Dos algoritmos para resolver el Problema del Recorrido del Caballo utilizando Prolog.

Sierra Casiano Vladimir

14 de febrero de 2021

Resumen

Para el presente proyecto se abordará una variante del Problema del Recorrido del Caballo. Se presentarán dos maneras (a comparar) para encontrar soluciones óptimas, la primera es utilizando la regla de Warnsdorff, y la segunda es adaptando el algoritmo bioinspirado Artificial Bee Colony. Ambas soluciones tendrán un enfoque declarativo.

1. Preliminares.

1.1. Problema del Recorrido del Caballo.

El problema data a mediados del Siglo VI en la India [4] (muy cerca del origen del ajedrez) y ha sido muy estudiado a lo largo de la historia por celebres matemáticos como Euler, quien enunció el problema de la siguiente manera: *¿Cómo se pueden encontrar todas las secuencias de movimientos de la pieza del caballo en un tablero de ajedrez de tal manera que cada casilla del tablero es visitada exactamente una vez?* [2]

El problema es una instancia del más general Problema del camino Hamiltoniano de la Teoría de Gráficas.

A lo largo del tiempo varios enfoques han sido propuestas para encontrar soluciones tales como la fuerza bruta, divide y vencerás, búsqueda DFS con backtracking e incluso redes

neuronales [1]. El problema es NP-completo, por lo que encontrar algoritmos eficientes para dar soluciones sigue siendo un reto interesante.

1.2. Definición del problema.

La pieza del caballo en el ajedrez tiene un movimiento en forma de L , como se ilustra a continuación.

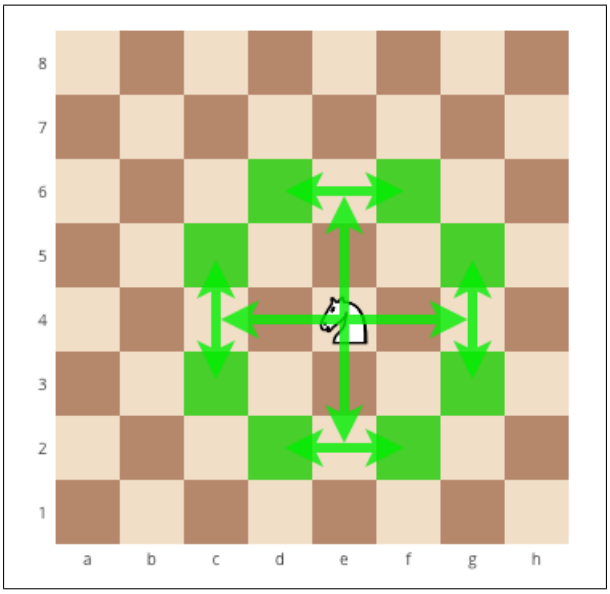

Figura 1: Posibles movimientos para el caballo.

El problema que queremos resolver es, dada una posición inicial y el tamaño del tablero, encontrar el camino más largo posible utilizando los movimientos del caballo y visitando cada casilla a lo más una vez.

1.3. Artificial Bee Colony.

El algoritmo Artificial Bee Colony (ABC) fue propuesto en el año 2005 por el Dervis Karaboga, y está inspirado en el comportamiento de abejas melíferas para encontrar fuentes de comida de calidad. Es uno de los algoritmos bioinspirados más populares para problemas de optimización numérica [1]. En esta sección describiremos brevemente su funcionamiento.

Componentes.

- Abejas, que se dividen en tres tipos: empleadas, observadoras y exploradoras.
- Se tiene un mismo número de abejas de cada tipo.
- Fuentes de comida, que representa una posible solución al problema.
- Cada abeja empleada es encargada de una fuente de comida (es decir, tenemos el mismo número de fuentes de comida que de abejas empleadas).
- Hay una función fitness, que evalúa la calidad de una fuente de comida.

Metodología.

- Se generan fuentes de comida de manera aleatoria.
- Se repite un número definido de veces los siguientes pasos
    - Todas las abejas empleadas buscan una abeja compañera para compartir información y actualizar la fuente de comida.
    - Las abejas observadoras toman las fuentes de comida de las abejas empleadas, y seleccionan solo a algunas para actualizar su información.
    - Las abejas exploradoras toman las fuentes de comida y las inspeccionan. Si encuentran alguna fuente de comida que no puede optimizarse más y que se encuentra por debajo de un límite se descarta y se cambia por una nueva posición generada aleatoriamente.

