Sistemas de Inteligencia Artificial

Métodos de Búsqueda No Informados e Informados

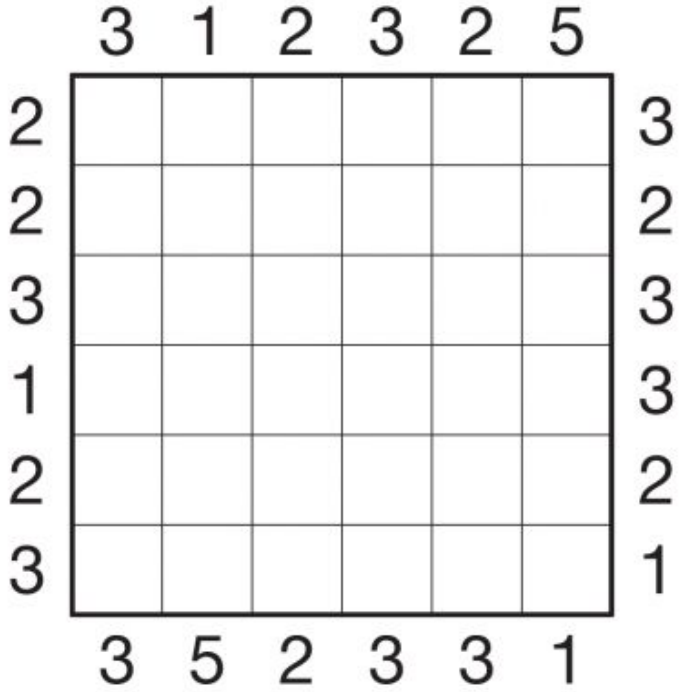
Grupo 1

Problema

Edificios

- Tablero de N x N
- Completar los casilleros con valores incluidos en [1, N]
  - La cantidad de pisos del edificio
- No pueden existir duplicados en fila y columna
- Restricción de visibilidad
  - La cantidad de edificios que puedo ver desde el borde

Edificios - Ejemplo



Implementación

Visibilidad

- Un vector built-in de tamaño N por cada borde
- Cada valor indica la cantidad de edificios visibles desde su posición y con respecto al borde

Tablero

- Matriz built-in de N x N
- Visibilidad
- Lista con casilleros fijados en el estado inicial

Se genera un nuevo tablero al aplicar cualquier regla

Estado

- Tablero
- Primer posición vacía para cada fila y columna

Problema

- *isGoal*
  - Matriz completa
  - No existan conflictos de duplicidad
  - No existan conflictos de visibilidad
- *getHValue*
  - Admisibles
  - No Admisibles

Reglas - Put

Idea: Colocar en la primera posición vacía del tablero un valor entre 1 y N inclusive

Compuesta por: Posición del tablero (fila y columna) y el valor a insertar

Costo: 1

Cantidad de reglas: N^3

Reglas - Put

Restricciones

- La primera posición vacía en el tablero se corresponda con aquella que compone a la regla
- No existan duplicados en fila o columna
- No existan conflictos de visibilidad

