Propuesta de construcción dinámica de espacios de búsqueda

Víctor Tomás Tomás-Mariano, Felipe de Jesús Núñez-Cárdenas, Efraín Andrade-Hernández

Escuela Superior de Huejutla, Universidad Autónoma del Estado de Hidalgo, Huejutla de Reyes, Hidalgo, México

{victor_tomasm, felipe.huejutla, andrade_a_h}@hotmail.com

Resumen. Se realiza el análisis de algoritmos para construir espacios de búsqueda tipo rejilla —laberintos— y su relación con grafos, los algoritmos analizados son: *Kruscal, Aldous-Broder, Prim* y *Recursivo Bactracker*. También se describe una propuesta para generar los gráficos por computadora. En el proceso de construcción se genera un grafo que permite aplicar algoritmos de búsqueda; así poder encontrar el camino entre dos nodos del grafo.

Palabras clave: Grafico por computadora, algoritmos de búsqueda, grafos.

1. Introducción

La construcción de un laberinto puede ser tan complicada como el resolverlo, los laberintos a realizar son de tipo cuadriculado o rejilla, éste debe ser perfecto y completamente conectado, es decir, se debe poder elegir cualquier punto del laberinto como entrada y cualquier otro punto como salida [1].

Un laberinto es conectado si hay un camino de cualquier celda a cualquier otra celda. Se desea que todos los laberintos sean conectados con el propósito de que cualquier celda pueda ser designada como inicio o salida, para poder encontrar una solución. Por consiguiente, se necesita poder garantizar que un laberinto sea conectado y que los algoritmos de construcción cumplan esta condición. En este proyecto se centra en explicar el comportamiento de los algoritmos de construcción de espacios de búsqueda –laberintos- y se recomienda una serie de algoritmos que se pueden aplicar en el medio de búsqueda para hallar una ruta entre dos vértices [2].

2. Construcción de laberintos de conexión simple

La construcción al azar permite generar gran cantidad de laberintos en poco tiempo y de forma sencilla, conteniendo corredores como las ramas de un árbol, persiguiendo la desorientación del explorador, siendo a su vez ésta la forma más fácil de

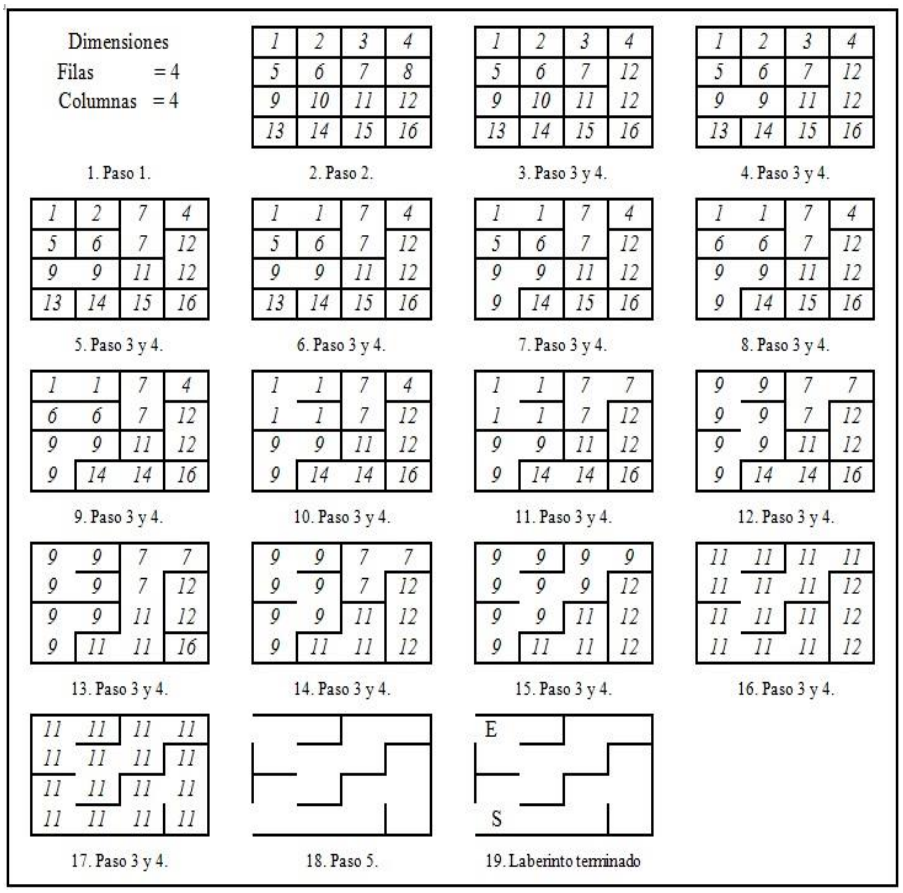


Fig. 1. Construcción de un Laberinto con el Algoritmo de Kruskal.

implementarse en un programa de computadora [1, 2, 3]. Los algoritmos explicados a continuación, generan laberintos completamente conectados.