De manera resumida, las abejas empleadas actualizan todas las fuentes de comida, las observadoras actualizan solo algunas soluciones (las más optimas al momento) y las exploradoras descartan soluciones subóptimas. Este proceso se repite un número finito de veces y al final observamos la solución global.

1.4. El papel de la Programación Declarativa.

Para las implementaciones de este proyecto se optó por el uso de Prolog, lenguaje que pertenece al paradigma de la Programación Lógica.
En Prolog la información es manejada con hechos y reglas de producción.

Como observaremos en la sección 2, una considerable cantidad de operaciones que se requieren para los algoritmos desarrollados en este proyecto consisten en

- Verificar que se cumplan condiciones (por ejemplo, para mutar información).
- Explorar varias posibilidades en un mismo punto (por ejemplo, explorar los posibles movimientos).

Ambas características están intimamente ligadas a la filosofía de Prolog, que nos regala la exploración a través de su árbol de resolución y además tiene el bactracking ya implementado. Estos aspectos hacen que una implementación en este lenguaje sea atractiva.

2. Desarrollo.

2.1. Representación de la información.

Para todos los métodos propuestos a continuación es importante entender cómo estamos representando los distintos elementos.

- La posición en el tablero es un par $(\mathbf{X}, \mathbf{Y})$, donde X es el número de la columna y Y el número del renglón. Ambos valores están en un rango entre 1 y N (longitud del lado del tablero).

  Por ejemplo, para un tablero de 3×3 los pares para cada posición son los siguientes

(1,1)	(2,1)	(3,1)
(1,2)	(2,2)	(3 ,2)
(1,3)	(2,3)	(3,3)

Figura 2: Representación de las posiciones en un tablero de 3×3.

- Un recorrido se representa con una lista, donde cada elemento es una posición del tablero. El orden en que se visitan las posiciones se lee de izquierda a derecha en la lista.

  Por ejemplo, dado el recorrido

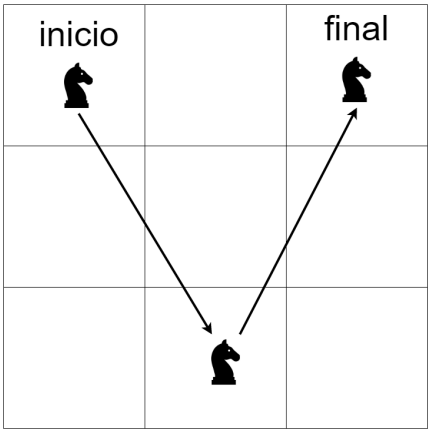

Figura 3: Ejemplo de secuencia de movimientos.

su representación es: $[(1,1)\ ,\ (2,3),\ (3,1)]$

Algunos aspectos particulares de cada método serán nombrados en su correspondiente sección.

2.2. Tratando de resolver el problema con fuerza bruta.

La solución más siple en la que podemos pensar es recorrer todos los caminos posibles, y al final elegir el más largo. Esto se puede hacer utilizando una búsqueda DFS y backtracking. El árbol de posibles movimientos y el backtracking es manejado gracias a Prolog, pues el recorrido DFS corresponde a la exploración que hace Prolog en el árbol de resolución. Mientras que el backtracking nos permite volver a posiciones anteriores para explorar otro posible camino.

Potencialmente, tenemos que elegir entre 8 casillas (a las que se puede mover) en cada paso. Como el número de casillas es N^2 el recorrer todos los caminos tiene una complejidad de $\mathcal{O}(8^{N^2})$.

