Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 2

Algoritmo Genético

Alumna: Romina Romero Oropesa

Profesor: Alexandre Bergel

Auxiliares: Juan Pablo Silva

Ayudantes: Alonso Reyes Feris
Gabriel Chandía

Fecha de entrega: 29 de noviembre de 2018
Santiago, Chile

Índice de Contenidos

Lista de Figuras

Lista de Códigos

Problema: N-queen

El problema seleccionado es el N-queen. La tarea es posicionar N reinas en un tablero de agedrez de NxN, sin que se ataquen entre si. Esto quiere decir que:

- No debe haber dos reinas en la misma vertical.
- No debe haber dos reinas en la misma horizontal.
- No debe haber dos reinas en la misma diagonal.

En el presente trabajo, se utiliza un algoritmo genético para resolver el problema.

El vector de genes corresponde a la concatenación de las filas. Cada posición representa a una celda. Si hay posicionada una reina en la celda, en su posición habrá un 1. De lo contrario, habrá un 0. Sea N el número de reinas a ubicar, el ancho y alto del tablero, la secuencia genética será de largo NxN,y el alfabeto será [0,1].

La función de fitness revisa cada una de las horizontales, verticales y diagonales. El algoritmo simplemente, si encuentra una reina en una línea, cada vez que encuentre otra suma uno al costo de la solución. La función de fitness es -1*costo. Así, la solución óptima entregará un cero.

$$fitness(t) = -1 * (t.reviewHorizontals() + t.reviewVerticals() + t.reviewDiagonals() + t.repeated())$$

Sumamos también la cantidad de índices repetidos, pues también se consideran reinas mal ubicadas.

Descripción del programa

Para poder ejecutar los tests y hacer uso de esta implementación, se debe instalar el paquete numpy y matplotlib de python con una terminal:

Código 1: Instalación de dependencias.

```
pip install numpy matplotlib scipy
```

En https://pip.pypa.io/en/stable/installing/ se explica como instalar pip.

Se implementa la clase Individual y Population para modelar al algoritmo genético. En la clase Individual se hace el crossover y mutación. En la clase Population se implementa la reproducción.

Para facilitar la modificación de hiperparámetros, se implementa la clase Tester, en cuyo constructor se define cada uno de ellos:

Código 2: Definición de hiperparámetros

```
class Tester:
```

```
  def __init__(self,fitnessFunction,alphabeth=[0,1],genSize=128,populationSize=100,threshold=0,
   ↪ cleanFunction=ident,mutateFunction=ident,visualizationFunction=display):
    self.fitnessFunction = fitnessFunction
    self.alphabeth = alphabeth
    self.genSize = genSize
    self.populationSize = populationSize
    self.threshold=threshold
    self.cleanFunction=cleanFunction
    self.mutateFunction=mutateFunction
    self.visualizationFunction=visualizationFunction
```

- **fitnessFunction:** función de fitness con que se mide cada solución.
- **alphabeth:** alfabeto al cual pertenece cada gen.
- **genSize:** tamaño de la secuencia genética.
- **populationSize:** tamaño de la población de cada generación.
- **threshold:** valor máximo que puede tomar la función fitness, valor que toma al alcanzar la solución.
- **cleanFunction:** función que se ejecuta luego de crearse una secuencia genética. Por defecto no se hace nada.
- **mutateFunction:** función que se ejecuta luego de mutarse la secuencia genética. Por defecto no se hace nada.
- **visualizationFunction:** función que ïmprime.[a]l individuo en pantalla. Por defecto se imprimen los genes.

Por otro lado, para facilitar el manejo y revisión de tableros, se creó la clase Board. Esta tiene unittest asociados. Para ejecutarlo, escribir en línea de comando:

Código 3: Unittest clase board.

```
python Board.py
```

Para la ejecución del algoritmo genético, cada generación tiene el mismo tamaño. Además, si tras 50 generaciones no se muestra un candidato mejor, se asume estancado, y se detiene el algoritmo.

Para terminar, se realizó 4 tests que se pueden ejecutar por línea de comando:

Código 4: Ejecutar tests

```
python test[nombre del test].py
```

Los archivos de test son los siguientes:

- **testSecretBitSequence128.py:** test con el reto de descubrir la secuencia de 128 bits oculta, cuya función de fitness es la cantidad de aciertos de cada individuo.

- **testSecretBitSequence1000.py:** mismo test anterior, pero con 1000 bits.
- **testSecretVowelSequence128.py:** mismo test, pero el alfabeto, en lugar de ser [0,1], es el conjunto de vocales.
- **testSecretLettertSequence128.py:** mismo test, pero el alfabeto corresponde al abecedario completo.

