SIMPLE SQUARES

IMPLEMENTACIÓN DE GPS

Grupo 4: Martina Arco, Joaquín Ormachea, Diego Orlando y Lucas Emery

Introducción

Para el trabajo práctico de algoritmos de búsqueda, se nos pidió que implementemos un GPSEngine con dichos algoritmos y un juego que luego utilice estos algoritmos para encontrar una solución. El juego que nos tocó fue Simple Squares.

El juego Simple Squares consiste en un tablero de casilleros con cuadrados y círculos de colores. Los círculos son fijos en su posición inicial, mientras que los cuadrados pueden realizar movimientos unitarios en su dirección. El juego consiste en lograr que todos los cuadrados están en el mismo casillero que sus círculos correspondientes. Los cuadrados pueden empujarse entre sí.

Implementación de Simple Squares

Se decidió armar objetos Circle y Square que contienen su información de color, dirección, etc. Luego estos son guardados en 2 mapas con la forma Map<Point, Circle> y Map<Point, Square>. De esta forma se evita tener que guardar y modificar un tablero.

Hay una única regla por cuadrado que consiste en mover el mismo en su dirección un casillero y si existe un cuadrado en ese casillero, este se mueve al siguiente. Esta propagación de movimiento la implementamos con una función recursiva.

Los costos en el juego son siempre 1 ya que equivale a un movimiento por turno y no hay diferentes reglas a aplicar.

Heurísticas

Para las heurísticas nos pareció dificultoso encontrar buenas estimaciones que no subestimen las mejores situaciones posibles. También nos costó interpretar la trivialidad de estas ya que como el juego es bastante simple, las heurísticas que se pueden aplicar parecen simples.

Para poder hacer el planteo y observar si es admisible, encaramos el problema desde los casos particulares más favorables. Ejemplo: XYZ000. Ahí podemos observar que cada cuadrado está a 3 de distancia de su objetivo pero si los movimientos los hago con X, todos llegan a su objetivo en solamente 3 turnos. A partir de esto decidimos que siempre que evaluaremos los movimientos a hacer tendríamos que utilizar el máximo de movimientos en el tablero y no la suma. Ya que sino estaríamos sobreestimando estos casos óptimos.

Por un lado decidimos utilizar máxima distancia manhattan de un cuadrado a su círculo correspondiente. Por cada cuadrado calculamos la suma de los movimientos necesarios en x e y para llegar a su objetivo

La segunda decidimos hacer máxima distancia lineal ya que claramente subestima qué tan lejos de la solución estamos, por lo que es admisible. Esta heurística no es tan eficiente porque estima usando diagonales que es un movimiento imposible.

Para la tercer heurística intentamos buscar algo más complejo por lo que decidimos disectar cada cuadrado en sus movimientos necesarios por dirección y sentido. Y luego realizar la suma de los máximos en cada dirección y sentido. Esto lo que nos permite es tener una idea de la cantidad de movimientos mínimos que se van a tener que hacer en cada dirección contemplando el caso de que en cada dirección el cuadrado más alejado puede empujar a los otros.

Conclusiones Generales

A* es el más eficiente en cuanto a nodos, es decir, en cuanto a su expansión y estados que analiza pero no con el tiempo debido a la heurística. Consideramos que es la mejor opción a utilizar si se desarrolla una heurística adecuada ya que es significativamente superior a los otros algoritmos (gráficos 1-4). Esto es para el caso de problemas con solución; para problemas sin solución es muy similar al resto de los casos (gráficos 5-8).

IDDFS es el menos eficiente en cuanto a nodos ya que es el que más expande y más nodos frontera tiene pero ayuda a brindar una solución parcial en cualquier momento. Con esto nos referimos a que si se frena en una cierta profundidad, se encontrará la solución más óptima en ese momento. Esto se cumple con costo 1, que es nuestro caso.

BFS es el que lleva más tiempo y analiza más estados pero garantiza solución óptima con costo 1. Podría usarse para este tipo de problemas y donde no importe la eficiencia pero se quiera una implementación rápida.

