Implementación Algoritmos Primero en Profundidad y Primero en Amplitud

Gómez, Jonathan; Pérez Oliva, J. Simón; Romero Caballero, Facundo N.

Inteligencia Artificial I,
Universidad Gaston Dachary,
Posadas (Misiones), Argentina
{chonidacharygomez@gmail.com, faroni1930@gmail.com - jsimon_perezoliva@hotmail.com}

Abstract. A lo largo de este trabajo se plasmará el funcionamiento de los Algoritmos Primero en Profundidad y Primero en Amplitud (en adelante, PP y PA), abordando la problemática que surge al momento de realizar búsquedas en un espacio de estados de tipo laberinto (10 x 10 casillas) . Una de las posibles soluciones a esta problemática son los algoritmos PP y PA, por un lado, el algoritmo PP proporciona el camino de solución entre dos puntos de manera más rápida generalmente, aunque se ve sesgado fuertemente por el orden de expansión del mismo. Por otro lado, el algoritmo PA garantiza hallar dicha solución a un coste de menor eficiencia. En una primera instancia se introduce al tema con una breve reseña sobre los algoritmos nombrados, abordando la problemática y cómo se logró resolverlo a través de cada algoritmo para luego exponer su implementación en el lenguaje de programación Python, dejando explícito también las diferentes modificaciones sobre los algoritmos que se han realizado para poder unir la interfaz gráfica con el modelo. Finalizando dicha sección, se presentan las diferentes configuraciones necesarias para lograr implementarlos. Además, consta de un apartado de Simulaciones y Resultados en el cual se concluirán las diferentes pruebas realizadas sobre cada algoritmo y cómo cada uno fue resolviendo y generando los diferentes árboles de expansión, y a su vez una comparación entre los resultados arrojados en cantidad de estados del espacio de estados explorados, así como cantidad de iteraciones, para finalmente concluir el trabajo según las simulaciones y resultados obtenidos.

Keywords: PP, PA, árbol, rama, nivel, camino, nodo, lista de nodos a explorar, lista de nodos verificados.

1 Introducción

Los problemas de búsquedas en grandes espacios de soluciones requieren la aplicación de técnicas que permitan reducir el análisis necesario de dichos espacios durante el proceso para poder llegar al objetivo, o garantizar que se halle una solución, si la hubiera; todo dependerá de la cantidad de recursos asignados.

El objetivo de este trabajo es dar solución al problema de búsqueda en grandes espacios de soluciones desarrollando una aplicación que implemente una versión propia de los algoritmos PP y PA. En el desarrollo de los algoritmos, se gestionan los datos con listas de arreglos, no obstante, en la interfaz gráfica se optó por utilizar, además de listas, matrices, árboles y diccionarios, los cuales son objetos disponibles para su uso, brindados por la librería gráfica seleccionada para el presente trabajo.

2 Marco Teórico

El algoritmo Primero en Profundidad (PP) es un método de búsqueda que se enfoca en obtener la ruta más rápida entre dos estados. No siempre se obtendrá el camino de solución de manera más rápida, ya que esto dependerá del orden de expansión escogido. Aún así, reduce el análisis de caminos ya que explora el espacio de búsqueda (representado por un árbol) por ramas, expandiendo una rama si en la rama previa ya no se halló la solución. Cabe mencionar, sin embargo, que pueden ocurrir casos en los que el algoritmo PP tarde demasiado en hallar la solución, o se generen bucles en la búsqueda, que impidan hallar siquiera la solución. [1]

El algoritmo Primero en Amplitud (PA) es un método de búsqueda que se enfoca en garantizar la obtención del camino de solución para el laberinto planteado en el escenario, sin priorizar la obtención de la misma de manera más rápida. El algoritmo PA garantiza que se obtendrá la solución del problema, junto con su camino si la hubiera y dicho camino será el mejor posible, a un costo de eficiencia, debido a que usualmente éste debe explorar casi todo el

espacio de estados, Un árbol es una estructura de datos que funge como representación gráfica de un espacio de estados, también conocido como espacio de búsqueda, el cual facilita la comprensión del proceso de obtención de un camino de solución, partiendo desde un estado inicial (Nodo inicial) hasta un estado objetivo (Nodo final). Los árboles están compuestos por Nodos (Estados individuales representados gráficamente mediante círculos), y Aristas (conexiones entre nodos representadas gráficamente por arcos o líneas que unen los nodos).