Si se requiere modificar los hiperparámetros, basta con hacerlo al momento ded instanciar la clase Tester.

Evaluación

Secuencia oculta de 128 bits

Población de 100 individuos con 128 genes.

- Mejor solución: encontrada
- Número de generaciones: 29
- Tiempo de ejecución: 1,65 segundos

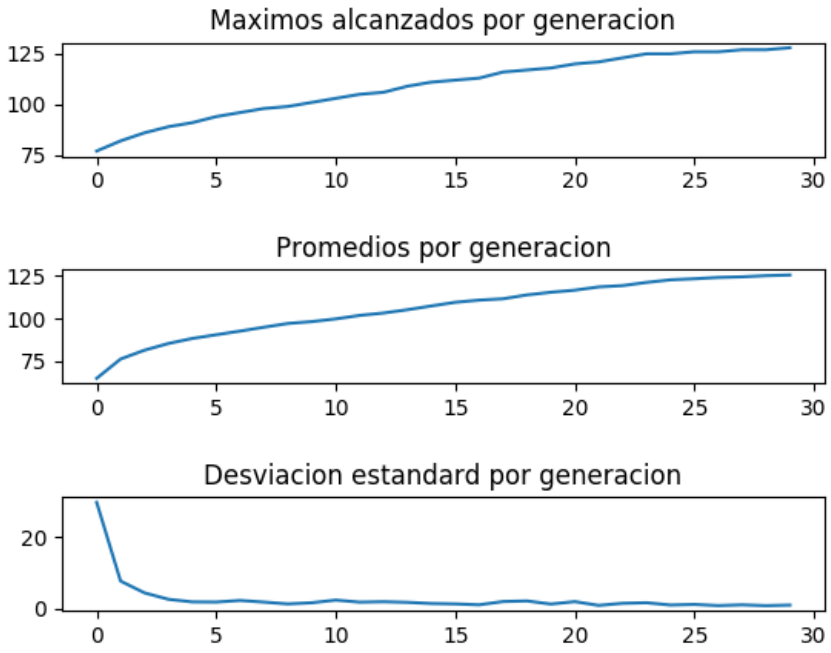

Figura 1: Métricas para secuencia oculta de 128 bits

Secuencia oculta de 1000 bits

Población de 200 individuos con 1000 genes.

- Mejor solución: 905 ítemes encontrados (óptimo no encontrado)
- Número de generaciones: 221
- Tiempo de ejecución: 355,8 segundos

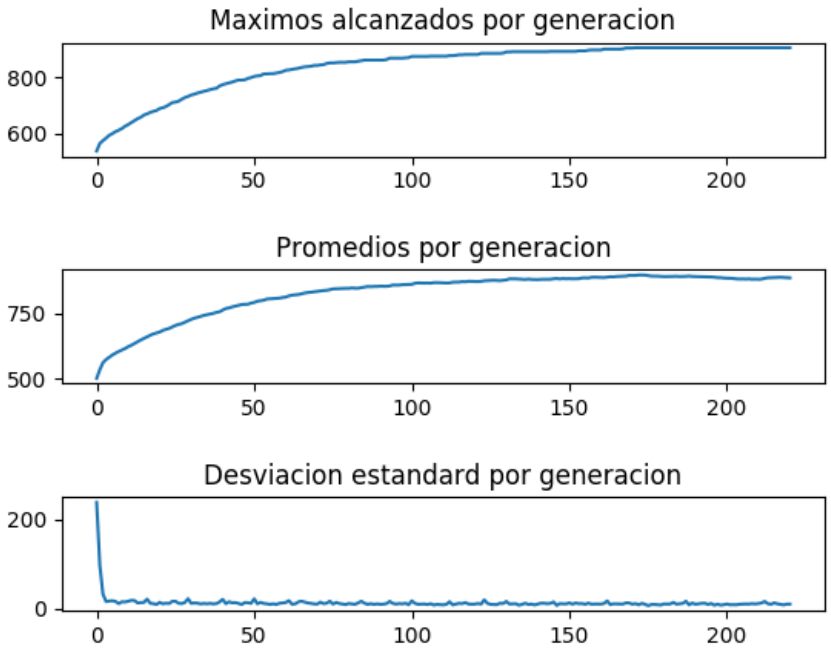

Figura 2: Métricas para secuencia oculta de 1000 bits

Secuencia oculta de 128 vocales

Población con 100 individuos con 128 genes.

