Cristalización Simulada jugando Ajedrez

Nicolás Robayo Pardo[1]

[1] Universidad de los Andes
Cra 1 Nº 18A - 12, 111711, Bogotá, Colombia
n.robayo10@uniandes.edu.co

Resumen

Esta tarea busca crear una heurística que halle una solución al problema de las n-reinas resuelto por Carl Gauss en 1850. Este busca posicionar en un tablero de ajedrez $n \times n$ n reinas sin que estas puedan atacarse. Este problema tiene varias soluciones fundamentales para tableros de todos los tamaños. Esta tarea usará la metaheurística de la cristalización simulada (*simulated annealing)* para encontrar una primera solución a este problema para tableros de tamaños de 5 a 500. Se usará un análisis estadístico para calibrar los diversos parámetros de la cristalización simulada y se concluirá sobre la efectividad de esta metaheurística para resolver este problema junto con los mejores parámetros por tipo de tablero.

1 Introducción

El problema de las n-reinas fue propuesto por Max Bezzel en 1848 para el tablero de 8×8. Este propone buscar un tablero de ajedrez con el mayor número de reinas puestas sin que se ataque entre ellas. El problema fue resuelto dos años después por Carl Friedrich Gauss quien probo que hay 92 soluciones (12 de ellas distintas) y que el máximo número es n [1]. De estas 12 soluciones distintas sale el restante a través de rotaciones o reflexiones. A continuación, mostramos un diagrama con estas 12 soluciones.

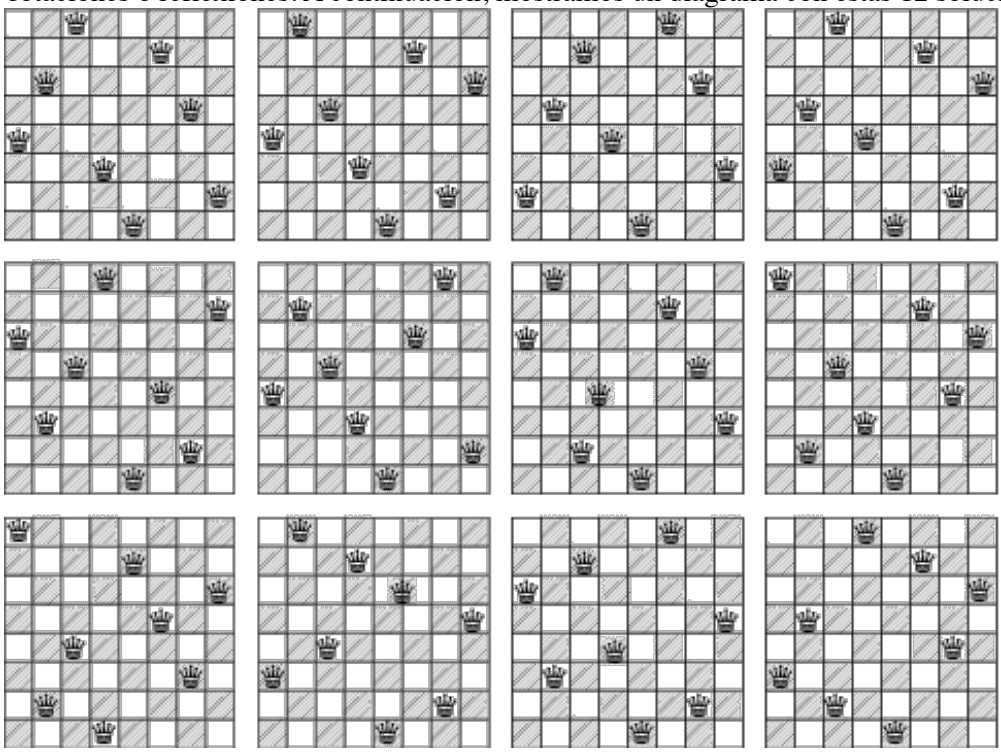


Ilustración 1 Soluciones únicas del tablero 8x8 [11]

Es conocido que la solución es $n - 1$ para tableros $n = 2\ o\ n = 3$ y n para todos los demás tableros.

Entre las aplicaciones conocidas de este problema se encuentran: la colocación de transmisores/receptores de comunicación sin que tengan interferencia entre ellos. Una solución del problema de n-reinas proporciona una colocación donde los receptores se pueden comunicar en 8 direcciones distintas sin interferencia. Otras aplicaciones incluyen prevenir trancones, control de tráfico e integración de circuitos en placas integradas.

El número de soluciones a este problema para los primeros 27 tableros son las siguientes:

N	A(N)
0	1
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884
21	314666222712
22	2691008701644
23	24233937684440
24	227514171973736
25	2207893435808352
26	22317699616364044
27	234907967154122528

Tabla 1 Número de soluciones diferentes al problema de n-reinas para tableros de n[1]

Como podemos apreciar, el número de soluciones crece casi exponencialmente con el tamaño de tablero. Además, un análisis del espacio de soluciones hecho por Masehian et al. muestra que estas se encuentran de manera aleatoria uniforme en el espacio. Las soluciones óptimas se encuentran diseminadas por todo el espacio de solución con baja variabilidad en la calidad y sin correlación entre la calidad y la distancia entre ellas. Es por esto por lo que un algoritmo de búsqueda local se puede desempeñar muy bien al tratar de encontrar una solución al empezar desde cualquier punto del espacio de solución. Esto se hace patente en el uso de otro algoritmo como la metaheurística del recocido simulado que se desempeña bien pero pierde mucho tiempo "enfriando" el algoritmo. Una metaheurística de poblaciones se tarda en converger a una primera solución dado que toma mucho tiempo en generar candidatos a través de operadores que no añaden eficiencia a la operación, pero luego se apalancan de un set diverso de soluciones para seguir encontrando soluciones al problema [2].

