Cómputo Evolutivo
Tarea 4

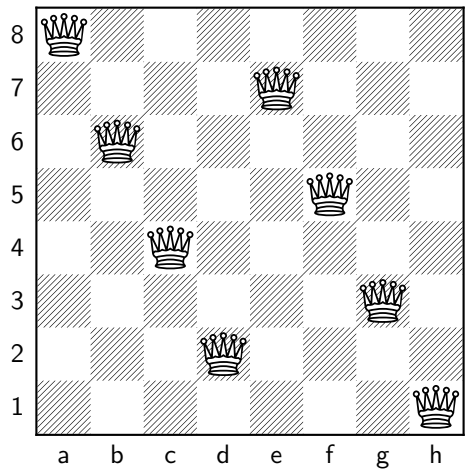
Adrián García Pérez
315131224

1 El problema de las N reinas

El problema de las N reinas consiste en encontrar la posición de N piezas de ajedrez en un tablero de $N \times N$, siendo estas piezas, reinas. La particularidad del problema es que las reinas no deben atacarse entre ellas, y esto es difícil de conseguir debido a que las reinas pueden moverse en cualquier dirección cualquier cantidad de veces, por lo que una solución no debe tener a dos reinas en la misma columna, fila o diagonal. Los ejemplares más comunes son aquellos con configuraciones de 8 reinas en un tablero de 8×8 reinas, donde un algoritmo genético generalmente encuentra soluciones rápidamente.

2 Representación de soluciones

La representación de las soluciones será una lista de N elementos, donde cada elemento será un número entero entre 0 y $N-1$, el cual indica el número de fila donde se encuentra posicionada cada reina. Por ejemplo, para $N = 8$ y la solución $[0, 2, 4, 6, 1, 3, 5, 7]$, el tablero correspondiente es:



Notemos que cada recuadro del tablero se identifica por un número y una letra. Además, la relación entre el arreglo y el dibujo no es directa, ya que debemos tomar N y restarle el elemento del arreglo para encontrar su posición en el tablero. Por ejemplo, para la segunda columna, $N - 2 = 8 - 2 = 6$; el número 2 en el arreglo indica que el tablero tiene una reina en la casilla b6.

Esta forma de representar las soluciones nos permitirá trabajar más fácilmente en el algoritmo genético al momento de cruzar y mutar nuestras soluciones. Si usaramos una matriz indicando en qué casillas hay una reina, tendríamos muchos espacios con 0 y la complejidad para hacer operaciones sobre las soluciones aumentaría de forma innecesaria.

El tamaño de búsqueda para las soluciones se ve reducido a comparación de otras representaciones, como el caso de una matriz, ya que esa representación podría provocar que más de una reina se ubique en una columna, y nuestra representación evita eso al ubicar una reina por columna. Así, nuestro espacio de búsqueda pasa a ser uno basado en permutaciones, provocando que tenga un tamaño de $n!$, reduciendo mucho la complejidad, aunque nuestro algoritmo no busca todas esas posibles respuestas porque no es de naturaleza de *fuerza bruta*. La implementación de la solución se encuentra en el archivo `Member.py`.

3 Función de Evaluación

La función de evaluación correspondiente a nuestras soluciones al problema de N reinas deben ser tales que, dada una solución, nos indique qué tan "buena" o "alejada" es la solución con respecto a nuestro objetivo.

Recordemos que el objetivo del problema de las N reinas es encontrar una configuración tal que ninugna reina ataque a otra. Si nuestra función de evaluación se basa en contar el número de reinas que se atacan entre sí, entonces nuestro objetivo sería minimizar la función, es decir, buscar que su valor sea 0 (no puede ser negativo).

Así, podemos considerar que una solución es "buena" entre más pequeño sea el valor de dicha función al evaluar la solución.

Por ejemplo, para el tablero indicado en la sección anterior, la función de evaluación da un resultado de 1, indicando que solamente hay un par de reinas que se atacan entre sí, es decir, el número de ataques es igual a 1 (la reina a8 y la h1).
La implementación de la solución se encuentra en el archivo `Member.py`, en el método `get_fitness`.

4 Algoritmo Genético

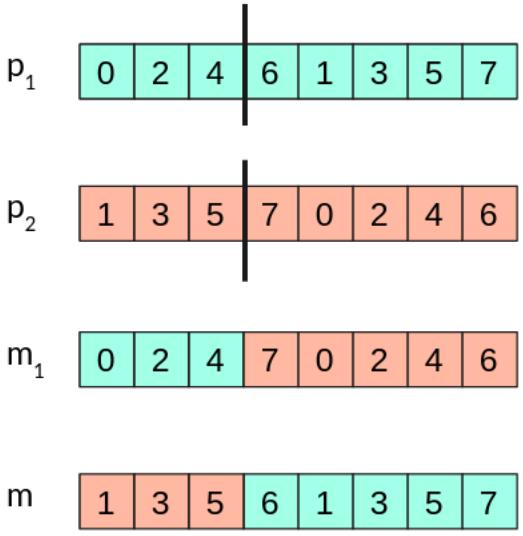
La implementación del algoritmo se encuentra en el archivo `Population.py`.

