© Revista de Matemática: Teoría y Aplicaciones 2019 **26**(2) : 319–337
cimpa – ucr issn: 1409-2433 (Print), 2215-3373 (Online)
doi: https://doi.org/10.15517/rmta.v26i2.38322

UN ALGORITMO ESTOCÁSTICO PARA RESOLVER LABERINTOS

A STOCHASTIC ALGORITHM FOR SOLVING MAZES

Iván Omar Cruz-Ruiz* Pedro Lara-Velázquez†
Miguel A. Gutiérrez-Andrade‡
Sergio G. de-los-Cobos-Silva§ Eric A. Rincón-García¶
Román A. Mora-Gutiérrez‖

Received: 10/Apr/2018; Revised: 16/May/2019;
Accepted: 23/May/2019

---

*Universidad Autónoma Metropolitana-Iztapalapa, Departamento de Ingeniería Eléctrica, Ciudad de México, México. E-Mail: iocr@xanum.uam.mx
†Misma dirección que/Same address as: I.O. Cruz-Ruiz. E-Mail: plara@xanum.uam.mx
‡Misma dirección que/Same address as: I.O. Cruz-Ruiz. E-Mail: gamma@xanum.uam.mx
§Misma dirección que/Same address as: I.O. Cruz-Ruiz. E-Mail: cobos@xanum.uam.mx
¶Misma dirección que/Same address as: I.O. Cruz-Ruiz. E-Mail: rincon@xanum.uam.mx
‖Universidad Autónoma Metropolitana-Azcapotzalco, Departamento de Sistemas, Ciudad de México, México. E-Mail: mgra@correo.azc.uam.mx

Resumen

El artículo describe un nuevo método para resolver laberintos cuadrados usando una versión aleatorizada de búsqueda a profundidad. El algoritmo propuesto se probó en dos familias de laberintos, una de ellas basada en el método de Aldous-Broder y la otra en el de Backtrack. El algoritmo de solución se compara con el método de Dijkstra, que es una técnica bien conocida para resolver este tipo de problemas. Este encuentra soluciones en menor tiempo en laberintos de gran tamaño(mayores a100x100 celdas).

Palabras clave: optimización combinatoria; laberintos cuadrados; algoritmos aleatorizados; árboles y gráficas.

Abstract

In this article a new method to solve square mazes using a randomized depth-first search algorithm is described. The algorithm was tested in two families of labyrinths, one of them based on the Aldous-Broder method and the other on Backtrack. The algorithm was compared against the Dijkstra method, a well-known technique to solve this kind of problems. The new method finds solutions in less time for large-size labyrinths (greater than 100 x 100 cells).

Keywords: combinatorial optimization; square mazes; randomized algorithms; trees and graphs.

Mathematics Subject Classification: 68W20, 90C27, 37E25.

1 Introducción

"Un laberinto es un lugar en el que caminos y encrucijadas forman un conjunto de rutas que vuelven compleja su solución y está diseñado para confundir a quien se adentre en él" [8].

Los laberintos son "figuras" interesantes que se encuentran ampliamente en la cultura popular, desde películas (*The Maze* de 1953, *The Shining* de 1980 o *Maze Runner* de 2014 entre muchas otras), juegos simples en papel o incluso videojuegos (desde *Maze Craze* para Atari de 1978 pasando por *Tomb Raider* de 1996 hasta *There is a Way* de 2018). Los laberintos siempre han sido atractivos para la psique humana, desde los más simples que se ponen en una manteleta de desayuno para niños hasta los más complejos, que pueden ser un reto incluso para los expertos.

Desde el siglo II antes de Cristo, en la isla de Creta, se habla del Laberinto del Minotauro, el cual era tan intrincado que Dédalo e Ícaro solo pudieron salir de él volando. En la misma naturaleza podemos encontrar sistemas de cavernas subterráneas que forman estructuras como laberintos. Posiblemente el sistema cavernoso que se encuentra en Gortina, al sur de Creta, sea el origen del concepto de los laberintos. Este sistema está formado por varios kilómetros de túneles y varias docenas de bifurcaciones. En este lugar es común que cuando una persona inexperta se adentra, puede tardar días en salir o incluso morir en el intento.

Existen laberintos de varias formas: circulares, octagonales, hexagonales, cuadrados, etc [9]. En este caso se abordan los laberintos cuadrados, que son los más conocidos y usados, ya que están delimitados por una figura cuadrada y al interior se tienen caminos rectangulares y cuadrados, que forman esquinas de 90 grados.

De acuerdo con la forma en que se llega a la meta, los laberintos se clasifican en dos tipos:

- **Laberinto clásico**: La meta se encuentra al centro, solo hay un camino para llegar a esta y solo hay una puerta para ingresar y salir de él.

- **Laberinto moderno**: Es aquel donde los caminos que lo conforman se interconectan entre sí y no poseen otros que llegan de nuevo al mismo punto de partida.

De todos los tipos y clasificaciones de laberintos, este artículo se centra en la generación y solución de laberintos cuadrados modernos.

