Sistemas de Inteligencia Artificial

TRABAJO PRÁCTICO ESPECIAL - INFORME

EDIFICIOS (SKYSCRAPER)

GRUPO 1 - 1C 2017

DOCENTES

- Parpaglione, Cristina
- Pierri, Alan

INTEGRANTES

- Comercio Vázquez, Matías Nicolás - 55309
- Ibars Ingman, Gonzalo - 54126
- Mercado, Matías - 55019
- Moreno, Juan - 54182

Índice

Introducción

El objetivo del presente informe es describir el trabajo realizado para el juego Edificios, exponer los puntos claves de la implementación, mostrar el análisis de los resultados y las conclusiones obtenidas.

Implementación

El lenguaje utilizado para el desarrollo fue Java 8, incluyendo la dependencia JUnit 4 para realizar testing. Para todas las clases mencionadas en esta sección, se implementó una interfaz a través de la cual una instancia de un estado puede acceder, sin conocer los detalles de implementación.

Visibilidad

Se utilizaron 4 vectores que representan los bordes superior, inferior, izquierdo y derecho. Al inicializar el programa se populan esos vectores nunca modificados en el flujo del programa, con lo cual se utiliza una única instancia para todo estado.

Tablero

Se utilizó una matriz built-in de enteros de N x N (siendo N el lado del tablero) ya que resulta más eficiente que guardar una instancia de una clase. Se reconoce que si bien se puede tener como estado inicial una matriz dispersa (sus elementos en gran parte son 0) sigue siendo más eficiente utilizar una matriz con sus elementos en 0 que utilizar, por ejemplo, un mapa que guarde entradas *((x, y) -> height)* tal que el *height* sea mayor a 0. Almacena además la Visibilidad, una lista con los casilleros fijados en el estado inicial (sus valores son diferentes a 0), y una variable con la cantidad de casilleros vacíos para evitar tener que recorrer la matriz cada vez que se genera un nuevo tablero.

Debido a que el backtracking (dependiendo la estrategia de búsqueda) se realiza en el motor de inferencia (y no en esta estructura en particular, deshaciendo los cambios), resulta más conveniente crear una nueva matriz para cada nuevo estado.

El tablero inicial se carga desde un archivo. Si la regla que se va a aplicar es Swap entonces se llena la matriz sin pisar los valores pre insertados y se verifica que se cumplan las restricciones de duplicidad por fila y columna (para que el tablero inicial tenga la menor cantidad de conflictos posible, con un preprocesamiento que aplica cierta heurística para ello).

Estado

Almacena el Tablero, la primer posición vacía en el tablero y un vector con la última posición vacía para todas las filas y columnas (utilizado por la regla Put).

Problema

En el método *isGoal* se revisa que la matriz esté completa, que para todo casillero no existan conflictos de duplicidad (en fila y columna) y que se cumplan las condiciones de visibilidad.

El método *getHValue* se detalla en la subsección Heurísticas. Si se detecta que todas las heurísticas que se están utilizando son admisibles (consultando a cada una a través de una interfaz) se retorna el valor máximo entre ellas, sino se retorna el mínimo.

Reglas

Put

Esta serie de reglas consisten en colocar en la primera posición vacía (recorriendo el tablero por filas y de izquierda a derecha) un valor entre 1 y N inclusive, siendo N el tamaño de un lado del tablero.

Cada regla está formada por una posición del tablero (fila y columna) y un valor a insertar. La aplicación de la misma depende de si la primera posición vacía en el tablero (esto es una heurística aplicada directamente sobre la regla, no como una función h) es igual a aquella que la compone y si al insertar el valor se cumplen las restricciones del siguiente párrafo.

No debe existir un duplicado en la fila o columna a la que pertenece. Con respecto a la visibilidad, se debe seguir cumpliendo que la cantidad de edificios visibles desde los bordes TOP y LEFT sea menor o igual a la restricción impuesta por el mismo. Si la inserción se realiza en la última posición de una fila se debe cumplir con que la cantidad de edificios visibles desde el borde LEFT y RIGHT sea igual a la restricción impuesta por cada uno. El caso es análogo para las inserciones en las últimas posiciones de cada columna, en donde se toma en cuenta las restricciones del borde TOP y del BOTTOM. Esta heurística fuerza a que el factor de ramificación disminuya considerablemente, comparado con el caso de ir insertando aleatoriamente.