En cuanto a conclusiones propias de este juego, se puede decir que la cantidad de reglas a generar eran pocas, por lo que los tiempos eran chicos ya que se generaban pocos nodos (cada padre generaba la cantidad de hijos igual o menor a la cantidad de colores). Además, cuanto mayor la cantidad de colores, más se apreciaba la diferencia entre algoritmos (ver tabla 1 versus tabla 2).

Asimismo, los problemas sin solución, al ser un tablero limitado se resolvían rápidamente. No había infinita cantidad de movimientos, cuando los cuadrados llegaban al borde del tablero y no era una solución, ese estado se consideraba inadmisible.

En cuanto a las heurísticas, como era esperable la mejor es la que suma la máxima distancia en cada dirección ya que es la que más se asemeja a un caso real

(ver tabla 4). Todas encuentran la solución de menor costo porque son todas admisibles.

Anexo

Algoritmo de búsqueda	Nodos expandidos	Estados analizados	Nodos frontera	Costo y profundidad	Tiempo (ms)
BFS	12	13	14	4	8
DFS	11	12	12	4	5
IDDFS	20	14	30	4	11
GREEDY	11	12	12	4	6
A*	7	8	10	4	8

Tabla 1: problema con 2 colores.

Algoritmo de búsqueda	Nodos expandidos	Estados analizados	Nodos frontera	Costo y profundidad	Tiempo (ms)
BFS	394	693	909	8	52
DFS	159	243	249	8	21
IDDFS	447	533	1176	8	36
GREEDY	136	249	260	10	35
A*	49	60	137	8	22

Tabla 2: problema con 3 colores.

Algoritmo de búsqueda	Nodos expandidos	Estados analizados	Nodos frontera	Costo y profundidad	Tiempo (ms)
BFS	93	183	182	9	20
DFS	93	183	182	5	18
IDDFS	435	183	978	5	44
GREEDY	93	183	182	3	17
A*	93	161	182	9	29

Tabla 3: problema con sin solución.

	Nodos expandidos	Estados analizados	Nodos frontera	Costo y profundidad	Tiempo (ms)
Máxima dirección	49	67	133	8	22
Máxima Manhattan	96	122	250	8	33
Distancia lineal	180	255	468	8	69

Tabla 4 : A* con diferentes algoritmos.

