Sistemas de Inteligencia Artificial

Trabajo Práctico Especial 1

Métodos de búsqueda
No informados e Informados

3 de Abril de 2019

Grupo 9

De Rienzo, Constanza - 56659
Godfrid, Juan - 56609
Osimani, Agustina - 57526
Radnic, Pablo Ignacio - 57013

Índice

Índice **2**

Introducción **3**

Simple Squares **3**

Clasificación 3

Representación 3

Reglas 4

Método de Input 4

Heurísticas 5

Problemas Encontrados y Decisiones de Implementación **5**

Ramas infinitas en el árbol de nodos 5

Performance IDDFS 5

Modificaciones al GPS **6**

Resultados y Observaciones **6**

Conclusiones **7**

Anexo **8**

JSON 8

Niveles utilizados para las pruebas 8

level_1: 8

level_2: 9

level_3: 9

no_solution: 9

gameaboutsquares_com_level_14: 9

gameaboutsquares_com_level_13: 10

stress_problem: 10

stress_problem_2: 11

Gráficos 12

Introducción

El objetivo del presente trabajo es dual. Por un lado se busca crear un sistema, denominado *Generic Problem Solver (GPS),* el cual dado un problema utilice distintos algoritmos de búsqueda para encontrar una solución. Y por otro lado, se busca modelar el juego asignado, “Simple Squares” y crear heurísticas correspondientes a este.

Simple Squares

El juego consiste en un tablero con una o más piezas (en la forma de cuadrados) de colores. Cada pieza tiene que llegar a un destino fijo (y conocido) pudiendo desplazarse solamente en una única dirección asignada a esta. En cada movimiento, se elige una pieza, la cual se moverá en su dirección asociada. Si en el destino al que se moverá tal pieza se encuentra otra, se desplazará a esta última en el sentido en que se mueve la primer pieza. A su vez, existen lugares del tablero donde se encuentran flechas que cambian la orientación de las piezas en el caso en que estas caigan en ese lugar.
Un estado es solución, si y sólo si, todas las piezas se encuentran en su destino asociado.

Clasificación

El entorno del juego es totalmente observable, ya que el agente tiene acceso a todos los aspectos relevantes de una acción. Es determinista, ya que la regla que se aplica en cada caso determina el siguiente estado. Es secuencial, ya que la decisión actual afecta futuras decisiones. Es estático, ya que el entorno no cambia mientras el agente delibera. Es discreto, ya que las reglas a aplicar son finitas. Es conocido, ya que el agente conoce el resultado de todas las acciones disponibles.

Representación

A la hora de pensar cómo representar internamente el juego se barajaron distintas posibilidades, qué objetos eran necesarios y qué atributos debería tener cada uno de ellos. Finalmente, se optó por representar el tablero como una matriz de objetos llamados Tile con sus coordenadas x e y, para de esta forma poder representar los Cuadrados y los “Changers” como objetos que hereden de este. El objeto Cuadrado posee un color, una dirección y su Tile objetivo. Este objetivo primero se consideró hacer un objeto aparte pero analizando los lugares donde se usaría resultó una mejor solución ponerlo como un atributo del cuadrado. El objeto Changer, la pieza que cambia la dirección de un cuadrado si este cae sobre ella, posee una dirección (la que le asignaría al cuadrado) y un cuadrado para representar el caso donde un cuadrado y un changer se encuentren ubicados en la misma posición. Finalmente, un Estado consiste de este tablero, junto con una lista de Cuadrados y Changers.

Reglas

Las reglas del juego modelado tienen el formato, “Mover pieza *i* en la dirección que esta posea”. Por este motivo, el conjunto de reglas del juego es dependiente en su totalidad del estado del mismo, analizando cuantas piezas movibles hay y creando una regla por cada una de estas piezas.