Una rama es una representación vertical de un subconjunto del espacio de estados dentro de un árbol. Su nombre surge de la semejanza gráfica entre dicho subconjunto de estados y las ramas de los árboles. Un nivel es una representación horizontal de un subconjunto del espacio de estados dentro de un árbol. Se identifica por poseer nodos "a la misma altura" gráficamente. Un camino es el conjunto de nodos (estados) conectados por aristas que unen el estado inicial con un nodo dado. En caso de que dicho camino una el estado inicial con el estado final (objetivo), dicho camino se denomina camino solución.

3 Implementación de los Algoritmos

A continuación, se realiza una descripción de las tecnologías utilizadas para la implementación de los algoritmos mencionados en la sección 2. Estas tecnologías incluyen: lenguaje de programación, IDE utilizado, librerías importadas, etc.

En la presente implementación, el lenguaje de programación utilizado es Python, debido a su sintaxis simple y fácil de utilizar para el desarrollo del núcleo de la aplicación, además de para las representaciones gráficas de matrices, árboles y grafos en ventanas a través de librerías.

En conjunto con Python, se utilizó el IDE Visual Studio Code para la programación en Python, debido a su versatilidad y facilidad de uso.

Como complemento al core de la aplicación desarrollada, se utilizaron las siguientes librerías:

- PyQt5: librería utilizada para la representación gráfica en ventanas del trabajo. [2]
- Matplotlib: librería utilizada para la representación gráfica de listas en 2 dimensiones. Es decir, se utiliza para la representación gráfica de grafos en ventanas de coordenadas. [3]
- NetworkX: librería utilizada para la construcción de grafos, los cuales incluyen nodos y aristas. [4]
- Random: librería propia de Python para la generación de números pseudo-aleatorios. [5]

4 Desarrollo de los Algoritmos

En la presente sección, se explica en detalle el desarrollo de los algoritmos PP y PA, adaptados a los lineamientos de implementación del presente trabajo, junto con extractos del código fuente, reflejando el desarrollo expuesto. Cabe mencionar que, en la implementación actual, el inicio de cada algoritmo se realiza luego de la generación de un laberinto aleatorio de 10 x 10 casillas.

La lista "aristas_G" contiene las aristas entre pares de nodos libres (nodos que se pueden transitar), pares de nodos libres y nodos bloqueados (estar en un nodo libre y encontrarse con una pared). Además, se hacen uso de dos listas, las cuales se encuentran vacías al inicio y dos variables auxiliares para realizar las verificaciones pertinentes del algoritmo: la lista de nodos a explorar "listaExplorar", la cual va a contener todos los nodos expandidos para explorar por el algoritmo en una instancia de la ejecución del mismo; la lista de nodos verificados "listaVerificados", la cual va a almacenar todos los nodos que ya fueron procesados por el algoritmo; la variable auxiliar "nodoCambioEstado", la cual cambia de estado el nodo actual de "Libre" a "Ocupado", así como también del estado "Bloqueado" a "Ocupado", y el nodo "nodoActual", el cual es el seleccionado para procesarse.

4.1 Algoritmo Primero en Profundidad (PP)

Los pasos realizados para la implementación del algoritmo PP, siguiendo los lineamientos del algoritmo fundamental, fueron los siguientes:

Se inicia el algoritmo luego de haber generado el laberinto, llamando al método

“primeroEnProfundidad”. Dicho método inicializa los nodos “ini” y “fin” que son los estados inicial y estado final representados por el par ordenado de filas y columnas (9,9) y (0,0) respectivamente; el nodo inicial se agrega a la lista a explorar.

