TEC | Tecnológico de Costa Rica

Primer Proyecto de Lenguajes de Programación: Soluciones al Problema de N-Reinas en Lenguajes de Programación Funcional

Diego Acuña Carnet:2018109507
Ronald Esquivel López Carnet: 2018093269
Ricardo Murillo Jiménez Carnet: 2018173697

Instituto Tecnológico de Costa Rica. Escuela de Computación.

Profesor: Eddy Ramírez.

Resumen

El popular problema de las NReinas consiste en colocar **n-reinas** en un tablero de ajedrez, de tal manera de que ninguna de las reinas quede atacando a otras. Este problema tiene 2 versiones. La más simple consiste en encontrar exactamente una solucion.Y la más compleja consiste en encontrar todas las soluciones posibles para un **nxn** tablero.

En esta ocasión se realizarán dos versiones de las soluciones ya que se estará implementando dos algoritmos, en diferentes lenguajes, y con distinta metodología, uno va a ser desarrollado mediante el uso de **Algoritmos Genéticos**, en el cual, gracias a las características de estos algoritmos, se tendrá como salida una población de los cromosomas más aptos, lo que se traduce como en las posibles soluciones más prometedoras al problema, mientras que el otro algoritmo será implementado con **Backtracking**, el cual es un algoritmo heurístico, en el cual el costo de tiempo es menor.

El objetivo de este proyecto y de este documento no es la implementación y estudio de estos dos algoritmos, sino más bien el estudio de los dos lenguajes de programación que van a ser empleados para dar solución a el problema mediante estos algoritmos.

La primera solución, implementada mediante el uso de algoritmos genéticos va a ser desarrollada en el lenguaje de programación funcional **Erlang**. Por otra parte, la implementación del algoritmo por backtracking va a ser desarrollada en **Scheme**.

Los resultados la implementación de las soluciones muestran resultados curiosos, donde destacan que la implementación de Scheme es exitosa, pero se presentan problemas de rendimiento y de eficiencia, lo que lo puede convertir en un resultado intratable a partir de 9 reinas. Mientras tanto la implementación de Erlang, en la cual se implementa los algoritmos genéticos, se presentó problemas en la implementación, pero las pruebas que se realizan con éxito muestra un tiempo de respuesta normal o tratable.

Además, en Erlang se podrá ver un comportamiento curioso, que se refleja en la implementación del Algoritmo Genético, unas perturbaciones de los timepos de ejecución muy interesantes, que suponen los comportamientos de los randoms a través del código.

1. Introducción

Para un conocimiento más a fondo acerca de los lenguajes, su funcionamiento y demás es necesario mencionar la historias del origen y la evolución de estos a lo largo del tiempo, mencionando posteriormente alguna de las caractrísticas fundamentales de cada uno de ellos. Pero previo a esto, es necesaria la explicación de algunos conceptos claves.

Es indispensable que cuando se habla de lenguajes funcionales, se deba comprender el significado de estos y las diferencias con otros tipos de lenguajes de programación que existen.

En un principio, los lenguajes de programación fueron creados para fines de controlar las computadoras o el hardware y hasta ahí. Luego, se entró en razón de que las operaciones que las computadoras son capaces de razonar de manera sencilla, para los humanos son muy complicadas, y además, se tenía que implementar un lenguaje que fuera común entre las computadoras, por lo que se comenzó a crear distintos tipos de lenguajes de programación, de los cuales surgieron los funcionales. [1]

Los lenguajes de programación **funcionales** como Erlang y Scheme consisten en que su computación o forma de computar está estrechamente relacionada con la evaluación de expresiones, además este tipo de paradigma de programación advoca a que los programas van a ser escritos muy rápidamente en comparación de otros tipos de lenguajes. Aparte de esto son concisos, de alto nivel y más recomendados para el análisis y razonamiento de arquitecturas paralelas y demás.

Resolver problemas como el de N-Reinas en estos lenguajes funcionales, implica que la abstracción de la solución del problema y de la implementación de los algoritmos es menos compleja en comparación con otros tipos de lenguajes. Tomando en cuenta lo anterior, se escogió Erlang como el lenguaje para la implementación del algoritmo genético por motivos de soporte, manejo, abstracción y principalmente documentación. Por otra parte, la implementación del backtracking en scheme es más sencilla, debido a que el manejo de listas es su principal característica de este lenguaje, lo que facilita la abstracción de la solución del problema en dicho algoritmo.