Método de Input

Al correr el trabajo, se deberá especificar el estado inicial del juego, la estrategia de búsqueda y la heurística correspondiente en caso de que se elija GREEDY o A* como estrategia. La manera de hacerlo es:

$> java -jar gps-1.0.jar -p filename -s strategy -h [heuristic]

-p determines the problem level. 'filename' should be the json with the level representation
-s determines the search strategy, 'strategy' value can be:
    BFS
    DFS
    GREEDY
    IDDFS
    ASTAR
-h determines the heuristic when strategy is GREEDY or ASTAR, 'heuristic' value can be:
    AVG
    MAX
    GOD

Un archivo de nivel debe ser un JSON con las siguientes propiedades:

- Dos enteros positivos, **width** y **height**
- Dos arrays, **squares** y **changers**

Un square es un JSON con las siguientes propiedades:

- Dos enteros positivos **X** e **Y** tales que $x < width \wedge y < height$
- Un string **color** que representa a un color, los colores posibles son “red”, “blue”, “green”, “purple”, “cyan”, “yellow”, “black”. En caso que el string no represente un color conocido por defecto será visualmente blanco.
- Un string **direction** que representa una dirección, puede ser “up”, “down”, “left” y “right”.
- Un **objective**, que es un JSON con dos enteros positivos **X** e **Y** tales que $x < width \wedge y < height$

Un changer es un JSON con las siguientes propiedades:

- Dos enteros positivos **X** e **Y** tales que $x < width \wedge y < height$
- Un string **direction** que representa una dirección, puede ser “up”, “down”, “left” y “right”.

En el anexo en la sección “JSON” hay ejemplos de niveles posible.

Heurísticas

Para los algoritmos Greedy y A* se implementaron 3 heurísticas diferentes:

- **MAX:** El máximo entre todos los caminos desde cada pieza a su objetivo
- **AVG:** El promedio entre todos los caminos desde cada pieza a su objetivo
- **SUMA:** Suma de máxima distancia horizontal positiva, máxima distancia horizontal negativa (en módulo), máxima distancia vertical positiva y máxima distancia vertical negativa (en módulo) de cada pieza a su objetivo.

Como se mostrará en la presentación, todas las heurísticas planteadas son admisibles.
En la sección “Gráficos” en el anexo, se pueden ver distintas estadísticas que se tomaron en cuenta para analizar cuál es la heurística óptima.
Los resultados fueron concluyentes en que SUMA fue la heurística más performante. Esta es la que mejor se atiene al concepto de no sobreestimar manteniendo el estimado más cercano posible al real debido a que no minimiza el problema a un solo cuadrado quedándose con el camino máximo o el promedio de los caminos.

Problemas Encontrados y Decisiones de Implementación

Durante la implementación surgieron varias problemáticas a resolver y decisiones a tomar que se plantean a continuación.

Ramas infinitas en el árbol de nodos

Se encontraron niveles particulares en los que no se llegaba a la solución debido a la existencia de ramas infinitas. Para dar un ejemplo, puede verse que en gameaboutsquares_com_level_14 (Sección Niveles utilizados para las pruebas en Anexo), si se moviera dos veces la pieza azul para abajo y luego la violeta para arriba, se llegaría a un estado repetido del cual se puede continuar con la misma secuencia infinitamente.
Para resolver esto, se guarda una referencia de los estados analizados en bestCosts para no volver a explorarlos si ya fueron alcanzados.

Performance IDDFS

En el caso en que un problema no tenga solución, en el algoritmo IDDFS debe validarse que se hayan recorrido todos los nodos del árbol para concluir esto.
En una primera instancia, esto se resolvía verificando que todos los nodos frontera estuvieran contenidos en los nodos visitados.
Luego, se modificó para que valide si la cantidad de nodos recorridos en la iteración actual coincide con la iteración pasada. Esto supuso una mejora en la complejidad temporal.

Modificaciones al GPS

Para realizar la implementación del problema se debieron adaptar algunos aspectos del motor dado por la cátedra.
En primer lugar, las reglas del problema son estáticas a lo largo del algoritmo, pero dependen de las piezas (incluidas en los estados). Para esto, a la hora de inicializar las reglas del problema se les pasa el estado. Notar que al estado se lo utiliza para obtener las piezas y generar las reglas de la forma “Mover la pieza i en la dirección j” y no para realizar especulaciones en cuanto a variables particulares de dicho estados.

Además, se le añadió un atributo *depth* a la Clase GPSNode, permitiendo que las instancias de los nodos almacenen su profundidad para recuperar fácilmente la profundidad de la solución y máxima profundidad recorrida.

Por otro lado, el GPSEngine sufrió algunas modificaciones, entre ellas:

- Se agregó un caso excepcional en el método findSolution() que ejecuta el algoritmo de busqueda IDDFS si es éste el recibido por parámetro.
- Se modificó el tipo de valor del mapa bestCosts. Previamente se almacenaba únicamente el costo del mejor camino al nodo llave, modificamos el mapa para que almacene también la profundidad del camino.
    - Esta modificación nos permite en IDDFS realizar comparaciones de profunidad de los nodos ya visitados y garantizar que la solución se encuentre en la menor profundidad posible.

