Implementación de Algoritmos de Inteligencia Artificial para la Resolución de Videojuegos Retro

INSTITUTO TECNOLÓGICO DE COSTA RICA

PROFESOR: EFRÉN JIMÉNEZ DELGADO II SEMESTRE 2022 GRUPO: 2

Isaac David Araya Solano
Ingeniería en Computación
Carnet: 2018151703

Alexia Denisse Cerdas Aguilar
Ingeniería en Computación
Carnet: 2019026961

***Abstract*—La motivación detrás de este proyecto consiste en el deseo de aplicar nuestros conocimientos adquiridos de la materia, con el fin de entender mediante comparaciones de diferentes resultados de prueba, y las razones del porqué los algoritmos genéticos y probabilísticos deben de implementarse con eficiencia para crear una inteligencia artificial que establezca escenarios futuros los juegos retro, respectivamente: Air Striker y Super Mario Bros. El propósito principal es analizar el comportamiento de los algoritmos implementados para los juegos elegidos, dependiendo de la variabilidad de los datos de entrada, con un máximo de 1000 registros. Después de ejecutar las pruebas, se realizó una evaluación de los resultados y se concluyó que el genético tiene una complejidad de tiempo lineal y que la cantidad de datos de entrada afecta completamente la eficiencia de este.**

***Keywords*—** eficiencia, complejidad del tiempo, o -grande, aprendizaje automático, python, juegos retro, algoritmo probabilístico, algoritmo genético, inteligencia artificial

I. Introducción

A medida que la tecnología evoluciona y el software y las computadoras se vuelven más comunes en todas las áreas del mundo actual, se deben considerar aspectos muy importante: la eficiencia y la optimización. Como cualquier otro producto en el mercado, el software también se rige por este principio; en este caso, que los recursos con más enfoque son la memoria, tiempo y la mejora de él mismo.

El objetivo final de este proyecto es conseguir una población de individuos que solventen las soluciones de al menos dos juegos, en el cual se estudiarán dos algoritmos: probabilístico y genético; enfocado principalmente en su implementación para los juegos retro mencionados. Esta investigación tiene como objetivo responder a esta pregunta: ¿cómo afecta la cantidad de datos de entrada al rendimiento de los algoritmos y la forma en la que avanza el nivel?

Para responder a la pregunta se realizarán una variedad de mediciones a cada algoritmo, considerando el tiempo de ejecución, las asignaciones en memoria, las comparaciones realizadas y la cantidad de líneas de código ejecutadas. Luego, también se hará un análisis exhaustivo de los resultados con la intención de determinar el comportamiento de los algoritmos mencionados y clasificarlos correctamente.

Como resultado, el proyecto está estructurado con las siguientes secciones: Trabajos Relacionados, Solución, Metodología con sus respectivos diagramas, figuras y tablas; Evaluación, Conclusiones, Anexos y finalmente las Referencias utilizadas a lo largo de la investigación.

II. Trabajos Relacionados

Para entender de una mejor manera y a profundidad los conceptos de los algoritmos de este proyecto, se realizó una revisión de la literatura. En primer lugar está el algoritmo genético, como se menciona en [1], el cual consiste en una población de soluciones codificadas de forma similar a cromosomas. Cada uno de estos cromosomas tendrá asociado un fitness que cuantifica su validez como solución al problema. En función de este valor se le darán oportunidades de reproducción donde, con cierta probabilidad, se realizarán mutaciones de estos cromosomas.

Además, según Bernal y Guevara, la definición de este algoritmo se basa en una técnica de programación que imita a la evolución biológica como estrategia para resolver problemas computacionales. Una de sus características principales es la de ir perfeccionando su propia heurística en el proceso de ejecución, por lo que no requiere largos períodos de entrenamiento especializado por parte del ser humano [2].

Por otro lado, el algoritmo probabilístico es un algoritmo que basa su resultado en la toma de algunas decisiones al azar, de tal forma que, en promedio, obtiene una buena solución al problema planteado para cualquier distribución de los datos de entrada [3].

Se utiliza en situaciones con limitaciones de tiempo o memoria y cuando se puede aceptar una buena solución de media, ya que a partir de los mismos datos se pueden obtener soluciones diferentes y algunas erróneas. Para que sea más probable ofrecer una solución correcta, se repite el algoritmo varias veces con diferentes submuestras aleatorias y se comparan los resultados [4].

