Sistemas de Inteligencia Artificial

Métodos de búsqueda no informados e informados

Trabajo Práctico Especial Número 1

Grupo 7

Alumnos:

Argañaraz, Matías - Legajo: 51340

Kenny, Kevin - Legajo: 49262

Videla, Máximo - Legajo: 51071

# 

# Índice:

[Enunciado](#h.hpqc5620skzu)

[Objetivo](#h.2b4litfshopz)

[Descripción del trabajo](#h.opc2yx6mhhmb)

[Problema a resolver](#h.gban1kpr7xh5)

[Definición del problema](#h.qlv952aze8x1)

[Representación del problema](#h.wz9gaagpkhyx)

[Estado inicial](#h.rb6m8h39jk8r)

[Condición de solución](#h.3hoethb6ikuq)

[Función de costo](#h.ifzaux20r2le)

[Búsquedas desinformadas](#h.vi6gnbfz33pl)

[Breadth First](#h.ifktv3t0bxpz)

[Depth First](#h.wduecsx6admw)

[Profundización Iterativa](#h.6bdrhevs07ih)

[Enfoque Hibrido](#h.is2bmurb79yi)

[Búsquedas informadas](#h.us8eaaks9hil)

[Heuristicas](#h.d3de2f96xs82)

[Heurística 1: “Bloques completados”](#h.i49z3b2f8ibx)

[Heurística 2: “Filas y columnas completadas”](#h.1v24xdlxhmib)

[Heurística 3: “Bloques, filas y columnas completadas”](#h.a6zo3xuxf9nk)

[Greedy Search](#h.cdf3a7kmtshn)

[A\*](#h.pkqznxhzfzl)

[Resultados](#h.bqy25epwf9z7)

[Conclusión](#h.bww3yjgliyhx)

# 

# 

# 

# Enunciado

## Objetivo

Se debe crear un Sistema de Producción que será usado para resolver el problema asignado a cada grupo.

Se les entregará un motor de inferencia reducido programado en Java. Cada grupo puede decidir entre

utilizar este motor y hacerle las modificaciones que sean necesarias para completar el trabajo o realizar dicho motor en el lenguaje que deseen.

## Descripción del trabajo

Se deberá realizar lo siguiente:

* Implementación de las estrategias de búsqueda no informadas: depth first, breadth first y profundización iterativa.
* Implementación de las estrategias de búsqueda informadas: greedy search y A\*.
* **Heurísticas**. Presentación de al menos dos (2) heurísticas. Las heurísticas deben ser no triviales. Enumerar las diferencias entre ellas. Todas las heurísticas presentadas deben estar codificadas y se debe poder ejecutar la búsqueda utilizando cualquiera de ellas. Especificar si son o no admisibles.
* **Función de costo**. Presentación de la/las funciones de costo creadas. Si hay más de una, enumerar las diferencias entre ellas. Todas las funciones de costo presentadas deben estar codificadas y se debe poder ejecutar la búsqueda utilizando cualquiera de ellas.
* Tanto las estrategias de búsqueda como las funciones heurísticas deberán ser entrada del programa a fin de poder intercambiarlas sin necesidad de recompilar el código fuente.
* De cada corrida se deberá analizar al menos: la profundidad de la solución, cantidad total de estados generados, número de nodos frontera, número de nodos expandidos y tiempo de procesamiento.

## Problema a resolver

CalcuDoku

Cada puzzle consiste de una grilla con bloques. El objetivo es completar todos los espacios vacíos para que los números del 1 a N (siendo N la cantidad de las o columnas en la grilla), aparezcan exactamente una vez en cada la y columna. Además, los números en cada bloque deberá producir el resultado que se muestra en el bloque, aplicando la operación matemática asignada a la grilla.

URL: <http://www.conceptispuzzles.com/index.aspx?uri=puzzle/calcudoku>

# 

# Definición del problema

## Representación del problema

El problema consiste básicamente en un tablero, una heurística y M (la cantidad de celdas vacías que serán elegidas para aplicar reglas). Este tablero contiene celdas, las cuales son agrupadas entre sí en diferentes grupos de cuatro tipos:

* Suma
* Resta
* Multiplicación
* División

Cada celda contiene un número y el grupo al que pertenece. Cada grupo, a su vez, contiene las celdas que pertenecen a él, la operación que tiene atribuida y el resultado de dicha operación. Para simplicidad, cada grupo es capaz de indicar que números acepta y, además, de reconocer si se encuentra completo y de forma correcta.

En el caso de los estados, se opta por utilizar una matriz de enteros en representación de las celdas, ya que copiar todo el tablero cada vez resulta muy ineficiente. De esta manera, basta con que cada estado tenga una referencia al tablero y lo actualice cada vez que se llegue al mismo.

## Estado inicial

El juego arranca con el tablero vacío.

## Condición de solución

El juego finaliza por condición y no por comparación de estado. La condición de finalización en éste caso es que el tablero esté lleno y se cumplan las siguientes condiciones:

1. No se repitan números dentro de una misma columna.
2. No se repitan números dentro de una misma fila.
3. La operación correspondiente (según indique el tablero: +, -, /, \*) entre los números de cada grupo debe dar el resultado indicado por el tablero.