2. Historia y de Erlang y Scheme

2.1. Scheme

19

Scheme comenzó en la década de 1970 como un intento de entender el **modelo Actor de Carl Hewitt**, para cuyo propósito Steele y Sussman escribieron un **"pequeño intérprete Lisp"** usando Maclisp y luego .ªgregaron mecanismos para crear actores y enviar mensajes".Scheme se llamó originalmente "Schemer", en la tradición de otros lenguajes derivados de Lisp como Planner o Conniver. El nombre actual resultó del uso por parte de los autores del sistema operativo ITS, que limitaba los nombres de archivo a dos componentes de un máximo de seis caracteres cada uno. Actualmente, "Schemer"se usa comúnmente para referirse a un programador de Scheme.[2]

Scheme es un lenguaje de programación funcional de alcance estático, de dialecto recursivo de cola, adecuado para el lenguaje de programación Lisp, el cual fué inventado por Guy Lewis Steele Jr. y Gerald Jay Sussman.

Scheme fue diseñado para tener una semántica excepcionalmente clara y simple y pocas formas diferentes de formar expresiones. Una amplia variedad de paradigmas de programación, incluidos estilos imperativos, funcionales y de paso de mensajes, encuentran una expresión conveniente en Scheme.[3]

Scheme fue uno de los primeros lenguajes de programación en incorporar procedimientos de primera clase como en el cálculo lambda, demostrando así la utilidad de las reglas de alcance estático y la estructura de bloques en un lenguaje tipado dinámicamente. Scheme fue el primer dialecto principal de Lisp en distinguir procedimientos de expresiones y símbolos lambda y utilizar un único entorno léxico para todas las variables. Al depender completamente de las llamadas a procedimientos para expresar la iteración, Scheme enfatizó el hecho de que las llamadas a procedimientos recursivas de cola son esencialmente **goto's** que pasan argumentos. Más recientemente, Scheme introdujo el concepto de números exactos e inexactos. Scheme es también el primer lenguaje de programación que admite macros que permiten la sintaxis de un lenguaje estructurado en bloques. [3]

2.2. Erlang

El nombre Erlang, supone que fue usado como una referencia al matemático e ingeniero danés Agner Krarup Erlang y una abreviatura silábica de .ᴱʳicsson Language". [4]
Erlang fue diseñado con el objetivo de mejorar el desarrollo de aplicaciones de telefonía. La versión inicial de Erlang se implementó en **Prolog** y fue influenciada por el lenguaje de programación **PLEX**. En 1988, Erlang había demostrado que era adecuado para la creación de prototipos de centrales telefónicas, pero el intérprete de Prolog era demasiado lento. En 1992, se inició el trabajo en la máquina virtual BEAM (VM) que compila Erlang a C. Según Armstrong [4], el lenguaje pasó de un producto de laboratorio a aplicaciones reales tras el colapso de la central telefónica AX de próxima generación llamada AX-N en 1995. Como resultado, Erlang fue elegido para la próxima central de transferencia asincrónica.

Erlang es un lenguaje de programación de propósito general y un entorno de ejecución. Erlang tiene soporte integrado para concurrencia, distribución y tolerancia a fallas. Erlang se utiliza en varios grandes sistemas de telecomunicaciones de Ericsson.[5]

3. IDE's Utilizados

En este proyecto se hará uso de dos IDE's, tanto para Scheme como para Erlang.

3.1. Dr. Rachet y Full-Swindle

Racket es un lenguaje de programación de amplio espectro de la familia de Lisp y Scheme. Es multiparadigma así como de propósito general. Tiene su propio IDE, llamado DrRacket. Racket viene con un conjunto completo de bibliotecas: una caja de herramientas GUI multiplataforma, un servidor web y miles de paquetes adicionales que están a un solo comando de distancia.[6] El paquete Swindle amplía Racket con muchas características adicionales. La característica principal que inició este proyecto es un sistema de objetos similar a CLOS, pero hay muchas características más. En este proyecto, se usará este paquete.