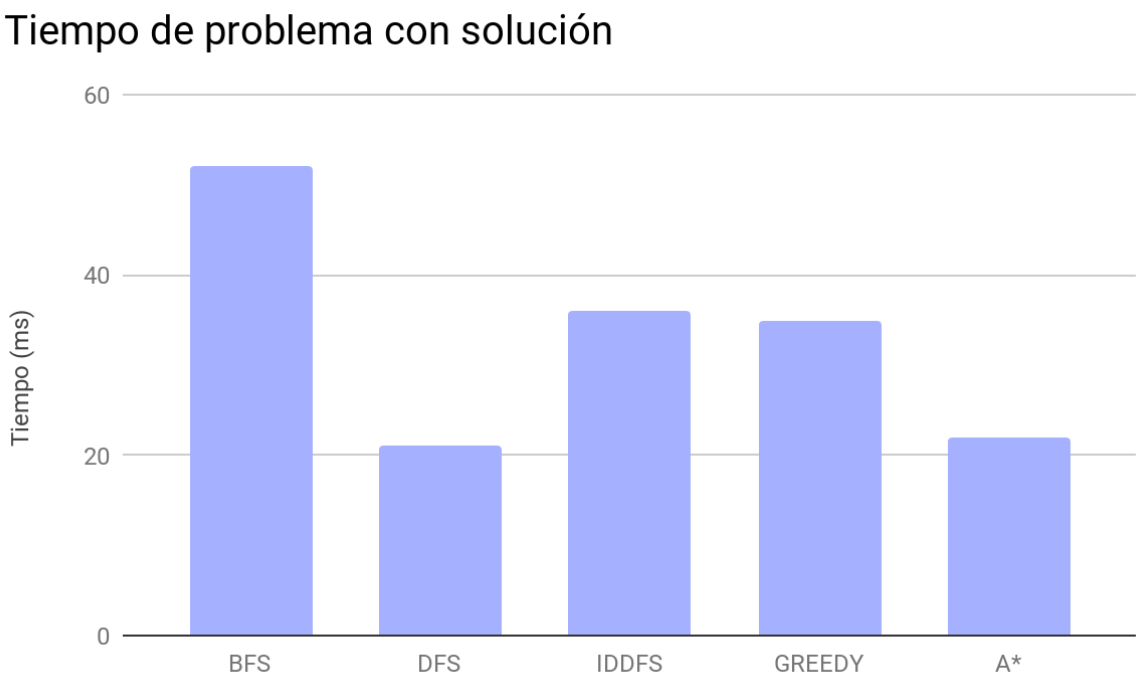

Gráfico 1

Gráfico 2

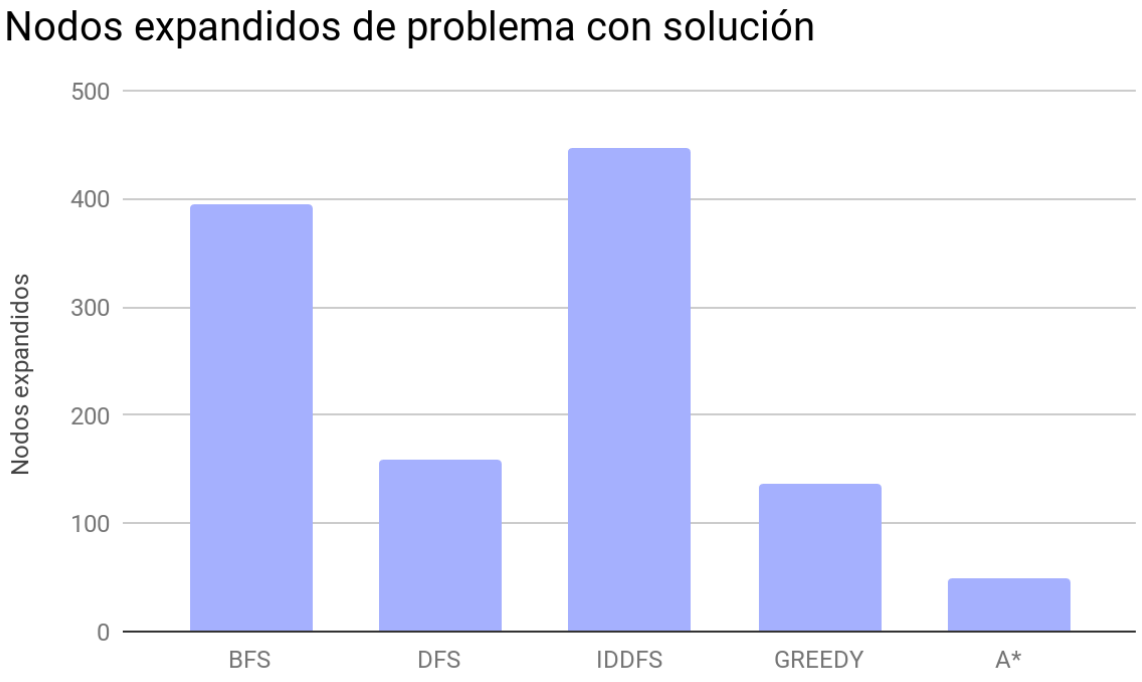

Gráfico 3