Finalmente, la complejidad de los algoritmos juega un papel importante a la hora de analizarlos ya que permite clasificar los algoritmos según su comportamiento. Tal como lo describen Abdiansah y Wardoyo [5], se calcula utilizando la notación Big-O y se puede dividir en dos tipos de complejidad: complejidad de tiempo (cuánto tiempo se ejecuta el algoritmo) y complejidad de espacio (cuánta memoria usa el algoritmo su algoritmo).

III. Solución

Dentro del código programado de cada algoritmo para la creación de inteligencia artificial se realizan cálculos para que las pruebas sean más analíticas e informativas. Ambos están programados en el lenguaje de programación Python y todos esos cálculos fueron estimados con algunos contadores para analizar de la mejor manera los resultados de los mismos, cada método ejecutado se explica y se detalla más en la Metodología de este proyecto.

En ambos algoritmos se utiliza la librería gym-retro que permite la ejecución de los juegos que se han mencionado, cada integración de juego tiene archivos que enumeran ubicaciones de memoria

para variables en el juego, funciones de recompensa basadas en esas variables, condiciones de finalización de episodios, estados de guardado al comienzo de los niveles y un archivo que contiene hashes de ROM que funcionan con estos archivos [6].

A. Algoritmo genético

Empleado en el juego Air Striker el cual funciona con un concepto biológico y distintas partes fundamentales que caracterizan a este algoritmo: selección, cruce y mutación. Cada individuo de la población representa un movimiento en el juego, dependiendo de la selección y el estado de la evaluación, son los que evolucionan en cada generación.

Para satisfacer esto, primero se crea una población inicial de *m* individuos con arreglos de dos elementos de forma aleatoria, los cuales hacen referencia el movimiento a la izquierda y derecha, respectivamente. Luego, se evalúa esta población en la función fitness, que se comporta como la función objetivo de la maximización de recursos, donde solo los individuos que sobrevivan se guardan como los "individuos fit". Luego de evaluar todos los individuos de la población se crea una nueva generación a partir de cruzar los individuos fit únicamente. Esta nueva generación es evaluada igualmente por la función fitness y posteriormente se repite el proceso de la creación de una nueva generación. Este proceso se repite hasta cumplir el número de generaciones deseado.

B. Algoritmo Probabilístico

Utilizado para el juego Super Mario Bros. La idea de este algoritmo es ir generando movimientos aleatorios variando dependiendo de la probabilidad. A partir de cada movimiento que se hiciera este se guarda en la lista de jugadas exitosas que se usan para generar la distribución de probabilidad. A partir de esta distribución se escoge aleatoriamente entre si crear el siguiente movimiento de forma aleatoria o si usar una de las jugadas exitosas antes mencionadas.

Cada vez que se muriera el personaje se repiten los mismos movimientos hasta justo antes de morir para así cada vez progresar más en el juego y finalmente llegar a completar el juego. Ya que siempre el algoritmo busca generar una respuesta a pesar de no ser la mejor o no ser correcta, este se clasifica como un algoritmo de Monte Carlo.

IV. Metodología

La metodología del presente proyecto se basa en obtener información precisa sobre estos algoritmos con las diferentes pruebas que se realizaron. Esta sección incluye las mediciones del método empírico, analítico, gráfica y factor de talla, las siguientes pruebas se componen de: conteo de asignaciones, conteo de comparaciones, conteo de líneas de código ejecutadas y tiempo de ejecución. Cada método y medida nos permitió hacer un análisis de la complejidad basado en el código de los algoritmos y poder clasificarlos según su notación Big-O.

A. Estrategia de ramificación escogida

A la hora de generar cada generación para el algoritmo genético se utiliza la estrategia de cruce y mutación. En este caso el cruce que se utiliza es un "single point crossover" que lo que hace es combinar la mitad de la información de un padre con la mitad de la información del otro padre y luego dependiendo de una pequeña probabilidad se hace una mutación en el hijo resultante.

Estas mitades de información son un botón de cada padre respectivamente. Estos cruces se hacen entre los padres que se consideran fit y estos se escogen dependiendo de si sobrevivieron o no en el juego.