Resultados y Observaciones

En el documento adjunto *‘Resultados’* se encuentran las evaluaciones de las métricas solicitadas para cada uno de los casos detallados en la sección *Niveles utilizados para las pruebas’.* Los algoritmos informados utilizaron la heurística AVG durante las pruebas.

En un principio el equipo comenzó a evaluar los algoritmos utilizando diseños de niveles recogidos al azar en internet para alimentar la biblioteca de niveles de prueba del repositorio. (*ver grilla 2 Resultados Iniciales*)

Estos alcanzaron para hacer las siguientes observaciones:

- Todos los algoritmos solucionan los problemas
- El algoritmo A* consistentemente consigue la solución de menor costo/profundidad.
- IDDFS alcanza soluciones equivalentes a las de BFS (en mayor tiempo).
- No se observan diferencias marcadas de performance entre los algoritmos.
- IDDFS es el algoritmo que más nodos expande y analiza.

Luego el equipo diseñó niveles de estrés (niveles con soluciones profundas pero fácilmente observables por un humano), con el propósito de que se pudiera observar mejor las diferencias de costos en memoria y en tiempo de los algoritmos: (*ver grilla 3 Resultados de Estrés*)

Estos alcanzaron para hacer las siguientes observaciones:

- Todos los algoritmos solucionan los problemas
- IDDFS alcanza las soluciones en tiempos mucho mayores que los otros algoritmos.
- El nivel *Stress_Problem* resultó configurar un caso en el que el algoritmo DFS alcanza la solución de manera más rápida incluso que los algoritmos informados.
- IDDFS es el algoritmo que más nodos expande y analiza.

Finalmente el equipo diseñó el nivel *dfs_worst_case* (nivel con solución de poca profundidad, fácilmente observable por un humano pero cuyo árbol de estados ramifica rápidamente), con el propósito de resaltar las debilidades del algoritmo DFS frente a los demás: (*ver grilla 4 Resultados DFS_Worst_Case*)

Éste alcanzó para hacer las siguientes observaciones:

- Todos los algoritmos salvo DFS solucionan el problema
- En una computadora del equipo luego de 22 minutos de procesamiento el algoritmo DFS lanza un OutOfMemoryException antes de encontrar la solución.
- BFS es el algoritmo que más nodos expande y analiza.

Conclusiones

Si bien se observaron casos donde los algoritmos no informados alcanzaron la solución de determinados niveles de forma más veloz que los informados, también se observaron marcadas debilidades de ellos.

- DFS: Es efectivo para esquemas donde la solución es profunda y el grafo de estados presenta relativamente pocas ramificaciones. Es notablemente débil en los esquemas opuestos
- IDDFS: Es efectivo para esquemas donde la solución no es profunda. Es notablemente débil en los opuestos.
- BFS: Es efectivo en todos los esquemas probados. Expande y analiza más nodos de que los algoritmos informados.

Analizando los algoritmos Informados vimos que su desempeño es relativamente similar a excepción de que A* alcanza soluciones de menor costo y también expande y analiza menos nodos.

Observando los resultados generales, en el conjunto de configuraciones de nivel probadas, el equipo decidió que utilizaría el algoritmo A* para solucionar problemas de Simple Squares ya que alcanza las soluciones de manera más rápida y con costo menor.

Anexo

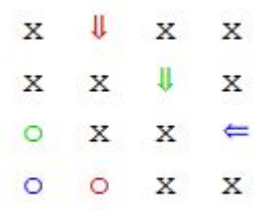
JSON

Ejemplo de nivel:

```
{
  "width": 15, "height":15, "squares": [
      {
            "x": 0, "y": 2, "color": "blue", "direction": "down",
            "objective":
            {
                  "x": 13, "y": 14
            }
      },
      {
            "x": 0, "y": 1, "color": "red", "direction": "right",
            "objective":
            {
                  "x": 14, "y": 14
            }
      }
  ],
  "changers": [
      {
            "x": 13, "y": 2, "direction": "down"
      },
      {
            "x": 0, "y": 14, "direction": "right"
      }
  ]
}
```

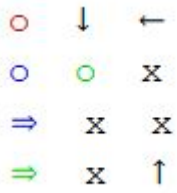
Niveles utilizados para las pruebas

level_1:

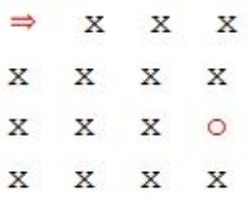


level_2:

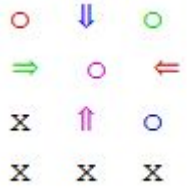
level_3:



no_solution:

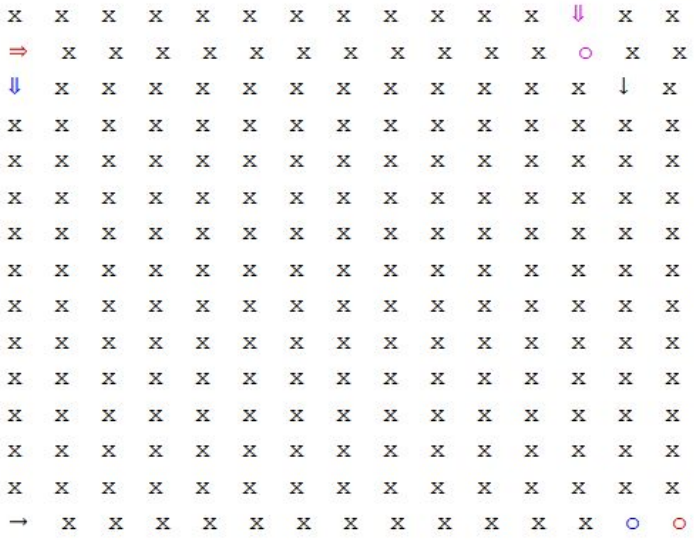


gameaboutsquares_com_level_14:

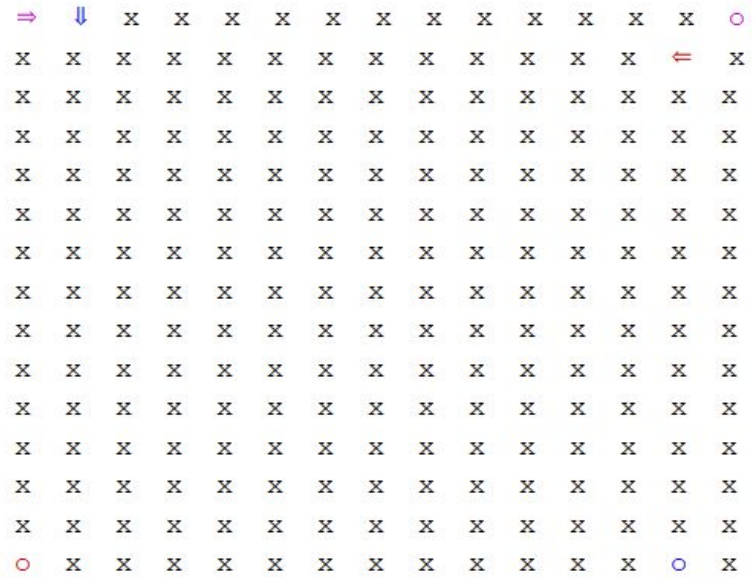


gameaboutsquares_com_level_13:

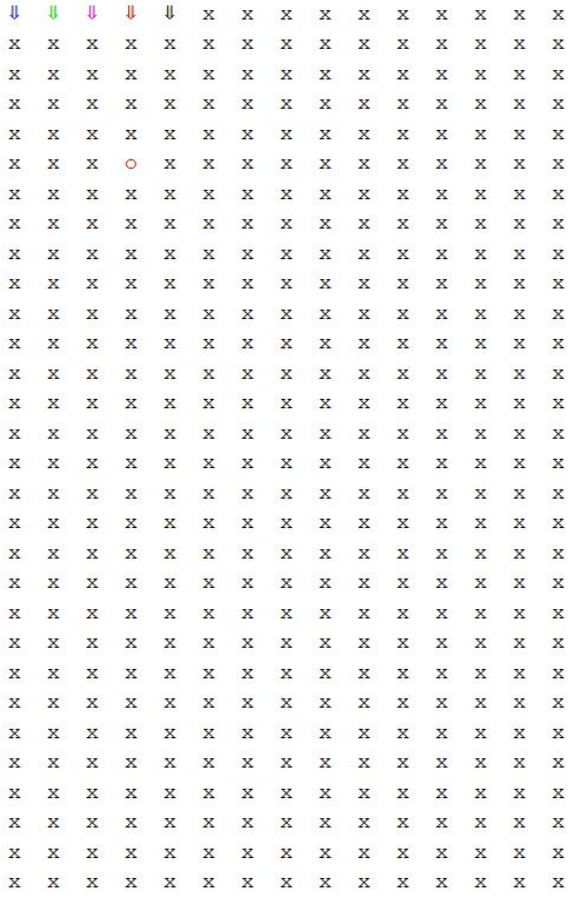
stress_problem:



stress_problem_2:



dfs_worst_case:



Aclaración: Los cuadrados azul, verde, rosa y negro se encuentran posicionados sobre sus objetivos.