2 Generación de instancias de laberintos

En esta sección se abordan dos formas para generar laberintos, los algoritmos **Aldous-Broder** y **Backtrack**, que son algoritmos bien conocidos que generan laberintos difíciles de resolver. Los dos modelos siguen el mismo principio, donde se hace una analogía en la que se tiene una construcción que está formada por una matriz de cuartos, que en principio, ninguno de ellos tiene comunicación y el algoritmo se encarga de ir rompiendo "paredes" y de acuerdo a sus reglas particulares formará el laberinto. Se usan los números 1 para las columnas de los cuartos, 2 para las paredes de los cuartos y 3 para indicar el espacio central que se genera al tener las cuatro paredes [1]. Esta información se guarda en una matriz cuadrada de tamaño $(2n + 1) \times (2n + 1)$, donde n es el número de cuartos por fila o por columna. Se puede ver en la Figura 1 la generación de una matriz inicial de tamaño 3×3 que requiere de un arreglo de tamaño 7×7 en la que la primer línea está formada por columnas "1" y paredes "2", la segunda línea por paredes "2" y espacios "3"y así sucesivamente [5].

1	2	1	2	1	2	1
2	3	2	3	2	3	2
1	2	1	2	1	2	1
2	3	2	3	2	3	2
1	2	1	2	1	2	1
2	3	2	3	2	3	2
1	2	1	2	1	2	1

Figura 1: Matriz de "cuartos" para la generación de laberintos. Los "1" representan columnas, los "2" paredes y los "3" cuartos. Laberinto de tamaño 3×3.

2.1 Algoritmo de Aldous-Broder

Esta es una de las técnicas más simples en la generación de laberintos complejos pero es una de las menos eficientes. D. Aldous y A. Broder trabajaron independientemente en árboles de expansión uniformes (árbol elegido al azar entre todos los posibles en un grafo). El algoritmo como tal, consiste en elegir arbitrariamente una localidad en la matriz (un cuarto) y moverse aleatoriamente entre celdas. Si se llega a una celda que no ha sido visitada, se une esta en donde se estaba con la nueva celda encontrada. El método finaliza cuando todas las celdas en la matriz han sido visitadas [2].

El pseudocódigo de Aldous-Broder se presenta en el Algoritmo 1.

En la línea 1 se declaran las variables que van a generar el laberinto, para esto se escoge una posición aleatoria en la matriz. En la línea 2 se entra a un ciclo en el que primero aumenta el valor de la casilla en la que se encuentra parado, si tiene valor 3 es que no ha sido visitada y si tiene valor 4 o mayor, ya lo fue. Después ingresa a un nuevo ciclo en la línea 4 donde primero se elige aleatoriamente hacia donde va a realizar el siguiente movimiento (Norte, Sur, Este u Oeste), una vez elegido el movimiento, entra a una serie de decisiciones desde la línea 6 a la 21 en que, dependiendo de la dirección a la que se mueve, se ingresa a tal condición, en caso de que se cumpla la dirección y esta nueva casilla no haya sido visitada anteriormente, entonces se procede a "romper la pared",

Algoritmo 1 Aldous Broder.

```
a1,a2
repetir
    matrix(a1,a2)=matrix(a1,a2)+1
    repetir
        aux= elige movimiento Norte, Sur, Este u Oeste
        si Norte y no visitado entonces
            romper pared Norte
            moverse a siguiente casilla
        fin si
        si Sur y no visitado entonces
            romper pared Sur
            moverse a siguiente casilla
        fin si
        si Este y no visitado entonces
            romper pared Este
            moverse a siguiente casilla
        fin si
        si Oeste y no visitado entonces
            romper pared Oeste
            moverse a siguiente casilla
        fin si
    hasta que hay casillas alrededor sin visitar
    repetir
        a1=rnd*nn : a2=rnd*mm
    hasta que se encuentre una casilla con vecinos sin visitar
hasta que haya casillas sin visitar
```

que es un método en el cual se cambia el valor de la casilla en la matriz; continuando con la analogía de los cuartos, la "pared" se transforma en un camino y posiciona las variables (a1,a2) en la nueva casilla abierta. Este ciclo se realiza mientras se tenga una casilla contigua a la que se pueda pasar. En la línea 23 entramos en un nuevo ciclo que busca una nueva casilla que tenga vecinos que no hayan sido visitados antes. El algoritmo continúa casillas por visitar.

Al analizar el pseudocódigo, se puede deducir que al inicio, el algoritmo se mueve rápidamente entre celdas vacías, pero al disminuir la cantidad de celdas vacías, se vuelve lento, ya que no termina hasta que toda la matriz sea visitada.

