Sistemas de Inteligencia Artificial

Trabajo Práctico Especial I

Grupo 4

Docentes

Parpaglione, María Cristina

Luciani, Ignacio José

Alumnos

Badi, Leonel

Farré, Lucas Andrés

Gómez, Jorge Exequiel

1. Descripción

1.1. Objetivo

Implementar un Sistema de Producción para resolver el juego *Fill Zone.* El mismo consiste en un tablero donde cada casillero está *pintado* de un color y se debe lograr *propagar* los colores hasta que el tablero se encuentre *pintado* de un mismo color en su totalidad. Se deberá terminar el nivel antes de superar el número de jugadas permitidas.

1.2. Implementación

1.2.1. Estado

El estado representado consiste en una *foto* del tablero para un turno determinado. El mismo fue implementado mediante una matriz de *bytes* (*byte[][]*) para ser eficientes en memoria.

1.2.2. Reglas

Las reglas en juego se basan en una misma regla general aplicada para cada uno de los distintos colores. La regla general consiste en *pintar* de un color el casillero del extremo superior izquierdo del tablero y *propagar* el mismo a los casilleros adyacentes de manera recursiva hasta que quede la zona de mayor tamaño posible *pintada* con ese color.

1.2.3. Algoritmos

Para el desarrollo de la búsqueda de la solución, se implementaron los siguientes algoritmos:

- Algoritmos de búsqueda desinformados
  - *DFS (Depth-First Search):* Se inicializa la lista *open* con una implementación propia que agrega los valores al comienzo. De esta manera nos aseguramos que se tomen siempre los nodos de mayor profundidad.
  - *BFS (Breadth-First Search):* Se inicializa con una *LinkedList* común, ya que ésta los agrega al final.
  - *Profundización Iterativa o IDDFS (Iterative Deepening Depth-First Search):* Se implementó utilizando una implementación propia de lista (La misma que en *DFS*), de esta manera nos aseguramos que el orden de búsqueda sea como el *DFS*. Para ir recorriendo por niveles se tiene una variable que va contando hasta qué profundidad tiene que alcanzar en cada iteración. Si el costo de los valores supera esta variable, simplemente no se agregan. Cuando la lista *open* se vacía, se vuelve a agregar el nodo inicial y se aumenta el valor de la variable que nos acota la profundidad. Cabe aclarar que reiniciamos la lista *closed* para poder agregar nodos que en otra iteración ya hemos agregado.

- Algoritmos de búsqueda informados
  - *A* (A-Star):* Se implementó utilizando una cola de prioridades y un comparador. El comparador tiene en cuenta la suma del valor heurístico y el costo para ordenar los elementos.

- *Greedy:* Al principio esta estrategia se implementó utilizando un comparador que compara primero el nivel y luego el valor heurístico. Esta era una manera elegante de ordenar los nodos, primero por costo, y luego por valor heurístico. El gran problema de esto era la complejidad que agrega, ya que aunque sean solo dos *ifs*, se ejecutan una gran cantidad de veces. Para solucionar y optimizar esto, se hizo de una manera totalmente diferente utilizando una lista que agrega al principio y una cola de prioridades. La cola de prioridades tiene un comparador que los mantiene ordenados por valor heurístico, de mayor a menor. A esta cola de prioridades se insertan los últimos nodos que se generaron en una explosión. Luego de aplicar todas las reglas a un nodo, se vuelca esta cola en la lista original *open*.

1.2.4. Heurísticas

Para la ejecución de los algoritmos de búsqueda informada se implementaron las siguientes heurísticas:

- *a. DistanceHeuristicAdmissible*

  Es una heurística admisible que aproxima la cantidad de pasos mínimos necesarios para lograr *pintar* los casilleros de las otras tres esquinas del tablero. Para esto, se genera una cola de prioridades con las cantidades de casilleros adyacentes de todos los colores. De esta manera, se puede acotar la cantidad de pasos mínimos a realizar para poder alcanzar cada uno de los casilleros de las esquinas. Cabe aclarar que la cola se calcula una única vez al comienzo, dado que es un proceso costoso.

  Esta heurística tiene la desventaja de que en momentos tardíos del juego, cuando ya se llegó o se esta por llegar a las esquinas, ofrece valores heurísticos muy similares.

- *b. DistanceHeuristicAdmissibleColor*

  Esta heurística es una variante de la heurística anterior. La diferencia es que además de utilizar la lista con las cantidades de casilleros adyacentes, utiliza el valor de la cantidad de colores restantes en el tablero luego de realizar el movimiento. Es decir que si el valor heurístico que va a devolver utilizando la heurística anterior es menor que la cantidad de colores, se cambia la manera de acotar la distancia y se retorna la cantidad de colores que tiene el tablero. De esta manera se consigue una cota más aproximada.