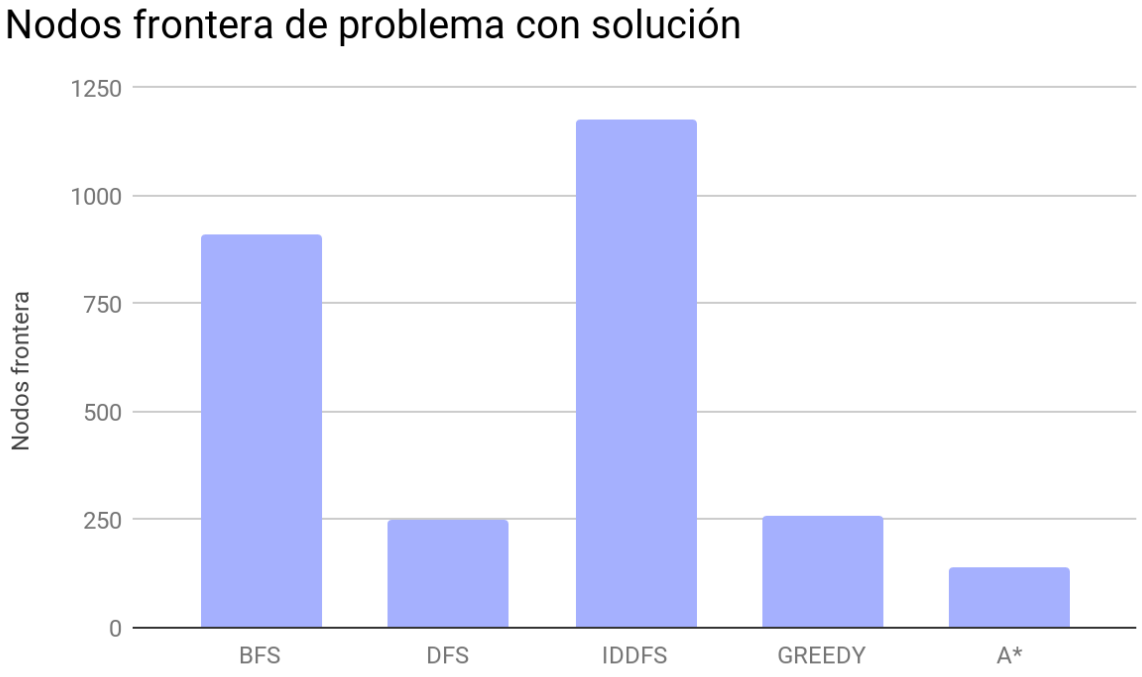

Gráfico 4

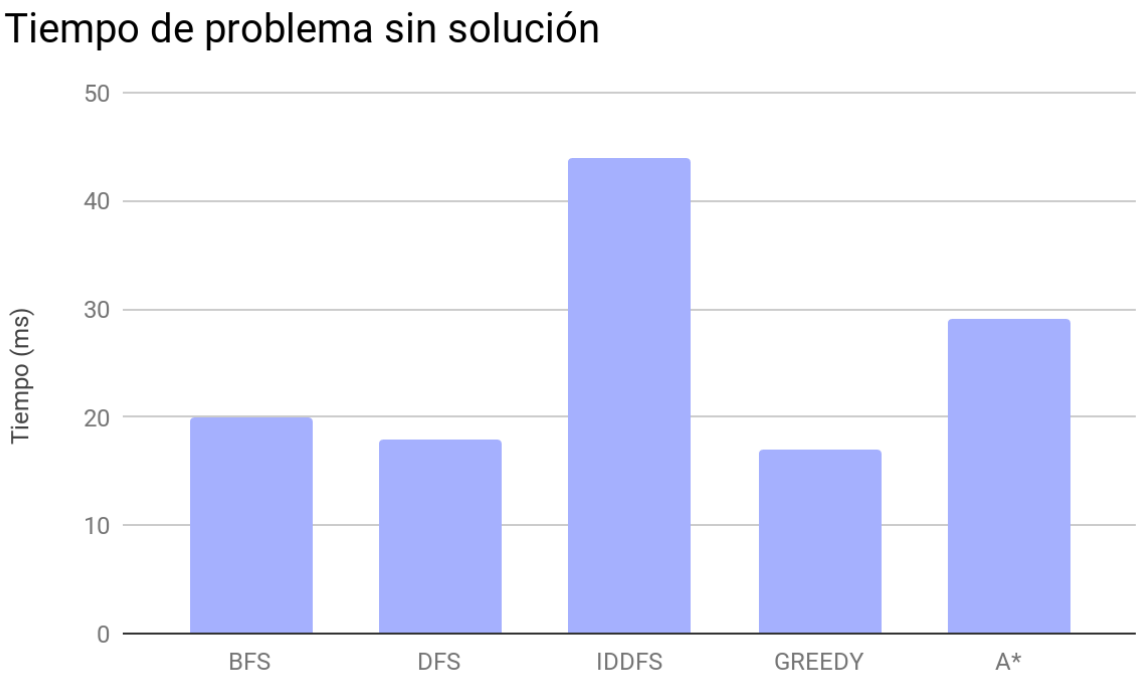

Gráfico 5

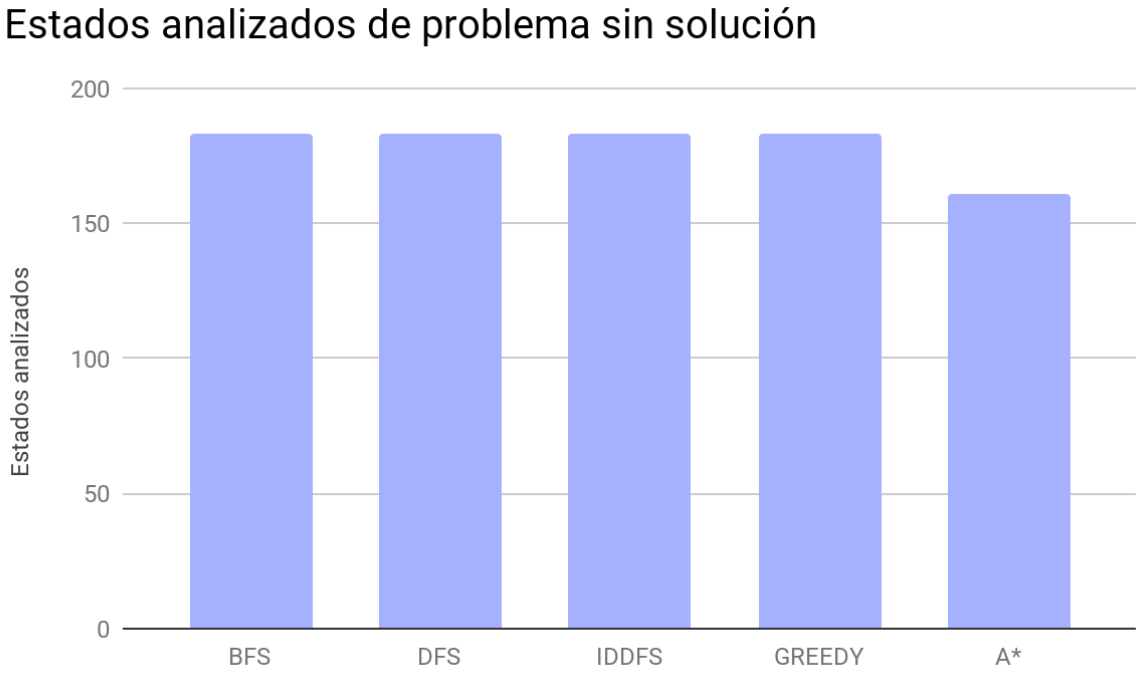

Gráfico 6