1.1 Definición del problema de n-reinas

El problema de n-reinas se puede definir con una pregunta muy sencilla: ¿se pueden colocar n reinas en

[1] Sloane, N. J. A. Secuencia A000170/M1958 en "The On-Line Encyclopedia of Integer Sequences."

un tablero de ajedrez de $n \times n$ de forma que ninguna reina ataque a otra? Una reina se dice que es atacada si hay otra que se puede mover legalmente hacia su posición.

1.2 Modelamiento Matemático

Se expone a continuación la formulación de Hoffman et al. [3]

1.2.1 Parámetros

$$n: Tamaño\ del\ tablero$$

1.2.2 Variables de decisión

$$x_{ij} = \begin{cases} 1 & si\ hay\ un\ reina\ ocupado\ la\ casilla\ (i,j) \\ 0 & dlc \end{cases}$$

1.2.3 Función Objetivo

Se busca maximizar el número de reinas en el tablero.

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$$

1.2.4 Restricciones

(0) Naturaleza de variables

$$x_{ij} \in \{0,1\} \qquad \forall i,j = 1,\dots,n$$

(1) Solo puede haber una reina por fila.

$$\sum_{j=1}^{n} x_{ij} \leq 1 \ \forall\ i = 1,\dots,n$$

(2) Solo puede haber una reina por columna.

$$\sum_{i=1}^{n} x_{ij} \leq 1 \ \forall\ j = 1,\dots,n$$

(3) Solo puede haber una reina por diagonal

$$\sum_{i=1}^{n} \sum_{j=1 \mid i+j=k}^{n} x_{ij} \leq 1 \ \forall\ k = 2,\dots,2n$$

(4) Solo puede haber una reina en la otra diagonal

$$\sum_{i=1}^{n} \sum_{j=1 \mid i-j=k}^{n} x_{ij} \leq 1 \ \forall\ k = 1-n,\dots,n-1$$

2 Revisión de Literatura

Resolver este problema por fuerza bruta no tendría sentido dado que la complejidad está en el orden de $O(n!)$. Muchos de los trabajos recientes se enfocan en encontrar algoritmos o técnicas de solución para encontrar una solución de manera rápida para cualquier tamaño de tablero. Uno de ellos muy utilizado es el mecanismo de retroceso el cual construye soluciones candidatas y retrocede algunos pasos al concluir que la solución que construye no puede convertirse en una solución final. Este algoritmo es muy útil para encontrar soluciones, pero se ha demostrado que para tamaños de tableros muy grandes no hay mucha variedad en las soluciones encontradas pues estas no son muy distintas entre ellas [4]. Además, su complejidad crece con el tamaño del problema [5].

La gran mayoría de los algoritmos clásicos se vuelven ineficientes para tablero de tamaños $n > 20$ por lo que el uso de metaheurísticas se ha vuelto atractivo para encontrar una solución en un tiempo computacional pequeño para problemas muy grandes, pero renunciando a encontrar una solución óptima. Por ejemplo, el uso de Recocido simulado, búsqueda tabu y algoritmos genéticos son de las heurísticas más usadas. Entre estas metaheurísticas la mejor resulta ser el recocido simulado debido a su menor número de veces que calcula la función objetivo mientras que en las otras dos metaheuristicas se calcula varias veces en cada iteración [6]. La búsqueda Tabu es buena para encontrar varias soluciones dado que, a diferencia de las otras dos metaheurísticas, no pierde eficacia después de encontrar la primera solución. Se recomienda entonces usar el recocido simulado en casos donde el tamaño del tablero sea muy grande (n>500000).

El recocido simulado es una metaheurística que empieza buscando en el espacio de solución de la solución inicial e iterativamente se acerca a la solución óptima a través de cambios de estado aceptando mejores soluciones o peores con una probabilidad de $P(\Delta E, T) = e^{-\frac{f(s')-f(s)}{T}}$ en donde el parámetro de control T simboliza la temperatura descendiente al enfriar un solido a través de un recocido y $f(s)$ es la función objetivo. La temperatura es reducida por un cronograma de enfriamiento. Este algoritmo continúa hasta encontrar la solución óptima o agotar un número máximo de iteraciones sin cambio en la solución [7].

Algunos parámetros importantes del recocido son la temperatura inicial y el cronograma de enfriamiento. La temperatura inicial debe ser lo suficientemente alta como para aceptar cualquier solución en las primeras iteraciones y está en términos de la función objetivo del problema. Algunos cronogramas de enfriamiento usados son el lineal, el geométrico y el logarítmico [2]. En el cronograma lineal, la temperatura es actualizada por restando a la temperatura actual T un escalar β o una fracción de este dependiendo de la iteración i:

$$T = T_o - i \cdot \beta$$

Otro cronograma de enfriamiento es el geométrico el cual es uno de los más populares. Este usa un factor $alpha \in (0,1)$ que es multiplicado iterativamente a la temperatura actual. Este factor debe ser lo suficientemente alto para acelerar la búsqueda y lo suficientemente bajo para no converger tempranamente. La ecuación usada es la siguiente:

$$T = \alpha \cdot T$$

El último cronograma de enfriamiento es el logarítmico. Este es especialmente lento, pero tiene la propiedad de converger a la solución óptima.

$$T_i = \frac{T_o}{\log(1+i)}$$

La búsqueda local es uno de los mejores algoritmos para encontrar soluciones al problema de las n-reinas. Entre alguno de los mecanismos de transición se encuentran los mínimos conflictos y Random Swap. El operador de mínimos conflictos crea una nueva solución al escoger una columna o fila aleatoria y calcular el número de conflictos de cada casilla. Luego se mueve la reina ubicada en esa fila o columna a esta casilla con mínimos conflictos y posteriormente se mueve la reina ubicada en la columna o fila de la nueva casilla a una posición con mínimos conflictos [8].

