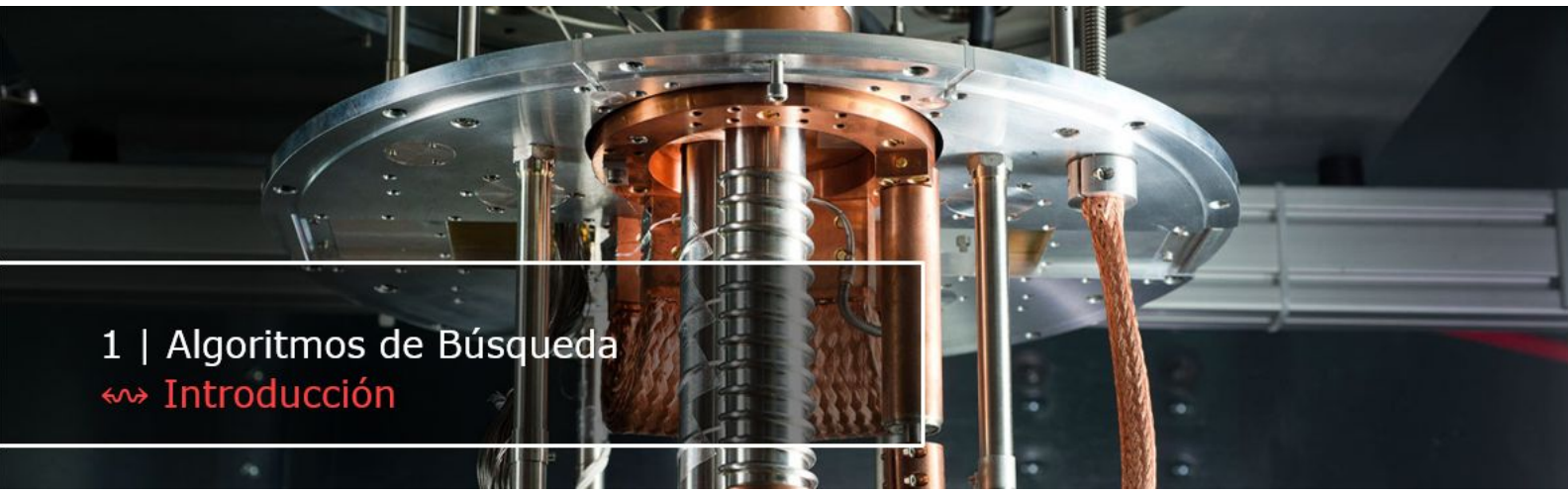
ITBA | Sistemas de Inteligencia Artificial

Algoritmos de Búsqueda

[SEARCH: FILL ZONE]

(54.623) Juan Dantur
(55.382) Ariel Debrouvier
(53.396) Agustín Golmar

❖ Tabla de Contenidos



1. Introducción

En el siguiente informe[1] se describe la construcción de un motor de resolución de problemas genérico (*General Problem Solver*), basado en estrategias de búsqueda desinformadas (*blind search*), e informadas (*heuristic search*).

Como caso de uso, la cátedra solicitó que se resuelva el problema ***Fill Zone***, un videojuego que consiste en propagar el color de una celda distinguida hasta cubrir el tablero completo, considerando como buena solución aquella secuencia que posee la menor cantidad de propagaciones.

Figura 1.1: Fill Zone, en su versión online[2].