El costo de aplicar la regla es 1.

Swap

Esta serie de reglas consisten en intercambiar los valores de 2 posiciones del tablero.

La regla es aplicable sólo si el tablero tiene todas sus posiciones ocupadas. Además, se valida que no se puedan intercambiar valores fijados en el estado inicial para garantizar la unicidad de la solución.

El costo de aplicar la regla es 12 por motivos detallados en la sección Heurísticas.

Métodos de búsqueda

Los métodos de búsqueda implementados son: BFS, DFS, IDDFS (Iterative deepening depth-first search), Greedy y A*. Los mismos se encuentran en el motor de

búsqueda, y su implementación es independiente de la base de conocimientos del problema a resolver.

Cabe destacar que en el caso de utilizar la estrategia A*, cada vez que se genera un nuevo nodo se revisa que *f(n)* del hijo creado sea menor que *f(n)* del padre; en caso de serlo, se utiliza el *f(n)* del padre. De esta manera, se ignoran los valores que puedan surgir al utilizar heurísticas que no sean monótonas crecientes, obteniendo una f no decreciente.

Heurísticas

Para los métodos informados (Greedy y A*) se implementaron dos heurísticas: una admisible y otra no admisible. Ambas heurísticas fueron desarrolladas solamente para la regla de Swap, debido a la facilidad de implementación y de contraste de casos teóricos con casos aplicados. Teniendo en cuenta esto, se procedió al desarrollo teórico de las mismas (previo a su implementación).

Admisible

Entre las condiciones para que A* encuentre el camino óptimo al objetivo, se pide que $h(n) \leq h^*(n)$ para todo nodo n. Al no contar con una h* conocida[1][2], se debió buscar una h' que cumpla que $h(n) \leq h'(n) \leq h^*(n)$ para todo nodo n, es decir, que asegure que esté acotada por h*(n) y por lo tanto, sea admisible.

Sabiendo que $h^*(n_0) = g^*(n_g) = g(n_g) = \#swaps * costoSwap$, y que un nodo goal representa un estado goal para el problema, se concluyó que

$$h^*(n) = \#\ swaps\ para\ solucionar\ todos\ los\ conflictos * costoSwap.$$

Si bien la cantidad de swaps para solucionar todos conflictos es desconocida, se puede encontrar una cota de la misma, suponiendo que cada swap resuelve todos los conflictos que se podrían resolver.

A continuación se detalla la cantidad máxima de conflictos que puede resolver un swap:

- Conflictos de repetidos
  - Filas: 2 (uno por cada fila involucrada en las celdas del swap)
  - Columnas: 2 (uno por cada columna involucrada en las celdas del swap)
- Conflictos de visibilidad
  - Filas: 4 (dos por cada fila que involucra una celda del swap: visibilidad izquierda y visibilidad derecha)
  - Columnas: 4 (dos por cada columna que involucra una celda del swap: visibilidad superior y visibilidad inferior)

De esta forma, la cantidad máxima de conflictos que se pueden resolver con un swap (de ahora en más: #MCRS) es 12.

Luego, suponiendo que cada swap resuelve la #MCRS (escenario ideal), se tiene que:

[1] Esto sucede pues la regla sobre la que se aplica la heurística es SWAP. Si fuese PUT, el costo de h* sería conocido e igual a la cantidad de espacios que todavía quedan por llenar.

[2] Si bien se conoce cómo calcular h* (fórmula que se deja expresa en los siguientes párrafos), no se conoce el valor numérico real de la # swaps para solucionar todos los conflictos, y es por esto que se la considera como *desconocida*.

$$h'(n) = \#\ ideal\ swaps\ para\ solucionar\ todos\ los\ conflictos * costoSwap \leq$$
$$\#\ swaps\ para\ solucionar\ todos\ los\ conflictos * costoSwap = h*(n)$$

Siendo $\#\ ideal\ swaps\ para\ solucionar\ todos\ los\ conflictos = \lceil \#conflictos / \#MCRS \rceil$

Habiendo encontrado una cota inferior de h*, la h elegida fue:

$$h(n) = \#conflictos / \#MCRS * costoSwap \leq \lceil \#conflictos / \#MCRS \rceil * costoSwap = h'(n)$$

Para facilitar los cálculos computacionales, se eligió que costoSwap = #MCRS, quedando entonces que $h(n) = \#conflictos$.