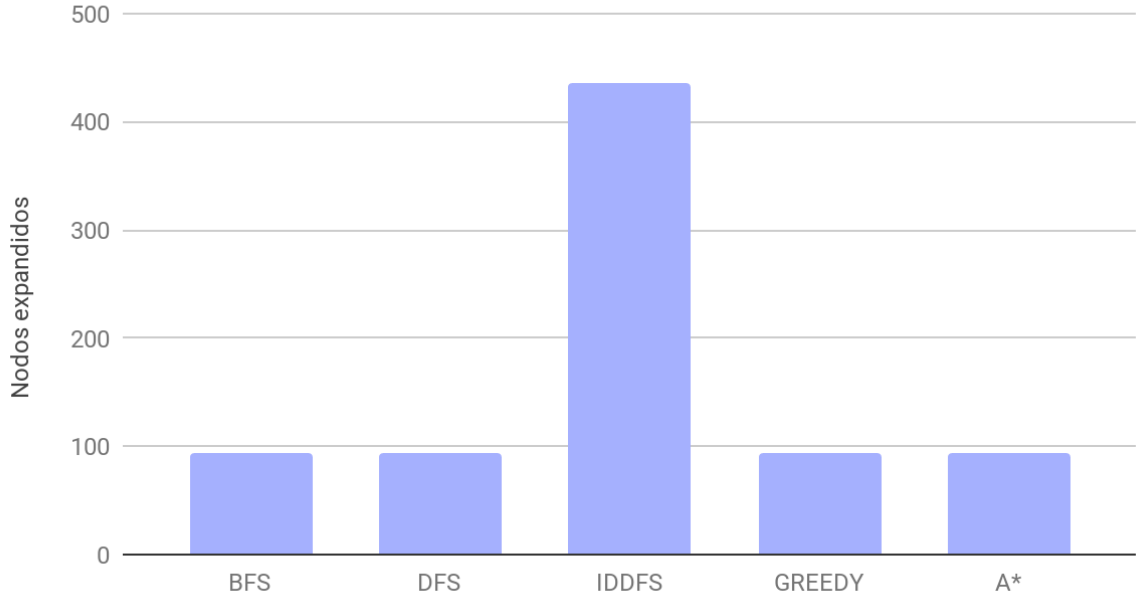

Gráfico 7

Gráfico 8

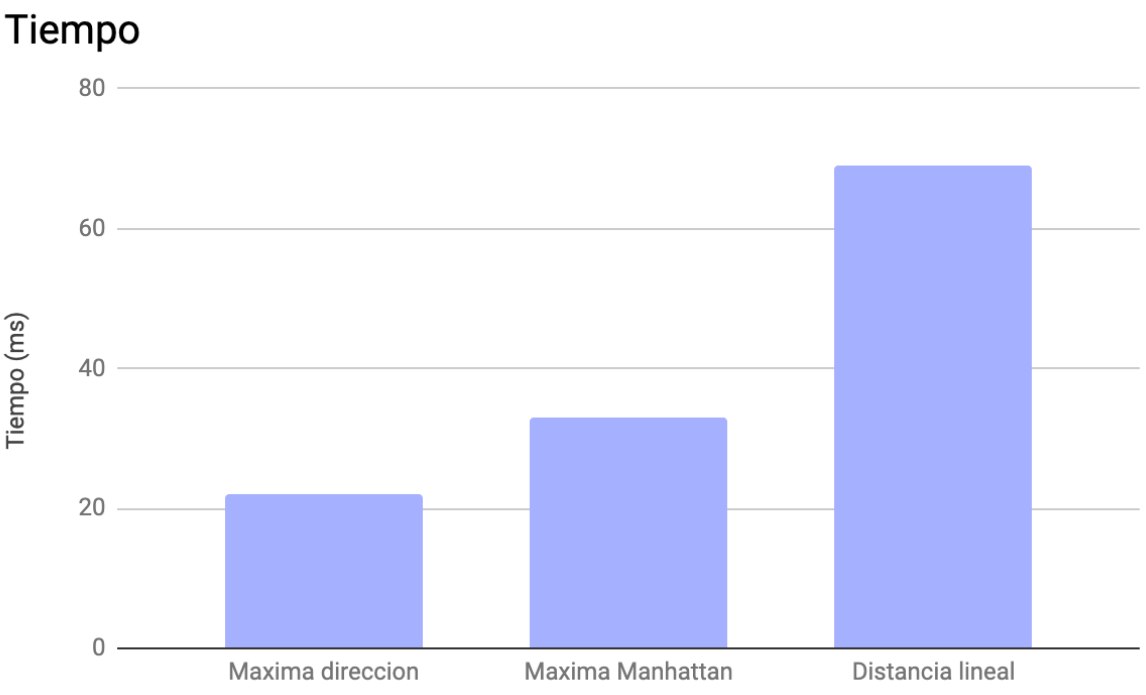

Gráfico 9 : comparación