  Esto tiene ventajas en los instantes finales del juego, ya que si un estado tiene menos colores que otro, es un estado más próximo a la solución.

- *c. DistanceHeuristic*

  Es una heurística no admisible que busca la aproximación a las esquinas utilizando la distancia de Manhattan. Esta heurística no es admisible, ya que no estima la cantidad de pasos para alcanzar el estado final, pues puede suceder el caso de *pintar* más casilleros con un solo movimiento, debido a la aleatoriedad del tablero.

- *d. CountBlockHeuristic*

  Esta heurística otorga un valor heurístico menor al estado que tiene más celdas adyacentes, del mismo color que el casillero de la esquina superior izquierda. Esta heurística no es admisible, pues el valor heurístico no estima la cantidad de pasos restantes y es simplemente la cantidad total de casilleros del tablero restada a la cantidad de casilleros *pintados* resultantes de aplicar la regla.

1.2.5. Costos

Para la ejecución de algoritmos de búsqueda se utilizó una función de costos constante de valor 1, donde cada paso incrementa en uno el valor del costo, es decir, que el costo es igual a la profundidad en el árbol. En este caso, el costo máximo permitido para la solución del problema, es la cantidad de pasos máximos permitidos.

2. Resultados

2.1. Detalles numéricos

Los detalles numéricos de los resultados obtenidos se encuentran en la tabla del punto A.1. del Anexo.

2.2. Análisis de los resultados

Teniendo en cuenta los valores obtenidos, podemos destacar las siguientes observaciones para cada uno de los algoritmos:

- *BFS*

Era de esperar que el tiempo de ejecución de este clásico algoritmo sea muy alto, pues el mismo recorre el árbol a lo ancho y consigue así la solución mínima, pero en contrapartida, debe expandir todos los nodos en cada nivel, y así, el tiempo de ejecución se vuelve altísimo.

- *DFS*

Con diferencia al algoritmo anterior, este otro algoritmo clásico, al recorrer el árbol en profundidad, consigue una solución muy rápidamente, pero ésta no es la mínima. Cabe destacar que el tiempo de ejecución del mismo, depende muchísimo de la cantidad de movimientos máxima admitida por el nivel del juego, esto se pudo observar en el tablero de 14x14, en el cuál el algoritmo no logró encontrar ninguna solución de a lo sumo 30 pasos en un tiempo menor a 5 minutos. En los tableros de 6x6 y de 10x10, el algoritmo finalizó muy rápidamente, pues los tableros eran más reducidos pero también tenían el mismo máximo de 30 movimientos posibles.

- *IDDFS*

Este algoritmo, al igual que el *BFS*, encuentra la solución mínima sin hacer uso de heurísticas. El mismo resultó ser esperadamente más rápido que el *BFS*, pero inesperadamente más rápido también que el A* haciendo uso de la heurística *a*. Esto nos dio la pauta de que la heurística *a* resultó ser verdaderamente mala.

- *Greedy*

Este algoritmo resultó muy bueno para cualquiera de las cuatro heurísticas propuestas, pues pudimos notar que, por más que las soluciones obtenidas no sean las mínimas, el mismo logró conseguir mejores soluciones que el algoritmo *DFS* en tiempos semejantes, e incluso menores en ciertos casos.

- *A**

La principal ventaja de este algoritmo es conseguir la solución mínima, pero para eso la heurística debe ser admisible. Pudimos verificar, en cierta medida, que las dos heurísticas admisibles que propusimos, realmente lo son, pues para el tablero de 6x6, ambas alcanzaron la misma solución que el algoritmo *BFS*, en un tiempo menor con la heurística *a*, y en uno considerablemente menor con la heurística *b*. Otro dato no menor, es que por más que una heurística no sea admisible, este algoritmo puede encontrar una solución no mínima en un tiempo excelente. Esto se pudo observar utilizando la heurística *d*, con la cual se logró resolver el tablero de 14x14 en tan solo un tercio de segundo, mientras que en ninguno de los otros casos se logró resolverlo en menos de 5 minutos.