La selección de padres consiste en tomar las soluciones que serán la base para la creación de una nueva generación de soluciones. Como se indica en la especificación de la tarea, se realiza un torneo binario. Esto es, se eligen aleatoriamente 2 soluciones de la población actual, y se hacen competir; es decir, tomamos la solución con el mejor valor de *fitness* (más bajo) y se añade a una lista de *padres*. Esto se encuentra implementado en el método `tournament`.

Después, se realiza la operación de cruza, que consiste en generar a los nuevos miembros de la siguiente generación tomando a los padres previamente seleccionados. La cruza se realizará condicionándola con una probabilidad. Se genera un número aleatorio entre 0 y 1, y si es menor igual a la proba de cruza, se obtienen dos hijos diferentes a los padres, y en otro caso, se hace una copia exacta de los padres como nuevos hijos.

En mi implementación opté por utilizar un operador de cruza de un punto, es decir, tomamos dos soluciones padres p_1, p_2, y generamos un número aleatorio r_i entre 0 y la longitud de la solución (la longitud de los genes). Posteriormente, creamos dos nuevas soluciones, donde la primera tiene los genes $p_1[: r_i]+p_2[r_i :]$ y la segunda tiene los genes $p_2[: r_i]+p_1[r_i :]$. Esto se encuentra implementado en la función `cross`.

Por ejemplo, si $p_1 = [0, 2, 4, 6, 1, 3, 5, 7]$, $p_2 = [1, 3, 5, 7, 0, 2, 4, 6]$ y $r_i = 3$, entonces los dos hijos resultantes son $m_1 = [0, 2, 4, 7, 0, 2, 4, 6]$ y $m_2 = [1, 3, 5, 6, 1, 3, 5, 7]$:



Posteriormente, se recorre la lista de los hijos creados, y con una probabilidad dada, se hace mutar un valor de sus genes. Es decir, para la solución $s = [3, 1, 4, 7, 2, 7, 0, 2]$, se recorre cada uno de sus elementos y aleatoriamente se cambia por otro valor entre 0 y $N - 1$, evitando que la mutación provoque que la solución deje de ser válida (por tener elementos negativos o mayores a $N - 1$). Esto se encuentra implementado en el método `mutate`.

El último paso consiste en remplazar la población anterior por la nueva generación. Existen muchas maneras de hacerlo, y por la forma que opté usar en mi implementación fue por tomar al mejor individuo de la generación actual (para cumplir con el elitismo requerido en la tarea) y remplazar al resto de miembros de la población actual por la nueva población. Esto se encuentra implementado en el método `replace`

5 Experimentación

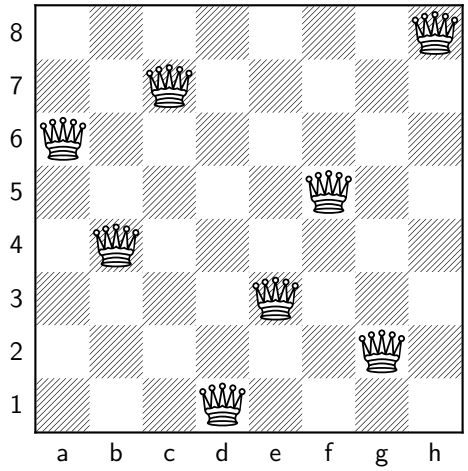
Se realizaron 25 pruebas para valores de N entre 8 y 12 y 5 pruebas para valores de N entre 13 y 20. Como nota, observamos que el tamaño de la población es igual a tres veces el número de reinas. A continuación se muestra una tabla con ejemplos de cada ejecución:

# reinas	Probabilidad de cruza	Probabilidad de mutación	# máximo de iteraciones	Tamaño de población
8	0.9	0.5	1000	24
9	0.9	0.5	1000	27
10	0.9	0.5	1000	30
11	0.9	0.5	1000	33
12	0.9	0.5	1000	36
13	0.9	0.5	1000	39
14	0.9	0.5	1000	42
15	0.9	0.5	1000	45
16	0.9	0.5	1000	48
17	0.9	0.5	1000	51
18	0.9	0.5	1000	54
19	0.9	0.5	1000	57
20	0.9	0.5	1000	60