El ciclo del algoritmo comienza ahora, ya que iremos a recorrer la lista a explorar mientras no esté vacía, en cada iteración del algoritmo, se escoge el “nodoActual” y “nodoCambioEstado”, siendo este el primer nodo de la lista de nodos a explorar y se elimina el nodo actual de la lista a explorar debido a que pasa ser un nodo explorado. Después se cambia el estado del “nodoCambioEstado” a ocupado y se lo agrega a la lista de nodos verificados, luego se verifica si el nodo actual coincide con el nodo objetivo (nodo fin), si lo es entonces se devuelven la lista de nodos verificados, lista de nodos a explorar y la lista de evolución, para utilizarlas en el apartado de la gráfica; sino se pasa a expandir el nodo actual siguiendo el orden de expansión Izquierda, Arriba, Derecha, Abajo, mediante el método “buscarNodosHijos” que además verifica que ninguno de los nodos hijos coincida con el nodo padre del nodo actual; En el presente algoritmo, si bien se opta por expandir todos los nodos hijos, dicha expansión representa la carga de los mismos a memoria, mas no el procesamiento de éstos. Luego de esto, se realiza un control sobre la lista de nodos verificados, en la cual se corrobora que no existan nodos ya procesados en la lista de nodos a explorar mediante el método “controlDeBucle2” y, de encontrarse alguno,se quita de la lista de hijos los nodos que coinciden con nodos de la lista de nodos verificados, después se quita de la lista a explorar los nodos que coinciden con nodos de la lista de hijos. Al finalizar este control, se agregan todos los nodos hijos al inicio de la lista de nodos a explorar.

Se repite el proceso descrito hasta encontrar el nodo objetivo (nodo fin) o hasta explorar todo el espacio de búsqueda. Cabe mencionar que también se hace uso de una lista denominada “listaEvolucion”. Sin embargo, la misma se utiliza únicamente para almacenar y posteriormente mostrar en la aplicación de escritorio el desarrollo paso a paso del algoritmo PP en formato de texto. No afecta en ninguna manera al desarrollo de dicho algoritmo.

4.2 Algoritmo Primero en Amplitud (PA)

Los pasos realizados para la implementación del algoritmo PA, siguiendo los lineamientos del algoritmo fundamental, fueron los siguientes:

Se inicia el algoritmo luego de haber generado el laberinto, llamando al método “primeroEnProfundidad”. Dicho método inicializa los nodos “ini” y “fin” que son los estados inicial y estado final representados por el par ordenado de filas y columnas (9,9) y (0,0) respectivamente; el nodo inicial se agrega a la lista a explorar.

El ciclo del algoritmo comienza ahora, ya que iremos a recorrer la lista a explorar mientras no esté vacía, en cada iteración del algoritmo, se escoge el “nodoActual” y “nodoCambioEstado”, siendo este el primer nodo de la lista de nodos a explorar y se elimina el nodo actual de la lista a explorar debido a que pasa ser un nodo explorado. Después se cambia el estado del “nodoCambioEstado” a ocupado y se lo agrega a la lista de nodos verificados, luego se verifica si el nodo actual coincide con el nodo objetivo (nodo fin), si lo es entonces se devuelven la lista de nodos verificados, lista de nodos a explorar y la lista de evolución, para utilizarlas en el apartado de la gráfica; sino se pasa a expandir el nodo actual siguiendo el orden de expansión Izquierda, Arriba, Derecha, Abajo, mediante el método “buscarNodosHijos” que además verifica que ninguno de los nodos hijos coincida con el nodo padre del nodo actual; En el presente algoritmo, si bien se opta por expandir todos los nodos hijos, dicha expansión representa la carga de los mismos a memoria, mas no el procesamiento de éstos. Luego de esto, se realiza un control sobre la lista de nodos verificados, en la cual se corrobora que no existan nodos ya procesados en la lista de nodos a explorar mediante el método “controlDeBucle1” y, de encontrarse alguno,se quita de la lista de hijos los nodos que coinciden con nodos de la lista de nodos verificados, después se quita de la lista a explorar los nodos que coinciden con nodos de la lista de hijos. Al finalizar este control, se agregan todos los nodos hijos al final de la lista de nodos a explorar.

Se repite el proceso descrito hasta encontrar el nodo objetivo (nodo fin) o hasta explorar todo el espacio de búsqueda. Cabe mencionar que también se hace uso de una lista denominada “listaEvolucion”. Sin embargo, la misma se utiliza únicamente para almacenar y posteriormente mostrar en la aplicación de escritorio el desarrollo paso a paso del algoritmo PA en formato de texto. No afecta en ninguna manera al desarrollo de dicho algoritmo.