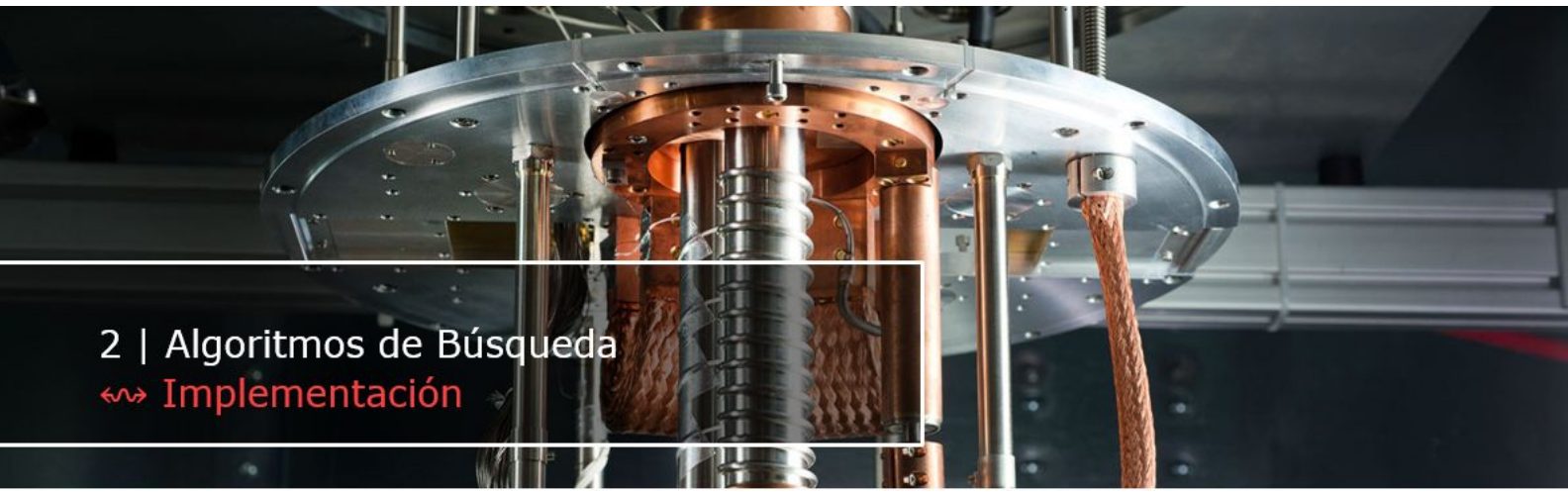
Como se observa en la *Figura 1.1*, el tablero se representa mediante una matriz cuadrada, y la paleta de colores de propagación posee *6* tonos, aunque en la implementación es posible proporcionar tableros genéricos (rectangulares inclusive), y paletas de cualquier dimensión.

Si bien el juego original está limitado por la cantidad máxima de propagaciones, esa limitación depende de la optimalidad de los algoritmos de búsqueda y de las heurísticas implementadas, por lo cual, la restricción fue eliminada.

---

[1] La imagen de la portada se corresponde con el bus principal del computador cuántico *2000Q*, desarrollado por la empresa *D-Wave Systems* (https://www.dwavesys.com/).

[2] http://www.mindjolt.com/fill-zone.html



2. Implementación

A continuación se detallarán las características funcionales y no-funcionales del sistema desarrollado, junto con las decisiones de diseño tomadas.

2.1. Arquitectura

Se desarrolló sobre la plataforma *Java SE 8 Release*, y puede extraerse en formato **.jar* utilizando un compilador *Maven 3*. Para más información acerca del proceso de generación y ejecución, remitirse al *README.md* en el repositorio[3].

Se utilizaron las dependencias de *Google Guice* para inyección de dependencias y desacoplamiento, *Logback + SLF4j* para logging y *JAXB* para el archivo de configuración, en formato **.xml*.

2.2. General Problem Solver

Se utilizó como base el motor provisto por la cátedra[4], sobre el cual se realizaron algunas modificaciones. En particular, se optimizó la escritura del código, aunque manteniendo las interfaces provistas. Se realizaron pequeñas modificaciones en el funcionamiento de los algoritmos, para incrementar su eficiencia. En particular, para garantizar la optimalidad del algoritmo *Iterative Deepening DFS*, se seleccionan los nodos de menor costo, lo que implica que las soluciones propuestas por esta estrategia sean tan eficientes como las que *BFS* ofrece, aunque con una marcada reducción de nodos explorados en la estructura *open list*.