La construcción al azar se basa en el enfoque que se llama "**Cavar túneles".** Como su nombre lo indica, se refiere a cavar túneles a lo largo y ancho del espacio de construcción hasta cubrirlo en su totalidad, como la explotación de una mina. A este tipo de construcción, pertenecen:

- Algoritmo de construcción Kruscal.
- Algoritmo de construcción Aldous-Broder.
- Algoritmo de construcción Prim's.
- Algoritmo de construcción Recursivo Bactracker.

Algoritmo de Construcción Kruskal. Este algoritmo genera laberintos de conexión simple (LCS). La construcción Kruskal genera laberintos simplemente conectados, este algoritmo cava túneles en varias secciones del laberinto, los pasos se enumeran a continuación, ver Figura 1:

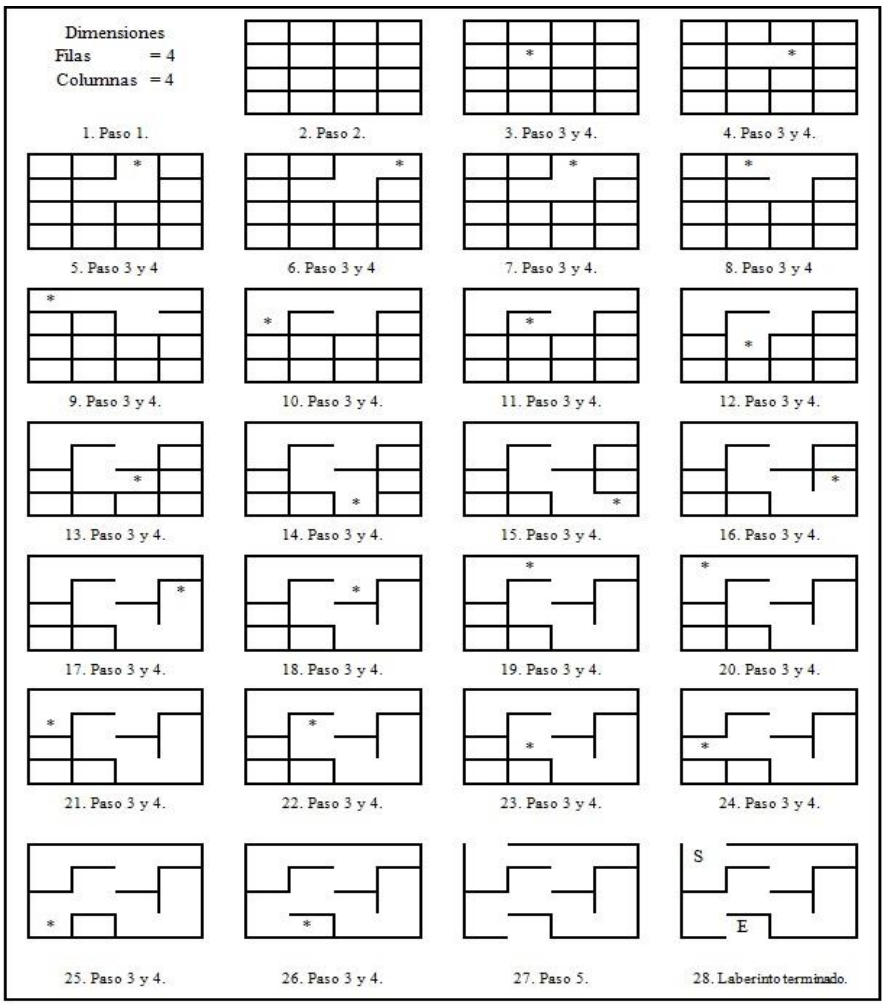


Fig. 2. Construcción de un Laberinto con el Algoritmo de Aldous-Broder.