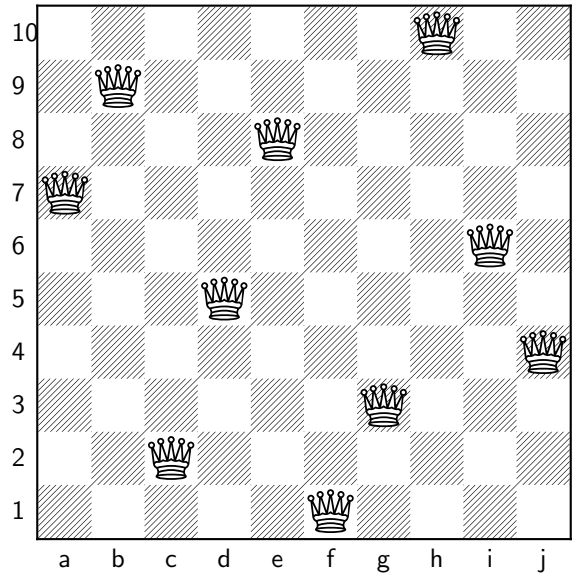
Y ahora se muestra una tabla de resultados de todos los experimentos. Notemos que no siempre se llegan a las mil iteraciones (sobre todo para valores pequeños de N), debido a que opté por detener la ejecución cuando el valor *fitness* del mejor miembro fuera igual a 0.

# reinas	# de repeticiones	Promedio de iteraciones	Tiempo de ejecución promedio	Porcentaje de éxito
8	25	471.84	1.55s	72%
9	25	327.48	1.37s	88%
10	25	690.28	3.65s	44%
11	25	544.32	3.49s	56%
12	25	670.4	5.53s	52%
13	5	640.6	6.46s	40%
14	5	642.6	7.96s	80%
15	5	822.0	12.07s	20%
16	5	494.6	8.75s	80%
17	5	621.4	12.55	60%
18	5	649.0	15.86s	60%
19	5	933.8	25.87s	20%
20	5	892.0	28.69s	40%

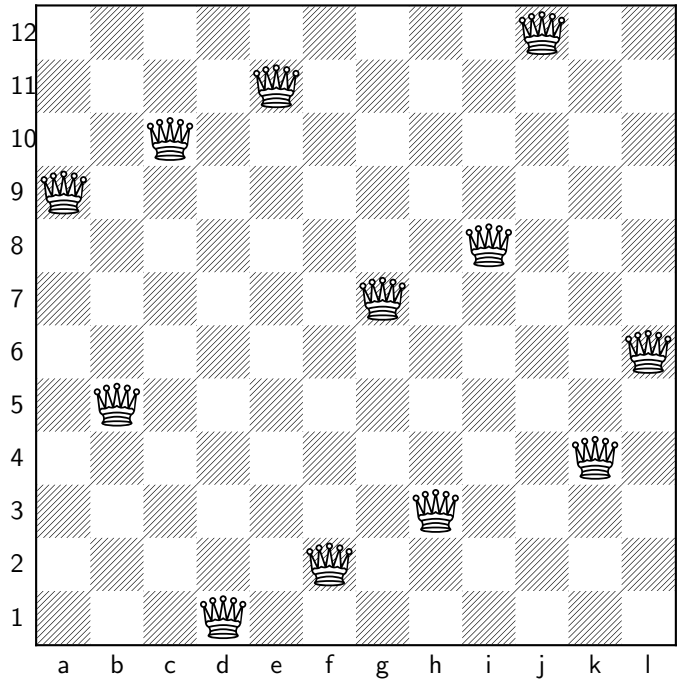
Finalmente, el programa cuenta con una funcionalidad para poder expresar soluciones en código de LaTeXpara mostrar un tablero correspondiente. Ahora, mostramos algunas soluciones encontradas:



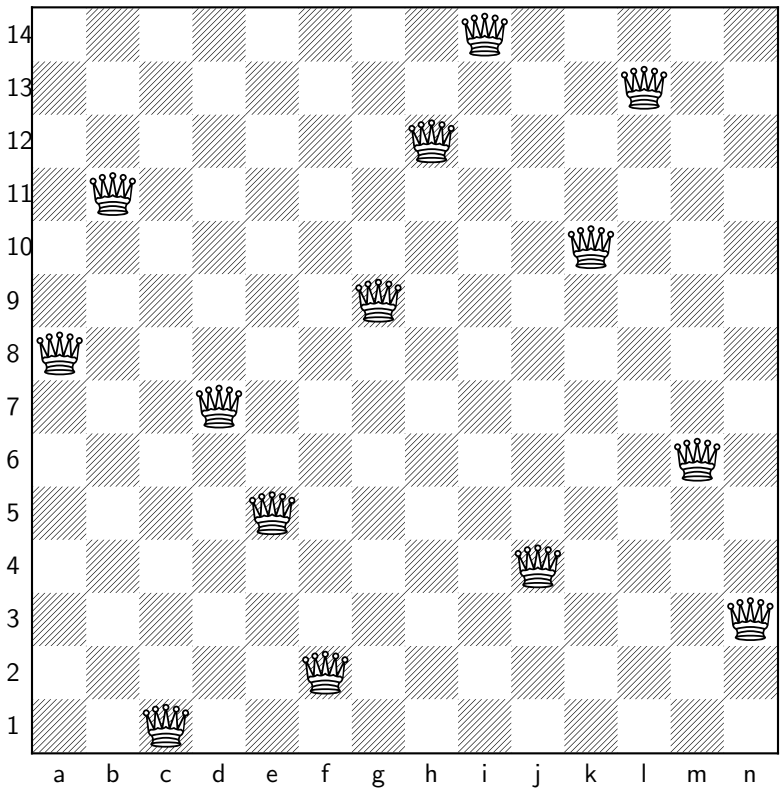
Solución: `[2, 4, 1, 7, 5, 3, 6, 0]`



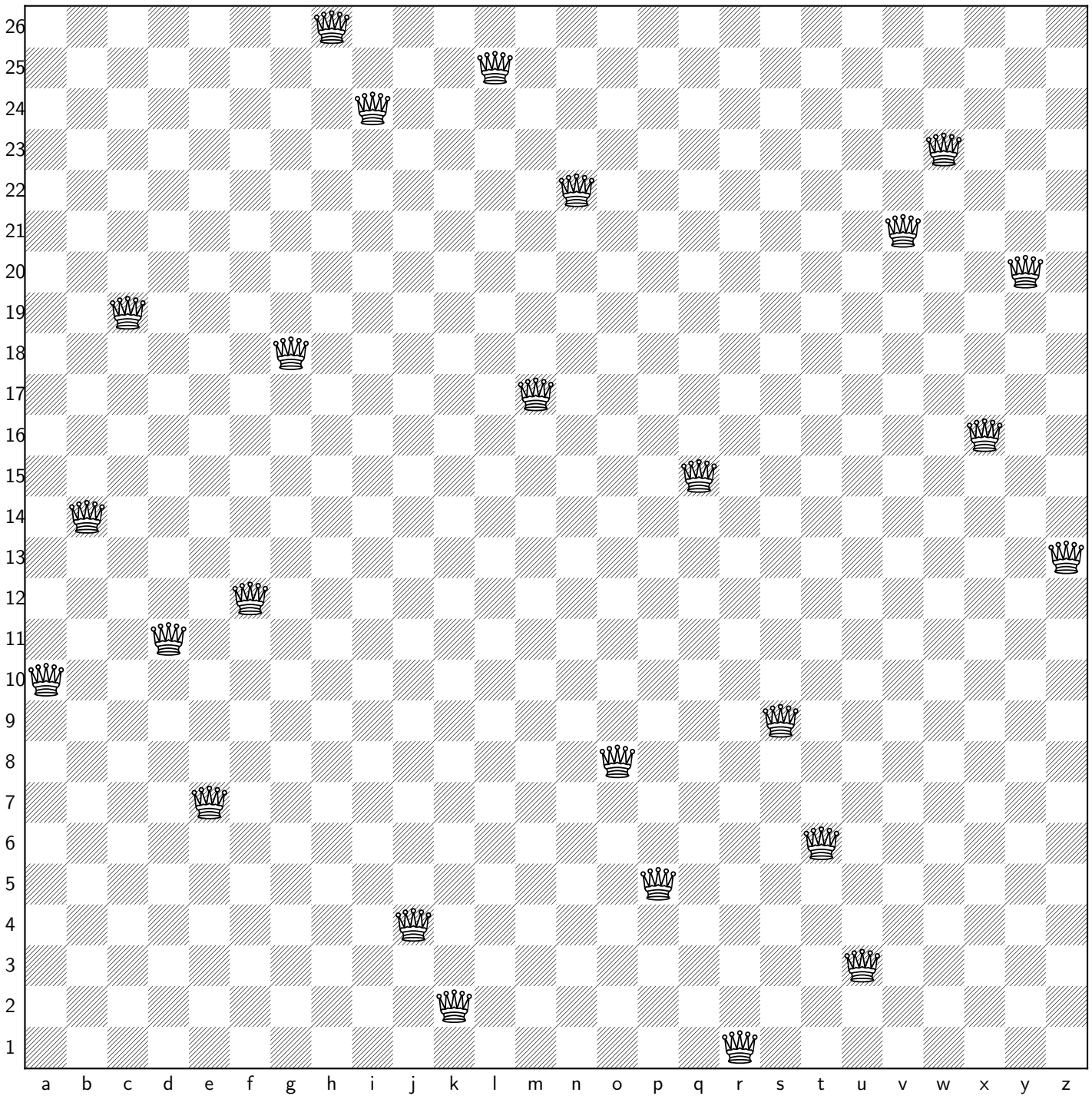
Solución: `[3, 1, 8, 5, 2, 9, 7, 0, 4, 6]`