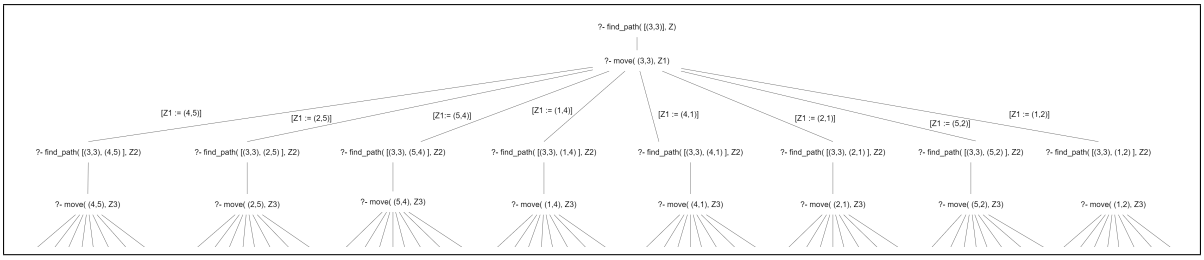

Figura 4: Primer nivel del árbol de búsqueda para un tablero 5×5.

Explorar todos los caminos es inviable, la capacidad de las computadoras de hoy en día es superada para un tablero de 8×8. Para reducir el espacio de búsqueda (y entonces obtener resultados en un tiempo razonable) se pueden utilizar muchos métodos, uno relativamente simple es el descrito en la sección siguiente.

2.3. Solución con la regla de Warnsdorff.

En 1823 H.C. Warnsdorff propusó una heurística muy eficiente para afrontar el problema, a dicha heurística se le conoce como regla de Warnsdorff [3] . Aunque la heurística fue propuesta para el problema original (encontrar el camino que pasa por todas las casillas) para nuestra variante también funciona, pues maximiza la longitud de los recorridos.

Llamamos una casilla adyacente a una casilla a la cual se puede ir desde la posición actual con un movimiento del caballo.

Para decidir el siguiente movimiento primero se observan las casillas adyacentes a la actual, para cada una de ella se obtiene la cantidad de casillas adyacentes no visitadas, al final se elige aquella con la menor cantidad.

Entonces el siguiente movimiento debe ser a aquella casilla adyacente no vistada que tenga la menos cantidad de casillas adaycentes no visitadas.

Por ejemplo, en la siguiente imagen la pieza del caballo está en su posición inicial, sus casillas adyacentes tienen el número entero correspondiente a la cantidad de casillas adyacentes no visitadas. El siguiente movimiento siguiendo la regla de Warnsdorff es hacia la casilla con el valor 2.

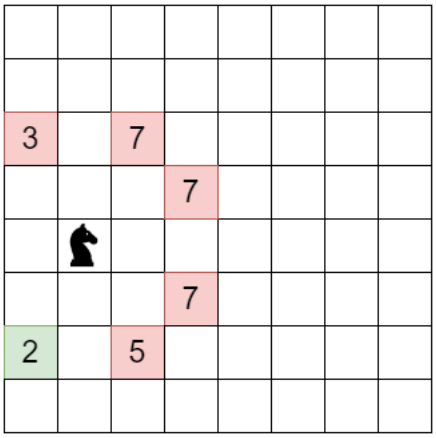

Figura 5: Ejemplo de regla de Warnsdorff.

El árbol de resolución de Prolog entonces ya no tiene que desarrollar las ramas para todos los posibles movimientos, con la heurística escogemos solo uno y sobre esa rama seguimos desarrollando.

Por ejemplo, para un tablero de 5×5 con posición inicial $(3, 3)$ el desarrollo de los primeros movimientos con la heurística puede ser observado en el siguiente árbol [1].

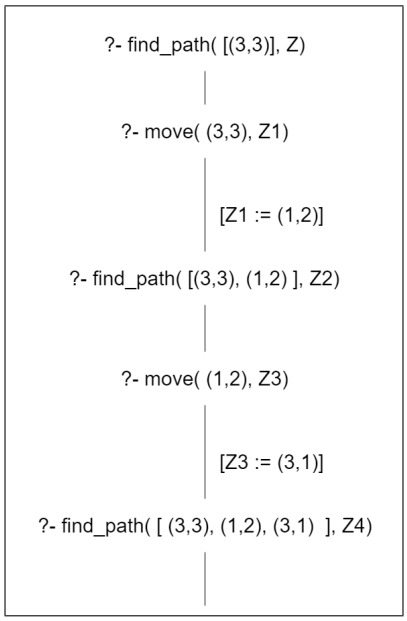

Figura 6: Primeros tres niveles del árbol de búsqueda con heurística.