No admisible

Para el desarrollo de la heurística no admisible, lo que se hizo fue comprobar, estadísticamente, cuál era el número de conflictos que se resolvían por swap, lo que llevó a la conclusión de que en los primeros pasos, el algoritmo resolvía de hasta tres conflictos a la vez, pero a medida que los iba resolviendo se estancaba en un número casi constante de conflictos por tablero hasta encontrar la solución. Esto llevó a determinar que, en promedio, se puede resolver hasta un conflicto por swap. Es importante aclarar que esta heurística no admisible porque, por ejemplo, podría existir el caso en que un tablero tuviese 12 conflictos que pueden ser solucionados con un único swap, y para ese caso, $h_i(n) > h^*(n)$.

Formalmente, la h no admisible elegida fue: $h_i(n) = \#conflictos * costoSwap$.

Resultados

A continuación se presentan los resultados de ejecutar el motor utilizando todas las estrategias de búsqueda contra distintos casos de prueba. Para cada resultado, se enuncian la profundidad alcanzada, el costo total, cantidad de nodos explotados, cantidad de nodos expandidos y el tiempo de procesamiento (este es el orden mostrado en los resultados para cada configuración). Los resultados que se obtuvieron aplicando la regla Put se detallan en la Tabla 1. Los resultados que se obtuvieron aplicando la regla Swap con métodos de búsqueda informados se muestran en la Tabla 2. Es importante aclarar que no se tomaron mediciones respecto de utilizar la regla Swap en métodos de búsqueda no informados ya que los resultados obtenidos no pueden ser comparables entre sí, ya que al no poseer información adicional sobre qué camino tomar, se realizarán swaps indefinidamente hasta llegar a una solución.

Adicionalmente se muestran gráficos que comparan la cantidad de nodos explotados (Gráficos 1 y 2) y el costo total (Gráfico 3) de llegar a la solución para diferentes configuraciones.

Conclusiones

Respecto de los resultados obtenidos, se puede concluir:

- Utilizando la regla Put:
    - Utilizar la estrategia DFS es más eficiente en cuanto a cantidad de nodos explotados para cualquiera de los tamaños probados, con respecto al BFS.

    - Al tratarse de un problema cuya solución se encuentra siempre a la misma profundidad a partir del estado inicial, no justifica aplicar IDDFS pues su performance será siempre inferior o igual a la que ofrece un DFS.
- Utilizando la regla Swap:
    - Los métodos de búsqueda no informados son muy poco performantes.
- Si h devuelve un valor muy alejado de h*, A* presenta un excesivo uso de memoria debido al alto factor de ramificación, y un excesivo consumo de procesamiento al tener que mantener los nodos ordenados por f creciente.
- Utilizar una h no admisible, cuyos valores fueron obtenidos estadísticamente, permite mejorar la performance de memoria de A*, aunque se pierda la certeza de encontrar la solución óptima.

Anexo

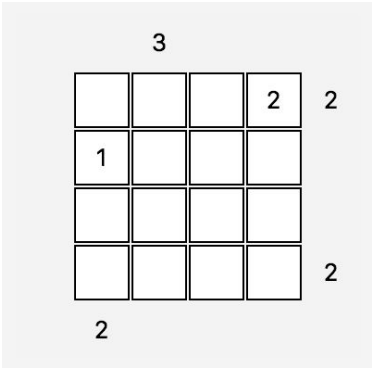
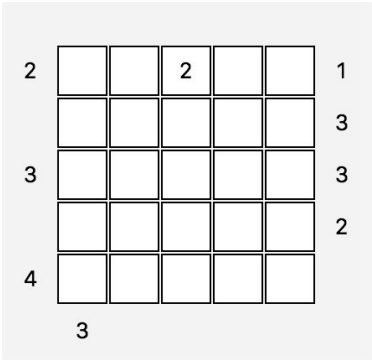
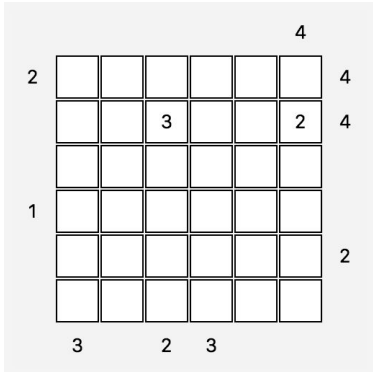
4x4	5x5	6x6
		