de tiempos de A* con diferentes heurísticas

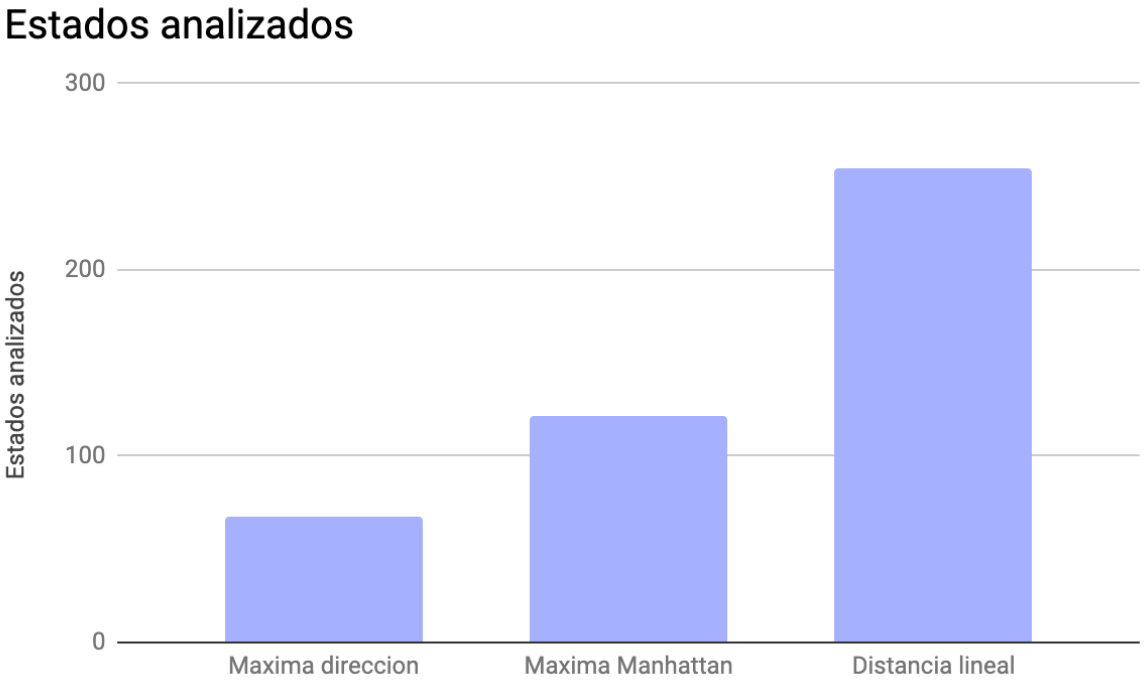

Gráfico 10 : comparación de estados analizados