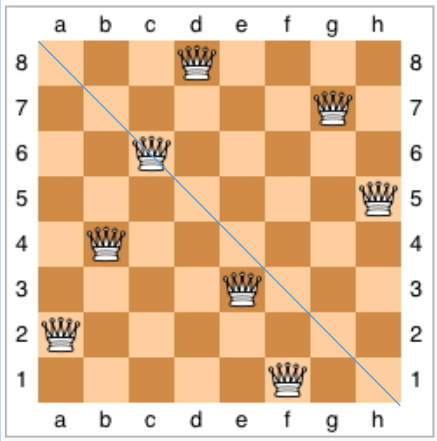
Un operador muy sencillo es el Swap aleatorio el cual intercambia dos reinas de posición. El tamaño de este vecindario es $\frac{n(n-1)}{2}$. Este operador no tiene certeza de mejorar la función objetivo por lo que Masehian et al. [2]crearon el Swap efectivo el cual actúa de forma inteligente considerando el número de ataques al hacer los intercambios. Este operador comienza contando el número de conflicto que hay en la diagonal principal y si este número no es 0 utiliza esta diagonal para hacer el cambio. Si no encuentra conflictos va buscando en el resto de las diagonales hacia arriba y abajo y en ausencia de conflictos mira la diagonal cruzada principal hasta revisar todas las diagonales. Luego, al encontrar una diagonal con conflictos, escoge una reina ubicada en la diagonal al azar y otra que no este en la diagonal. Este swap no garantiza un mejoramiento en la función objetivo, pero por lo menos se dedica a disminuir los conflictos en la diagonal seleccionada. Experimentalmente se puede concluir que reduce el número de conflictos mucho mejor que el Swap aleatorio.

Existen otras metaheurísticas que han sido usadas para resolver este problema. Homaifar en 1992 describió lo bueno que los operadores de algoritmos genéticos solucionaban problemas combinatorios y de satisfacción de restricciones como el de n-reinas, además presentaron resultados para tamaños n<200 [9]. Dirakkhunakon y Suansook compararon los resultados del algoritmo de recocido simulado con el mismo algoritmo mejorado iterativo. Los resultados mostraron que el algoritmo modificado encontraba mejores soluciones para el problema de n-reinas. [10]

3 Metodología de solución

Para esta tarea implementaremos en Python una metaheurística de Cristalización Simulada usando como operador de búsqueda local el swap efectivo y como mecanismo de enfriamiento la formula geométrica. Adicionalmente no usaremos una longitud L de iteraciones por pasos dependiente del número de iteraciones ya que su incremento hace que el algoritmo no converja para tableros muy grandes.

Dadas las propiedades de los movimientos de la reina, una solución tendrá una reina por columna y por fila. Usando esto como codificación creamos un vector de n posiciones simbolizando cada columna. En este vector se guardará un número de 1 a n que simboliza la fila donde se ubica la reina en cada columna. Esta codificación garantiza que no haya reinas compartiendo filas o columnas y hace que el único conflicto por el que preocuparse sean las diagonales.



A	B	C	D	E	F	G	H
$fila\ 2$	$fila\ 4$	$fila\ 6$	$fila\ 8$	$fila\ 3$	$fila\ 1$	$fila\ 7$	$fila\ 5$

Diagonal Principal:{ 2:{$fila\ 2$}, 5:{$fila\ 4$}, 6:{$fila\ 1$}, 7:{$fila\ 3$}, 8:{$fila\ 6$}, 11:{$fila\ 8$}, 12:{$fila\ 5$}, 13:{$fila\ 7$}}

Diagonal Cruzada: {3: {$fila\ 1$}, 5: {$fila\ 5$}, 6: {$fila\ 3$}, 8: {$fila\ 7$}, 9: {$fila\ 2$ }, 10: {$fila\ 4$} , 11: {$fila\ 6$ }, 12: {$fila\ 8$} }

Table 2 Codificación Usada

De esta codificación se desprende otra que conserva esta propiedad y ayuda a calcular el número de conflictos en las diagonales. Se tendrá un set por cada diagonal del juego de ajedrez guardado en un diccionario. En este set se incluirá los números de fila de las reinas que están ubicadas sobre esta diagonal. Al haber más de un número en cada set se empezará a contar los conflictos. Es así como el cálculo de la función objetivo se hace simple al solo tener que contar el número de reinas mayores a 1 por cada set y sumarlas. Ahora, para actualizar estos sets al cambiar la solución se hace uso de las siguientes formulas:

$$diagonal_{arriba} = fila + columna$$
$$diagonal_{abajo} = n - columna + fila$$

Esto logra que el cálculo de la función objetivo sea lineal y reduce enormemente los tiempos de ejecución para instancias muy grandes. También ayuda a rápidamente seleccionar la reina a swapear dado que solo se necesitan ver los sets con mas de una fila.