Además, durante la explosión de los nodos, los mismos se ingresan a una estructura `ArrayList`, en todos los casos, para luego transferirse a la *frontera* correspondiente (según cada estrategia). Esto permite un incremento substancial en la *performance* de los algoritmos, en particular

[3] https://bitbucket.org/itba/sia-2017a-11
[4] https://github.com/apierri/GeneralProblemSolver

de las estrategias informadas *Greedy* y *A**, debido a que los mismos requieren que la estructura *open list* retenga un orden global basado en la ***función de evaluación***[5] sobre los nodos, y no de forma local, como lo expresaba el motor original.

Algunas implementaciones utilizan una estructura `PriorityQueue` como base de todos los algoritmos, pero se decidió utilizar una lista enlazada (`LinkedList`) para las estrategias *blind* debido a que posee mayor desempeño durante las inserciones y extracciones en sus extremos. De hecho, es innecesaria la utilización de una estructura consciente del orden: en *DFS*, *BFS*, e *Iterative Deepening* no existe tal cosa como la *prioridad*.

DFS e *Iterative Deepening* son equivalentes: el primero opera hacia una profundidad máxima teóricamente inalcanzable (`Integer.MAX_VALUE`), y el segundo cicla el árbol de forma incremental (aumentando la profundidad máxima en cada caso). Para ello, se agregó una propiedad en la clase *GPSNode* que representa la profundidad de dicho nodo. Esto evita ejecutar una función de complejidad proporcional a $O(n)$.

Ya que el motor utiliza una estructura *close list* para evitar la repetición de estados durante la exploración, la búsqueda en general se considera *Graph Search* en lugar de *Tree Search*.

2.3. Fill Zone

Se define el problema a resolver formalmente, según sus componentes principales[6]:

- **Initial State**: el estado inicial se extrae de la especificación en formato **.sia*, y se almacena en una matriz de enteros. En caso de utilizar la heurística *graph*, se almacena en un grafo no-dirigido. En cualquier caso, la paleta de colores no se almacena de ninguna forma (sólo su dimensión).
- **Actions**: las acciones aplicables se corresponden simplemente con propagar uno de los k colores de la paleta, siempre y cuando no se aplique el mismo color que la celda superior izquierda (*celda distinguida*).
- **Transition Model**: el resultado de aplicar un color, implica modificar el color de la celda distinguida, y luego propagar el mismo a través de las celdas adyacentes (arriba, abajo, izquierda y derecha, pero no en diagonal), de forma recursiva. Esto modifica las celdas de la matriz, o aglomera los nodos en el caso del grafo.
- **Goal Test**: el estado objetivo se alcanza cuando todas las celdas de la matriz son iguales (poseen el mismo color), o lo que es equivalente, cuando el grafo posee sólo un nodo.
- **Path Cost**: denotado usualmente por $g(n)$, y definido como la suma del costo de las acciones aplicadas, o *step cost*, donde el mismo es $c(n, a, n') = 1$, ya que la *performance* se mide en

[5] *"Artificial Intelligence: A Modern Approach"*, Sección 3.5: *Informed (Heuristic) Search Strategies*, Parte II (*Problem Solving*), Russell & Norvig, 3rd Ed., 2010, Prentice Hall.
[6] *"Artificial Intelligence: A Modern Approach"*, Sección 3.1.1: *Well-defined Problems and Solutions*, Parte II (*Problem Solving*), Russell & Norvig, 3rd Ed., 2010, Prentice Hall.

función de la cantidad de acciones aplicadas, y además es uniforme con respeto a todos las acciones aplicables.

2.4. Heurísticas

Se implementaron un total de *3* heurísticas, de las cuales *2* son *admisibles*, y además (aunque aquí no se demuestra), son *consistentes*.

2.4.1. Distinct

Consiste en computar la cantidad de colores distintos presentes en la matriz. Esto implica un límite inferior en la cantidad de propagaciones a realizar: si hay k colores distintos, es necesario aplicar un mínimo de $k-1$ acciones. Debido a ello, la heurística es ***admisible*** (*Figura 4.1.1*). Durante el *goal test*, solo hay un color, y por lo tanto la cantidad de colores distintos es *0*. Debido a que la cátedra consideró que esta heurística es trivial, se optó por implementar otra estrategia admisible (*graph*).