Gráficos

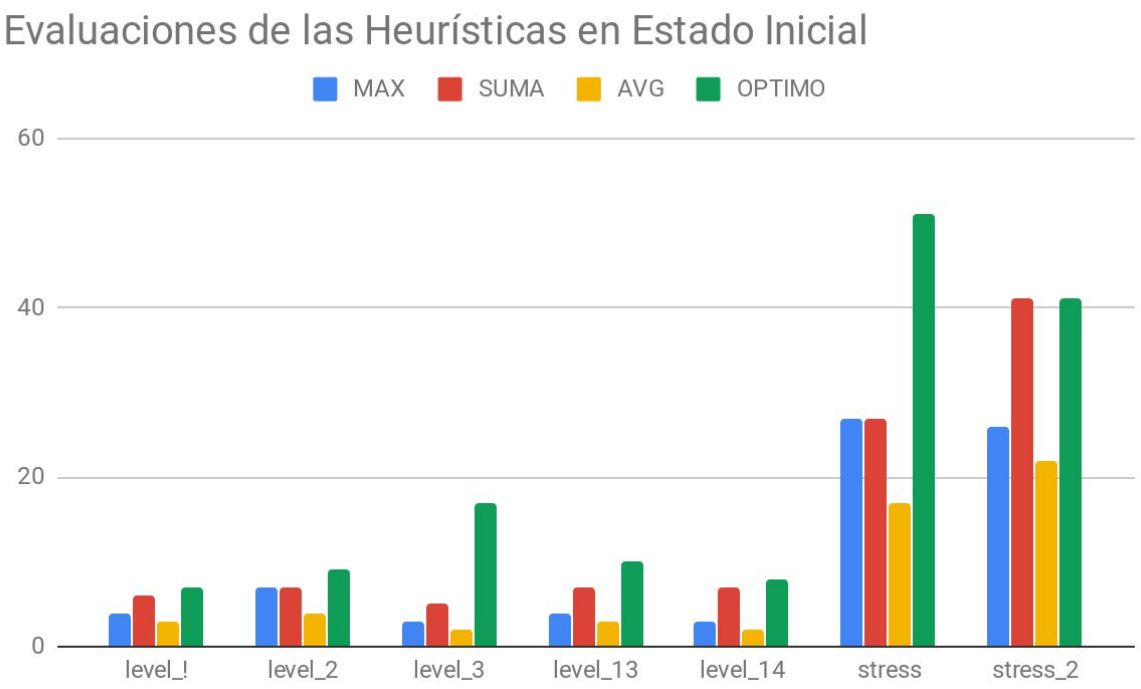


Gráfico 1

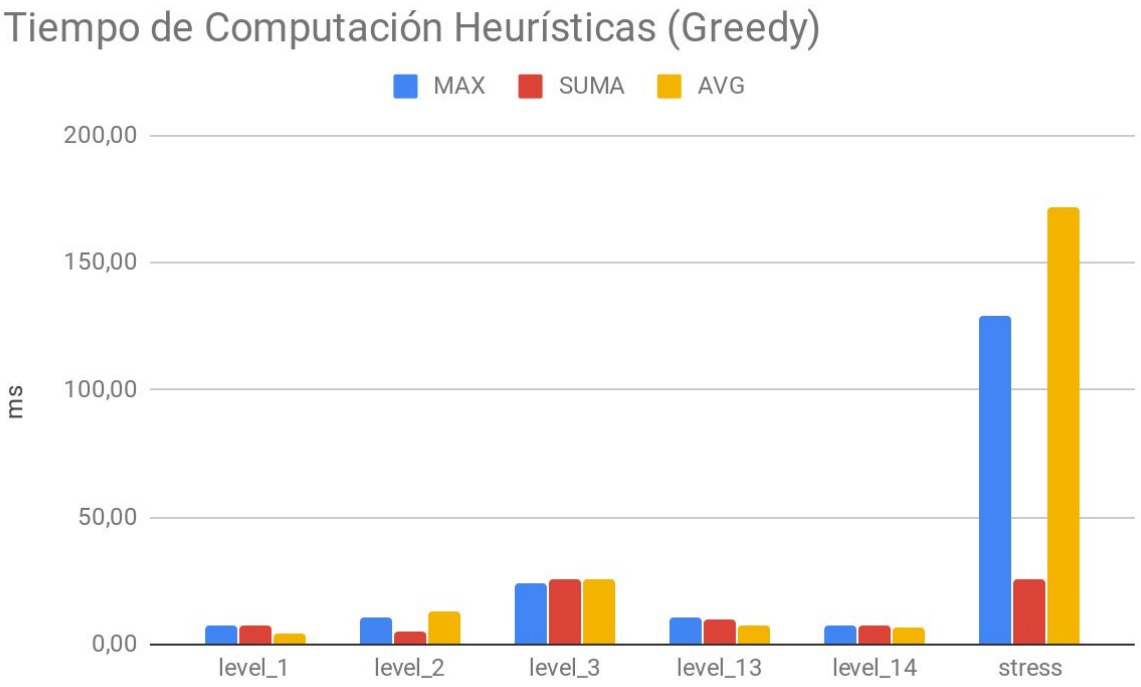


Gráfico 2

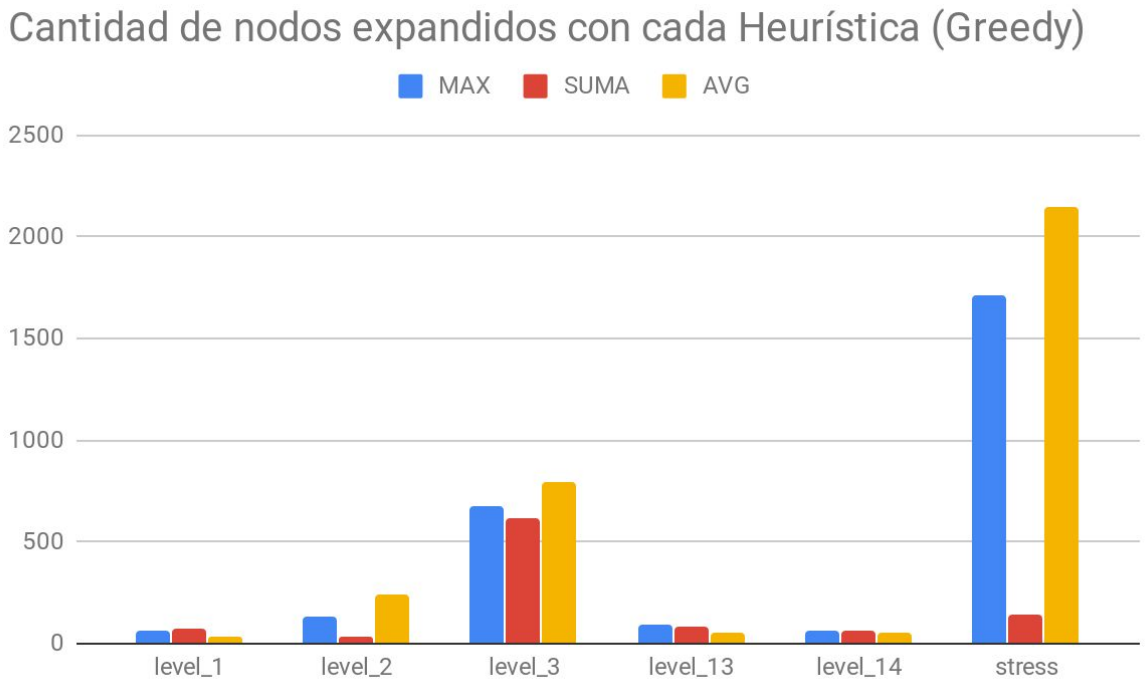


Gráfico 3

En el nivel stress, los valores para las tres heurísticas se dividieron por 3 para que se puedan visualizar todas las barras.
Para calcular los valores de los últimos dos gráficos. se hizo el promedio de 10 muestras ya que el algoritmo Greedy tiene un componente aleatorio al elegir el estado a analizar cuando hay más de uno con el mismo valor obtenido por la heurística.