**Reglas**

El juego consiste en llenar cada casillero con un número entero. El mismo debe ser mayor a cero y menor o igual a la cantidad de filas/columnas del tablero (el tablero siempre es cuadrado). Por lo tanto a priori las reglas eran:

* Poner 1 en alguna celda vacía.
* Poner 2 en alguna celda vacia.
* …
* Poner ***N*** en alguna celda vacia.

Siendo ***N*** la cantidad de filas/columnas del tablero.

Debido a que al aplicar esta cantidad de reglas en cada nodo, el crecimiento exponencial de la ramificación nivel a nivel es muy grande, decidimos acotarlas. Las redefinimos como:

* Poner 1 en alguna de las primeras ***M*** celdas vacías.
* Poner 2 en alguna de las primeras ***M*** celdas vacías.
* …
* Poner ***N*** en alguna de las primeras *M* celdas vacías.

***M*** es un valor fijo, elegido al momento de correr el programa.

## Función de costo

Dado que cuesta lo mismo poner cualquier número en cualquiera de las celdas, el costo es uniforme, y lo definimos como 1. En definitiva, todos los nodos en la misma profundidad tendrán el mismo costo, es decir, todos los nodos con la misma cantidad de celdas ocupadas tendrán el mismo costo.

# Búsquedas desinformadas

Dado que estos algoritmos no utilizan ningún tipo de heurística para elegir un camino más apropiado, es conveniente que la cantidad de reglas sea menor para minimizar la ramificación. Ésto lo podemos controlar eligiendo un ***M*** chico (preferentemente 1).

## Breadth First

Este algoritmo no es apropiado para este problema en particular, ya que al encontrarse la solución en un nivel fijo, no es necesario expandir todo el árbol gradualmente. Basta con encontrar un camino que lleve a la solución. El nivel en el que alcanza la solución siempre es N; siendo N igual a la cantidad de filas por la cantidad de columnas.

## Depth First

Este algoritmo de búsqueda desinformada es más eficiente que *Breadth First* en este problema en particular, debido a que llegará a la solución expandiendo menos nodos, y la solución encontrada será la óptima, dado que todas las soluciones tienen el mismo costo.

## Profundización Iterativa

Igual que en el caso de *Breadth First* no tiene sentido utilizar *Depth First* con profundización iterativa en este problema, ya que sabemos que la solución será encontrada recién en la última iteración; donde el límite de profundidad sea igual a la cantidad de filas por la cantidad de columnas del tablero.

### Enfoque Híbrido

Esta variación de profundización iterativa también resulta ineficiente en este problema, porque al utilizar Breadth First, se expanden nodos no necesarios para encontrar la solución.

# 

# Búsquedas informadas

## Heurísticas

Al tratarse de un problema de satisfacción de múltiples restricciones, se decide utilizar heurísticas que dependan de la satisfacción de alguna de ellas.

Se proponen 2 heurísticas independientes, y una que resulta de la unión de ambas, listadas a continuación:

### 

### Heurística 1: “Bloques restantes”

Esta heurística es en función de la cantidad de bloques restantes, por lo tanto favorece a completar un grupo, si es una posibilidad.

***h1(n)*** =

### Veamos que *h1* es admisible: Se puede ver que el máximo valor se alcanza cuando la cantidad de bloques no completados es igual a la cantidad total. Cuando este ocurre, h() toma como valor la cantidad de celdas no completadas, que es el valor de *h\*()*, si es que el nodo n pertenece a un camino solución.

Entonces se cumple ***h1(n)*** <= ***h\*(n)***.

### 

### Heurística 2: “Filas y columnas restantes”

Ésta heurística tiene en cuenta la cantidad de filas y columnas restantes, por lo tanto favorece a completar una fila o una columna, si es una posibilidad. Esta heurística es admisible ya que luego de aplicar la regla se sabe que faltarán completar al menos las filas y columnas que no fueron completadas en ese estado. Siendo:

***h2(n)*** =

### La admisibilidad de *h2* es análoga a la de *h1*.

### 

### Heurística 3: “Bloques, filas y columnas restantes”

Esta heurística es una combinación de las dos anteriores. El máximo entre 2 funciones admisibles, es admisible, y además las domina.

***h3(n)*** = max{ ***h1(n)***, ***h2(n)*** }

## 

## Greedy Search

Para este algoritmo es conveniente que ***M*** sea grande, debido a que va a poder elegir la mejor opción entre más nodos hijos. En caso de estar utilizando por ejemplo la Heurística 1, elegirá primero aquel subnodo que complete algún grupo, si es que hay alguno, haciendo mucho más eficiente la búsqueda.