B. Diagramas de flujo

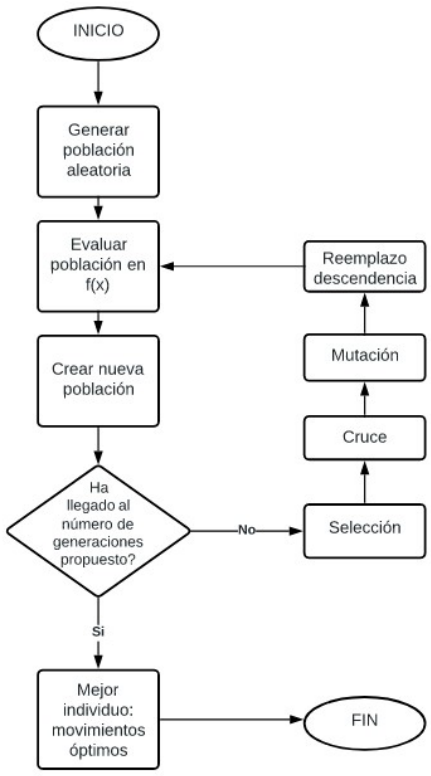

Fig. 1. Diagrama Flujo Algoritmo Genético

C. Tipos de cruces genéticos realizados

El tipo de cruce realizado fue el single-point crossover [7], el cual consiste en seleccionar un punto de cruce en la cadena del organismo principal. Todos los datos más allá de ese punto en la cadena del organismo se intercambian entre los dos organismos principales, los arreglos se caracterizan por el sesgo posicional. La Figure 2 representa el tipo de cruce elegido de una forma más simple:

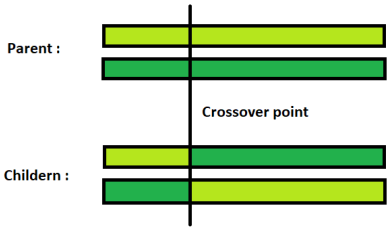

Fig. 2. Single-Point Crossover de GeeksforGeeks

D. Tipo de mutación que se aplicó

El tipo de mutación aplicado fue el bit flip [8] que se basa en la selección de uno o más genes (índices de matriz) y se cambian sus valores, es decir, cambiamos 1 a 0 y viceversa. A continuación, se explica mejor usando el diagrama dado en Figure 3:

Fig. 3. Bit Flip Mutation de GeeksforGeeks

E. Medición Empírica

Las pruebas que se ejecutan tienen las siguientes cantidades de datos de entrada: 10, 50, 100, 200, 500 y 1000, con el objetivo de observar el comportamiento de ambos algoritmos. Con la información obtenida de las pruebas, nos permite analizar e identificar el impacto que posee la cantidad de elementos en el rendimiento.

Como se muestra en Table I, cada resultado es creciente dependiendo de dos variables que se expone en la figura, las cuales son: Operaciones y Tamaño del arreglo. En consecuencia, en este escenario se puede clasificar con una complejidad de tiempo lineal o más específicamente de notación: **O(n)** donde *n* es la cantidad de datos de entrada que probamos.

TABLE I
MEDICIÓN EMPÍRICA ALGORITMO GENÉTICO

Operaciones	Tamaño del arreglo					
	10	50	100	200	500	1000
Asignaciones	113921	597901	1203173	2412597	6043629	12093497
Comparaciones	14233	75248	151591	303997	761905	1524622
Cantidad Lineas Ejecutadas	89415	473395	953667	1913091	4794123	9593991
Tiempo de Ejecución	12.19	51.77	111.15	193.54	485.81	1072.67
Cantidad Lineas de Código	108					

F. Medición Analítica

Consiste en un completo análisis de la complejidad algorítmica para obtener sus expresiones polinomiales y correspondientes representaciones.

Para calcular esta medición, el código fue analizado por sus funciones en conjunto, cada una de ellas indica la expresión polinomial. En la Table II, a pesar de que en la función main existe un ciclo anidado, se reconoció una constante *n* la cual representa el número de generaciones propuesto y *m* que representa el número de individuos de la población; sin embargo, no afecta de manera considerable el algoritmo.

Adicionalmente, es significativo para nosotros señalarlo y evidenciar que podría ocurrir la mayor parte del tiempo analizando otros algoritmos, por lo que la expresión polinomial de cada uno es de complejidad lineal, clasificada como notación **O(n)**. Estos son los resultados del análisis:

TABLE II
MEDICIÓN ANALÍTICA ALGORITMO GENÉTICO

Algoritmo Genético Airstrike	
Función	Pasos
generarPoblacionInicial	6m
randomPadre	5
crearAccion	4
cruceMutacion	16
crearNuevaGeneracion	26m+1
Total (función main)	49mn+4n
Clasificación O Grande	Lineal

G. Medición Gráfica

Este procedimiento se basa en ejecutar las pruebas con menos datos con la intención de crear una representación gráfica de los resultados de la asignación y los resultados de la comparación para visualizar los cambios en el rendimiento y cómo la complejidad del algoritmo afecta estos aspectos. Las cantidades utilizadas fueron: 10, 20, 30, 40, 50, 60, 70, 80, 90 y 100.