2.4.2. Frontier

Esta heurística es ***no admisible*** (*Figura 4.1.2*). Implica hallar todas las celdas de la matriz adyacentes inmediatamente al *cluster distinguido* (aquel que posee la celda distinguida y un coloreo uniforme). Luego se computa la cantidad de celdas en esta frontera para cada color de la paleta. Finalmente se selecciona el valor máximo (*max*), y se computa:

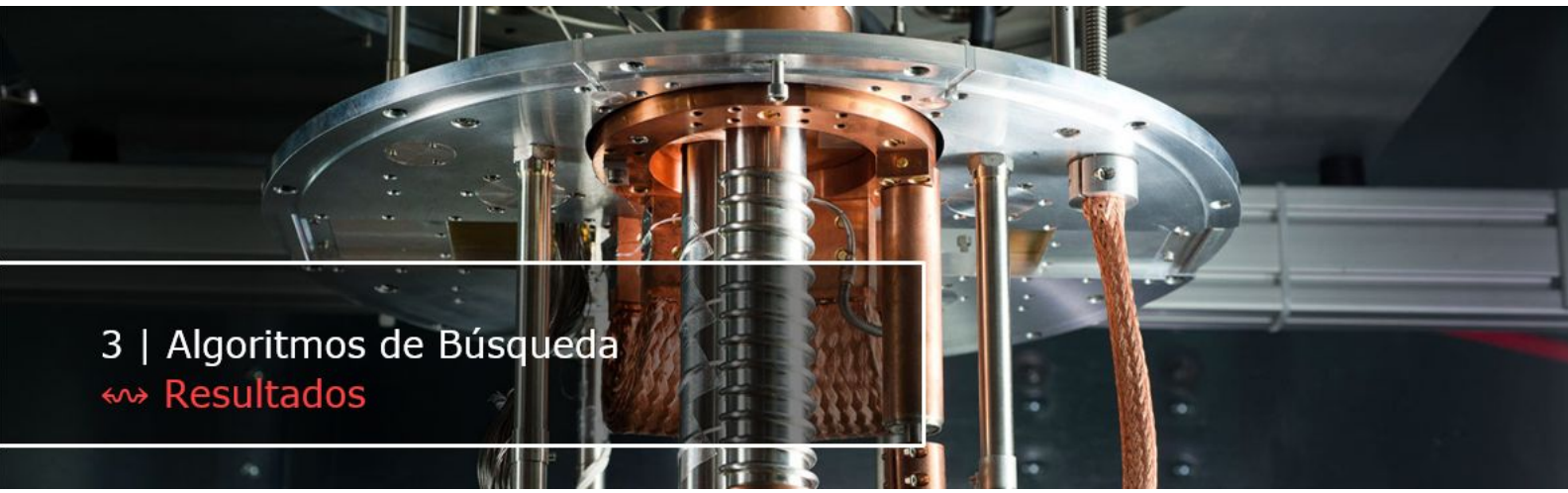
$$h(n) = 1 + frontier\ size - max$$

En caso de que el máximo sea *0*, $h(n) = 0$.

2.4.3. Graph

En este caso, se utiliza la representación basada en un *grafo no-dirigido*. Cada nodo del grafo representa un *cluster* donde sus celdas son adyacentes y del mismo color. Cada arista representa la adyacencia entre dos clusters. La heurística consiste en hallar la distancia de máxima longitud utilizando un recorrido *BFS* desde el *cluster distinguido* sobre un grafo *no-ponderado* y devolver el largo de este camino, como valor heurístico (*Figura 4.1.3*). Debido a que cada cluster es maximal en la cantidad de celdas del mismo color adyacentes que posee, el largo del camino representa una cota inferior en la cantidad de propagaciones a realizar, y por lo tanto el costo nunca es sobreestimado, por lo que la heurística es ***admisible***.