[1] El árbol de la imagen no muestra todo el desarrollo de Prolog, solo es una versión simplificada para observar la reducción de ramas.

2.4. Adaptando Artificial Bee Colony para resolver el problema.

2.4.1. Representación de las fuentes de comida.

El algoritmo ABC está planteado originalmente para problemas con valores continuos. Como nuestro problema es discreto se deben realizan unas adaptaciones. La siguiente adaptación está basada en la propuesta de Banharnsakun [1].

Los 8 patrones de movimiento del caballo se identificarán con un número:

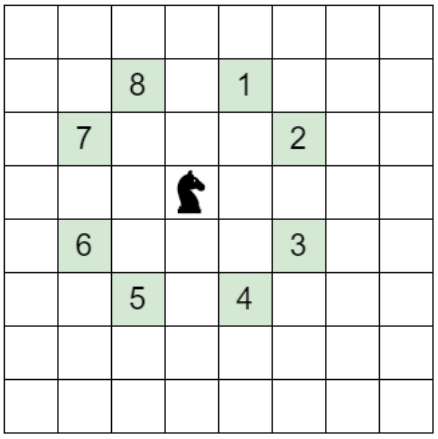

Figura 7: Identificadores de los patrones.

Las posibles soluciones para un tablero de $N \times N$ se representan con una lista de longitud $N^2 - 1$, donde cada elemento de la lista es un número entre 1 y 8.

3	2	5	2	7	1	3	8	4	3	6	1	7	5	3

Figura 8: Ejemplo de posible solución para un tablero de 4×4.

El significado operacional de esta secuencia es, dada la posición inicial, aplicar el patrón de movimiento que se encuentra en la cabeza de la lista, una véz llegada a la posición destino aplicar el patrón de movimiento que guarda el siguiente elemento de la lista.

Es claro que las listas pueden guardar movimientos inválidos, es decir, al aplicar el patrón de movimiento se regresa a una casilla previamente visitada o se sale del tablero.

El objetivo del algoritmo es entonces maximizar los movimientos válidos.

2.4.2. Función fitness.

La definición de nuestra función fitness está dada recursivamente:

$$f(\,[\,]\,) = 0$$
$$f(x:xs) = \begin{cases} 1 + f(xs) & \text{si } x \text{ es un movimiento válido} \\ 0 & \text{si } x \text{ es un movimiento inválido} \end{cases} \quad (1)$$

Decimos que la fuente de comida V_i tiene un valor fitness de $f(V_i)$.
Observe que con esta definición entre más grande es el valor de la función fitness más largo es el recorrido que representa la lista.

2.4.3. Actualización de las soluciones.

Para actualizar una fuente de comida V_1, primero se escoge aleatoriamente una fuente de comida distinta V_2 , el intercambio de información se realiza con tres cortes e intercalando los fragmentos de ambas fuentes, como se ilustra en la siguiente imagen:

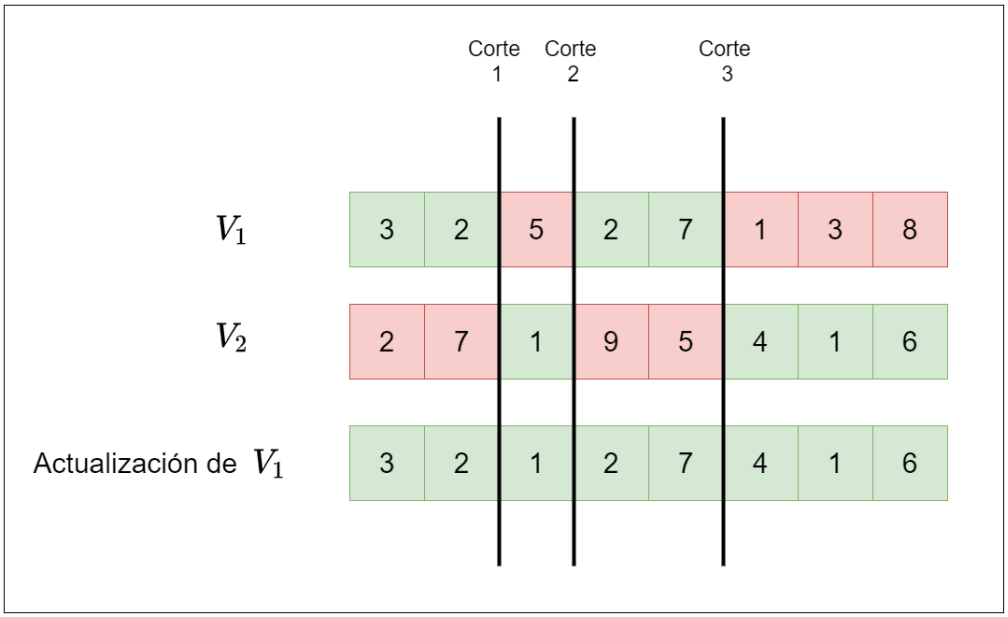

Figura 9: Proceso de actualización de una fuente de comida.

Los puntos de corte son escogidos aleatoriamente.

Recuerde que en todo momento se mantiene una estrategia glotona, cuando una fuente de comida muta se comprueba si el valor de la función fitness se incrementa o no, en caso de que no se incremente se descartan los cambios.

2.4.4. Algoritmo.

Como mencionamos en la sección 1.3, los trabajos realizados por los tres tipos de abejas conforman un ciclo de ejecución.

Abejas empleadas.

Todas las abejas tienen asociada una fuente de comida, de manera aleatoria escogen a una compañera y actualizan la posición de su fuente de comida siguiendo el método descrito en la subsección 2.4.3.

Abejas observadoras.

Las abejas observadoras toman todas las fuentes de comida que tienen las empleadas, cada fuente de comida tendrá asociada una probabiliad para ser seleccionada.

La probabilidad de que una fuente de comida V_i sea seleccionada está dada por la siguiente función

$$g(V_i) = \frac{f(V_i)}{\sum_{V_j \in S} f(V_j)} \tag{2}$$

Donde S es el conjunto de todas las fuentes de comida y f es la función fitness.

Cada abeja observadora selecciona una fuente de comida (utilizando g), diferentes abejas pueden escoger la misma fuenta. Posteriormente las fuentes de comida seleccionadas se actualizan, siguiendo el mismo método que las abejas empleadas.

Abejas exploradoras.

Las abejas exploradoras toman todas las fuentes de comida (después de que las observadoras hayan realizado sus modificaciones), observan cuál es el valor más grande que toma la función fitness en ese momento para alguna fuente de comida, llamemos fit_{max} a este valor, y para todas las fuentes de comida cuyo valor fitness asociado sea menor a fit_{max} se trata de extender la solución.

Observe que el valor fitness nos indica la cantidad de movimientos válidos, por lo que

para una fuente de comida V_i, el elemento en la posición $f(V_i)+1$ es el primer movimiento no válido de la secuencia. Las abejas exploradoras tratan de modificar este movimiento por otro que sea válido, si no existe ningún movimiento posible válido entonces V_i es una solución subóptima (está por debajo del límite actual y no hay manera de que se incremente), por lo que es descartada y reemplazada por una nueva fuente de comida generada aleatoriamente. En caso de que sí exista un movimiento válido se realiza la modificación a V_i cambiando únicamente ese valor.

Adicionalmente a esto, la fuente de comida con mayor fitness value también se trata de extender, sin embargo, en caso de no poder extenderse más no se descarta, pues es potencialmente la solución global.

El flujo de ejecución se puede observar en el siguiente diagrama.

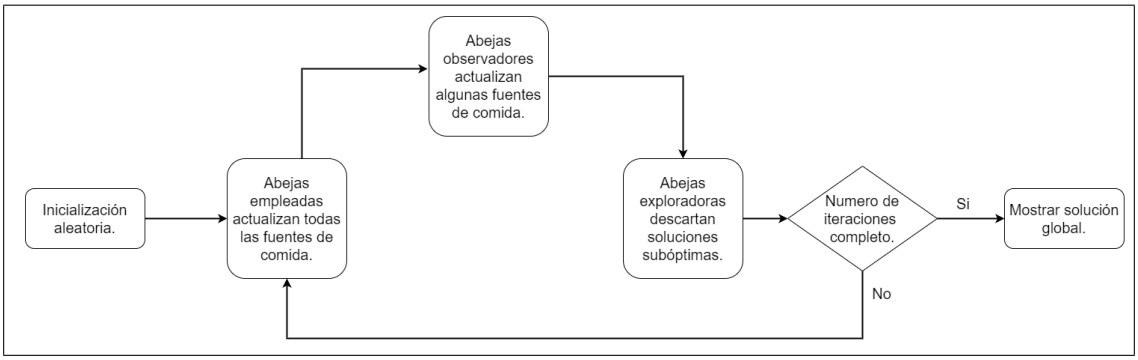

Figura 10: Flujo de ejeción.