1. Especificar el tamaño o las dimensiones del laberinto de acuerdo con las habitaciones que se desea que contenga.
2. Dividir el laberinto en habitaciones en base a sus dimensiones y etiquetarlas con una identificación única.
3. Seleccionar una habitación al azar y cavar un túnel hacia una habitación adyacente (en caso de existir más de una, elegir al azar) sólo si tiene diferente etiqueta, y etiquetar la habitación o habitaciones que se unieron con la identificación de la habitación inicial; repitiendo este proceso hasta que todas las habitaciones tengan la misma etiqueta.

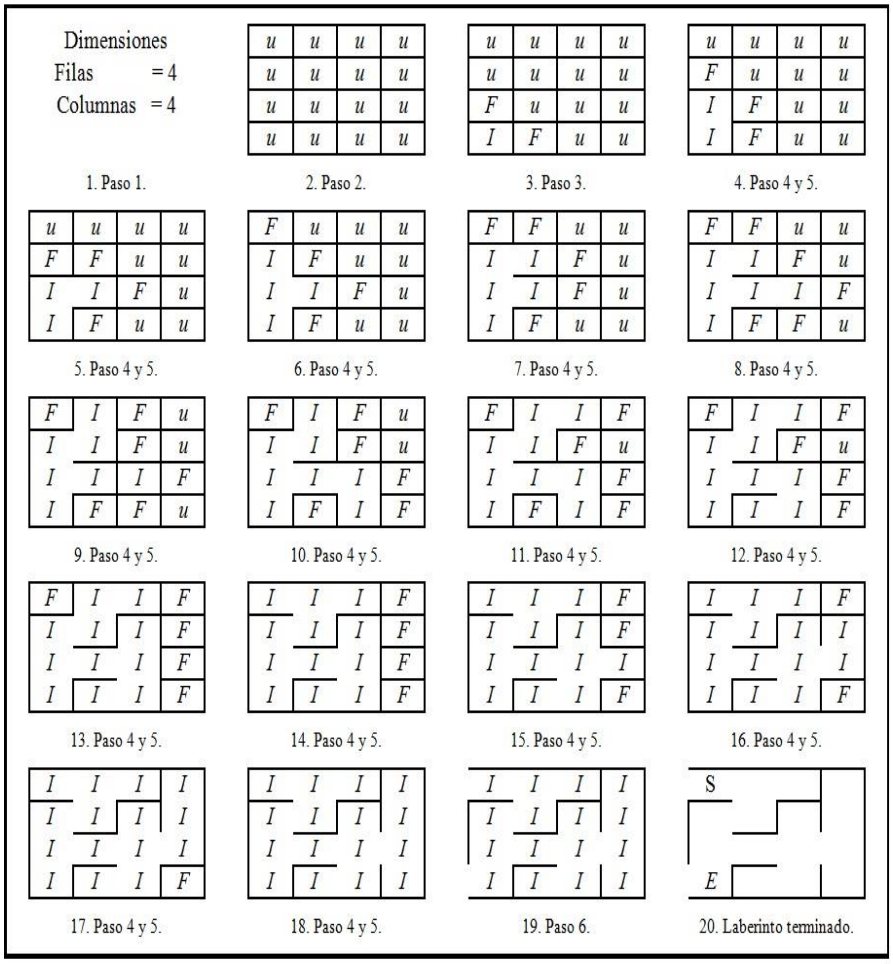


Fig. 3. Construcción de un Laberinto con el Algoritmo de Prim's.

4. Marcar la ubicación de la entrada o del punto inicial de la exploración y la ubicación de la salida.

Algoritmo de Construcción Aldous-Broder. Este algoritmo genera laberintos de conexión simple.

Es una técnica simple, pero puede llevar gran tiempo la construcción de un laberinto, es porque se mueve al azar dentro del laberinto sin discriminar las habitaciones en las que ya se cavaron túneles. Los pasos se enuncian a continuación, ver la Figura 2:

1. Especificar el tamaño o las dimensiones del laberinto de acuerdo con las habitaciones que se desea que contenga.
2. Dividir el laberinto en habitaciones en base a sus dimensiones.