Cuando se encuentra en un *goal state*, el grafo posee sólo *1* nodo, y el largo del camino es cero (*i.e.*, $h(n) = 0$).

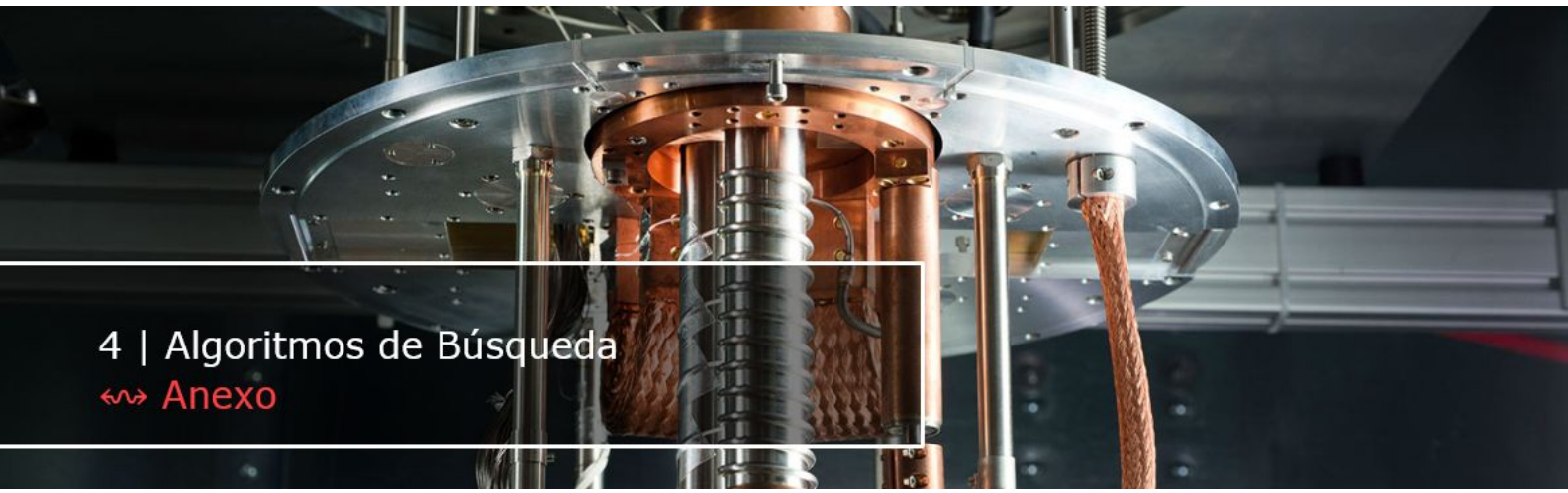


3. Resultados

Debido a la especificación del problema *Fill Zone*, es posible resolver cualquier tablero simplemente aplicando las reglas de forma sistemática (*i.e.*, aplicando todos los colores en secuencia, y luego volver a comenzar, hasta propagar por todo el tablero). Es por ello que dentro de las técnicas desinformadas, *DFS* presenta mayor eficiencia (*Tabla 4.2.1/4/7*), en cuanto al espacio de búsqueda explorado. Tanto *BFS* como *Iterative Deepening* devuelven soluciones óptimas, pero saturan la memoria disponible debido a su explosión de búsqueda. Por otro lado, las estrategias informadas extienden el horizonte de problemas tratables considerablemente. En particular, la estrategia *Greedy* expande menos nodos que *A**, pero carece de optimalidad (*Tabla 4.2.2/5/8*), debido a que es independiente del costo $g(n)$. Adicionalmente, *A** es un caso intermedio entre ambos grupos: es más costoso que *Greedy*, pero es mejor que los algoritmos desinformados (*Tabla 4.2.3/6/9*).