La solución inicial es construida de forma aleatoria ubicando una reina por columna y por vector y luego traduciendo esta codificación a la codificación de diagonales. Hay que tener en cuenta que la codificación inicial (de filas en vector de columnas) está implícita en todo el algoritmo al hacer los swaps de las reinas

conservando la fila y haciendo el intercambio entre columnas. La función objetivo es la suma del número de conflictos que hay en cada diagonal donde los conflictos se miden como el número de reinas mayor a uno que hay presentes en la diagonal.

Varios parámetros son necesarios para el uso del recocido simulado por lo que se hizo un estudio estadístico para tener los mejores parámetros de temperatura inicial, número de iteraciones por paso de temperatura, alpha de decrecimiento de temperatura. El número de MaxIteraciones se dejo en un número muy alto que permitiera al algoritmo converger a una solución óptima ya que al final esto será penalizado por el tiempo computacional.

3.1 Pseudocódigo

Input: n:Tamaño del tablero, T_{init}: Temperatura Inicial, α: Tasa de enfriamiento, L: Número de iteraciones por paso de temperatura, maxiter: Número máximo de iteraciones sin cambio de solución, $initSol$: Solución Inicial
Output: sol:Mejor Solución
iter_no_sol = 0
$sol = initSol$
T = T_{init}
while: iter_no_sol < maxiter:
    for i = 1 to L **do**
        reinaEmproblemada ← Escoger reina en diagonal con conflictos
        otraReina ← Escoger reina en otra diagonal
        cambioFO ← Calcular cambio en FO al hacer Swap
        if cambioFO $> 0 \vee \left(\text{cambioFO} < 0 \wedge e^{-\frac{cambioFO}{T}} > rand(0,1)\right)$ **do**
            sol ← Intercambiar columnas de reinaEmproblemada y otraReina
            Actualizar Función Objetivo
            iter_no_sol = 0
        else
            iter_no_sol +=1
    T = T$* \alpha$
return sol

3.2 Análisis Estadístico

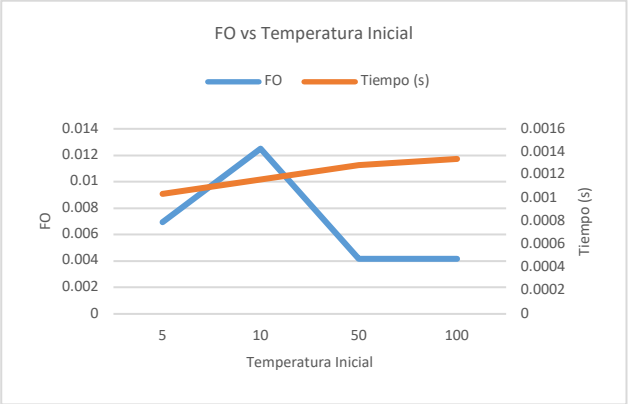
Para calibrar las soluciones se utilizaron los siguientes valores para los parámetros del algoritmo. Para la temperatura inicial se escogieron valores que aceptaran el peor cambio en función objetivo generado por el swap en un rango de 55% a 97% teniendo en cuenta las recomendaciones de la metaheurística incluyendo un valor que acepte todos los swaps en las primeras iteraciones. Para α se usaron valores dentro del rango recomendado $(0.9, 0.8)$. El último parámetro es el número de iteraciones por temperatura, este varia entre 1 el cual cambiaría la temperatura por cada iteración y 20.

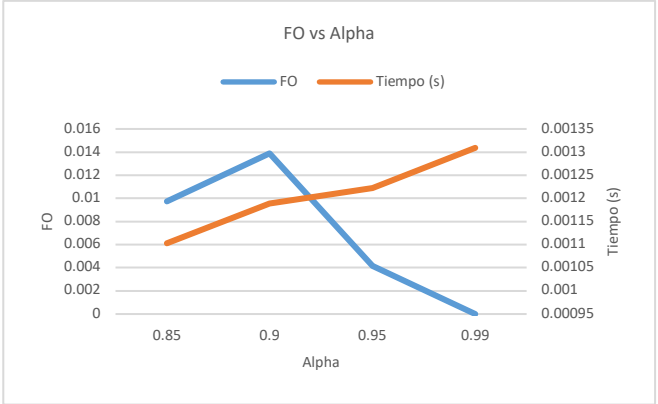
$$T_{init}: \{5,10,50,100\}$$
$$\alpha: \{0.99, 0.95, 0.9, 0.85\}$$
$$L: \{1, 2, 5, 10, 15, 20\}$$

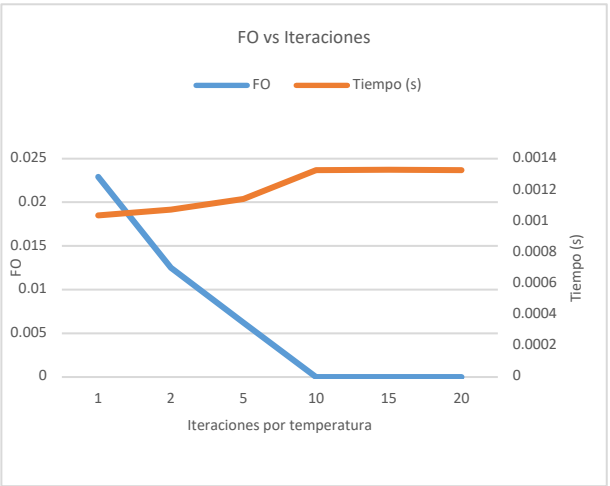
$$P(aceptar\ solucion\ muy\ mala(\Delta = -4)|T = 100) = e^{-\frac{4}{100}} = 96\%$$