- Mejor solución: 127 ítemes encontrados (óptimo no encontrado)
- Número de generaciones: 160
- Tiempo de ejecución: 9,2 segundos

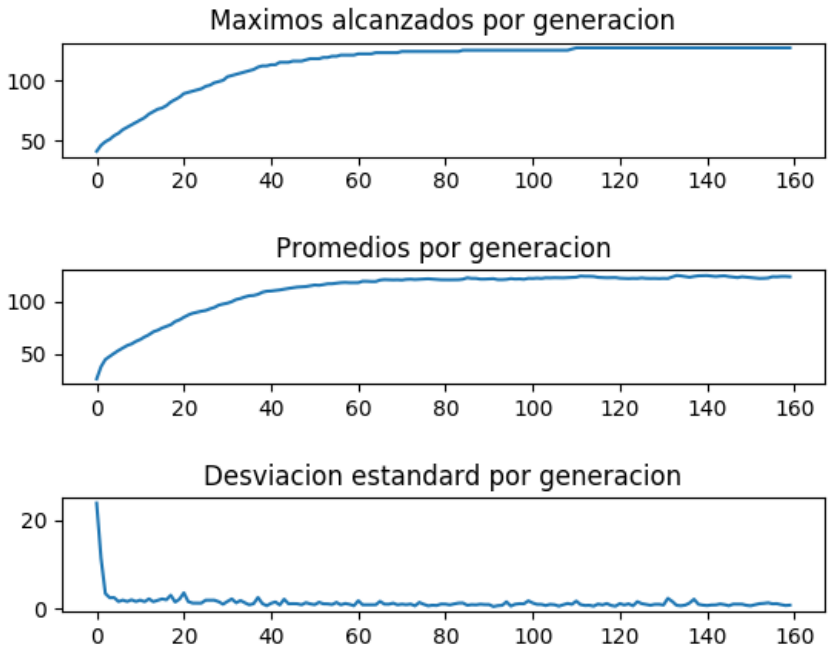

Figura 3: Métricas para secuencia oculta de 128 vocales

Secuencia oculta de 128 letras

Población con 200 individuos con 128 genes.

- Mejor solución: 119 ítemes encontrados (óptimo no encontrado)
- Número de generaciones: 181
- Tiempo de ejecución: 37,38 segundos

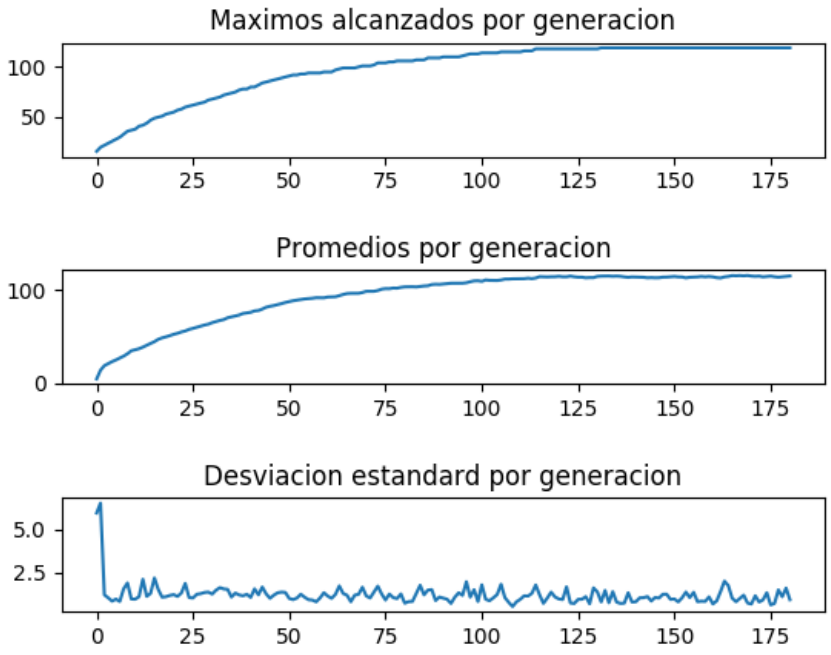

Figura 4: Métricas para secuencia oculta de 128 letras

NQueen N=10

Población con 200 individuos con 100 genes.