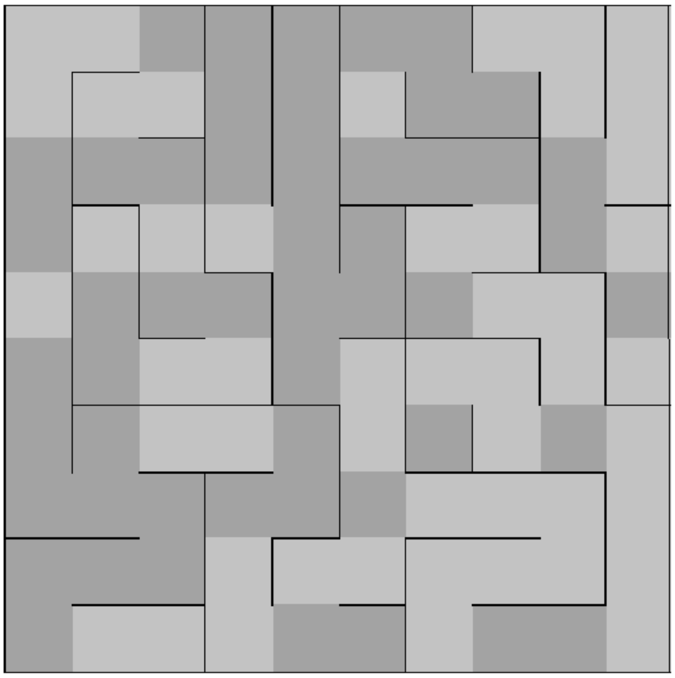

Figura 2: Implementación del método de Aldous-Broder para la generación de un laberinto de tamaño 10×10.

Al realizar distintas corridas del algoritmo (Figuras 2 y 3), se observan algunos recuadros en gris oscuro, se debe a que el algoritmo visitó estas celdas varias veces para encontrar las que faltan por visitar, debido a la forma aleatoria en que se busca una casilla cuya vecindad no ha sido visitada anteriormente, el proceso se hace más lento conforme se visitan más de estas.

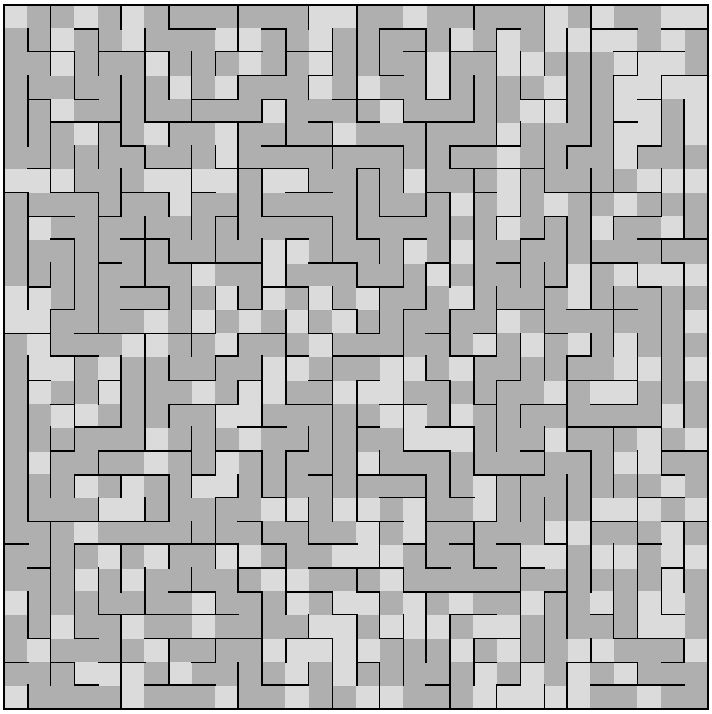

Figura 3: Implementación del método de Aldous-Broder para la generación de un laberinto de tamaño 30×30.

2.2 Algoritmo Backtrack

Backtrack es una técnica de búsqueda sistemática a través de todas las configuraciones posibles dentro de un espacio de soluciones. En este algoritmo se utiliza el inicio del algoritmo anterior (Aldous-Broder) en el que aleatoriamente se selecciona una casilla inicial de la matriz y también, aleatoriamente se elige a qué casilla contigua se mueve, si esta ya fue visitada con anterioridad, entonces elegimos otra casilla. Esto se realiza hasta que no sea posible elegir una nueva casilla de las cuatro posibles que se pueden elegir. Al quedar estancado, el algoritmo regresa por el camino que ha ido formando, evaluando si hay alguna casilla sin visitar en el camino, al encontrarse una casilla sin visitar, vuelve a realizar lo anterior, elige aleatoriamente la posición a la que se va a mover y si esta no ha sido visitada, procede a formar el camino. El método es más complejo que el anterior, pero la ventaja es que el tiempo de ejecución es menor, debido a que resuelve de una manera más eficiente la búsqueda de una vecindad no visitada.

El pseudocódigo de Backtrack se presenta en el Algoritmo 2.