3.2. Eshel V10.6.4

Eshel es un shell para el lenguaje Erlang. El shell es un programa de interfaz de usuario para ingresar secuencias de expresión. Las expresiones se evalúan y se devuelve un valor. Un mecanismo de historial guarda los comandos anteriores y sus valores, que luego se pueden incorporar en comandos posteriores. El usuario puede determinar cuántos comandos y resultados guardar. El shell también permite al usuario iniciar varios trabajos simultáneos. Un trabajo puede considerarse como un conjunto de procesos que pueden comunicarse con el shell.[7]

4. Descripción de la Solución

4.1. Algoritmo Genético

La implementación del algoritmo genético se hizo en el lenguaje funcional Erlang, el cual se hizo la granularidad de las funciones de acuerdo a la solución y funcionalidad de un algoritmo genético. Primero, es necesario entender cuales son los procesos necesarios para poder realizar un algoritmo genético de manera exitosa en la solución del problema de las N-Reinas. Para comenzar es necesario mencionar que los algoritmos genéticos se basan en el análisis, cruzamientro, mutación y filtración de poblaciones, haciendo alusión a la genética de Darwin. Para poder solucionar el problema mediante estos algoritmos se deben generar poblaciones iniciales, generadas aleatoriamente de acuerdo a la cantidad de n reinas que se quiere buscar en un tablero nxn. Después, se evaluan las poblaciones más adecuadas mediante una función de fitness, que hace una especie de filtración basada en que cada uno de los cromosomas que perternerce a la población inicial. Esta función hace una evaluación, buscando los cromosomas elite, o los más aptos mediante una calificación a partir de criteios de fitness. Estos criterios de fitness son los responsables de escoger cuales cromosomas son los elegidos para que pertenezcan a la población elite y cuales no. Para este problema, el criterio consiste en buscar un cromosoma, en donde cada reina dentro de la combinación, no choque entre si. Si el cromosoma presenta una calificación muy buena, se va acercando al 0, el cual significaque es un cromosoma perfecto u el más apto de la población.

Si se hace la escogencia y no se encuentra al cromosoma que contiene la respuesta, se procede al cruce de la población, en donde se cruzan dos cromosomas padre y madre con las mejores calificaciones fitness y se mutan. El concepto de mutación en los algoritmos genéticos es muy similar al de la genética biológica. Después del cruce de los cromosomas padres fit, se debe mutar al hijo, para que no sea exactmente igual a sus padres, podemos considerar a la mutación como uno de los factores decisivos para que estos algoritmos funcionen, ya que pueden empeorar a la población, o pueden causar la creación de la solución del problema. Una vez termina el cruce y la mutación, se vuelve a evaluar la población y si se encuentra el resultado, se termina el ciclo, o sino se realiza otra recursión. Los interesante de estos algoritmos es que si se hacen bien cada uno de sus pasos, se va a encontrar una solución, o por lo menos se va a generar una población final muy prometedora al final del cilo del algoritmo.

4.2. Algoritmo por Backtracking

Se hace la implementación del algoritmo por backtracking en el lenguaje de programación Scheme. De igual forma, a manera de introducción a los algoritmos por backtracking, estos algoritmos son algoritmos heurísticos, de general la caracteística de backtracking, y de manera poco técnica es

que estos algoritmos buscan las soluciones dentro de un árbol de procesos, si el algoritmo detetecta una anomalía o una indicación de que un proceso o posible solución es errónea, se devuelve y busca la solución por otro camino. Conforme avanza y prueba, el algoritmo va a hallar el camino a la solución.

En la solución al problema de este proyecto, se inserta una reina, en una lista simple, en el cual, el número que está en X columna es el número de la fila, luego se verifica si la solución tiene repetidos, si sí tiene, se llama a una función que devuelve el siguiente número o combinación de reinas, este es llamado en una función llamada "backtrackingNReinas", la cual verifica si hay repetidos y si el largo de la solución temporal es igual al largo de las reinas que se desean meter, si no hay repetidos y la condición de largo = N se cumple y es una solución, se suma 1 a la cantidad de soluciones encontradas y se agrega la solución, caso contrario pasa a la siguiente verificación, que ve si hay choque de reinas en las diagonales de la solución temporal. Como se pudo notar, este algoritmo busca más que una solución, precisamente busca la cantidad de soluciones que se le indica, si este número es mayor a la cantidad de soluciones máximas encontradas, se devuelve estos máximos resultados en una matriz.

4.3. Problemas encontrados y limitaciones en el desarrollo de la soluciones

A continuación se desarrollan algunos de los problemas encontrados a la hora de la implementación de estos algoritmos.