Imagen 1: Tableros iniciales utilizados en la obtención de resultados

		Estrategia				
		BFS	DFS	IDDFS	Greedy	A*
Tamaño de lado	4	14	14	14	14	14
		14	14	14	14	14
		111	85	666	75	111
		0	3	3	3	0
		24 ms	8 ms	26 ms	17 ms	35 ms
	5	24	24	24	24	24
		24	24	24	24	24
		2420	405	24861	855	2419
		0	11	11	7	1
		106 ms	15 ms	443 ms	17 ms	192 ms
	6	34	34	34	34	34
		34	34	34	34	34
		97742	35783	1287514	58210	97742
		0	18	18	14	0
		2250 ms	885 ms	29481 ms	1429 ms	8556 ms

Tabla 1: Aplicación de regla Swap en métodos de búsqueda no informados

		Estrategia			
		Heurística admisible		*Heurística no admisible*	
		Greedy	A*	Greedy	A*
Tamaño de lado	4	10	4	10	4
		120	48	120	48
		10	20316	10	49
		595	1245938	595	2996
		88 ms	29708 ms	84 ms	187 ms
	5	53		53	10
		636		636	120
		53		53	5924
		9354		9354	1060278
		559 ms		492 ms	22970 ms
	6	375		375	
		4500		4500	
		375		375	
		133692		133692	
		8841 ms		8835 ms	

Tabla 2 : Aplicación de regla Swap en métodos de búsqueda informados
Aclaración: Las celdas en rojo corresponden a resultados vacíos pues la cantidad de recursos necesaria (procesamiento y memoria) para llegar a la solución son muy altas

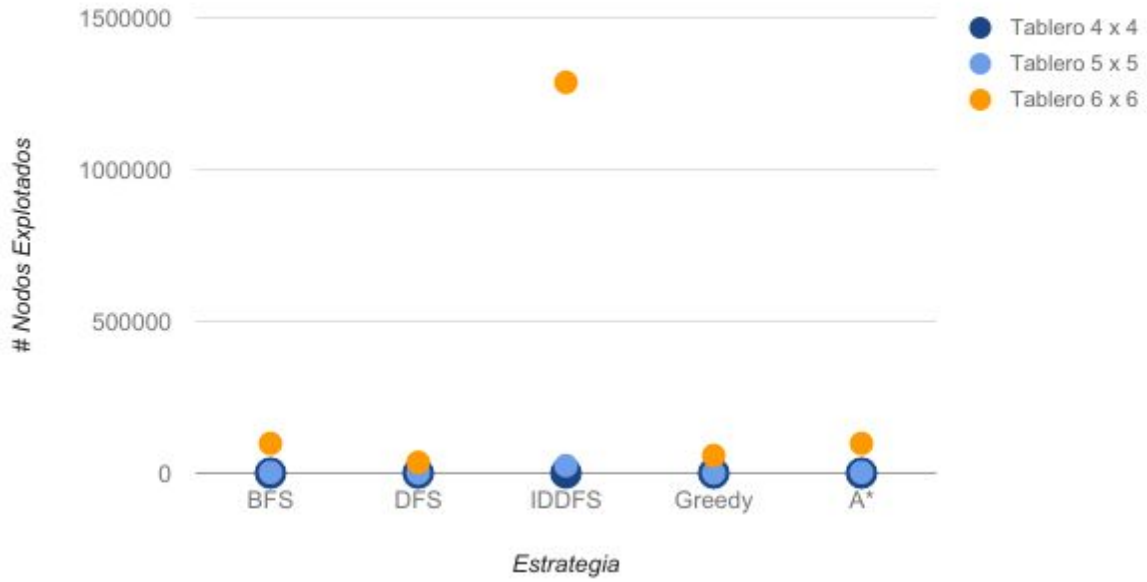


Gráfico 1: Comparación entre nodos explotados para diferentes estrategias contra diferentes tamaños de tableros