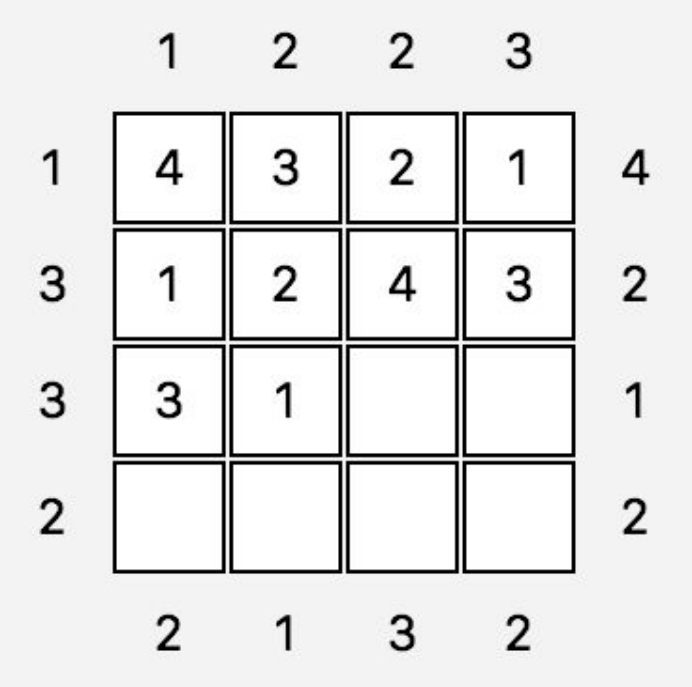
Reglas - Put

Restricciones de visibilidad

- En cualquier posición se revisa que la visibilidad SUPERIOR e IZQUIERDA sea válida

La cantidad de edificios que se pueden ver desde cada borde debe ser menor o igual al límite impuesto por el mismo

Reglas - Put



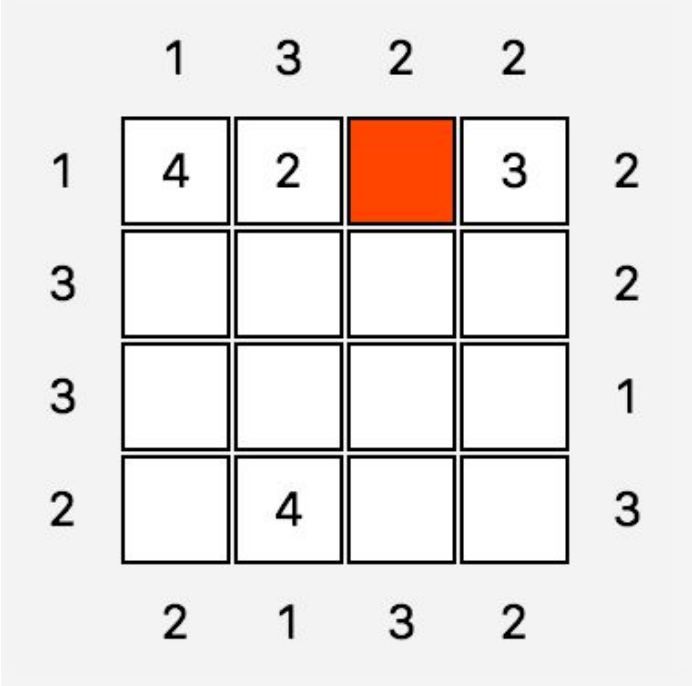
Reglas - Put

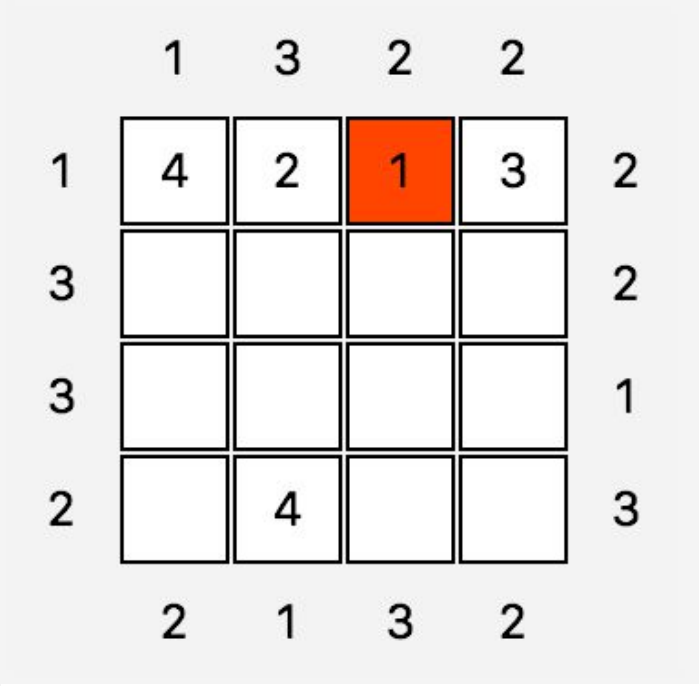
Restricciones de visibilidad (cont.)