Algoritmo 2 Backtrack.

```
1: a1,a2
2: repetir
3:     matrix(a1,a2)=matrix(a1,a2)+1
4:     repetir
5:         aux= elige movimiento Norte, Sur, Este u Oeste
6:         si Norte y no visitado entonces
7:             romper pared Norte
8:             moverse a siguiente casilla
9:             Guardar posición
10:        fin si
11:        si Sur y no visitado entonces
12:            romper pared Sur
13:            moverse a siguiente casilla
14:            Guardar posición
15:        fin si
16:        si Este y no visitado entonces
17:            romper pared Este
18:            moverse a siguiente casilla
19:            Guardar posición
20:        fin si
21:        si Oeste y no visitado entonces
22:            romper pared Oeste
23:            moverse a siguiente casilla
24:            Guardar posición
25:        fin si
26:    hasta que hay casillas alrededor sin visitar
27:    repetir
28:        volver una posición
29:        si hay casillas sin visitar alrededor entonces
30:            comenzar en esta posición
31:        fin si
32:    hasta que haya posiciones guardadas o posición encontrada
33: hasta que haya posiciones guardadas
```

En la línea 1 se declaran las variables que van a generar el laberinto, para esto se escoge una posición aleatoria en la matriz. En la línea 2 entra a un ciclo en el que primero se va a aumentar el valor de la casilla en la que se encuentra parado, si tiene valor 3 no ha sido visitada y si tiene valor 4 o mayor, sí ha sido visitada. Después, entra a un nuevo ciclo en la línea 4 donde elige aleatoriamente hacia donde realizar el siguiente movimiento (Norte, Sur, Este u Oeste), una vez elegido el movimiento entra a una serie de condiciones (línea 6 a 25) donde, dependiendo de la dirección, es como entra a tal condición, en caso de que se cumpla la dirección y esta nueva casilla no haya sido visitada anteriormente, se "rompe la pared", que es un método en el cual se cambia el valor de la casilla en la matriz; continuando con la analogía de los cuartos, en vez de ser "pared" pasa a ser un camino, posiciona las variables (a1,a2) en la nueva casilla abierta, este ciclo se realiza mientras se tenga una casilla alrededor a la cual se pueda pasar y guarda la nueva posición en una pila. Al continuar en la línea 27, entra en un nuevo ciclo donde extrae de la pila la nueva posición de búsqueda, si esta posición tiene vecinos sin visitar, entonces se regresa al ciclo anterior, en caso contrario se siguen extrayendo posiciones de la pila hasta que encuentre una nueva posición en la cual continuar la búsqueda hasta que se vacíe la pila, es decir, hasta que se haya visitado toda la matriz.

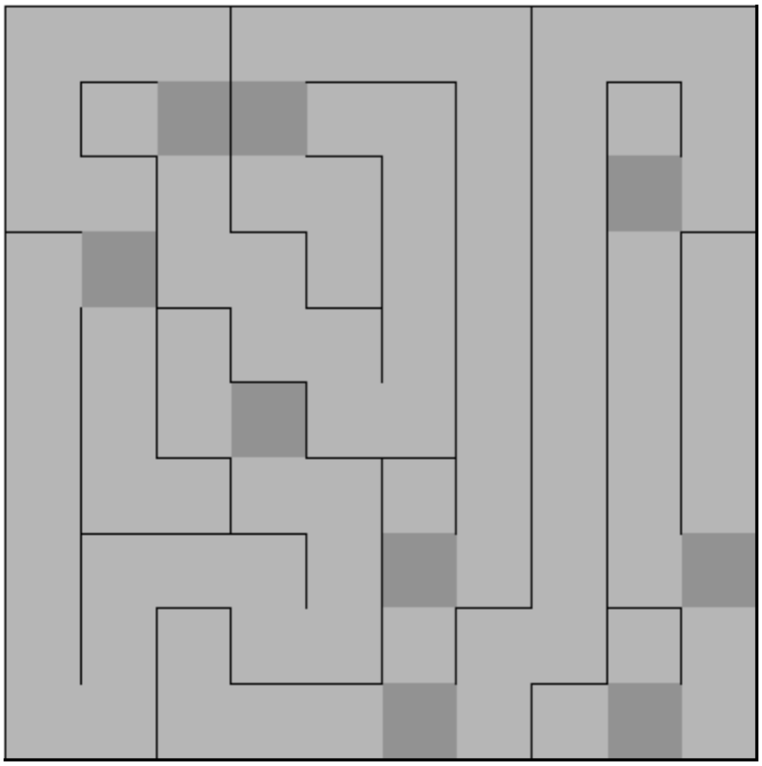

Figura 4: Método de Backtrack, laberinto de tamaño 10×10.

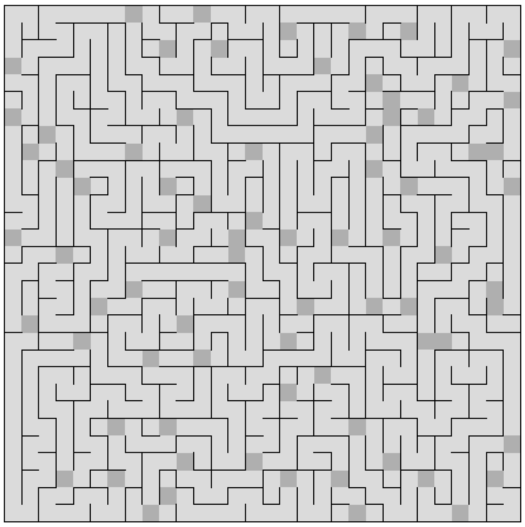

Figura 5: Método de Backtrack, laberinto de tamaño 30×30.