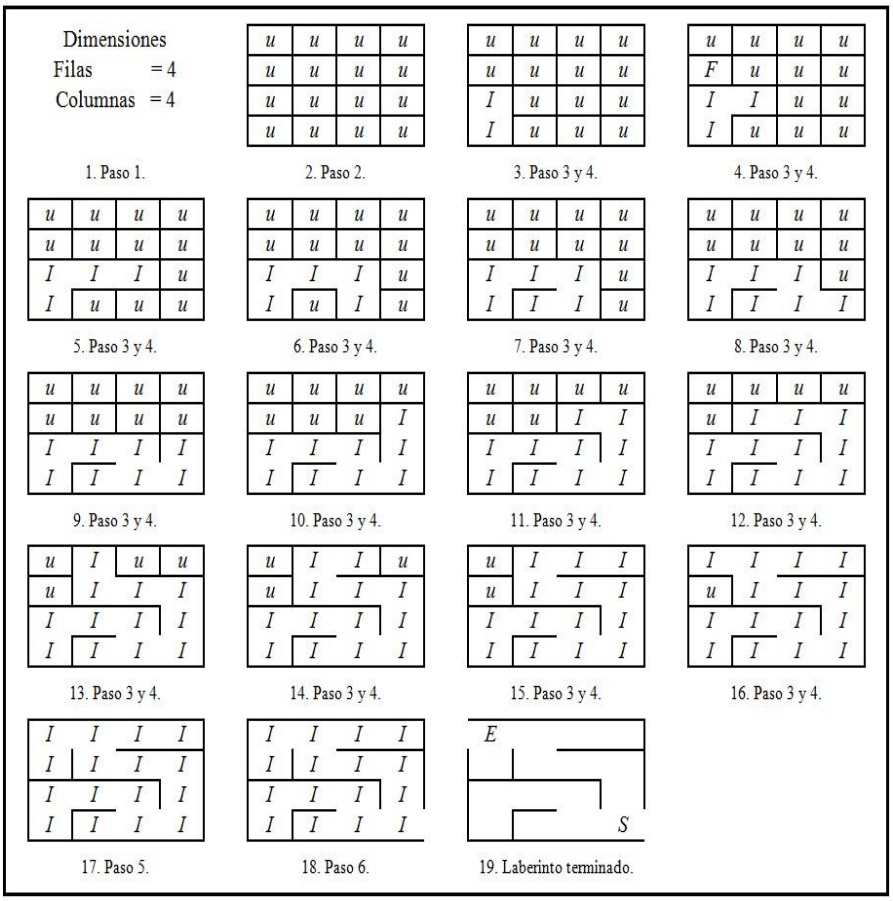


Fig. 4. Construcción de un Laberinto con el Algoritmo Recursivo Backtracker.

3. Elegir al azar una habitación dentro del laberinto.
4. Moverse a una habitación vecina adyacente al azar y cavar un túnel hacia la habitación anterior en el caso de que la nueva habitación no sea parte de otro túnel, continuando con este paso hasta que todas las habitaciones sean parte del laberinto.
5. Marcar la ubicación de la entrada o del punto inicial de la exploración y la ubicación de la salida.

Algoritmo de Construcción Prim's. Este algoritmo genera laberintos simplemente conectados, el procedimiento es el siguiente (ver la Figura 3):

1. Considera dos tipos de habitaciones:
2. Habitaciones etiquetadas con I son aquellas que forman parte del laberinto y han sido cavadas.

3. Habitaciones etiquetadas con F que no han sido cavadas, pero que son adyacentes a una habitación I.
4. Especificar el tamaño o las dimensiones del laberinto de acuerdo con las habitaciones que se desea que contenga.
5. Dividir el laberinto en habitaciones en base a sus dimensiones y etiquetarlas con una identificación única.
6. Seleccionar una habitación inicial, etiquetarla como I y etiquete sus habitaciones adyacentes con F.
7. Seleccione una habitación F aleatoriamente. Cavar un muro de la habitación I a la habitación F; cámbielo a I y cambie sus habitaciones adyacentes que no son I a F.
8. Repita este proceso mientras haya habitación F.
9. Marcar la ubicación de la entrada o del punto inicial de la exploración y la ubicación de la salida.

Algoritmo de Construcción Recursivo Backtracker. Este algoritmo genera laberintos de conexión simple. El procedimiento es el siguiente (ver la Figura 4):

1. Especificar el tamaño o las dimensiones del laberinto de acuerdo con las habitaciones que se desea que contenga.
2. Dividir el laberinto en habitaciones en base a sus dimensiones y etiquetarlas con una identificación única
3. Elegir al azar una habitación dentro del laberinto y empiece a cavar. Siempre cave hacia habitaciones adyacentes que no han sido cavadas, si las hay, si no, regrese a buscar uno, por el túnel ya cavado, hasta encontrar una.
4. Repita este procedimiento, mientras no se regrese al punto inicial.
5. Marcar la ubicación de la entrada o del punto inicial de la exploración y la ubicación de la salida.