4.3.1. Eficiencia

La eficiencia es un tema que debe (más bien tiene que) interesar mucho a los programadores. Encontrarse o desarrollar un programa con tiempos de respuestas no muy buenos o escencialmente malos, invalucra o implica el fracaso parcial o completo de la implementación. De igual forma un resultado funcional, pero ineficiente, con tiempos de ejecución bastantes altos no es muy bien visto.

En la implementación del algoritmo por Backtracking en Scheme, se puede ver que los tiempos de respuesta son muy altos, casi intratables. Pese a que se hace la verificación y las funciones implementadas dentro de la solución son correctas. (Correctas, en el sentido que realizan su función de manera exitosa). La causa radica escencialmente en la falta de conocimiento acerca del los lenguajes y básicamente la falta de técnica o conocimiento, ya que no se hace uso o no se aprovecha las características del lenguaje.

5. Resultados de la Solución

Como bien se en las gráficas mostradas más adelante en este documento. El algoritmo de backtracking tiene una eficiencia muy baja, llegando a subir sus tiempos de forma exponencial. Por lo que superó de mala manera las expectativas que se tenían en cuanto a su velocidad de resolución, a pesar de esto el algoritmo logra dar las respuestas correctas.

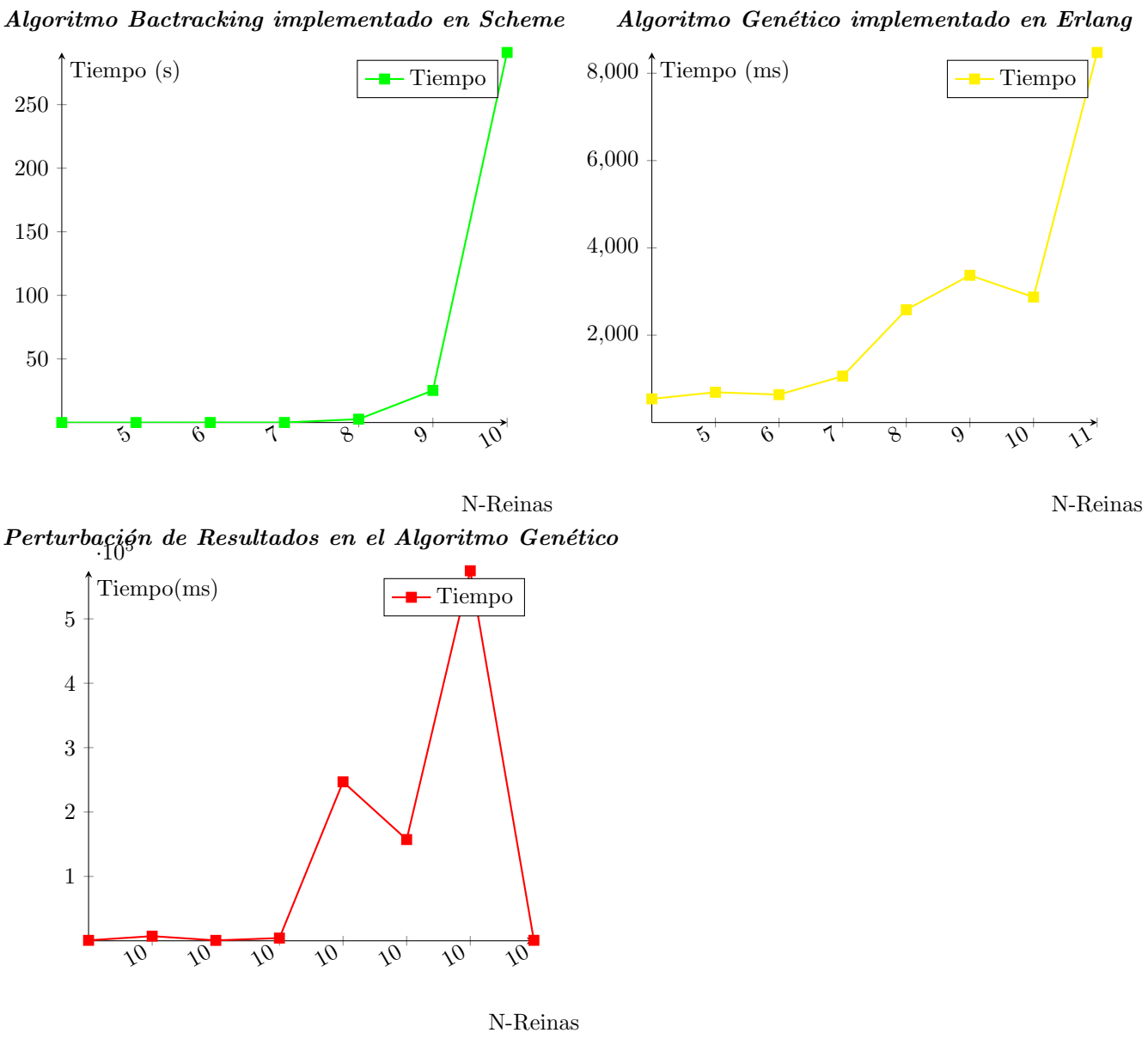

Figura 1: Comparación de tiempos

6. Conclusiones

Al observar los resultados del algoritmo genético podemos observar que tienen una cierta perturbación, cuanto a los resultados de tiempo además se puede notar que cuando se insertan valores de N mayores que 13 el algoritmo tiene una mayor probabilidad de no encontrar el resultado esto debido a la variabilidad que obtiene el algoritmo al trabajar en base a randoms para escojer su población, además de esto se puede observar esta variabilidad también cuando se ejecuta varias veces el mismo algoritmo con el mismo N cantidad de reinas, se observa que los tiempos puede oscilar y tender a generar picos en de tiempo en algunas ocupaciones, sin embargo el algoritmo realiza las operaciones en un rango de tiempo aceptable, como posible mejora se podía encontrar una manera de cruzar la población de tal manera que aproveche más a la población elite, heran-doles así a los cromosomas sus características óptimas para llegar a una solución más rápido y tal vez más estable.

En el caso del algoritmo de backtracking se cree que se puede mejorar el código implementando una función que generé la siguiente combinación de posiciones a probar de una forma mucho más efectiva que la actual.