de A* con diferentes heurísticas

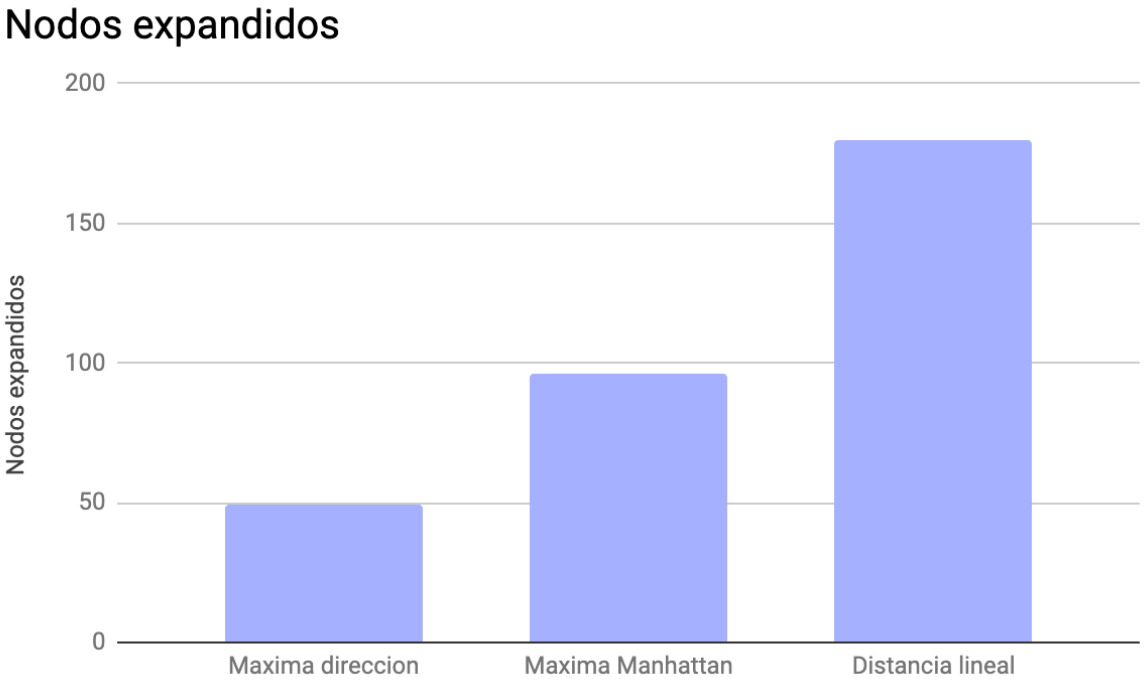

Gráfico 11 : comparación de nodos expandidos de A* con diferentes heurísticas