Solución: `[3, 7, 2, 11, 1, 10, 5, 9, 4, 0, 8, 6]`



Solución: `[6, 3, 13, 7, 9, 12, 5, 2, 0, 10, 4, 1, 8, 11]`

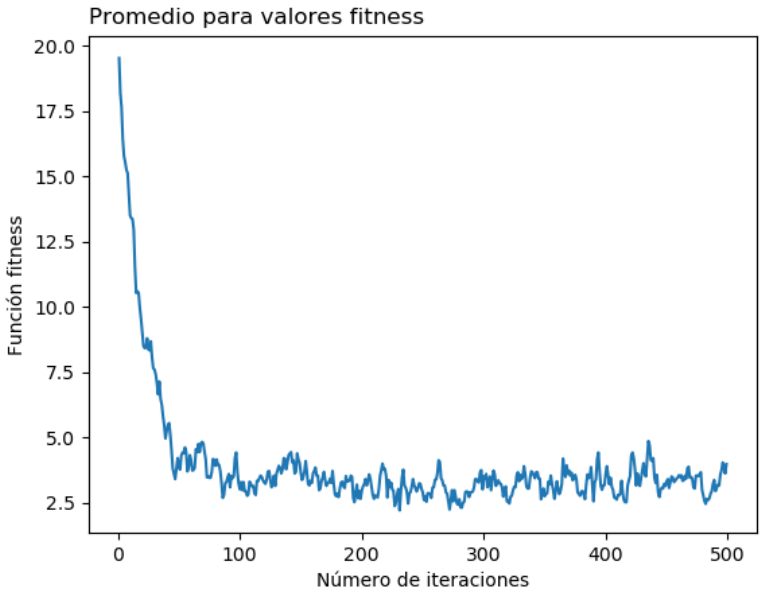


Solución:

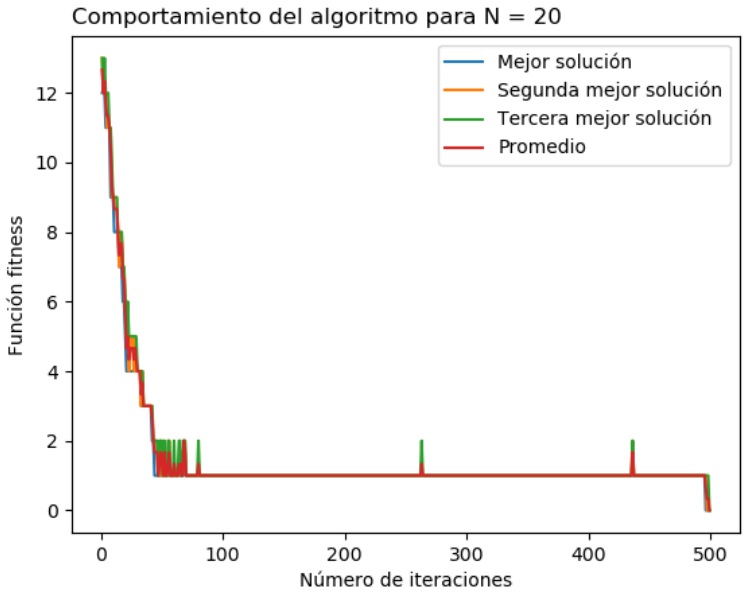
```
[16, 12, 7, 15, 19, 14, 8, 0, 2, 22, 24, 1, 9, 4, 18, 21, 11, 25, 17, 20, 23, 5, 3, 10, 6, 13]
```

Finalmente, opté por generar tres tipos de gráficas: una que muestra el promedio de la función fitness de toda una población a lo largo de una ejecución, otra que muestra el comportamiento de la función fitness para las b mejor soluciones (donde b puede cambiarse como parámetro del algoritmo) y otra que identica a la anterior pero solo con la mejor solución.

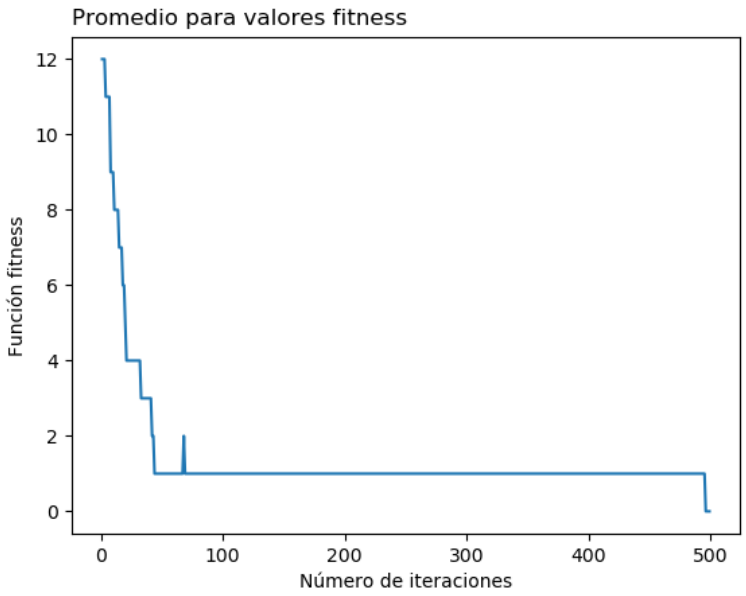
Por ejemplo, esta es una ejecución para 20 reinas:



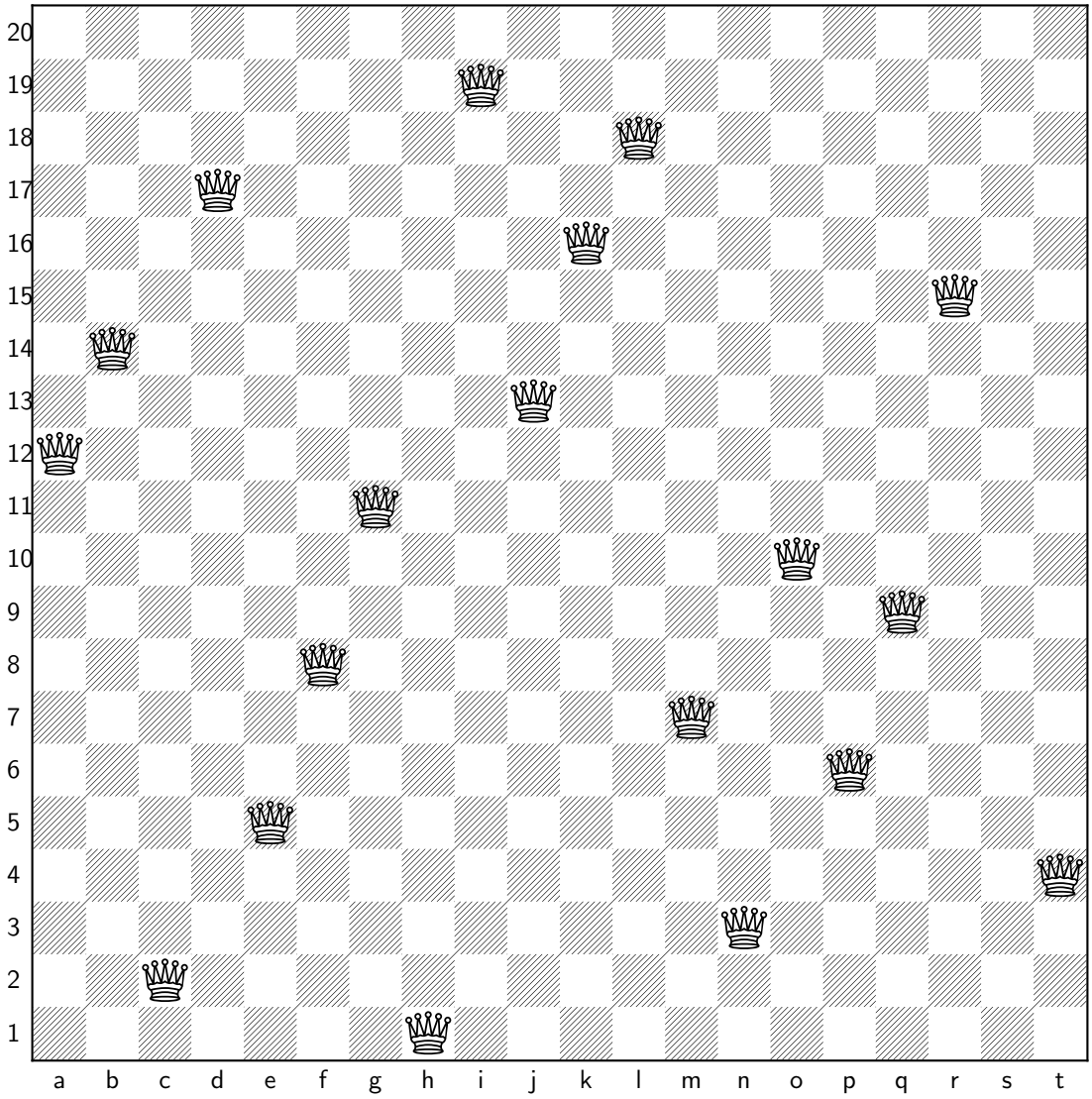
En esta gráfica se ve el promedio de los valores de *fitness* de toda la población



En esta gráfica se ve el comportamiento de las 3 mejores soluciones en la ejecución, así como el promedio de las mismas.