El algoritmo genético, mostrado en Figure 4 y Figure 5, presenta un crecimiento proporcional a la cantidad de datos de entrada, el cual puede llegar a ser intratable si se trata de una cantidad inmensa de datos. Por lo tanto, posee un comportamiento de complejidad lineal representado como **O(n)** donde *n* es la una de las cantidad probadas.

Realizar este método permite entender y visualizar el comportamiento de este algoritmo y cómo se representan los datos, además ayuda a detectar patrones, tendencias, relaciones y estructuras en los datos de entrada probados para este proyecto. A continuación se muestran las gráficas de los resultados obtenidos:

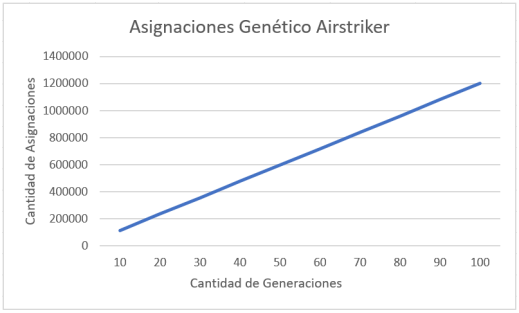

Fig. 4. Asignaciones Algoritmo Genético

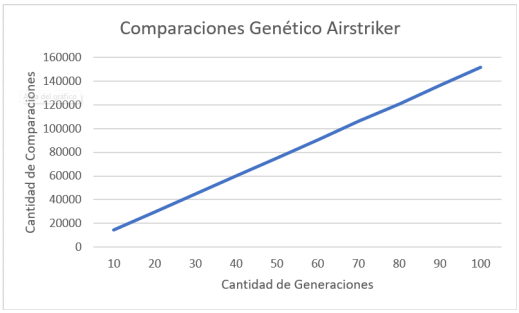

Fig. 5. Comparaciones Algoritmo Genético

H. Factor de Crecimiento

El factor de talla es la forma en que podemos comparar la relación de tamaño de dos datos de entrada y los otros resultados de la relación de tamaño obtenidos mediante la realización de pruebas con esas cantidades de datos de entrada. Este factor mostrará si el aumento de los datos es proporcional o no a los cambios en el rendimiento y dará una pista o una guía sólida sobre cómo se comporta el algoritmo. En este caso se utilizan las siguientes cantidades de datos de entrada: 10, 50, 100, 200, 500 y 1000.

En Table III se observa la conducta del algoritmo genético. Evidentemente, las columnas justo después del factor de tamaño son números cercanos al factor de tamaño de cada tamaño calculado. Está aumentando linealmente y su comportamiento de cuánto aumenta desde el primer dato de entrada hasta el otro es muy similar o cercano, por lo tanto su complejidad es **O(n)**.

TABLE III
FACTOR DE CRECIMIENTO ALGORITMO GENÉTICO

Talla	Factor Talla	Asignaciones	Comparaciones	Lineas Ejecutadas	Tiempo Ejecución
10 a 50	5	597901/113921 = 5.25	75248/14233 = 5.29	473395/89415 5.29	51.77/12.19 = 4.25
50 a 100	2	1203173/597901 = 2.01	151591/75248 = 2.01	953667/473395 = 2.01	111.15/51.77 = 2.15
100 a 200	2	2412597/1203173 = 2.005	303997/151591 = 2.005	1913091/953667 = 2.006	193.54/111.15 = 1.74
500 a 1000	2	12093497/6043629 = 2.001	1524622/761905 = 2.001	9593991/4794123 = 2.001	1072.67/485.81 = 2.21
50 a 500	10	6043629/597901 = 10.10	761905/75248 = 10.13	4794123/473395 = 10.13	485.81/51.77 = 9.38
100 a 1000	10	12093497/1203173 = 10.05	1524622/151591 = 10.06	9593991/953667 = 10.06	1072.67/111.15 = 9.65

V. EVALUACIÓN

Después de todas las pruebas realizadas en el algoritmo probabilístico se planteó la pregunta inicial *¿Cómo afecta la cantidad de datos de entrada al rendimiento de los algoritmos y la forma en la que avanza el nivel?*: La cantidad de datos de entrada puede afectar el rendimiento de un algoritmo y la forma en que afecta depende de cómo se programa el algoritmo. Si la complejidad del algoritmo es cuadrática, aumentar ligeramente la cantidad de datos puede disminuir significativamente la eficiencia, pero en otros tipos de complejidad puede no ser tan significativo. Usando la información recopilada por todas las pruebas ejecutadas, como se ve en Table IV, podemos concluir el algoritmo genético tiene una complejidad lineal que es la notación **O(n)**.

