del vector son permutaciones de N (no se repite ningún valor de x).

Esto nos elimina automáticamente muchas soluciones inválidas de la población, agilizando el algoritmo y permitiendo encontrar una solución mucho más rápidamente.

3.2. Función de fitness.

Se plantean dos funciones de fitness para el problema a tratar:

1. En primer lugar, y considerando que no llegamos a posicionar N número de reinas, se considera la función $F = \frac{n}{N} - \frac{b}{n}$, siendo n el número total de reinas posicionadas sobre el tablero frente a N (número ideal de reinas y dimensión del lado del tablero), y b el número de reinas mal posicionadas. Esta función vale 1 cuando todas las reinas están bien posicionadas, y 0 cuando están todas mal posicionadas, o no hay reinas sobre el tablero.

2. Considerando que, debido a nuestra codificación, no existe la posibilidad de que n sea distinta de N (siempre hay N reinas sobre el tablero, aunque estén mal situadas), se propone la función $F = 1 - \frac{b}{N}$, que no es más que una substitución de $n = N$ en la función propuesta anteriormente. Esta es la función implementada.

3.3. Métodos implementados

3.3.1. Selección

El método de selección aplicado para nuestro problema es una selección por torneo.

Para esta selección tomamos un parámetro K que se corresponde con el tamaño de la muestra de población que tomamos para nuestro torneo. A continuación, evaluamos la muestra y seleccionamos al mejor individuo un número de veces definido por el parámetro L.

Originalmente, se implementa el método de selección sugerido en el libro *An Introduction to Genetic Algorithms* (Melanie, M. 1996), en el cual la selección por torneo ordenaba la muestra de tamaño K por su grado de fitness, y seleccionaba los dos mejores como padres de la nueva población. Por razones de simplificación, se decide que el corte se realiza en L, quedando por defecto un tamaño de muestra K y un tamaño de ganadores del torneo de L.

Sin embargo, se llegó a la conclusión de que era mejor repetir el proceso de torneo L veces hasta conseguir el número deseado de padres.

...desarrollar

[GRÁFICOS para demostrarlo]

El método de selección escogido va acompañado de la política de reemplazo.

3.3.2. Cruce

El cruce que se implementa es una recombinación discreta entre los dos padres con cuidado de no repetir ninguno de los anteriores valores de N

Se genera un descendiente eligiendo un gen de cada progenitor de forma aleatoria

Métodos de cruce:

1. Partially matched crossover (PMX): In this method, two crossover points are selected at random and PMX proceeds by position wise exchanges. The two crossover points give matching selection. It affects cross by position-by-position exchange operations. In this method parents are mapped to each other, hence we can also call it partially mapped crossover.

2. Cycle crossover (CX): Beginning at any gene i in parent 1, the i-th gene in parent 2 becomes replaced by it. The same is repeated for the displaced gene until the gene which is equal to the first inserted gene becomes replaced (cycle).

3. Order crossover operator (OX1): A portion of one parent is mapped to a portion of the other parent. From the replaced portion on, the rest is filled up by the remaining genes, where already present genes are omitted and the order is preserved.

4. Otros:

   a) order-based crossover operator (OX2)

   b) position-based crossover operator (POS)

   c) voting recombination crossover operator (VR)

   d) alternating-position crossover operator (AP)

   e) sequential constructive crossover operator (SCX)

3.3.3. Mutación

1. Primera aproximación: se mutan los dos hijos con probabilidad p_m. Aproximación clásica: por cada bit del individuo, se evalúa su probabilidad de mutación, y se cambia el bit (0-1). Al no ser codificación binaria, lo cambiamos por valores 0-N.

2. Segunda aproximación: con (1) obtenemos resultados repetidos, nuestra codificación es una permutación de columnas (valores 0-N). Por lo tanto, evaluamos la probabilidad del individuo entero de mutar, y si es positiva hacemos un reordenamiento de sus valores (shuffle).

3. Tercera aproximación: con (2) obtenemos valores demasiado aleatorios con una probabilidad de mutación elevada. Nuestra aproximación esta vez consiste de nuevo en evaluar la probabilidad de mutación bit a bit, y en caso de ser positiva, intercambiando dicho bit con otro bit del individuo de manera aleatoria. De esta forma, en caso de mutar 1 bit, el individuo sólo diferirá del original en 2 bits.

4. Cuarta aproximación: con (3) se estanca mucho. Queremos mutar el individuo más, introducimos un índice de diversidad. Este índice multiplica la probabilidad de mutación hasta por 10, momento en el cual se resetea. Cambiamos también el planteamiento de (3), ahora muta todo el individuo con probabilidad pm, no bit a bit.

Nota: la probabilidad de mutar depende de N, ya que el número de bits a mutar depende también del tamaño del individuo, así que para hacerlo más constante tomamos la probabilidad de 1/N y la multiplicamos por la probabilidad por individuo.

- Cruce:

3.4. Parametrización

De Jong's experiments indicated that the best population size was 50–100 individuals, the best single-point crossover rate was about 0.6 per pair of parents, and the best mutation rate was 0.001 per bit - De Jong (1975) mencionado en *An Introduction to Genetic Algorithms* (Melanie, M. 1996)

Nota: Grefenstette (1986) noted that, since the GA could be used as an optimization procedure, it could be used to optimize the parameters for another GA! (A similar study was done by Bramlette (1991).) In Grefenstette's experiments, the "meta-level GA.^{ev}olved a population of 50 GA parameter sets for the problems in De Jong's test suite. Each individual encoded six GA parameters: population size, crossover rate, mutation rate, generation gap, scaling window (a particular scaling technique that I won't discuss here), and selection strategy (elitist or nonelitist).

Result: These settings were similar to those found by Grefenstette: population size 20–30, crossover rate 0.75–0.95, and mutation rate 0.005–0.01

Parámetros propuestos, e impacto de los mismos en los resultados, incluyendo gráficas y tablas que sean pertinentes

- Capas: **3**.
- Momentum: **0.2455**.
- Número de ciclos: **400**.
- El conjunto de datos, con un número reducido de patrones de entrada, requiere un modelo con una tasa de aprendizaje alta. Se fijó este valor en **0.61** y sin decaer durante el entrenamiento.

La elección de dichos parámetros se basó en una comparativa de los distintos valores posibles, como podemos ver en la siguiente figura:

Figura 1: Evolución del acierto en la clasificación según los parámetros escogidos para el entrenamiento

3.5. Resultados

Resultados obtenidos y número de evaluaciones necesarias

Nota: tiempos de ejecución del algoritmo principal con parámetros por defecto (en ms): Params: python3 AG_N_reinas.py 16 100000 200 4 2 0.001 0.4 0.8

Tiempo init: 3.2100677490234375 Tiempo evaluacion: 1.2519359588623047 Tiempo selección: 0.07486343383789062 Tiempo cruce: 0.1010894775390625 Tiempo mutación: 0.06699562072753906

4. Conclusiones, Problemas Encontrados y Opiniones Personales

Problemas: estancamiento.

De menor a mayor estancamiento en cruce: - cruce SCX aleatorio - cruce SCX no ordenado - cruce SCX

5. Anexo