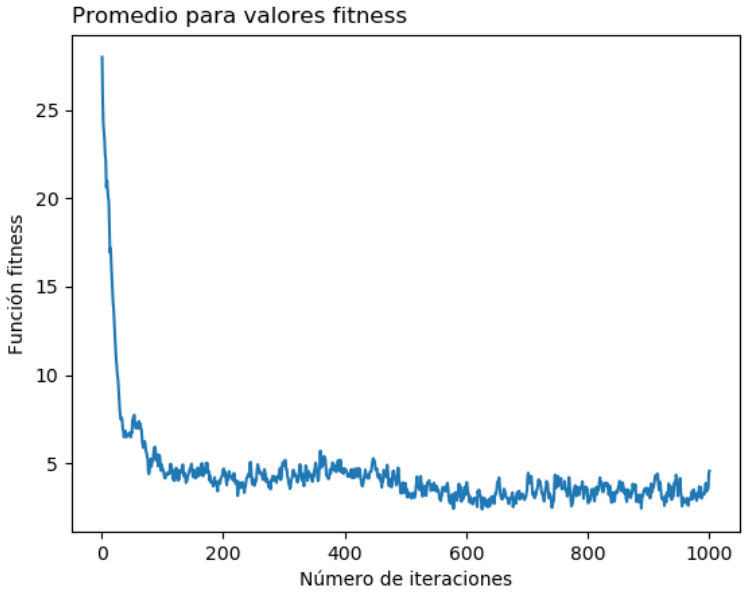
En esta gráfica se ve el comportamiento de la función *fitness* del mejor individuo de la población.



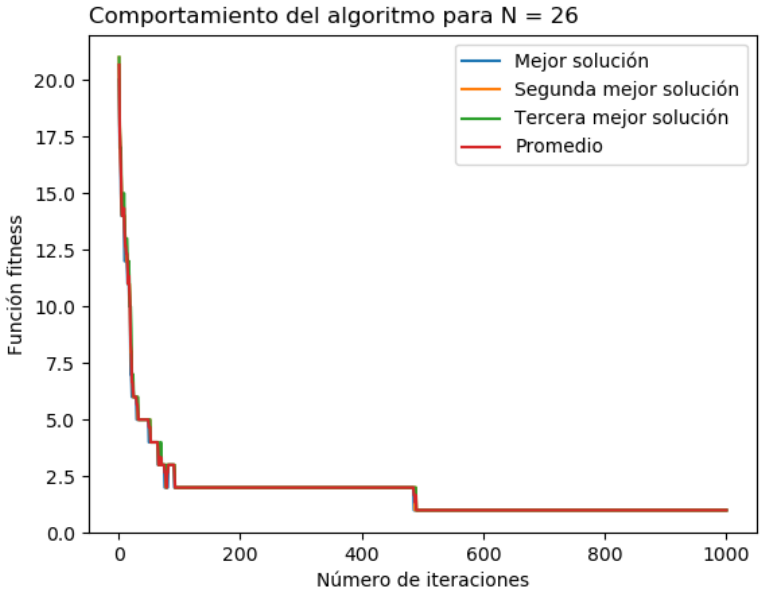
Solución:

```
[12, 14, 2, 17, 5, 8, 11, 1, 19, 13, 16, 18, 7, 3, 10, 6, 9, 15, 0, 4]
```

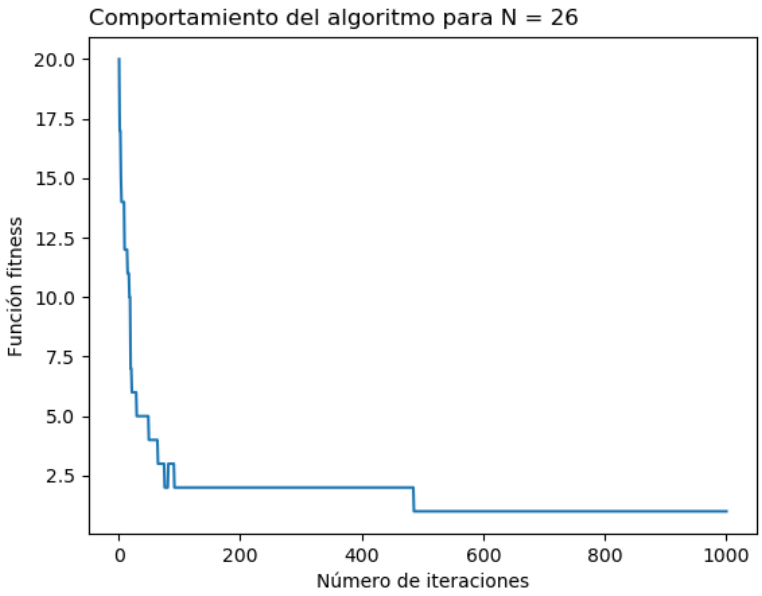
Una ejecución para 26 reinas, que no resuelve el problema por 1 conflicto entre reinas (reina j1 y u1):