Los algoritmos anteriores generan laberintos de conexión simple (LCS), se pueden construir laberintos de conexión múltiple (LCM) –laberintos con circuitos internos, "cavando" paredes en el interior del laberinto en forma aleatoria o en donde se formen callejones sin salida.

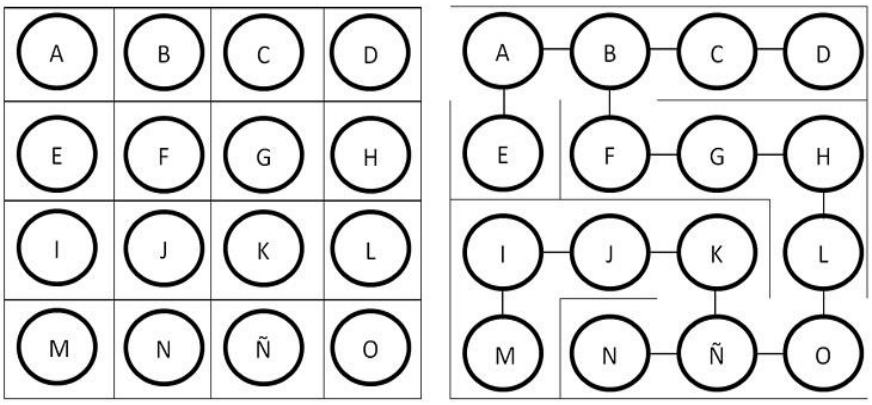
Los LCM, a diferencia de los LCS, que solo tienen una ruta como solución, estos laberintos, tienen varias "rutas solución", además tienen islas o circuitos internos, no tienen puntos muertos. Esto, dificulta encontrar la ruta entre dos puntos dentro del laberinto, pues se puede dar giros en un mismo segmento en forma indefinida. Por lo que de alguna manera hay que llevar el control de los lugares ya visitados. La propuesta planteada permite manipular este tipo de laberintos, al crear un grafo con circuitos internos.

3. Propuesta de construcción y graficación

En la construcción de un laberinto se manejan 2 matrices:

3.1. Matriz de vértices y aristas

Inicialmente esta matriz, se conforma por el número de vértices con base en las dimensiones que se desee tenga el laberinto. El grafo es completamente desconectado, tal como se ve en la figura 5a. Conforme se aplican los pasos de construcción, por ejemplo, el mostrado en la figura 4, se crea la arista entre los vértices respectivos y al final de la construcción, se obtiene un grafo completamente conectado, ver figura 5b.



a) b)

Fig. 5. Construcción laberinto a) grafo desconectado, b) grafo conectado asociado a un laberinto.

En la figura 5, se utilizan símbolos alfabéticos para los vértices, sin embargo, se puede utilizar símbolos numéricos para tener un mayor margen de amplitud en el número de vértices manejados.

3.2. Matriz de coordenadas

En esta matriz se representan las coordenadas del espacio de graficación. Se hace un mapeo en pixeles de las dimensiones de la matriz del laberinto, en la figura 6 se muestran las coordenadas para un laberinto de 4 filas, 4 columnas, y de las posiciones [i, j] respectivamente. Los movimientos permitidos en la construcción son: arriba, abajo, izquierda, derecha. Se propone el cálculo de coordenadas de graficación con base en la posición [i, j] de partida en el movimiento respectivo. Inicialmente, se traza el cuerpo del laberinto, extremadamente desconectado y se van cavando túneles a lo largo del proceso de construcción**.**

Movimientos permitidos, cavar túneles:

Arriba: [i, j] a [i-1, j]

Posición [i, j] origen: [2, 2]; posición [i, j] destino [1, 2]

Formula: [x0+ j*dx, y0+i*dy] , [x0+(j+1)*dx, y0+i*dy]

Posición origen: [i=2, j=2]:

Se obtiene: [0+2*10, 0+2*10], [0+3*10, 0+2*10] = [20, 20], [30, 20]