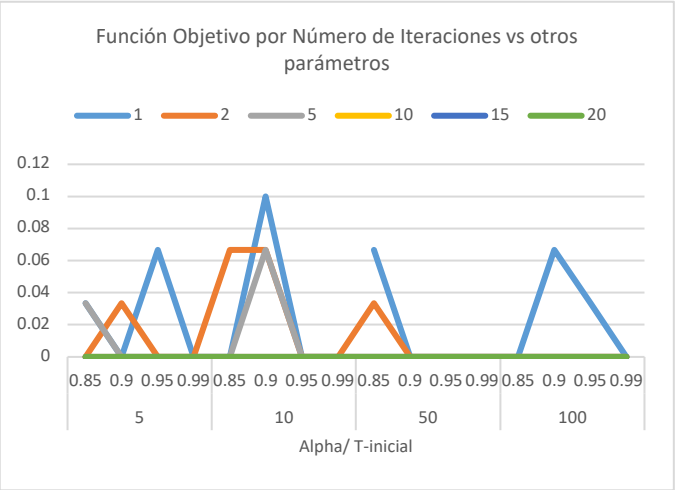
Se realizaron 30 realizaciones del algoritmo por cada combinación de parámetros para cada uno de los tamaños de tablero a analizar $\{8,25, 50, 75, 100, 200, 300, 500\}$. Los resultados indican que para tableros mayores a 50, el algoritmo no es sensible a los parámetros encontrando siempre una solución al

problema variando solamente el tiempo computacional para lograrlo. Es por esto por lo que solo se mostrará resultados de función objetivo para los tableros de 8 y 25 y para el resto se mostraran tiempos computacionales. Para realizar este análisis se uso 500 como el número máximo de iteraciones sin solución para parar el algoritmo.









3.2.1 Tablero 8×8

Para el tablero vemos con un número de iteraciones igual o mayor a 10 o con un α de 0.99 obtenemos fácilmente una solución al problema. Esto debido a que en estas situaciones el enfriamiento es lento y le damos las iteraciones suficientes para que halle el equilibrio en cada paso de temperatura. La combinación de parámetros que obtuvo siempre una solución con el menor tiempo computacional (0.0197s) es:

$$(\alpha = 0.85, L = 1, T_{max} = 100)$$

3.2.2 Tablero 25×25

Para el tablero de 25×25 las únicas combinaciones de parámetros que no lograron llegar a una solución en todos los 30 intentos fueron en su mayoría con temperaturas de 50, iteraciones L de 1 o con $\alpha = 0.9$.

T_{MAX}	A	L
5	**0.85**	20
10	**0.9**	5
10	**0.99**	1

50	**0.85**	1
50	**0.9**	2
50	**0.9**	15
50	**0.95**	1

Table 3 Parámetros que no llegaron a una solución para n=25

El resto de las combinaciones de parámetros llegaron a una solución y a continuación mostramos los tiempos de las mejores 11 combinaciones. Observamos que el mejor parámetro es $T_{max} = 5, \alpha = 0.85, L = 1$