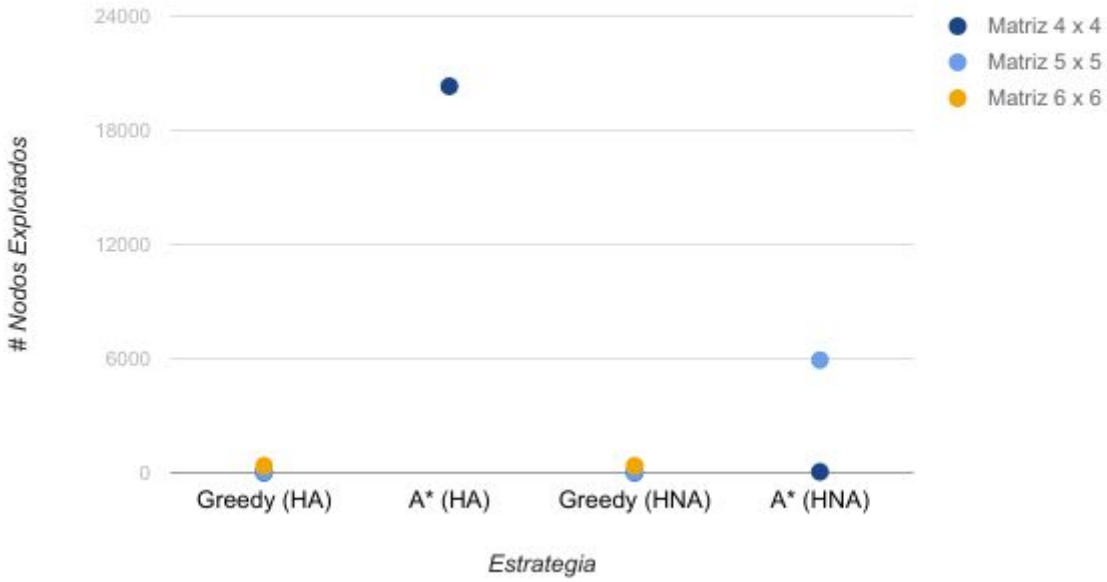


Gráfico 2: Comparación entre nodos explotados para diferentes estrategias (informados)contra diferentes tamaños de tableros

Aclaración: HA corresponde a heurística admisible y HNA a heurística no admisible

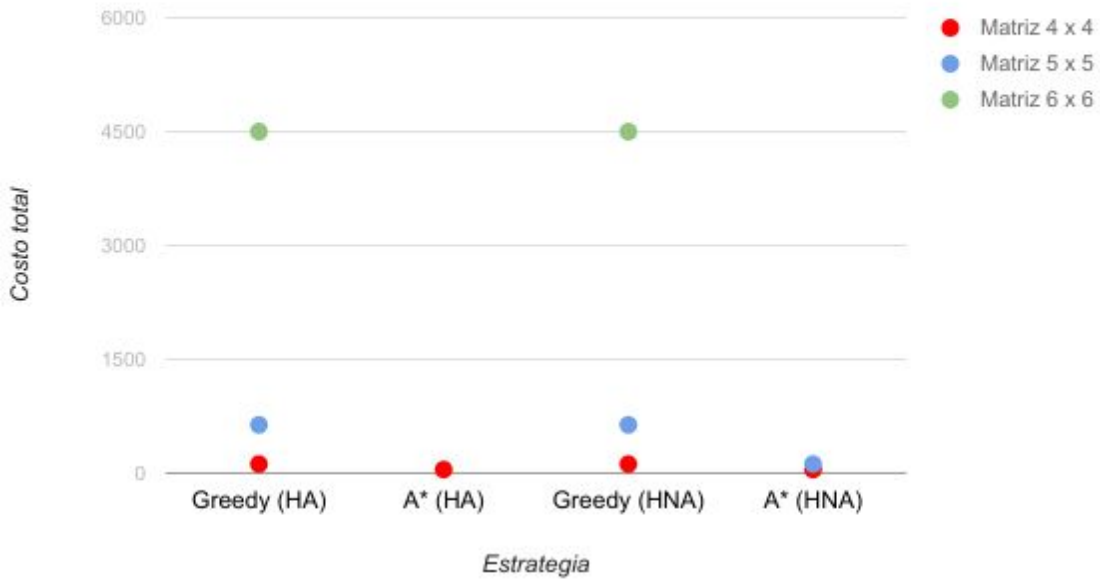


Gráfico 3: Comparación entre costo para diferentes estrategias informadas contra diferentes tamaños de tableros

Notas adicionales

- Otra regla posible a implementar es tomar 1 vector de los N! posibles, tales que cada vector representa una permutación de los valores entre 1 y N. En general, ocurre que $N! > N^3$, con lo cual la cantidad de reglas posibles a aplicar por nodo es mayor, pero la profundidad del árbol será, a lo sumo, de tamaño N.