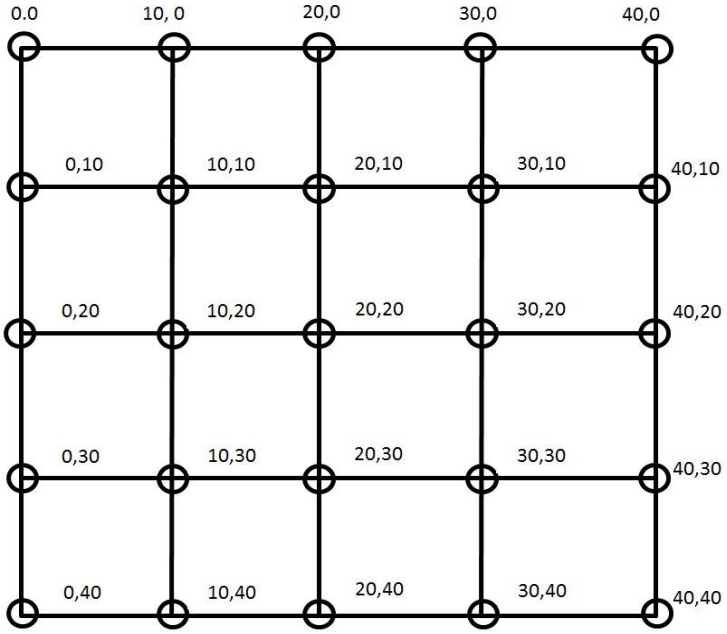


Fig. 6. Coordenadas del espacio de graficación.

Cavar_túnel (20,20, 30, 20)
Crear_vertice(nodoOri, nodoDes)
Comentario: incrementa en dx, constante en dy.

Abajo: [i, j] a [i+1, j]

Posición [i, j] origen:[2, 2]; posición[i, j] destino [3, 2]
Formula: [x0+j*dx, y0+(i+1)*dy], [x0+(j+1)*dx, y0+(i+1)*dy]
Posición origen: [i=2, j=2]:
Se obtiene: [0+2*10, 0+3*10] - [0+3*10, 0+3*10] = [20,30]-[30,30]
Cavar_túnel (20,30, 30, 30)
Crear_vertice(nodoOri, nodoDes)
Comentario: incrementa en dx, constante en dy.

Izquierda: [i, j] a [i, j-1]

Posición [i, j] origen: [1, 1]; posición [i, j] destino [1,0]
Formula: [x0+j*dx, y0+i*dy] , [x0+j*dx, y0+(i+1)*dy]
Posición origen: [i=2, j=2]:
Se obtiene; [0+2*10, 0+2*10] – [0+2*10, 0+3*10] = [20,20]-[20,30]
Cavar_túnel (20,20, 20, 30)
Crear_vertice(nodoOri, nodoDes)
Comentario: constante en dx, incrementa en dy.

Derecha: [i, j] a [i, j+1]

Posición [i, j] origen: [1, 1]; posición [i, j] destino [1,2]
Formula: [x0+(j+1)*dx, y0+i*dy]; [x0+(j+1)*dx, y0+(i+1)*dy]
Posición origen: [i=2, j=2]:

Se obtiene; [0+3*10, 0+2*10] – [0+3*10, 0+3*10] = [30,20]-[30,30]
Cavar_túnel (30,20, 30, 30)
Crear_vertice(nodoOri, nodoDes)
Comentario: constante en dx, incrementa en dy.

Los movimientos y valores de coordenadas obtenidas se pueden verificar en la figura 6, para el este ejemplo, se tiene un área limitada para el caso de 4 filas y 4 columnas.

Se inicia la graficación en las coordenadas iniciales x0=0, y0=0, con incrementos en dx y dy que representan la distancia entre celdas horizontal y vertical respectivamente, en este caso se maneja un valor igual a 10. Los valores relativos a las posiciones [i, j] dentro del arreglo dependen de la celda que se esté procesando durante el proceso de construcción.

Para un espacio de graficación de n * m, se obtiene un conjunto de coordenadas tal como se muestra en la figura 7:

Sean ***n*** filas en dx: 1, 2, 3…n.
Sean ***m*** columnas en dy: 1, 2, 3…m

[x0],[y0]	[x0+1dx], [y0]	[x0+2dx], [y0]	[x0+3dx], [y0]	…	[x0+n*dx], [y0]
[x0],[y0+1dy]	[x0+1dx],[y0+1dy]	[x0+2dx],[y0+1dy]	[x0+3dx],[y0+1dy]		
[x0],[y0+2dy]	[x0+1dx],[y0+2dy]	[x0+2dx],[y0+2dy]	[x0+3dx],[y0+2dy]		
[x0],[y0+3dy]	[x0+1dx],[y0+3dy]	[x0+2dx],[y0+3dy]	[x0+3dx],[y0+3dy]	…	
…..					
[x0],[y0+mdy]					[x0+n*dx],[y0+m*dy]

Fig. 7. Matiz de coordenadas para un laberinto de *m filas* por *n columnas*.