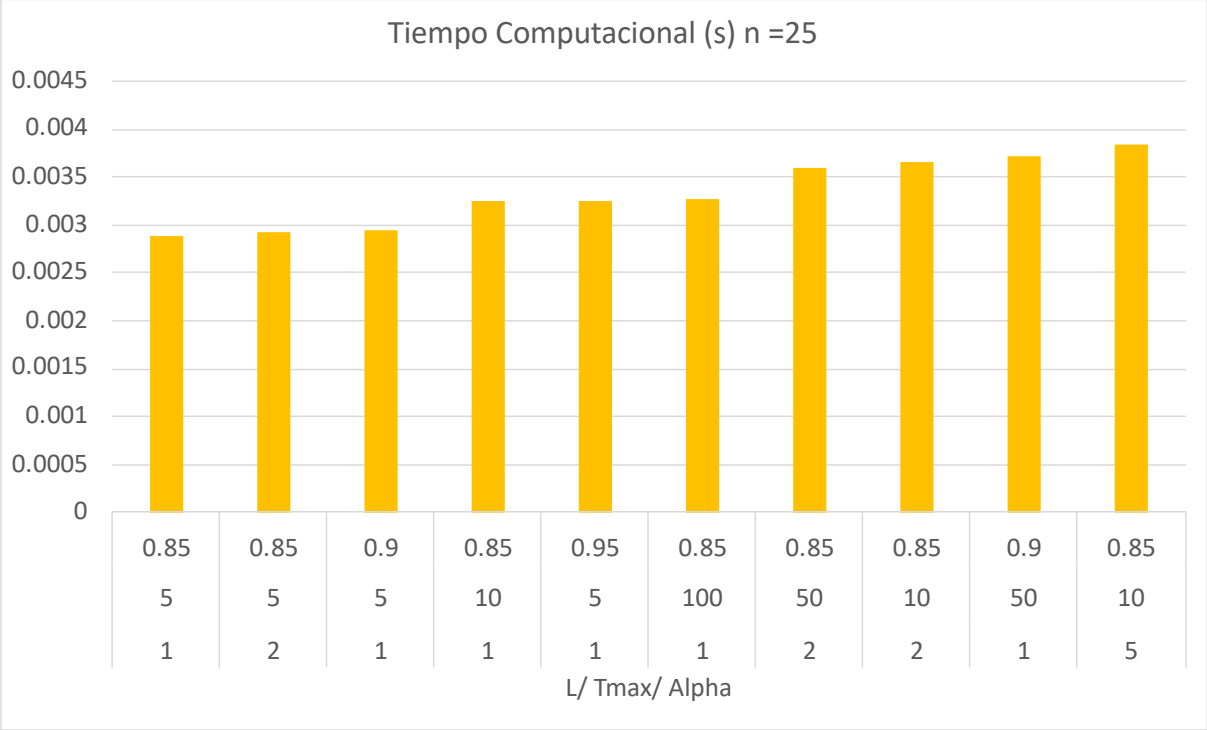
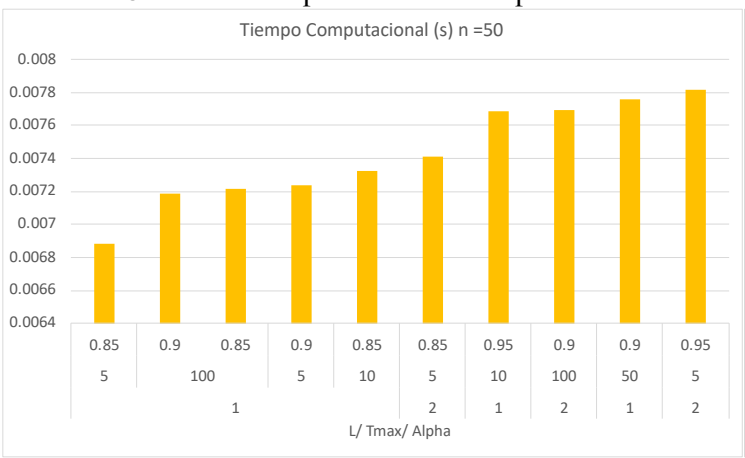
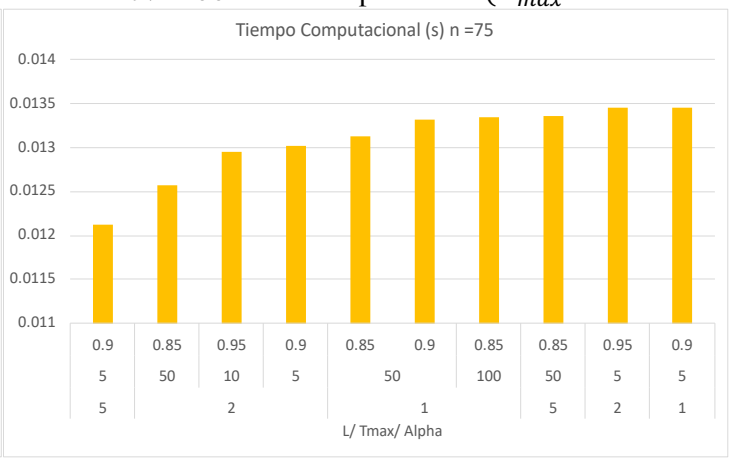


Figure 1 Mejores tiempos computacionales para n=25

3.2.3 Tableros $n \geq 50$

Para los tableros con $n \geq 50$ vemos que la combinación de parámetros ($T_{max} = 100, L = 1, \alpha = 0.9$) esta en las primeras posiciones. (1 o 2) para los tableros de 50, 100 Y 200 Por esta razón se usarán estos parámetros para presentar los resultados finales de tableros $50 \leq n \leq 200$. Para los tableros $n < 50$ se usaran los parámetros ($T_{max} = 5, L = 1, \alpha = 0.9$) dado que los anteriores hacían que el tablero de $n = 8$ no diera siempre una solución óptima. Para las instancias con $n > 200$ se usará el parámetro ($T_{max} =$





$5, L = 15, \alpha = 0.85$) el cual tiene buen desempeño con instancias muy grandes.

3.3 Detalles de Implementación

Se implementaron los algoritmos en Python 3.8 usando una conceptualización de clases para priorizar el ahorro de memoria. La implementación recibe como parámetros el

```
// python HW3.py n Tinit alpha L MAXITER
// para correr heurística con n=500, Temperatura incial de 20, alpha de 0.9 y L = 2, maxiter =5000:
$ python HW3.py 500 20 0.9 2 5000
```

También se genero un archivo ejecutable para equipos que no tengan Python instalado para equipos tipo Linux. Las instancias se pueden correr con un archivo .sh para correr todos los experimentos acá presentados. Se incluyo un README en los soportes de esta tarea que indican como ejecutar estos archivos.

4 Experimentación y Análisis de Resultados

Estas fueron corridas en un procesador dual core “Broadwell” 2.7 GHz Intel “Core i5” (5275U) con 8 GB 1866 MHz LPDDR3 SDRAM corriendo macOS Catalina 10.15.6 con Python 3.8. Dada la aleatoriedad de los algoritmos, estos fueron ejecutados 30 veces por algoritmo para la experimentación y discusión de resultados.

TAMAÑO TABLERO	α	T_{max}	L	TIEMPO (S)	DESVIACIÓN ESTANDAR TIEMPO
8	**0.9**	5	1	0.00075861	0.00064024
25	**0.9**	5	1	0.00294522	0.00154522
50	**0.9**	100	1	0.00718567	0.00220651
75	**0.9**	100	1	0.0140683	0.00452365
100	**0.9**	100	1	0.0206978	0.00695374
200	**0.9**	100	1	0.06606694	0.01659674
300	**0.85**	5	15	0.14870519	0.02805776
500	**0.85**	5	15	0.41250609	0.05394621