3. Conclusiones

- Si bien las heurísticas no admisibles no nos aseguran conseguir la mejor solución respecto a costo, es posible encontrar una solución muy cercana a la misma en un tiempo considerablemente menor. Esto nos deja la moraleja de pensar verdaderamente cuál es nuestro objetivo final a la hora de establecer las heurísticas: ¿Queremos una solución perfecta?, ¿o preferimos una solución relativamente buena en un tiempo más acotado? Esto claramente dependerá del problema. En este juego, por ejemplo, como el jugador tiene un máximo de movimientos permitidos, no tiene la necesidad de ganar utilizando la mínima solución, le basta con encontrar una solución menor a la cota dada.

- La heurística admisible *a* resultó ser realmente pésima. En tableros de pequeñas dimensiones, los algoritmos de búsqueda desinformada *BFS* y *IDDFS* consiguieron la solución mínima en tiempos menores que el A* utilizando esta heurística. El problema es que esta heurística subestima muchísimo, y la cola de prioridades termina albergando valores muy similares, lo que hace que se comporte como un algoritmo *BFS*, pero termina siendo aún más lento que este último, debido al tiempo que lleva calcular los valores heurísticos.

- La heurística admisible *b*, en cambio, resultó ser excelente, pues por ejemplo para el tablero de 8x8, la misma consiguió la solución mínima utilizando el algoritmo A* en tan solo 7 segundos, mientras que al clásico algoritmo *BFS* le llevó más de 2 horas y al *IDDFS* alrededor de 45 minutos.

- Una heurística mala puede llegar a ser muy buena luego de hacerle alguna modificación o cambiar la estrategia de calcular la heurística a partir de cierto paso. Esto se observó luego de probar las dos heurísticas admisibles. La heurística *b* resultó ser muchísimo mejor que la heurística *a*, y sin embargo, ambas actúan de manera idéntica al principio y sólo se diferencian en los últimos pasos.

- Siempre hay que tener en cuenta la complejidad de los métodos que se van a invocar en todos los pasos y no subestimar la complejidad de calcular un valor heurístico, dado que si esto es muy costoso puede llegar a tardar más inclusive si recorre menor cantidad de nodos.

A. Anexo

A.1. Tabla de Resultados

Tablero	Algoritmo	Heurística	Profundidad	Nodos Expandidos	Cantidad de Estados	Nodos Frontera	Tiempo
6x6	BFS	-	9	17439	32510	15071	71.22 s
	DFS	-	28	29	140	111	83 ms
	IDDFS	-	9	1862447	9312236	17	13.19 s
	Greedy	a	25	26	125	99	18 ms
		b	18	18	88	70	15 ms
		c	27	28	133	105	15 ms
		d	25	25	123	98	15 ms
	A*	a	9	14829	30571	15742	45.52 s
		b	9	268	867	599	95 ms
		c	10	531	1745	1214	179 ms
		d	12	24	113	89	12 ms
8x8	BFS	-	12	189214	360825	171611	2.42 h
	DFS	-	29	37	151	114	19 ms
	IDDFS	-	12	203328794	1016643970	32	45 min
	Greedy	a	30	45	161	116	31 ms
		b	22	22	110	88	33 ms
		c	25	62	161	99	32 ms
		d	27	32	140	108	28 ms
	A*	a	12	167195	358540	191345	10.31 h
		b	12	3674	10248	6574	7.38 s
		c	13	2026	5974	3948	2.39 s
		d	17	39	170	131	31 ms

Tablero	Algoritmo	Heurística	Profundidad	Nodos Expandidos	Cantidad de Estados	Nodos Frontera	Tiempo
10x10	BFS	-	-	-	-	-	> 5 min
	DFS	-	30	201	314	114	37 ms
	IDDFS	-	-	-	-	-	> 5 min
	Greedy	a	30	1207	1317	110	341 ms
		b	30	708	821	113	159 ms
		c	30	221	335	94	63 ms
		d	30	3382	3489	107	1.12 s
	A*	a	-	-	-	-	> 5 min
		b	-	-	-	-	> 5 min
		c	-	-	-	-	> 5 min
		d	23	44	196	152	24 ms
14x14	BFS	-	-	-	-	-	> 5 min
	DFS	-	-	-	-	-	> 5 min
	IDDFS	-	-	-	-	-	> 5 min
	Greedy	a	-	-	-	-	> 5 min
		b	-	-	-	-	> 5 min
		c	-	-	-	-	> 5 min
		d	-	-	-	-	> 5 min
	A*	a	-	-	-	-	> 5 min
		b	-	-	-	-	> 5 min
		c	-	-	-	-	> 5 min
		d	30	1683	1844	161	372 ms

Nota: Las pruebas fueron realizadas con una computadora con procesador Intel Core i5 de 1.3 GHz y 4GB de memoria RAM.