- Si es la última posición vacía de una fila se revisa que la visibilidad IZQUIERDA y DERECHA sea exacta

La cantidad de edificios que se pueden ver desde cada borde debe ser igual al límite impuesto por el mismo

Reglas - Put





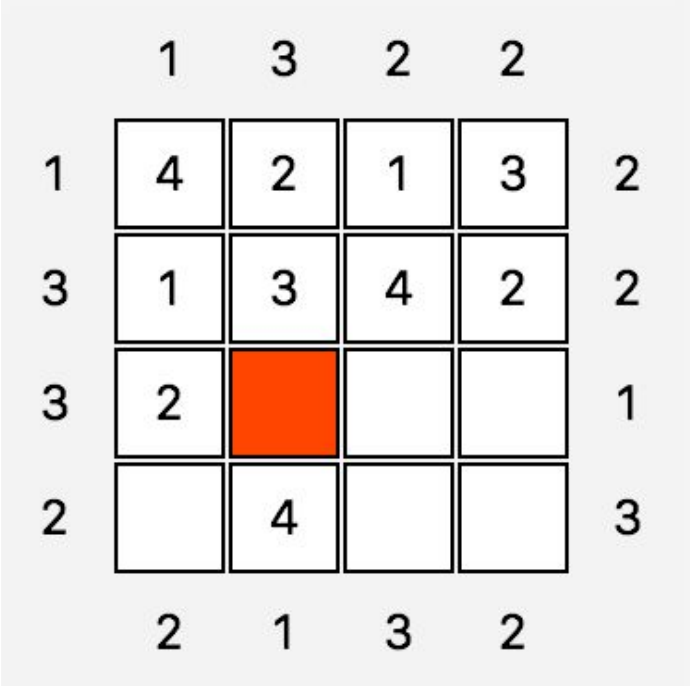
Reglas - Put

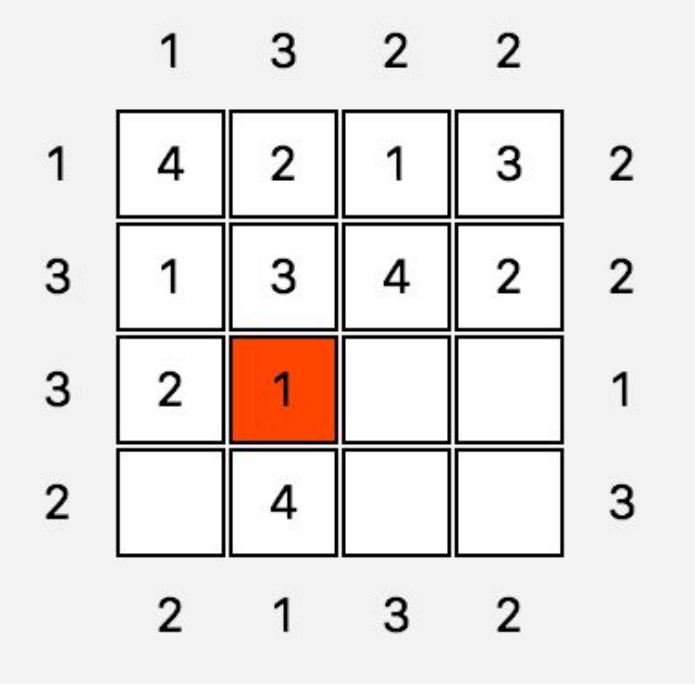
Restricciones de visibilidad (cont.)

- Si es la última posición vacía de una columna se revisa que la visibilidad SUPERIOR e INFERIOR sea exacta

La cantidad de edificios que se pueden ver desde cada borde debe ser igual al límite impuesto por el mismo