3. Resultados.

Se realizaron pruebas para tableros de longitud 8×8, 12×12 y 16×16 (cinco ensayos por cada uno) con posicione aleatorias. Las medias aritméticas tanto del tiempo de ejecución como la longitud del camino más largo encontrado se presentan en las siguientes gráficas. El algoritmo Artificial Bee Colony se ejecutó con 32 fuentes de comida y 200 iteraciones.

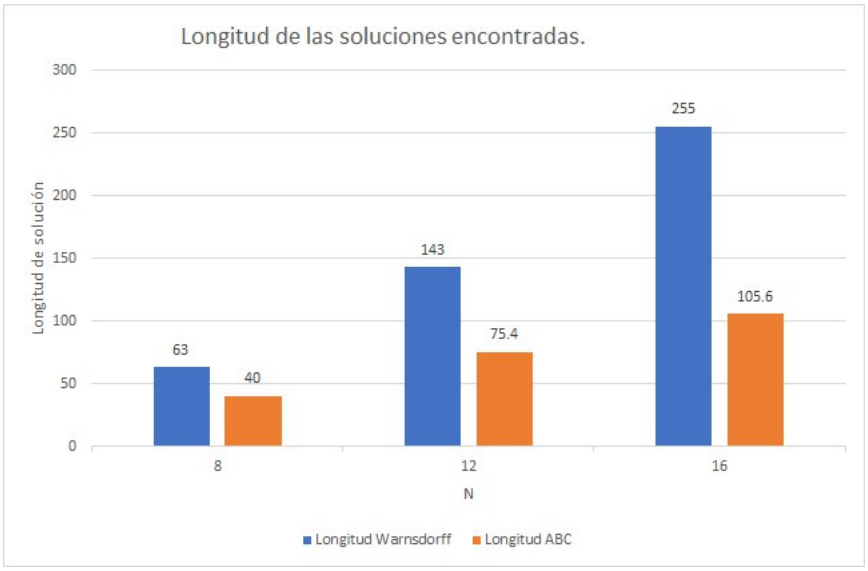

Figura 11: Longitud de los caminos resultantes.

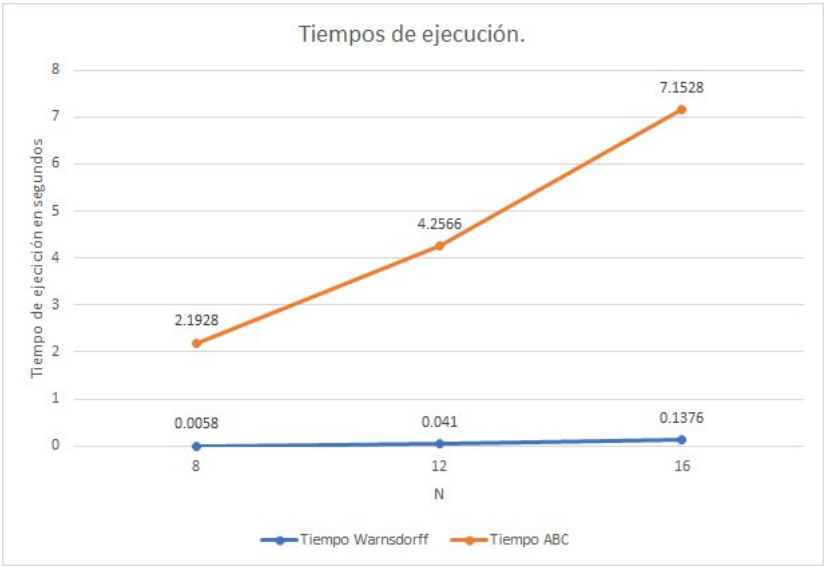

Figura 12: Tiempos de ejecición de las pruebas.