Es importante hacer hincapié en el hecho de que la representación del problema es crucial para el desempeño de la búsqueda. En particular, las heurísticas *frontier* y *graph* son costosas, lo que puede incurrir en la preferencia de un método más simple como *distinct*. Debido a ello se plantean algunas optimizaciones futuras:

- En la ***representación*** del tablero, quizás sea útil utilizar una matriz inicial y luego una estructura que almacene las diferencias al aplicar una acción de propagación. Esto permitiría deshacer las acciones aplicadas y utilizar la misma matriz para todo el árbol de búsqueda, reduciría la memoria consumida, y extendería el rango de tractabilidad de todos los algoritmos.
- Al computar el largo del camino durante la heurística *graph*, se utiliza una búsqueda en anchura (*BFS*), la cual, en el peor de los casos (donde cada nodo representa un *cluster* de una sola celda), puede incurrir en una alta degradación de la *performance*. Como es vital no sobreestimar el largo de este camino para mantener la admisibilidad de la heurística, podría aplicarse ***Limited BFS*** para cierto l, estableciendo un límite máximo de exploración.
- En la explosión de nodos, es relativamente fácil razonar un motor de búsqueda que implemente ***multi-threading***, despachando cada nodo en un *thread* particular, y permitiendo que la expansión se realice en paralelo, para cada subárbol, hasta que algún hilo encuentre la solución, sincronizando las estructuras *open* y *close list*.



4. Anexo

En esta sección se despliegan las figuras referenciadas y los resultados numéricos obtenidos durante la etapa de *testing* y *benchmarking*.

4.1. Figuras

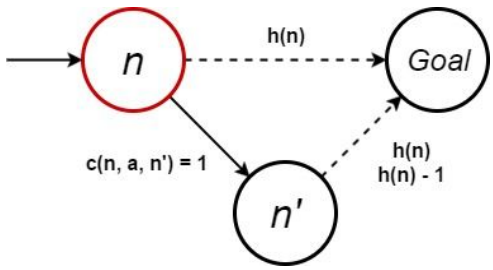


Figura 4.1.1: Demostración de consistencia para la heurística *distinct*, donde claramente se observa que la cota inferior siempre es *h(n)*, para todo *n*.

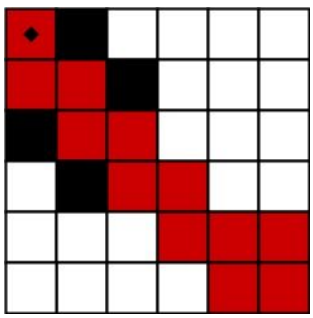


Figura 4.1.2: Contraejemplo para demostrar que la heurística *frontier* no es admisible. En este caso *h(n) = 1 + 9 - 5 = 5*, pero en solo *2* pasos se puede resolver el tablero, y por lo tanto hay sobreestimación.

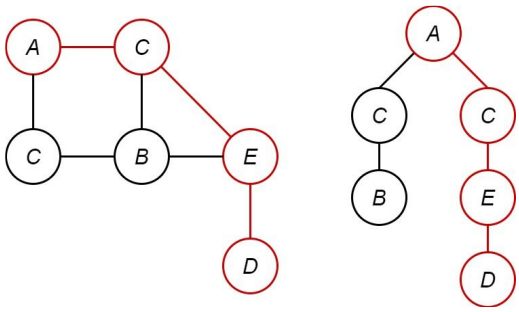


Figura 4.1.3: A la izquierda, una representación basada en grafos.
A la derecha, el camino más largo obtenido mediante *BFS*. En este caso, *h(n) = 3*.

4.2. Benchmarks[7]

Todas las pruebas se ejecutaron sobre un procesador doble núcleo *Intel Celeron N2840* de *2.16 GHz*, con *4 Gb.* de memoria *RAM*, bajo un sistema *Windows 10 Home x64*.

Problem		Dimension	Palette
`light4x5_6.sia`		*4x5*	*6*
Strategy	**BFS**	**DFS**	**ID-DFS**
Depth/Cost[8]	7	21	7
Explosions	539	21	682
Frontier Size	1131	67	16
Time [sec.]	0.062	0.007	0.075

Tabla 4.2.1: Blind search, *4x5* + *6* colores.