En las Figuras 4 y 5 las casillas gris obscuro son las casillas en donde el algoritmo Backtrack encontró una nueva salida para continuar con la generación del laberinto, estas casillas son menos que las del algoritmo anterior, es decir, visita menos casillas ya revisadas y por lo tanto, este método termina en menos tiempo.

3 Solución de laberintos

En esta sección se comparan dos algoritmos: el algoritmo de Dijkstra y el denominado Búsqueda Profunda. Para la ejecución de estos algoritmos, se utilizó una computadora con procesador Core i3, 4Gb de memoria RAM, 500 Gb de disco duro, sistema operativo Linux Mint y se programaron en FreeBASIC. Se realizaron ejecuciones para laberintos de distintos tamaños (10, 30, 60, 90, 120, 150, 180, 210, 240, 270 y 300), para cada tamaño de laberinto, se generaron diez instancias usando los algoritmos de Aldous-Broder y Backtrack mencionados anteriormente y se resolvieron con los algoritmos de Dijkstra y Búsqueda profunda respectivamente. En la Sección 4 se reportan los tiempos promedio de las ejecuciones de ambos algoritmos, así como la comparación de los mismos de acuerdo a su eficiencia temporal. Por consistencia y sin perder generalidad, se decidió colocar en la esquina inferior izquierda el inicio y el final en la esquina superior derecha, aunque ese criterio es arbitrario, debe ser idéntico para todas las instancias ya que se está evaluando la topología del laberinto, no la relevancia de los puntos de entrada y salida del mismo.

3.1 Benchmark: algoritmo de Dijkstra

El conocido algoritmo de Dijkstra se implementó como punto de comparación [7]. Como se puede observar en la Figura 6, al crecer el índice en las celdas visitadas se puede ver el orden en que se van reconociendo las celdas. La solución se puede reconstruir a partir del punto final (salida) del laberinto. La ruta está dada por la única trayectoria que llega hasta el índice 1 (Figura 6). El algoritmo muestra un efecto de “inundación” en donde abarca todo el laberinto explorando todos los caminos posibles hasta alcanzar la meta. Ver Figuras 7 y 8.

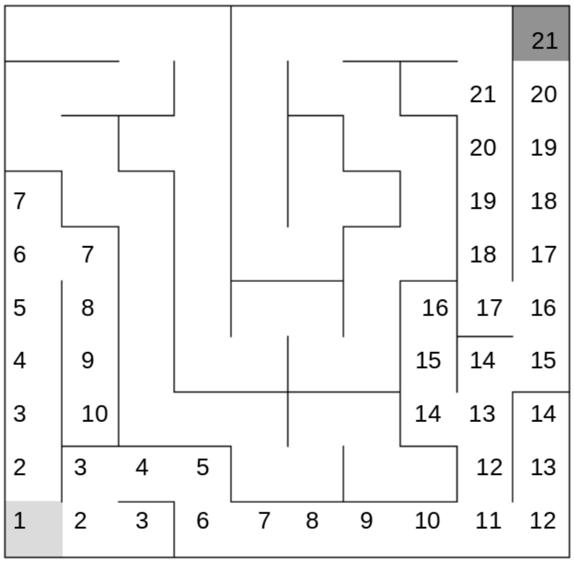

Figura 6: Método de Dijkstra, laberinto de tamaño 10×10 generado por Backtrack. Comparar con la Figura 7.

El pseudocódigo de Dijkstra se presenta en el Algoritmo 3.

En la línea 1 se declaran las variables (a1,a2) que son las que indican el inicio del laberinto y (f1,f2) que son las que indican el final, despues guarda en una pila la posición inicial (a1,a2). En la línea 3 entra en un ciclo donde explora los vecinos de la celda en las direcciones Norte, Sur, Este y Oeste. En caso de que se encuentre un camino hacia una dirección, entonces, ingresa la casilla encontrada a la pila y marca la casilla en la matriz de acuerdo al nivel que lleve en el árbol que se forma, después saca el siguiente elemento de la pila en la línea 20 y repite el procedimiento hasta que se encuentre la posición final (f1,f2).

Algoritmo 3 Dijkstra.

```
(a1,a2,f1,f2)
Guardar en una fila la celda inicial
repetir
    si hay paso hacia el Norte entonces
        Ingresar posición a la pila
        Marcar casilla en la matriz
    fin si
    si hay paso hacia el Sur entonces
        Ingresar posición a la pila
        Marcar casilla en la matriz
    fin si
    si hay paso hacia el Este entonces
        Ingresar posición a la pila
        Marcar casilla en la matriz
    fin si
    si hay paso hacia el Oeste entonces
        Ingresar posición a la pila
        Marcar casilla en la matriz
    fin si
    Sacar el siguiente elemento de la pila
hasta que se encuentre el final
```

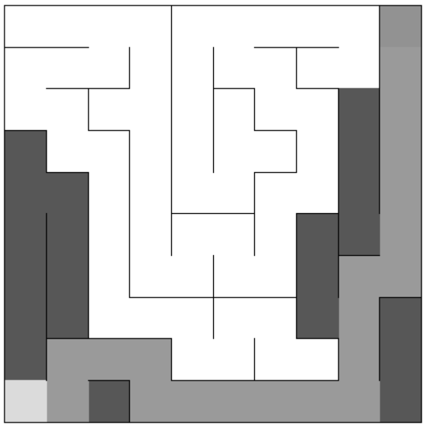

Figura 7: Método de Dijkstra, generado por Backtrack. Laberinto de tamaño 10×10.