5 Simulaciones y Resultados

Para corroborar el funcionamiento de los algoritmos descritos en la sección anterior, bajo la condición de llegar al objetivo, la simulación se realiza utilizando un conjunto de cinco laberintos aleatorios. De esta forma el algoritmo debe recorrer el laberinto por el camino que se encuentre de manera sistemática hasta el objetivo, es decir, siguiendo los pasos expresados previamente según el algoritmo elegido en las secciones 4.1 y 4.2. A continuación se presentan, a modo de ejemplo, los resultados de una prueba aleatoria de resolución de un laberinto en su forma matricial, su árbol de expansión y su evolución paso a paso, para denotar gráficamente la interfaz de la aplicación de escritorio desarrollada.

Primero en Profundidad. **Primero en Amplitud.**

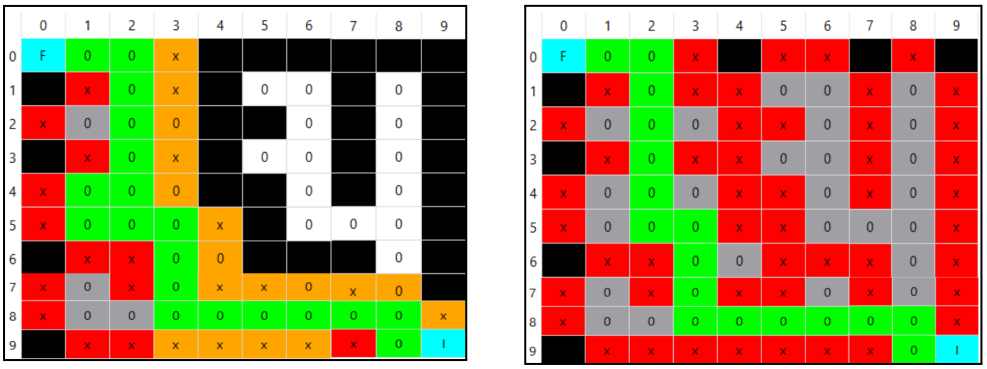

Figura 1. Laberinto aleatorio resuelto mediante PP y PA, de izquierda a derecha respectivamente.

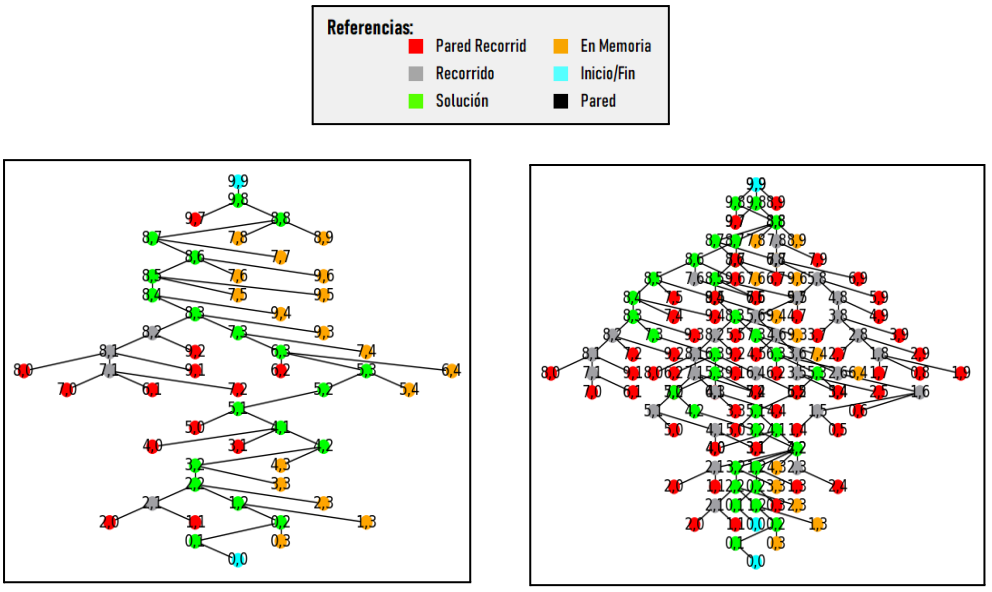

Figura 2. Árboles de expansión generados por PP y PA, de izquierda a derecha respectivamente.

Figura 3. Extracto final de la Evolución Paso a Paso generada por PP y PA, de izquierda a derecha respectivamente.