Problem		Dimension	Palette
`light4x5_6.sia`		*4x5*	*6*
Strategy	**Greedy**		
Heuristic	`distinct`	`frontier`	`graph`
Depth/Cost	7	11	7
Explosions	13	53	11
Frontier Size	52	136	44
Time [sec.]	0.018	0.120	0.063

Tabla 4.2.2: Greedy heuristic search, *4x5* + *6* colores.

[7] Las mediciones se realizaron promediando *3* ejecuciones sobre el mismo problema, para reducir el error muestral obtenido. Sólo se muestran tres cifras significativas (decimales), aunque se utilizó un cronómetro nanométrico.
[8] Para este problema (*Fill Zone*), el costo siempre es equivalente a la profundidad de la solución.

Problem		Dimension	Palette
`light4x5_6.sia`		*4x5*	*6*
Strategy	**A***		
Heuristic	`distinct`	`frontier`	`graph`
Depth/Cost	7	7	7
Explosions	39	65	173
Frontier Size	136	172	580
Time [sec.]	0.029	0.176	0.294

Tabla 4.2.3: A* heuristic search, *4x5* + *6* colores.

Problem		Dimension	Palette
`medium30x30_5.sia`		*30x30*	*5*
Strategy	**BFS**	**DFS**	**ID-DFS**
Depth/Cost	-	106	-
Explosions	-	106	-
Frontier Size	-	288	-
Time [sec.]	*OOM*[9]	0.130	*OOM*

Tabla 4.2.4: Blind search, *30x30* + *5* colores.

Problem		Dimension	Palette
`medium30x30_5.sia`		*30x30*	*5*
Strategy	**Greedy**		
Heuristic	`distinct`	`frontier`	`graph`
Depth/Cost	85	-	38
Explosions	159	-	52
Frontier Size	437	-	156
Time [sec.]	0.264	*TO*[10]	6.796

Tabla 4.2.5: Greedy heuristic search, *30x30* + *5* colores.

[9] *OOM* se entiende por *Out Of Memory*, un error generado por la máquina virtual de *Java* cuando la aplicación excede la cantidad de memoria disponible.
[10] *TO* se entiende por *Time Out*, donde el límite de tiempo impuesto es de *10* minutos. Si una búsqueda supera el límite sin consumir la memoria disponible, se etiqueta con *TO*, en lugar de *OOM*.

Problem		Dimension	Palette
`medium30x30_5.sia`		*30x30*	*5*
Strategy	**A***		
Heuristic	`distinct`	`frontier`	`graph`
Depth/Cost	-	-	-
Explosions	-	-	-
Frontier Size	-	-	-
Time [sec.]	*OOM*	*OOM*	*OOM*

Tabla 4.2.6: A* heuristic search, *30x30* + *5* colores.

Problem		Dimension	Palette
`heavy100x100_5.sia`		*100x100*	*5*
Strategy	**BFS**	**DFS**	**ID-DFS**
Depth/Cost	-	446	-
Explosions	-	446	-
Frontier Size	-	1261	-
Time [sec.]	*OOM*	0.728	*OOM*

Tabla 4.2.7: Blind search, *100x100* + *5* colores.

Problem		Dimension	Palette
`heavy100x100_5.sia`		*100x100*	*5*
Strategy	**Greedy**		
Heuristic	`distinct`	`frontier`	`graph`
Depth/Cost	333	-	-
Explosions	650	-	-
Frontier Size	1850	-	-
Time [sec.]	3.049	*OOM*	*TO*

Tabla 4.2.8: Greedy heuristic search, *100x100* + *5* colores.

Problem		Dimension	Palette
`heavy100x100_5.sia`		*100x100*	*5*
Strategy	**A***		
Heuristic	`distinct`	`frontier`	`graph`
Depth/Cost	-	-	-
Explosions	-	-	-
Frontier Size	-	-	-
Time [sec.]	*OOM*	*TO*	*OOM*

Tabla 4.2.9: A* heuristic search, *100x100* + *5* colores.