Figure 2 Resultados Recocido Simulado en n-reinas

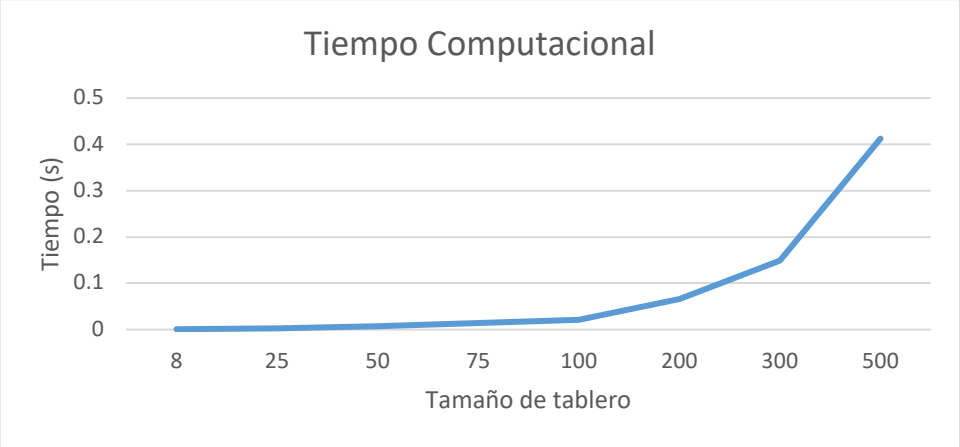


Ilustración 2 Tiempo Computacional

Observamos que para los tableros más pequeños los parámetros óptimos usan un $\alpha = 0.9$ junto con un número de iteraciones por temperatura de 1. Vemos así que el problema requiere un enfriamiento lento sin que tenga que hallar un equilibrio entre temperaturas. Además, para instancias medianas la temperatura inicial es lo suficientemente alta para aceptar cualquier swap en las primeras iteraciones lo que indica que el problema se beneficia de la disminución lenta de aceptación de soluciones malas. Para instancias grandes sin embargo los parámetros usados indican que el problema necesita una temperatura inicial baja (T=15) junto con una disminución rápida de temperatura donde la estabilización por paso de temperatura es alta (15 iteraciones). Vemos entonces que para estas instancias el problema tarda más en buscar un equilibrio por paso de temperatura a costa de una disminución más rápida.

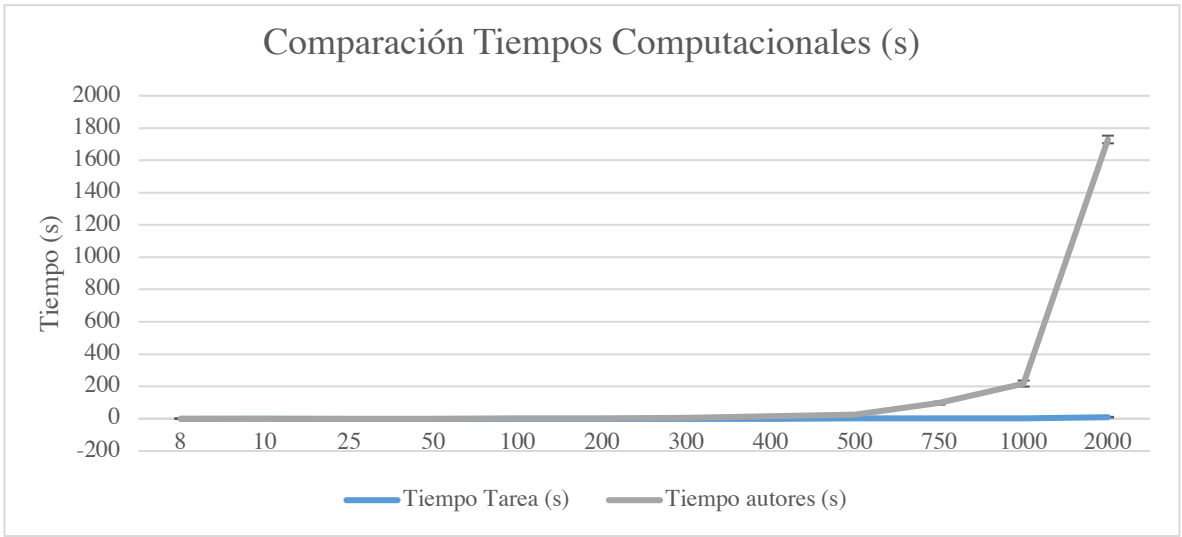
En términos de tiempos computacionales, para las instancias usadas ($n \leq 500$) los tiempos son rápidos, pero se hace evidente que para $n \gg 0$ el tiempo computacional se vuelve de tipo polinómico de grado 2. Para solucionar esto se podría usar en próximos trabajos estructuras de datos no convencionales que agilicen más selección de reinas y calculo de la función objetivo.

El análisis estadístico generado indica que, en realidad, encontrar una solución a este problema es fácil. Tal ves esto sea debido a que las soluciones se encuentran uniformemente esparcidas por el espacio de búsqueda (y por la potencia del operador usado) y por esto no le toma mucho encontrar una solución. Ahora, en términos de tiempo computacional puede que una búsqueda local sea mejor que usar un algoritmo complejo como el recocido simulado ya que esta gasta mucho tiempo al aceptar soluciones infactibles que puede que no sean necesarias para llegar a una solución [2].