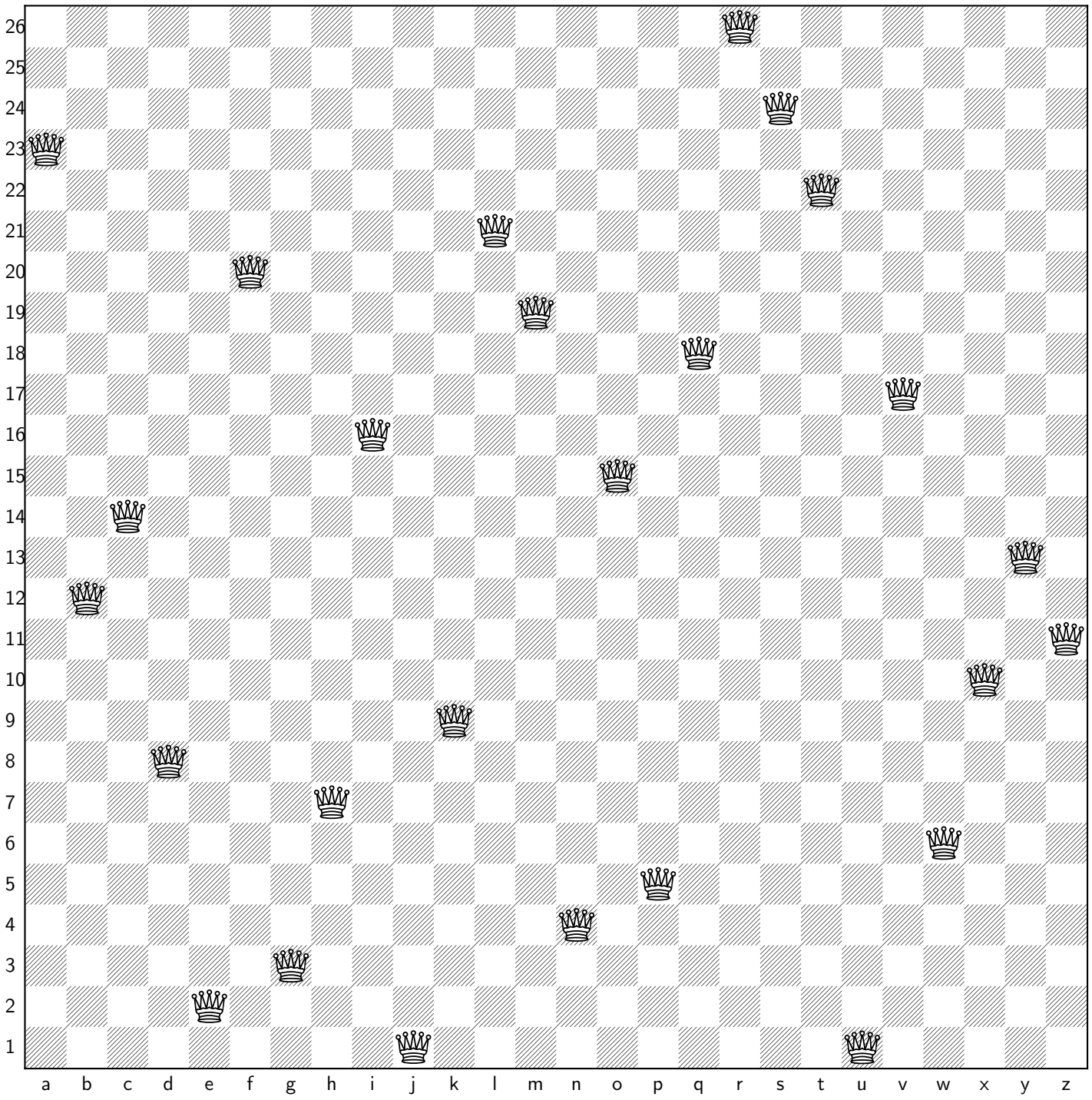
En esta gráfica se ve el promedio de los valores de *fitness* de toda la población



En esta gráfica se ve el comportamiento de las 3 mejores soluciones en la ejecución, así como el promedio de las mismas.



En esta gráfica se ve el comportamiento de la función *fitness* del mejor individuo de la población.



Solución:

```
[3, 14, 12, 18, 24, 6, 23, 19, 10, 25, 17, 5, 7, 22, 11, 21, 8, 0, 2, 4, 25, 9, 20, 16, 13, 15]
```

6 Comentarios y conclusiones

El trabajo me pareció muy interesante debido a que, a pesar de haberlo conocido previamente en Inteligencia Artificial, ahora tuvimos un verdadero enfoque como un algoritmo genético, y pudimos verlo con más profundidad y me gustó. Además, me pareció interesante automatizar un poco el proceso para generar el código en LaTeXpara los tableros solución, aunque fue relativamente sencillo pero ayuda mucho a visualizar el resultado.