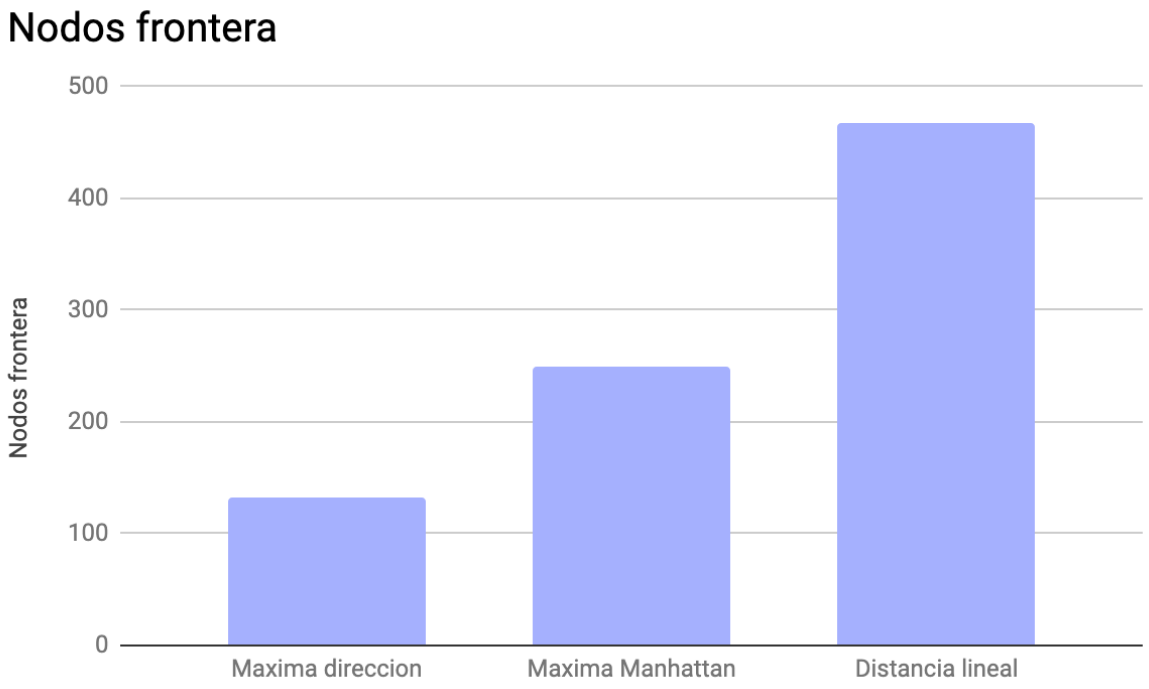

Gráfico 12 : comparación de nodos frontera de A* con diferentes heurísticas