Y también al ser formas "nuevas"de programar, no se logra aprovechar al máximo las características del Lenguaje, lo que puede llegar a afectar al funcionamiento en general.

7. Aprendizaje

Ronald: En este proyecto aprendí una nueva forma de programar, un poco compleja debido a que apenas estamos comenzando a programar en estos lenguajes funcionales. Me parece muy importante e interesante aprender a programar en estos lenguajes, pienso que si me especializo en ellos, mis habilidades de abstraer soluciones va a mejorar. Además me permite conocer diferentes formas de solucionar problemas, sin estr llamando variables, como estaba acostumbrado hasta este momento. Considero que las personas que apenas estamos aprendiendo a programar y apenas estamos comenzando a adentrarnos en este mundo, deberiamos comenzar a aprender con estos lenguajes, y no acostumbrarnos a depender de variables temporales y de la iteración.

Ricardo: Después de 2 años de carrera programando casi que unicamente en iterativo, costó mucho acostumbrarse a la forma en que hay que pensar en los lenguajes funcionales, lo cual es de mucha ayuda, más abordando un problema tan conocida como el de las n-Reinas.

Diego: Los aprendizajes adquiridos después de realizar el proyecto son muy amplios y variados, el hecho de cambiar la forma de trabajar los problemas y abrir la mente para abstraer los problemas de forma correcta es un ejercicio que como programadores en el futuro ya se para en lenguajes funcionales o no, nos va a servir para poder apreciar y atacar un problema desde diferentes ángulos. Además el manejo de lenguajes funcionales como lo hemos visto puede ser muy útiles para determinados casos en los que en otros tipos de lenguaje como java o C será un completo tormento realizar.

Referencias

[1] "Paul hudak(1989). conception, evolution, and application of functional programming languages," 1989. Web; Accedido el 17-10-2020.

[2] "Gerald jay sussman guy l. steele, jr.(1998), the first report on scheme revisited," 1998. Web; Accedido el 16-10-2020.

[3] "Chris hanson(2003). the scheme programming," 2003. Web; accedido 18-10-2020.

[4] "Armstrong, joe (2007). history of erlang. hopl iii: Proceedings of the third acm sigplan conference on history of programming languages.," 2007. Web; Accedido el 16-10-2020.

[5] "Ericsson(s.f). in a nutshell, what is erlang?," s.f. Web; Accedido el 16-10-2020.

[6] "tenth racketcon(2018). racket," 2018. Web; Accedido el 17-10-2020.

[7] "Robert virding, peter andersson(1991-2005). the erlang shell," 1991-2005. Web; Accedido el 17-10-2020.

[8] "tenth racketcon(2018). package: Swindle," 2018. Web; Accedido el 17-10-2020.