4. Conclusiones.

Como podemos observar en los resultados, de los dos métodos que se presentaron como alternativa fue la implementación de la heurística de Warnsdorff la que dio mejores resultados, encontrando caminos más largos que ABC y en menos tiempo.

La diferencia en la calidad de las solución fue bastante significativa. Aunque los tiempos de ABC dependan directamente de las iteraciones y fuentes de comida, los tiempos con parámetros moderados crecen de manera mucho más rápida a comparación de la heurística.

5. Trabajo futuro.

Aunque los resultados obtenidos con el algoritmo ABC no son satisfactorios muestra la explotación de ciertos puntos importantes, y con algunas variantes se puede intentar obtener mejores soluciones.
Para mejorar los resultados se deben realizar modificaciones siguiendo algunos de los siguientes puntos

- Modificar el número de agentes y el número de iteraciones. De manera más general encontrar las cotas apropiadas para estos valores dependiendo del valor de la variable N.
- Modificar la manera en que se comparte la información entre los agentes.
- Modificar el proceso de las abejas exploradoras, para tratar de extender de manera más inteligente los caminos y evitar repeticiones en la información.

Ambos puntos requieren un análisis profundo, deben ser implementados y probados para saber si mejoran las soluciones. Realizar variantes queda fuera del alcance de este proyecto pero deben ser tomadas en cuenta para trabajos futuros.

6. Detalles de la implementación.

6.1. Versión.

Todas las implementaciones presentadas fueron realizadas con SWI-Prolog 8.2.4.

6.2. Disponibilidad de la implementación.

Los archivos relacionados al proyecto se encuentran disponibles en el repositorio https://github.com/ciencias-unam/proyecto-final-VladimirSierra

Le invitamos a leer el archivo README.md para conocer las instrucciones precisas de ejecución.

La estructura del proyecto es:

```
proyecto-final-VladimirSierra
├── reporte
│   ├── bibliography
│   ├── images
│   ├── Reporte.tex
│   └── Reporte.pdf
├── src
│   ├── BeeColony
│   │   └── bee_colony.pl
│   └── Busqueda
│       ├── heuristic_solution.pl
│       └── simple_solution.pl
└── README.md
```

En la carpeta *reporte* se encuentra el código fuente del reporte y su correspondiente pdf.

En la carpeta *src/BeeColony* se encuentran los archivos de la implementación para ABC.

En la carpeta *src/Busqueda* se encuentran dos archivos, uno que implementa la búsqueda completa (*simple_solution.pl*), y otro que implementa la búsqueda con heurística (*heuristic_solution.pl*).

Referencias

[1] Anan Banharnsakun. "Artificial Bee Colony Algorithm for Solving the Knight's Tour Problem". En: *Intelligent Computing & Optimization*. Ed. por Pandian Vasant, Ivan Zelinka y Gerhard-Wilhelm Weber. Cham: Springer International Publishing, 2019, págs. 129-138. ISBN: 978-3-030-00979-3.

[2] Martin Charles Golumbic y André Sainte-Laguë. "IX Knight's tour". En: *The Zeroth Book of Graph Theory: An Annotated Translation of Les Réseaux (ou Graphes)—André Sainte-Laguë (1926)*. Cham: Springer International Publishing, 2021, págs. 71-76. ISBN: 978-3-030-61420-1. DOI: 10.1007/978-3-030-61420-1_10. URL: https://doi.org/10.1007/978-3-030-61420-1_10.

[3] M. Pranav, S. Nithin y N. Guruprasad. "A Comparison of Warnsdorff's Rule and Backtracking for Knight's Tour on Square Boards". En: *Emerging Research in Electronics, Computer Science and Technology*. Ed. por V. Sridhar, M.C. Padma y K.A. Radhakrishna Rao. Singapore: Springer Singapore, 2019, págs. 171-185. ISBN: 978-981-13-5802-9.

[4] John J. Watkins. "Introduction". En: *Across the Board: The Mathematics of Chessboard Problems*. Princeton University Press, 2004, págs. 1-24. ISBN: 9780691154985. URL: http://www.jstor.org/stable/j.ctt7s2g5.4.