A continuación se toma una muestra de veinte pruebas, generando laberintos de 10 x 10 casillas, utilizando la aplicación desarrollada y tomando los siguientes ítems como datos relevantes de dichas pruebas:

- Cantidad de Iteraciones: cantidad total de iteraciones realizadas por el algoritmo para hallar la solución.
- Tamaño del Camino Solución: cantidad total de nodos que componen el camino de solución hallado por el algoritmo.
- Porcentaje del Espacio de Estados Explorado (%): porcentaje total del espacio de estados explorado por el algoritmo para hallar la solución. Este porcentaje sigue la forma:

$$Porcentaje\ Explorado = \frac{Cantidad\ de\ Nodos\ Explorados}{Cantidad\ Total\ de\ Nodos\ del\ Espacios\ de\ Estados}$$

- Porcentaje en Memoria (%): porcentaje total del espacio de estados que fue cargado en memoria, pero no fue procesado por el algoritmo. Este porcentaje sigue la forma:

$$Porcentaje\ en\ Memoria = \frac{Cantidad\ de\ Nodos\ Cargados\ en\ Memoria}{Cantidad\ Total\ de\ Nodos\ del\ Espacios\ de\ Estados}$$

N° Prueba	1		2		3		4		5	
	PP	PA	PP	PA	PP	PA	PP	PA	PP	PA
Cantidad de Iteraciones	68	89	47	94	51	95	54	91	48	91
Tamaño del Camino Solución (N° de Nodos)	19	19	21	19	19	19	25	21	19	19
Porcentaje del Espacio de Estados Explorado (%)	67 %	88 %	46 %	93 %	50 %	94 %	53 %	90 %	47 %	90 %
Porcentaje del Espacio de Estados cargado en Memoria (%)	19 %	1 %	16 %	1 %	16 %	0 %	20 %	0 %	13 %	0 %

Tabla 1. Resultados de las pruebas realizadas, desde la prueba N° 1 hasta la prueba N° 5.

N° Prueba	6		7		8		9		10	
	PP	PA	PP	PA	PP	PA	PP	PA	PP	PA
Cantidad de Iteraciones	54	89	40	91	32	85	30	85	35	93
Tamaño del Camino Solución (N° de Nodos)	19	19	27	19	19	19	19	19	23	19
Porcentaje del Espacio de Estados Explorados (%)	53 %	88 %	39 %	90 %	31 %	84 %	29 %	84 %	34 %	92 %
Porcentaje del Espacio de Estados cargado en Memoria (%)	11 %	1 %	23 %	1 %	31 %	0 %	17 %	1 %	22 %	0 %

Tabla 2. Resultados de las pruebas realizadas, desde la prueba N° 6 hasta la prueba N° 10.

N° Prueba	11		12		13		14		15	
	PP	PA	PP	PA	PP	PA	PP	PA	PP	PA
Cantidad de Iteraciones	58	91	49	91	48	90	29	94	49	90
Tamaño del Camino Solución (N° de Nodos)	19	19	19	19	21	19	19	19	21	21
Porcentaje del Espacio de Estados Explorados (%)	57 %	90 %	48 %	90 %	47 %	89 %	28 %	93 %	48 %	89 %
Porcentaje del Espacio de Estados cargado en Memoria (%)	16 %	0 %	17 %	0 %	16 %	0 %	18 %	1 %	15 %	1 %

Tabla 3. Resultados de las pruebas realizadas, desde la prueba N° 11 hasta la prueba N° 15.

N° Prueba	16		17		18		19		20	
	PP	PA	PP	PA	PP	PA	PP	PA	PP	PA
Cantidad de Iteraciones	42	91	65	92	31	90	31	75	31	91
Tamaño del Camino Solución (N° de Nodos)	25	19	19	19	21	19	19	19	21	19
Porcentaje del Espacio de Estados Explorados (%)	41 %	90 %	64 %	91 %	30 %	89 %	30 %	74 %	30 %	90 %
Porcentaje del Espacio de Estados cargado en Memoria (%)	22 %	0 %	16 %	1 %	15 %	0 %	13 %	4 %	21 %	1 %

Tabla 4. Resultados de las pruebas realizadas, desde la prueba N° 16 hasta la prueba N° 20.

Cabe mencionar que se dividen los resultados en 4 tablas con el fin de mantener la legibilidad de los datos intacta.