4. Resultados

En la representación de la figura 6, el grosor de las líneas del laberinto es de un píxel, sin embargo, la propuesta realizada permite manipular también el "grosor" de las líneas, algunos resultados obtenidos se visualizan en la figura 8, en esta figura se muestra un laberinto con dimensiones de 50 filas por 50 columnas y grosor de 1 pixel, figura 8 a); laberinto con dimensiones de 50 filas por 50 columnas y grosor de 4 pixeles, figura 8 b); laberinto con dimensiones de 50 filas por 50 columnas y grosor de 10 pixeles, figura 8 c).

De los resultados obtenidos en las graficas con grosor de un píxel, se observa que no son perceptibles a simple vista, por lo que resultan difíciles de explorar en la interfaz de una computadora, sin embargo, se obtienen mejores visualizaciones graficas al incrementar el grosor de las líneas a partir de 4 píxeles o mayor, otro factor que influye es el número de filas y columnas del espacio de búsqueda, es decir, en laberintos pequeños resulta fácil explorarlos, sin embargo, el grado de dificultad incrementa conforme se aumenta el numero de celdas que se desee tenga el espacio de búsqueda.

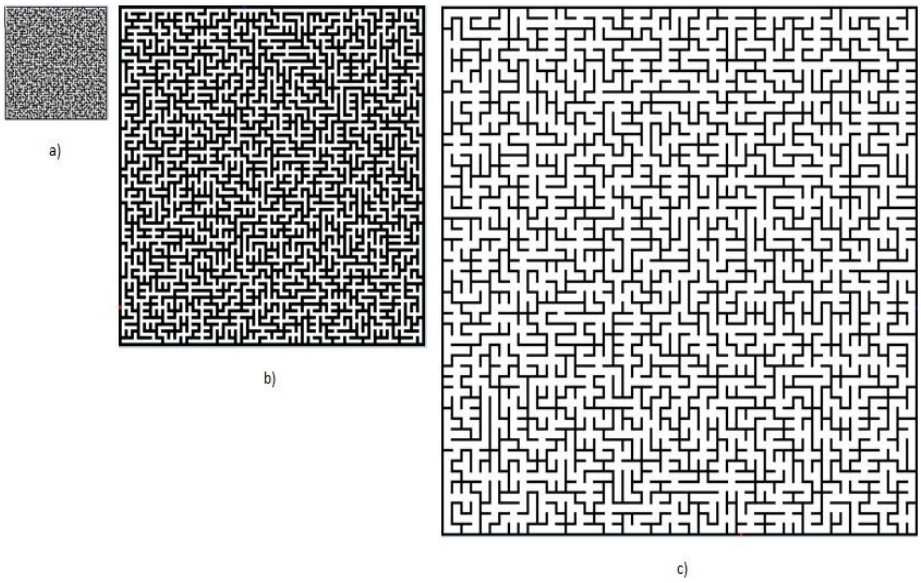


Fig. 8. Grosor líneas laberintos. a) Dimensiones: 50f, 50c, 1p; b) Dimensiones: 50f, 50c, 4p; c) Dimensiones: 50f, 50c, 10p.