TABLE IV
EVALUACIÓN FINAL

Algoritmo	Medición	Clasificación
Genético	Empírica	Lineal
	Analítica	Lineal
	Gráfica	Lineal
	Factor talla	Lineal
Probabilístico	Empírica	-
	Analítica	-
	Gráfica	-
	Factor talla	-

VI. CONCLUSIONES

Durante la realización de este proyecto, aprendimos cuán importante puede ser la eficiencia de un algoritmo. Incluso los recursos computacionales son limitados y la necesidad de optimizarlos es una realidad a la que nos enfrentamos a diario. Nuestro propósito con este documento fue proporcionar un enfoque analítico a este problema para ayudar a las personas a comprender cómo funciona la complejidad temporal de los algoritmos y cómo la estructura y el diseño del código pueden ser cruciales al optimizar un algoritmo.

Determinamos que el algoritmo genético posee una complejidad lineal, lo que significa que el crecimiento en el tiempo y la memoria será constante dependiendo de la cantidad de datos proporcionados. Esto muestra cómo el único hecho que afecta el tiempo y la memoria que utilizan estos algoritmos es el tamaño de los datos de entrada, respondiendo a la pregunta que nos propusimos al inicio del proyecto. Además, existen algunos errores ya que aprende con la función fitness escogida, pero no lo suficiente para pasar el nivel en su totalidad, lo cual puede mejorarse con más trabajo y algunos cambios en el código programado.

Para el algoritmo probabilístico, no se resolvió el problema dado. Se obtuvo un código; no obstante, faltan detalles y desarrollo para que se ejecute de manera correcta y pueda pasar el nivel del juego correspondiente. Por consiguiente, nos dimos a la tarea de investigar el cómo funciona el algoritmo mencionado, pero sus mediciones y análisis no fueron calculados debido a su estado de completitud y la falta de tiempo para el equipo de terminarlo totalmente.

En el futuro, para ampliar este proyecto y dar aún más evidencia de los efectos que la cantidad de datos de entrada tiene sobre un algoritmo, podríamos analizar diferentes algoritmos además de los ya estudiados en este trabajo o incluso realizar pruebas en diferentes computadoras ya que la potencia computacional es uno de los factores principales de la velocidad de ejecución de un programa y no se está considerando en esta investigación. Asimismo, perfeccionar ambos algoritmos para poder observar las diferencias a la hora de optimizarlos de la mejor manera.

VII. ANEXOS

A. Figuras

B. Tablas

C. Link Repositorio GitHub

- https://github.com/Isaac4918/SegundoProyectoAnalisis

REFERENCES

[1] M. G. Pose, "Introducción a los algoritmos genéticos." [Online]. Available: https://cursa.ihmc.us/rid=1KNKMJ4LN-11XXFSG-1KV5/Algoritmos

[2] J. Bernal and O. Guevara, "Algoritmos genéticos en entornos virtuales," 2007. [Online]. Available: https://repositorio.uci.cu/jspui/bitstream/ident/TD_0502_07/1/TD_0502_07.pdf

[3] Wikipedia, "Algoritmo probabilista," 2022. [Online]. Available: https://es.wikipedia.org/wiki/Algoritmo_probabilista

[4] R. M. de Vega, "Qué es un algoritmo informático: características, tipos y ejemplos," 2021. [Online]. Available: https://profile.es/blog/que-es-un-algoritmo-informatico/

[5] A. Abdiansah and R. Wardoyo, "Time complexity analysis of support vector machines (svm) in libsvm," 2015. [Online]. Available: https://www.ijcaonline.org/research/volume128/number3/abdiansah-2015-ijca-906480.pdf

[6] Readthedocs, "Gym retro," 2019. [Online]. Available: https://retro.readthedocs.io/en/latest/index.html

[7] GeeksForGeeks, "Crossover in genetic algorithm," 2019. [Online]. Available: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

[8] ——, "Mutation algorithms for string manipulation (ga)," 2022. [Online]. Available: https://www.geeksforgeeks.org/mutation-algorithms-for-string-manipulation-ga/