Reglas - Put


1 3 2 2
1 4 2 1 3 2
3 1 3 4 2 2
3 2 1
2 4 3
2 1 3 2


1 3 2 2
1 4 2 1 3 2
3 1 3 4 2 2
3 2 1 1
2 4 3
2 1 3 2

Reglas - Swap

Idea: Intercambiar los valores de 2 posiciones del tablero

Compuesta por: 2 Posiciones del tablero

Costo: 12

Cantidad de reglas: $\binom{N^2}{2}$

Reglas - Swap

Restricciones

- El tablero debe poseer todas sus posiciones ocupadas

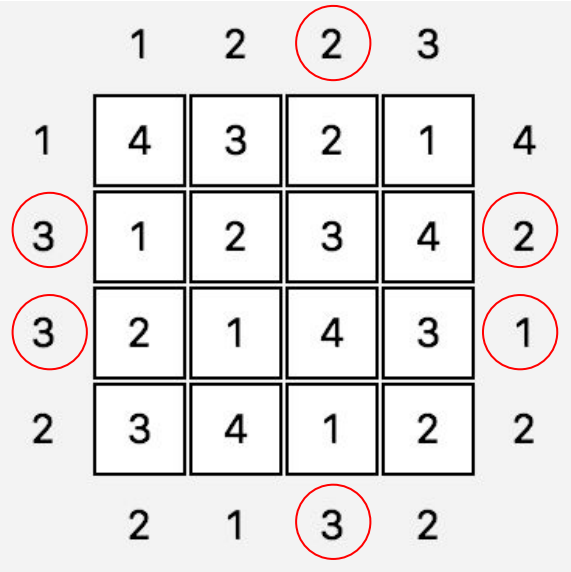
- No se pueden intercambiar valores fijados en el tablero inicial

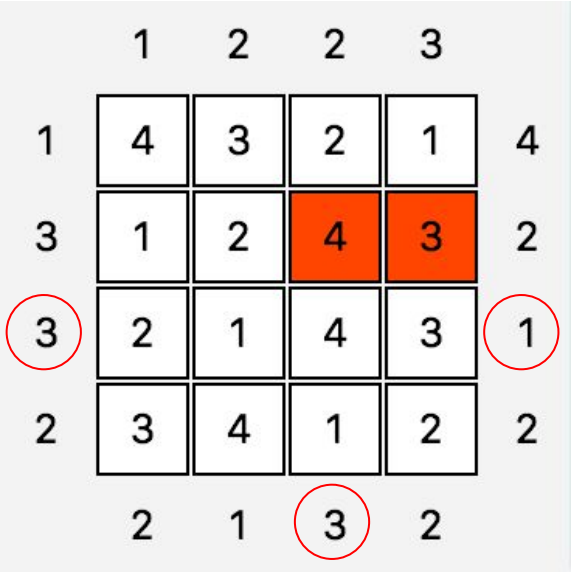
Reglas - Swap

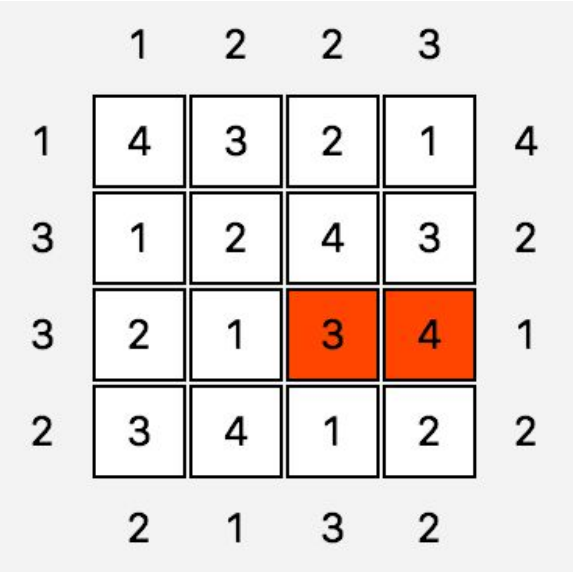
Tablero Inicial

- Se carga el tablero con los valores iniciales fijos
- Para cada posición, se tiene una lista de los posibles valores que puede tomar
- Cuando se inserta un valor, se elimina como posible valor de todos los casilleros de la fila y columna
- Se llena el tablero sin conflictos de duplicidad

Reglas - Swap


1 2 2 3
1 4 3 2 1 4
3 1 2 3 4 2
3 2 1 4 3 1
2 3 4 1 2 2
2 1 3 2


1 2 2 3
1 4 3 2 1 4
3 1 2 4 3 2
3 2 1 4 3 1
2 3 4 1 2 2
2 1 3 2


1 2 2 3
1 4 3 2 1 4
3 1 2 4 3 2
3 2 1 3 4 1
2 3 4 1 2 2
2 1 3 2

Heurísticas

- h sólo para regla Swap
  - Profundidad conocida para regla Put
  - Heurísticas (no h) para regla Put