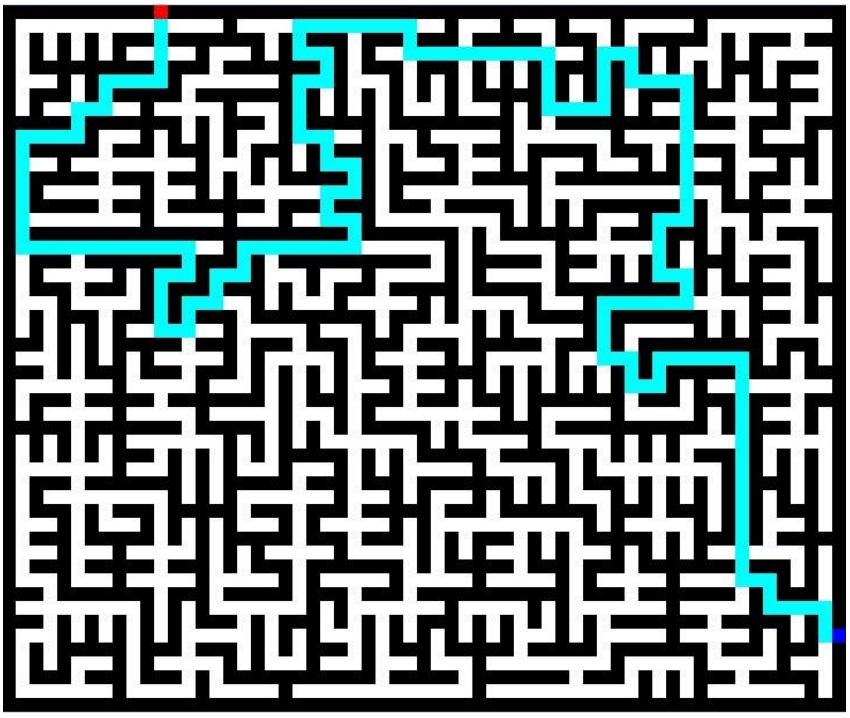


Fig. 9. Laberinto con solución. Dimensiones: 25f, 30c, 15p.

La entrada del laberinto se simboliza con un cuadro en color rojo y la salida en color más claro.

5. Búsqueda de solución en espacio de estados

La propuesta de generación de un grafo en el proceso de construcción de un laberinto, permite modelar el problema para aplicar algoritmos de búsqueda en grafos sin pesos, con la finalidad de encontrar una ruta entre dos vértices; entre las técnicas de la inteligencia artificial (IA) [4, 5] que se pueden aplicar en este tipo de problemas están: *Primera Búsqueda en Profundidad, Primera Búsqueda en Amplitud, Dijkstra y el A estrella*, los dos primeros hacen una búsqueda a ciegas y se aplica en casos en donde el punto de salida es desconocido y en las aristas no hay pesos[6]. El algoritmo A estrella es otro de los más aplicados en técnicas de búsqueda, y se aplica cuando el punto de salida es conocido, su funcionamiento tiende a realizar movimientos en diagonal, horizontal y vertical, permite asignar un costo en los movimientos. Este algoritmo utiliza una búsqueda heurística para encontrar la ruta óptima entre dos puntos [5, 7]. Maneja tres funciones: F, G y H.

- La función G es el costo del mejor camino desde la celda inicial a la celda n obtenido hasta el momento durante la búsqueda.
- La función H es el costo del camino más corto desde la celda n a la celda objetivo más cercano n.
- F = G + H, Es decir, F es el costo del camino más corto desde la celda inicial a la celda objetivo.

El algoritmo de Dijkstra [4] permite encontrar la ruta más corta entre dos vértices de un grafo, en las aristas se pueden colocar pesos positivos que significan costos que implica ir de una vértice a otro vértice.

Una posible ruta entre la entrada y salida que se obtiene con el grafo generado se muestra en la figura 9.

6. Conclusiones

Se han analizado los diferentes algoritmos para la construcción de laberintos de conexión simple. El proceso de construir un laberinto involucra que el resultado final de la construcción debe ser un laberinto completamente conectado, los algoritmos analizados construyen laberintos completamente conectados.

La técnica propuesta genera un grafo completamente conectado en un espacio de estados, y a partir del grafo, hacer una búsqueda de una solución que conecte a dos vértices, se pueden aplicar algoritmos para recorrido de grafos. Con respecto a la graficación del espacio de búsqueda se generan gráficos dinámicos, se contempla las posiciones [i, j] para el cálculo de coordenadas de graficación y los cambios en función de los movimientos permitidos en el algoritmo de construcción.

Referencias

1. Suits, David B.: Playing With Mazes. Department of Philosophy, Rochester Institute of Technology (1995)
2. Bernabe, S. E.: Construcción y Recorrido de Laberintos. Tesis de Licenciatura, Pachuca de Soto, Hidalgo, México (Julio 2004)
3. Depue, K.: Maze Complexity. Hastings College, Hastings, Nebraska, USA, B.A., Mathematics and Computer Science (May 2005)
4. Johnsonbaugh, R.: Matemáticas discretas. Pearson Educación (2005)
5. Marín, R., Palma, J.T.: Inteligencia Artificial: métodos, técnicas y aplicaciones (2008)
6. Molina, V. J., Torres, P. C., Restrepo, P. C.: Técnicas de inteligencia artificial para la solución de laberintos de estructura desconocida, Scientia et Technica UTP (2008)
7. Lester, P.: A* pathfinding for beginners. Almanac of Policy Issues: see http://www.policyalmanac.org/games/aStarTutorial.htm. (2005)