5 Comparación de Resultados con la Literatura

Compararemos la metaheurística realizada con una similar descrita por Masehian et al. [2] quienes usan también un recocido simulado con swap efectivo pero usando los parámetros $\alpha = 0.95, T = 5, L = 1$ para todos los tamaños de tablero. Los autores hicieron un *tunning* de los parámetros a través de un análisis TOPSIS que buscaba reducir el número de cálculos de función objetivo y el tiempo de corrida exclusivamente para el tablero de $n = 300$. El análisis TOPSIS usa métricas de separación entre las mejores y peores soluciones para encontrar los mejores parámetros. Los resultados publicados fueron corridos en un Intel Core i7 2.00 GHz CPU con 4.00 GB of RAM con un código en Matlab. A continuación, mostraremos una comparación de los tiempos computacionales corriendo nuestra heurística con los mismos parámetros usados por Masehian et al. para una justa comparación. Adicionalmente usaremos todos los tamaños de tablero usados por los autores

N	TIEMPO METAHEURÍSTICA DESARROLLADA (S)	TIEMPO MASEHIAN ET AL. (S)	DESVIACIÓN ESTÁNDAR METAHEURÍSTICA DESARROLLADA (S)	DESVIACÓN ESTÁNDAR MASEHIAN ET AL. (S)
8	0.002327831	0.07	0.00913459	0.02
10	0.001884121	0.09	0.0017281	0.07
25	0.00370901	0.18	0.00173863	0.06
50	0.007097649	0.27	0.00256976	0.05
100	0.021716209	0.6	0.00539211	0.12
200	0.074340022	1.88	0.01751624	0.31
300	0.164979451	4.01	0.02005417	0.72
400	0.311725637	13.75	0.05372006	1.93
500	0.514141589	26.19	0.16272136	3.22
750	1.044323959	96.39	0.12768053	9.56
100	1.967088071	217.66	0.39996794	18.65
2000	9.249476272	1729.35	2.27172389	23.8



La comparación en tiempos computacionales es demoledora. Evidentemente, aunque sean casi la misma metaheurisitica siendo evaluada con los mismos parámetros, la implementación de esta tarea tiene tiempos muchísimo menores a los logrados por Mashian et al. en el 2013. Esto puede ser debido a la codificación usada en esta tarea, el lenguaje de programación utilizado y la estructura del código usada. En

todo caso, se puede concluir que la metaheurística hecha es mucho mejor que la realizada por los autores.

6 Conclusiones

- Se encontró una solución para todas las instancias requeridas del problema de n-reinas usando la metaheurística del recocido simulado. Para instancias muy grandes, este algoritmo prefiere un tiempo de estabilización entre temperaturas largo y un enfriamiento rápido.
- El operador swap efectivo resulto ser una pieza fundamental del algoritmo al mejorar la escogencia de las reinas a intercambiar al escoger aquellas que ya estuvieran causando conflicto. Además, el uso de estructuras de datos basadas en diagonales y la conceptualización del problema de filas y columnas lograron reducir enormemente el espacio de búsqueda y agilizar el cálculo de función objetivo.
- Los tiempos computacionales logrados sugieren que el algoritmo tiene complejidad $O(n^2)$ pero es efectivo para encontrar una solución al problema. Cualquier combinación probada de parámetros obtiene una solución en instancias muy grandes por lo que es necesario revisar los tiempos para escoger una combinación. Esto se debe a que existen muchas soluciones para tableros muy grandes y estas se encuentran esparcidas por todo el espacio de solución.
- Con respecto a la literatura, la metaheurística implementada supera con creces en tiempo computacional a los demás algoritmos que usan recocido simulado. Esto puede ser debido a la codificación usada y a la implementación eficiente.

7 Referencias

[1] H. Noon and G. V. Brummelen, "The Non-Attacking Queens Game," *The College Mathematics Journal,* vol. 37, no. 3, pp. 223-227, 2006.

[2] . E. Masehian , H. Akbaripour and N. Mohabbati-Kalejahi, "Landscape analysis and efficient metaheuristics for solving the n-queens problem," *Comput Optim Appl,* vol. 56, pp. 753-764, 2013.

[3] E. Hoffman, J.C. Loessi and R. Moore, "Constructions for the solution of the n queens prob- lem," *Mathematics Magazine,* pp. 66-72, 1969.

[4] R. Sosi and J. Gu, "Efficient local search with conflict minimization: A case study of the n- queens problem," *IEEE Transactions on Knowledge and Data Engineering,* vol. 6, no. 5, pp. 61-68, 1994.

[5] K. Agarwal, A. Sinha and M. H. Bindu, "A Novel Hybrid Approach to N-Queen Problem," in *Advances in Computer Science*, Springer-Verlag Berlin Heidelberg, 2012, pp. 519-527.

[6] I. Martinjak and M. Golub, "Comparison of Heuristic Algorithms for the N-Queen Problem," in *ITI 2007 29th Int. Conf. on Information Technology Interfaces*, Cavtat, Croatia, 2007.

[7] D. Delahaye, S. Chaimatanan and M. Mongeau, "Simulated Annealing: From Basics to Applications," in *Handbook of Metaheuristics*, International Series in Operations Research & Management Science , 2019, pp. 1-20.

[8] R. Sosic and J. Gu, "Efficient local search with conflict minimization," *IEEE Trans. Knowl. Data Eng,* vol. 6, pp. 661-668, 1994.

[9] A. Homaifar and J. A. S. Turner, "The n-queens problem and genetic algorithms," in *Proceedings IEEE Southeast Conference*, 1992.

[10] S. S. Y. Dirakkhunakon, "Simulated annealing with iterative improvement," in *International Conference on Signal Processing Systems* , 2009.

[11] E. W. Weisstein, "Queens Problem," MathWorld--A Wolfram Web Resource, [Online]. Available: https://mathworld.wolfram.com/QueensProblem.html. [Accessed 2 Noviembre 2020].