Heurísticas

- $h^*(n_0) = g^*(n_g) = g(n_g) = \#swaps * costoSwap$
- *h*(n) = # swaps para solucionar todos los conflictos * costoSwap*

⇒ h* desconocida

Heurística - Admisible

- Buscar h’ /
  - $h'(n) \leq h^*(n)$

- *# ideal swaps para solucionar todos los conflictos ≤ # swaps para solucionar todos los conflictos*

Heurística - Admisible

- Cantidad máxima de conflictos en un swap:
  - Repetidos
    - Filas: 2
    - Columnas: 2
  - Visibilidad
    - Filas: 4
    - Columnas: 4
- #MCRS = 12

 # ideal swaps para solucionar todos los conflictos = ⌈#conflictos / #MCRS⌉

Heurística - Admisible

*h’(n) = # ideal swaps para solucionar todos los conflictos * costoSwap*

h(n) = # swaps para solucionar todos los conflictos * costoSwap*

$$h'(n) \leq h^*(n)$$

Heurística - Admisible

*h(n) = #conflictos / #MCRS * costoSwap ≤*

*⌈#conflictos / #MCRS⌉ * costoSwap = h'(n)*

- *Si costoSwap = #MCRS ⇒ h(n) = #conflictos*

⁂ *h(n) ≤ h'(n)*

⁂ *Por transitividad, h(n) ≤ h'(n) ≤ h*(n)*

⇒ h es admisible

Heurística - No Admisible

Experimento: Comprobación estadística de #conflictos que se resuelven por swap

Resultados:

- Primeros pasos aprox. hasta x conflictos
- Se estanca h/ hallar solución

Decisión: #conflictos resueltos por swap = 1

Heurística - No Admisible

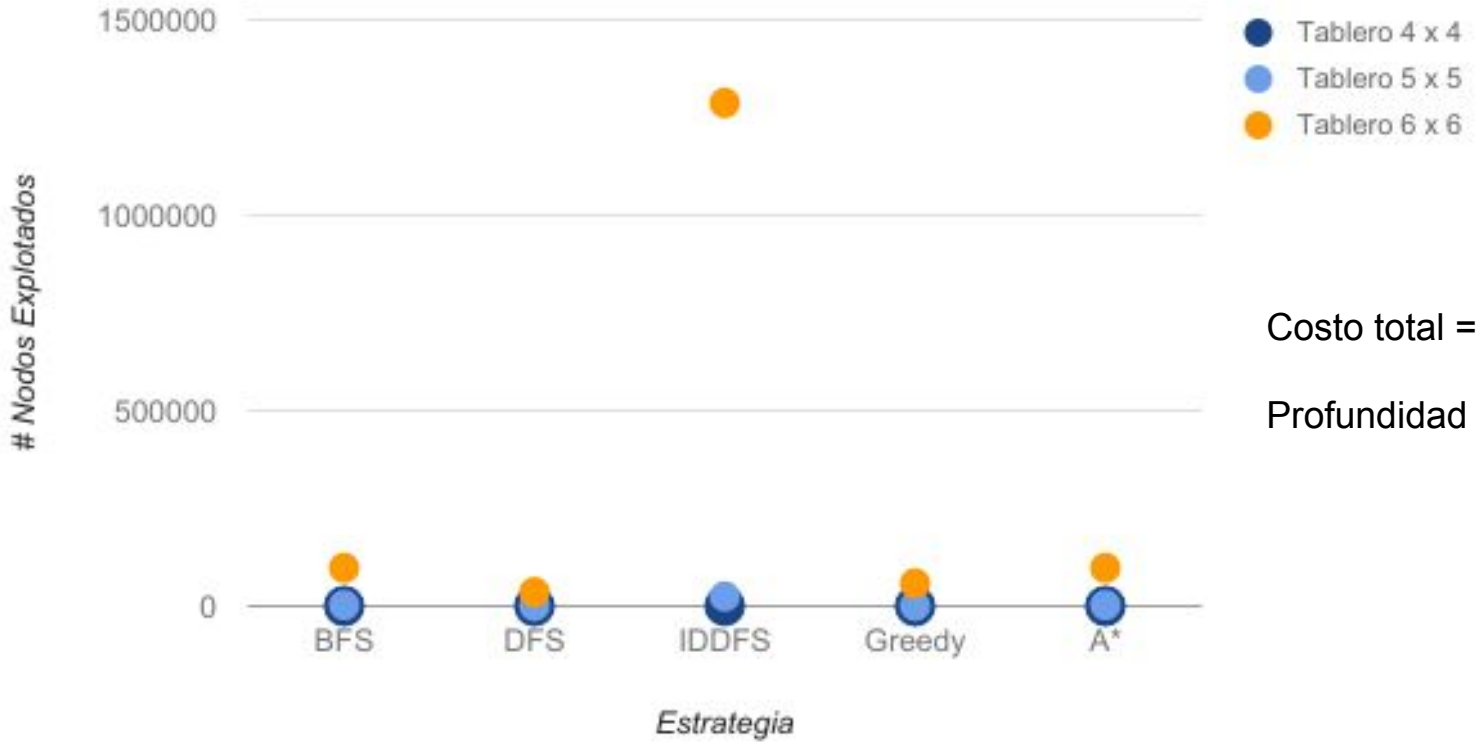
*h(n) = #conflictos * costoSwap*

¿Por qué no es admisible?