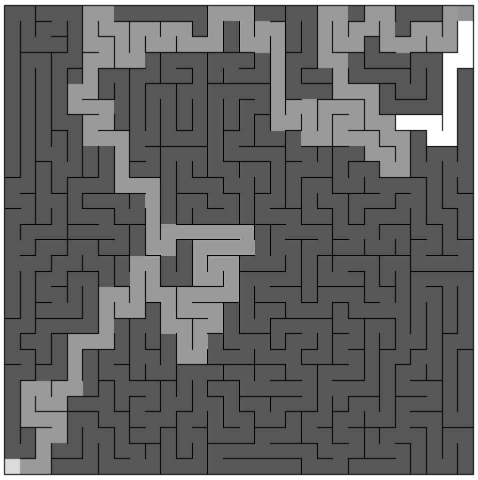

Figura 8: Método de Dijkstra, generado por Aldous-Broder. Laberinto de tamaño 30×30.

3.2 Algoritmo propuesto: búsqueda profunda aleatorizada

El algoritmo busca aleatoriamente una ruta para llegar a la casilla final, en cada paso verifica si hay más de un camino que se puede recorrer. Si es el caso, guarda la posición en una pila para que cuando encuentre un camino sin salida, regrese a la casilla anterior de la pila y continúe la búsqueda desde esa posición, este proceso se repite hasta encontrar la salida. Ver las Figuras 9 y 10, donde se muestran en gris obscuro las rutas que llevaban a callejones sin salida, la solución es la ruta en gris claro que va de la posición inicial (esquina inferior izquierda) a la salida (esquina superior derecha). En blanco se muestran los caminos que no fue necesario explorar para resolver el problema.

El pseudocódigo de Búsqueda Profunda Aleatorizada se presenta en el Algoritmo 4.

En la línea 1 se declaran las variables (a1,a2) que son las que indican el inicio del laberinto y (f1,f2) indican el final, también se guarda la posición inicial de la pila (a1,a2). En la línea 3 se entra a un ciclo anidado, en donde se escoge una dirección aleatoriamente (Norte, Sur, Este u Oeste) y si en la dirección escogida hay un camino, entonces la posición se actualiza a dicha casilla. Si la casilla tiene más de un camino, entonces se agrega a la pila, este procedimiento se repite mientras se tenga un camino que recorrer, si se llega a un callejón sin salida, entonces se toma el valor siguiente de la pila y se regresa al ciclo anterior. El algoritmo termina cuando se alcanza la casilla final.

Algoritmo 4 Búsqueda Profunda Aleatorizada.

```
(a1,a2,f1,f2)
Guardar en pila la celda inicial
repetir
    repetir
        dirección= escoger dirección aleatoriamente
        si salidas(dirección)=1 entonces
            Mueve a siguiente posición
            Marca casilla
        fin si
        si tiene más movimientos posibles entonces
            agregar a pila
        fin si
    hasta que haya camino
    si no hay camino entonces
        a1,a2 toman valor siguiente de pila
    fin si
hasta que no haya llegado a casilla final
```

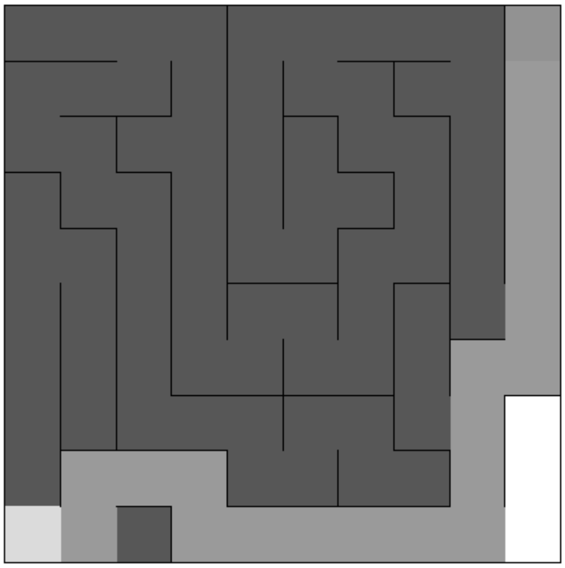

Figura 9: Método de Búsqueda Profunda. Laberinto de tamaño 10×10 generado por Backtrack.