- Mejor solución: 1 ataque de reinas (óptimo no encontrado)
- Número de generaciones: 109
- Tiempo de ejecución: 185.5 segundos

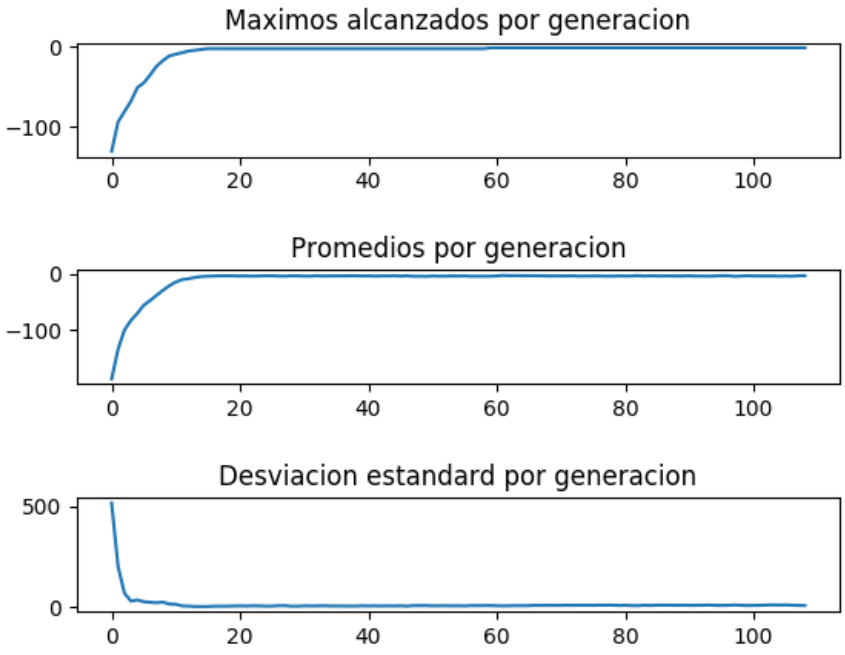

Figura 5: Métricas para nqueen, N=10

NQueen N=20

Población con 200 individuos con 400 genes.

- Mejor solución: 3 ataques de reinas (óptimo no encontrado)
- Número de generaciones: 177
- Tiempo de ejecución: 1098.3 segundos

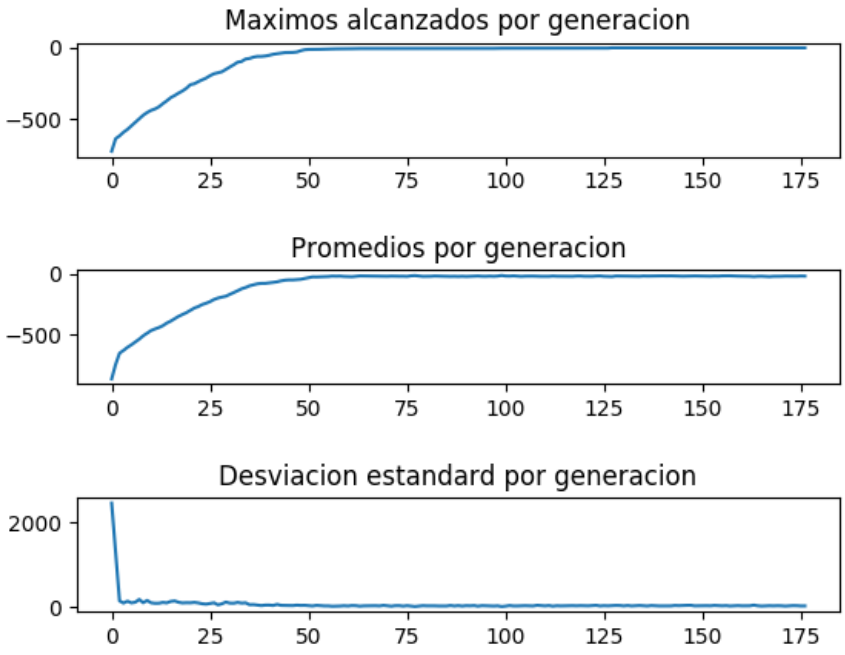

Figura 6: Métricas para nqueen, N=20

Discusión

El problema no se resolvió en su totalidad. A medida que la cantidad de reinas aumenta, el número de errores también.

Sin embargo, en todos los ejemplos podemos ver cómo se va mejorando generación tras generación. En el caso de la secuencia oculta de bits, se obtiene buenos resultados. A medida que se va complejizando el problema (largo de alfabeto y largo de secuencia genética), esto va dificultándose. Llegado un instante, las mejoras se producen tras demasiadas generaciones (como se va viendo en todas las curvas). Fue necesario determinar cuando el algoritmo quedó estancado, para no dejar el algoritmo ejecutándose por siempre.

Para mejorar el algoritmo, podría probarse performance distintas de mezclado, hacer algún procesamiento sobre los datos una vez mezclados, entre otras cosas. Por esto se deja disponible la opción de agregar funciones de procesamiento. Pero, de todas formas, si se encuentra una manera más simple de modelar el problema, sería la mejor alternativa para mejorar la performance del algoritmo.