- Caso real en que 1 swap resuelve 12 conflictos y el problema es resuelto

Resultados

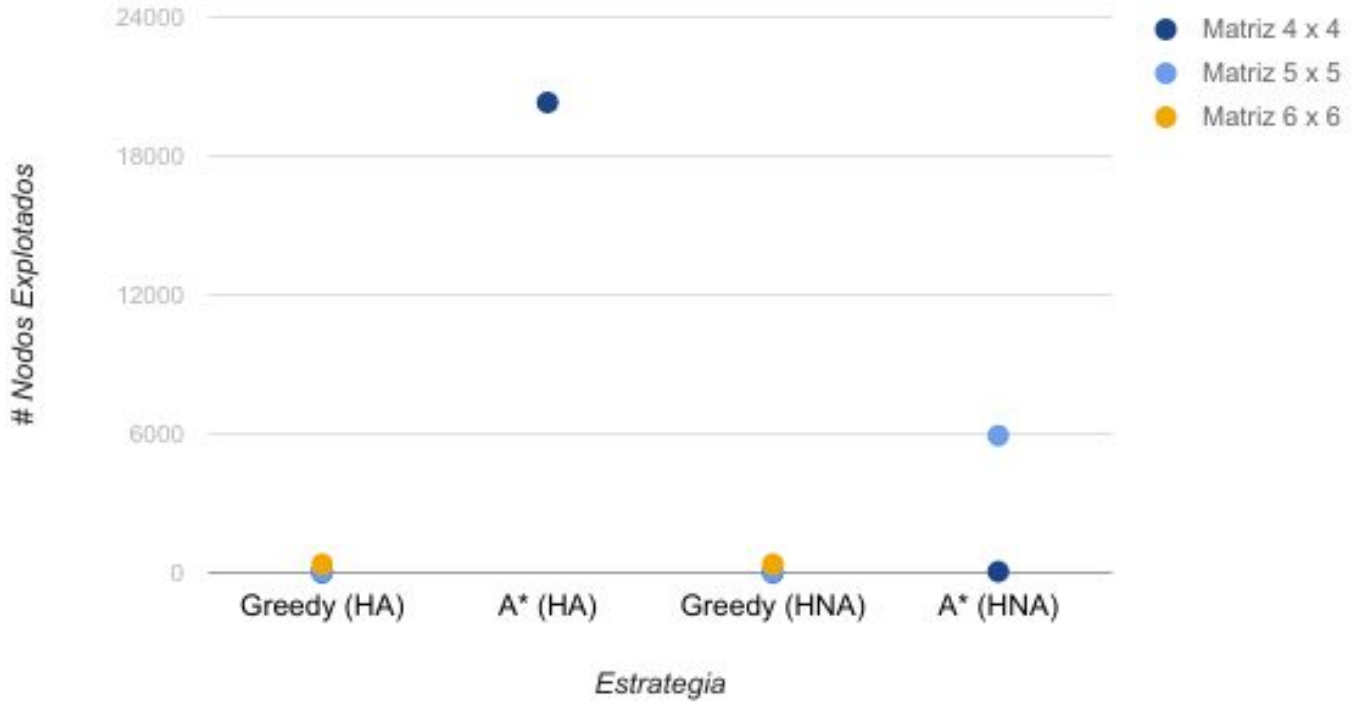
Resultados - Put



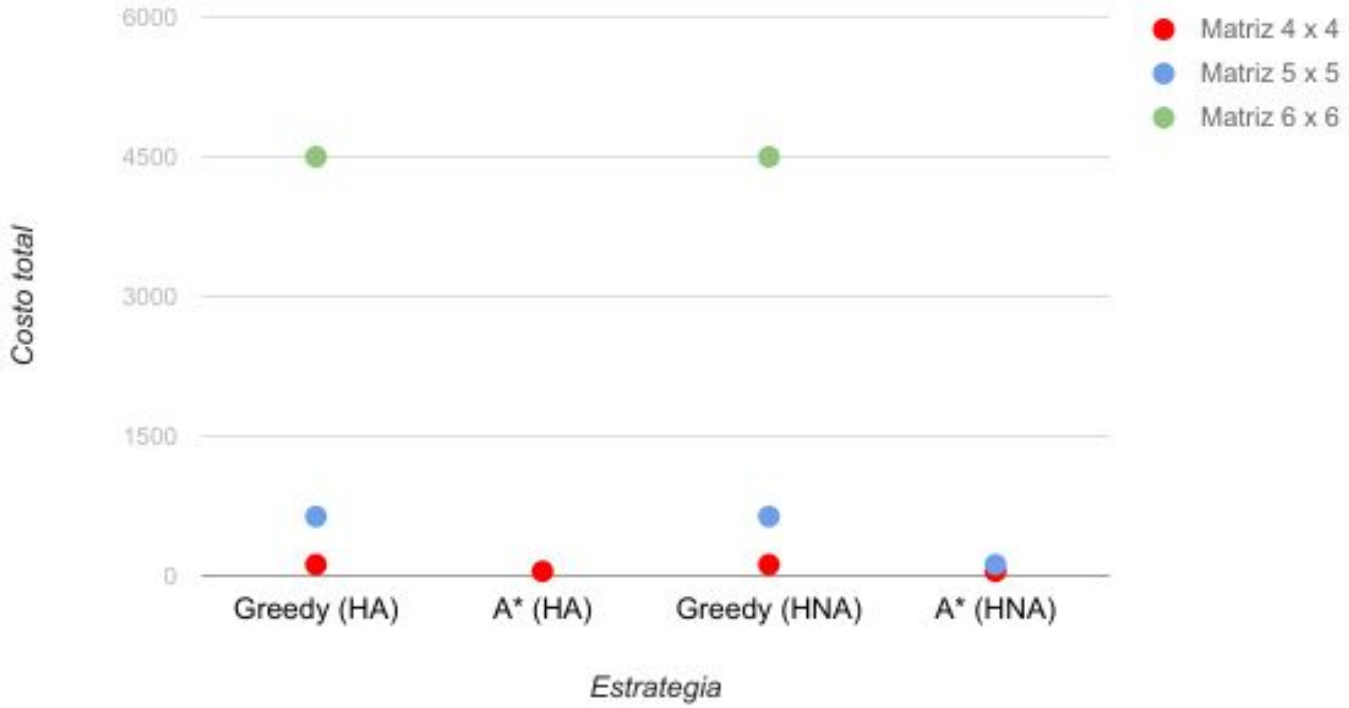
Costo total = #espacios vacíos

Profundidad = Costo Total

Resultados - Swap



Resultados - Swap



Conclusiones

Conclusiones

- 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.

Conclusiones

- 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.

Conclusiones

- 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.