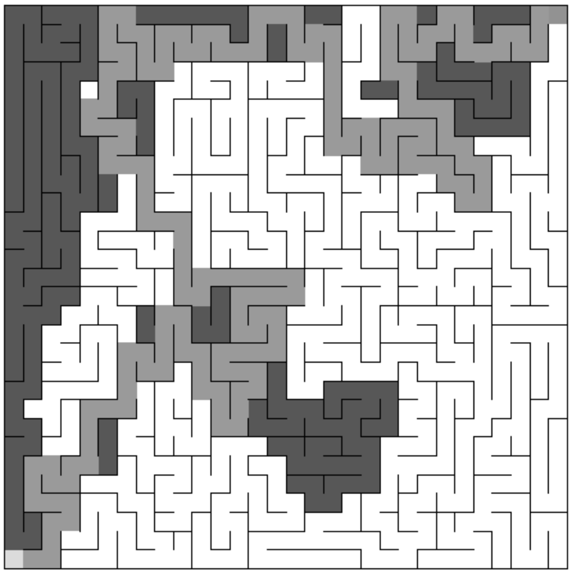

Figura 10: Método de Búsqueda Profunda. Laberinto de tamaño 30×30 generado por Aldous-Broder.

4 Resultados

Para evaluar los algoritmos de solución de laberintos, se generaron diez instancias de cada tamaño con los algoritmos Aldous-Broder y Backtrack y se tomó el tiempo de ejecución de cada corrida, así como el tiempo promedio de cada tamaño de instancia. En la Tabla 1 se muestran los tiempos promedio de ejecución de las 10 instancias de cada familia en milisegundos.

Para evaluar los algoritmos de solución de laberintos, se realizaron 10 ejecuciones de cada instancia y se probó con laberintos generados por los algoritmos de Backtrack y Aldous-Broder, se tomó el tiempo de ejecución a cada corrida y se promediaron los tiempos. En la Tabla 1 se muestran los resultados en milisegundos.

La Figura 11 muestra los resultados de ejecución de los algoritmos de solución para cada familia de laberintos, en ésta se observa que el algoritmo de Dijkstra es un método mas lento por varios ordenes de magnitud en comparación al algoritmo de Búsqueda Profunda Aleatorizada, en ambas familias de laberintos generados.

Tabla 1: Estudio comparado de tiempos del algoritmo Benchmark (Dijkstra) contra el algoritmo propuesto (Búsqueda Profunda Aleatorizada, BPA) en dos familias de instancias.

Tamaño	Al-Br con BPA ms	Al-Br con Dij ms	BT con BPA ms	BT con Dij ms
10	0.15	0.063	0.17	0.04
30	0.95	3.98	1.55	3.62
60	3.69	39.49	5.80	44.73
90	9.35	141.49	12.62	143.48
120	16.72	507.45	14.70	419.19
150	23.58	747.85	28.25	1,308.92
180	26.92	2,664.40	44.58	2,754.01
210	42.31	2,896.74	57.10	4,393.62
240	56.26	7,842.85	74.78	7,804.36
270	64.38	13,995.30	83.26	12,711.58
300	73.97	18,332.43	98.38	17,569.41

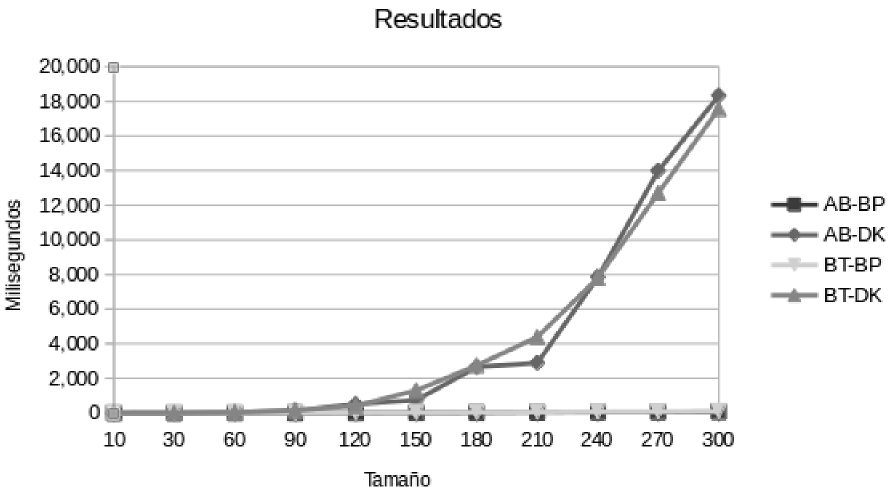

Figura 11: Comparación de tiempos de ejecución de todos los algoritmos.

En la Figura 12 podemos observar que la solución de los laberintos con ambos algoritmos de generación es parecido, en laberintos de menor tamaño, el tiempo de resolución es similar para ambos métodos, pero al aumentar el tamaño de estos, el algoritmo Aldous-Broder aparentemente es un poco más complicado de resolver. En la Figura 13, el algoritmo de búsqueda profunda tiene un comportamiento lineal y se puede notar la diferencia entre los algoritmos de generación. Para este método toma más tiempo resolver por el método de Backtrack. En estas dos gráficas se tienen dos escalas de tiempo totalmente diferentes, debido a que Dijkstra es aproximadamente 160 veces más lento que el algoritmo de Búsqueda Profunda Aleatorizada.