	PP	PA
Promedio de Iteraciones	44,6	89,9
Promedio de Tamaño de Camino Solución	20,7	19,2
Tamaño Mínimo de Camino Solución	19	19
Tamaño Máximo de Camino Solución	27	21
Promedio de Espacio de Estados Explorado	43,60 %	88,90 %
Promedio de Espacio de Estados cargado en Memoria	17,25 %	0,65 %

Tabla 5. Resultados estadísticos obtenidos en base a las veinte pruebas realizadas.

6 Conclusión

En este informe se presenta el desarrollo de los algoritmos PP y PA, implementados como resolución a la problemática de hallar un camino que conecte la entrada con la salida en un laberinto de 10 x 10 casillas (Espacio de Estados de 100 nodos). Para tal fin, se han propuesto veinte pruebas, con el fin de demostrar el correcto funcionamiento de cada algoritmo y sus comportamientos respecto a los diferentes datos de entrada (Laberintos). Para dichas pruebas, se han obtenido los siguientes datos estadísticos:

1. El 100% de las veces, tanto el algoritmo PP como el PA encuentran un camino de solución que conecte la entrada con la salida de cada laberinto que se le presente al modelo propuesto.
2. El promedio de iteraciones del algoritmo PP es de 44,6, mientras que el promedio de iteraciones del algoritmo PA es de 89,9. Esto significa que, en promedio, el algoritmo PP necesita menos de la mitad de iteraciones que el algoritmo PA para hallar una solución.
3. El tamaño mínimo del camino de solución corresponde al 19% del espacio de estados. Es decir, cada algoritmo debe explorar, como mínimo, un 19% de todo el espacio de estados para hallar una solución.
4. En base a las estadísticas obtenidas de máximo y mínimo de tamaño de camino solución, se denota que el algoritmo PP, a pesar de necesitar la mitad de iteraciones que el algoritmo PA para hallar una solución, existe un mayor rango de variación entre el mínimo y el máximo mencionado (variación de 8 nodos), mientras que dicho rango de variación en el algoritmo PA es sólo de 2 nodos.
5. El algoritmo PP, en promedio, necesitó explorar un 43,60% del espacio de estados, con un conjunto de estados cargados en memoria, más nunca procesados de 17,25%. Para el caso del algoritmo PA, el mismo necesitó explorar en promedio un 88,9% del espacio de búsqueda, con un conjunto de estados cargados en memoria, pero nunca procesados correspondiente a 0,65%.

Con todos los ítems expuestos previamente, además de los datos obtenidos de las pruebas efectuadas, se verifica el correcto funcionamiento de los algoritmos PP y PA para el escenario propuesto, debido a que halla todas las veces un camino de solución para la problemática planteada en este trabajo. Por otro lado se concluye que, luego de redactar el informe, se denota que ambas implementaciones de los algoritmos siguen una misma secuencia y que se diferenciaban en su control de bucle y que, en promedio, el algoritmo PP utiliza más eficientemente los recursos que el algoritmo PA para hallar una solución a un laberinto presentado, dependiendo del orden de expansión de los nodos. Esto quiere decir que su eficiencia se ve sesgada por la elección del orden de expansión de los nodos. Sin embargo, se denota que el algoritmo PA, a pesar de ser menos eficiente que el algoritmo PP para hallar la solución, halla la mejor solución para el problema presentado (el camino solución más corto). Esto concuerda con la naturaleza de ambos algoritmos, descritas en la sección 2 de este trabajo; Por lo cual, se recomienda utilizar el algoritmo PP para escenarios en donde se prioricen la eficiencia de los recursos, debido a diversas causas, y se recomienda utilizar el algoritmo PA para escenarios en los que se priorice la obtención de la mejor solución posible, a coste de un mayor uso de recursos.

Referencias

[1] Rich, E. Knight, K. Inteligencia Artificial. Mc Graw Hill. 1996. Capítulo 2.

[2] Proyecto PyQt. Available: https://pypi.org/project/PyQt5/

[3] Proyecto Matplotlib. Available: https://matplotlib.org/

[4] Proyecto NetworkX. Available: https://networkx.org/

[5] Documentación de Librería Random - Proyecto Python. Available: https://docs.python.org/3/library/random.html