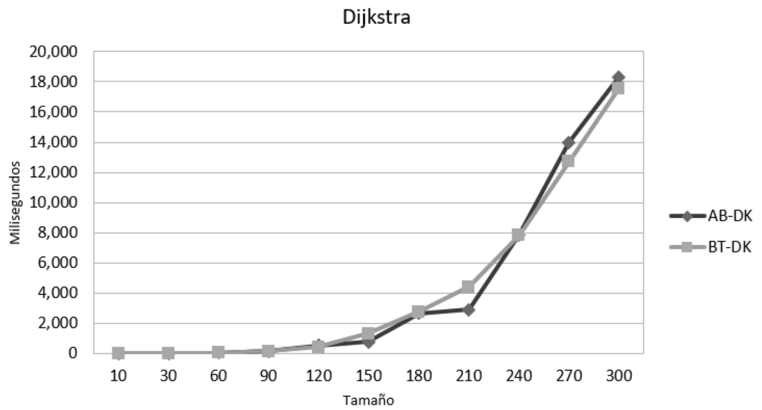

Figura 12: Comparación de tiempos de ejecución solamente con el algoritmo de Dijkstra.

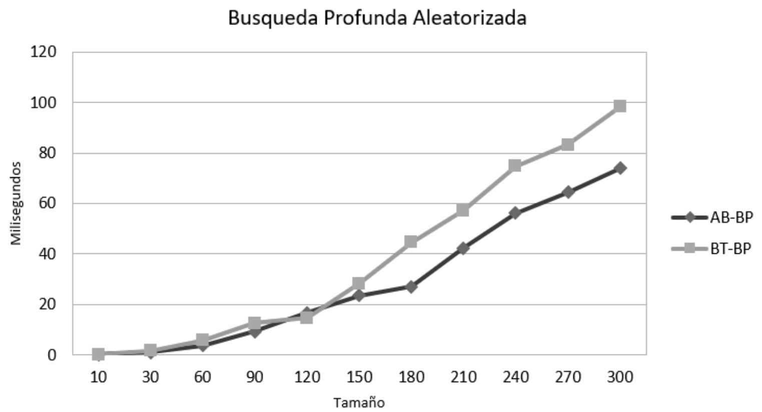

Figura 13: Comparación de tiempos de ejecución solamente con el algoritmo de Búsqueda Profunda.

5 Conclusiones

De acuerdo a los resultados obtenidos, se concluye que el algoritmo de Búsqueda Profunda es un método muy eficiente para la solución de laberintos, observando los tiempos de ejecución, sin importar que método de generación se haya utilizado, el mejor es el propuesto por una diferencia de dos ordenes de magnitud. Esto conlleva a que se puedan encontrar más rápidamente la solución de laberintos Benchmark.

Entre las aplicaciones que se pueden dar a este algoritmo se encuentran en el campo de la robótica, ya que a medida que un robot explora un laberinto, se van eliminando rutas sin salida y busca nuevas opciones [6, 3, 4]. Otra aplicación es comparar este método de solución respecto a cómo resuelven el problema algunos animales (por ejemplo, mamíferos pequeños o incluso humanos sin conocimiento de estos temas), se puede comparar el número de pasos (de casillas exploradas) que cada uno realiza, respecto al método aquí presentado, el cual nunca explora una ruta más de una vez. También, se puede usar en la exploración automatizada de cavernas con rovers o drones que permita explorar exhaustiva y eficientemente el espacio disponible. Finalmente, se puede usar en vehículos autónomos que buscan personas en zonas de desastre o derrumbes.

Agradecimientos

Agradecemos al Posgrado en Ciencias y Tecnologías de la Información de la Universidad Autónoma Metropolitana - Iztapalapa por las facilidades prestadas y especialmente al Consejo Nacional de Ciencia y Tecnología (CONACyT) el cual financió parcialmente esta investigación.

Referencias

[1] D. Ashlock, C. Lee, C. McGuinness, *Search-based procedural generation of maze-like levels*, IEEE Transactions on Computational Intelligence and Al in Games **3** (2011), no. 3, 260-273.

[2] J. Buck, *Mazes for Programmers: Code Your Own Twisty Little Passages*, The Pragmatic Bookshelf, Texas, 2015.

[3] D. C. Dracopoulos, *Robot path planning for maze navigation*, 1998 IEEE International Joint Conference on Neural Networks Proceedings, Vol. 3, IEEE World Congress on Computational Intelligence, 1998.

[4] A. S. Fraenkel, *Economic traversal of labyrinths*, Mathematics Magazine **43** (1970), no. 3, 125-130.

[5] K. Hamada, *A picturesque maze generation algorithm with any given end-points*, Journal of Information Processing **21** (2013), no. 3, 393-397.

[6] G. E. Jan, K-Y. Chang, I. Parberry, *A new maze routing approach for path planning of a mobile robot*, 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Vol. 2, The Institute of Electrical and Electronics Engineers, Inc., Japan, 2003.

[7] M. T. Jones, *Artificial Intelligence: A Systems Approach*, Infinity Science Press, USA, 2008.

[8] W. H. Matthews, *Mazes and Labyrinths: Their History and Development*, Dover Publications, New York, 1970.

[9] V. T. Tomás, M. Pozas, J. Hernández, *Propuesta para la generación de laberintos ampliados en 2D*, *Ciencia Huasteca*, UAEH, México, 2011.