## A\*

Implementar este algoritmo nos permite ver cuán importante es la heurística a utilizar. En un principio la Heurística propuesta era simplemente la cantidad de bloques, columnas o filas restantes, y dado que la función costo es una constante (todos los nodos a misma profundidad tienen mismo costo) la función ***f(n)*** no variaba mucho entre los distintos nodos. Dado que este algoritmo elige el nodo a explotar dependiendo de ***f(n)***, esto hacía que el algoritmo sea muy ineficiente, recorriendo muchos caminos y expandiendo demasiados nodos. Observando malos resultados con este algoritmo se comenzó a analizar la posibilidad de cambiar las heurísticas propuestas, teniendo en cuenta que no aportan demasiada información a algoritmos informados como ***A\**** y ***Greedy Search.*** Ahora, utilizando las nuevas heurísticas, citadas en la sección correspondiente, la eficiencia y performance de estos algoritmos informados se incrementó notoriamente ya que la función ***f(n)*** es mucho más representativa.

# Resultados

Las pruebas realizadas consisten en la ejecución de los distintos algoritmos implementados utilizando tres tableros diferentes (QuadOp3x3, QuadOp4x4 y QuadOp6x6) y cuatro valores para ***M*** (1,2,3 y 5). Además para el caso del algoritmo de profundización iterativa con enfoque híbrido se utiliza una condición de corte para la altura del recorrido BFS de la mitad de las celdas del tablero y las iteraciones DFS se realizan avanzando de a un nivel por vez. En el caso de los algoritmos de búsqueda informados se realizan pruebas con las tres heurísticas.

En todos los casos la búsqueda con profundización iterativa es la peor opción. Esto es lógico ya que la profundidad para la solución es constante y no se llegará al resultado nunca sin recorrer varias veces los estados. Dado que el problema finaliza siempre al mismo nivel, todas las iteraciones de profundidad menor, son un desperdicio de recursos.

El algoritmo BFS resulta ineficiente (sin importar el ***M***) ya que recorre inútilmente todas las posibilidades cuando recién al llegar al final del árbol (que será siempre de la misma altura) obtendrá el resultado.

Todos los algoritmos de búsqueda no informados resultan ineficientes cuando se utiliza un ***M*** > 1 ya que recorren varios caminos para llegar a una solución pudiendo recorrer sólo uno.

En el caso de ***M*** = 1 el algoritmo Greedy se comporta igual que DFS mientras que a medida que se incrementa el ***M***, Greedy mejora y DFS resulta menos eficiente.

Para los casos de ***M*** > 1 los algoritmos de búsqueda informada resultan más convenientes ya que las heurísticas encontradas ayudan a encontrar la solución más rápidamente al verificar la proximidad de varios nodos en simultáneo.

Se encuentran en el anexo los resultados de algunas de las pruebas realizadas.

# 

# Conclusión

Como la solución al problema es alcanzada en una profundidad constante, lo métodos de búsqueda que se expanden a lo ancho son poco eficientes debido a que nunca encontrarán la solución sin antes expandir todos los nodos hasta las hojas del árbol. En otras palabras, el algoritmo BFS resulta muy perjudicado.

En caso de definir una única regla que implique llenar la primer celda vacía, no es tenida en cuenta la heurística utilizada prácticamente ya que no puede comparar esa celda con ninguna otra, es por esto, que en caso de usar algoritmos de búsqueda informada es necesario (para que tenga sentido) utilizar un ***M*** > 1. Sin embargo, al aumentar el ***M***, también aumentan los caminos que conducen a la solución y, en caso de utilizar algoritmos de búsqueda no informada, solamente se evidencia una drástica caída en la performance al realizar varias veces caminos diferentes que si bien son distintos, todos conducen a la solución. Esto se debe a que el orden en el que se ubiquen los números no afecta al resultado.

En el caso de la profundización iterativa, resulta altamente ineficiente ya que nunca llegará a la solución mientras que la profundidad máxima no sea el tablero completo, es decir, se realizarán n\*n - 1 iteraciones sin sentido.

Si la heurística elegida es buena, las búsqueda informadas resultan en general, más eficientes, debido a que además de avanzar en profundidad como un DFS, intentan primero ubicar dentro de las ***M*** celdas, las que completen algún grupo, fila o columna, dependiendo de la heurística elegida. Esto hace muy eficiente al algoritmo ya que la última celda de cada grupo, fila o columna tiene sólo un número posible a ubicar reduciendo las iteraciones y evitando que se pruebe con ese número en la misma fila o columna.

## 

# Anexo

## Pruebas realizadas en las computadoras del ITBA con procesador Intel i7:

El número indicado entre paréntesis es el M elegido.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QuadOp6x6 (1) | Tiempo [ms] | Nodos exp. | Nodos frontera | Estados generados |
| BFS | 212 | 4743 | 0 | 4743 |
| DFS | 196 | 3382 | 7 | 3389 |
| HIDFS (n²/2) | 225 | 5227 | 15 | 5242 |
| Greedy H1 | 233 | 3382 | 7 | 3389 |
| Greedy H2 | 211 | 3382 | 7 | 3389 |
| Greedy H3 | 228 | 3382 | 7 | 3389 |
| AStar H1 | 173 | 1659 | 8 | 1667 |
| AStar H2 | 146 | 1418 | 10 | 1428 |
| AStar H3 | 161 | 1418 | 10 | 1428 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QuadOp6x6 (2) | Tiempo [ms] | Nodos exp. | Nodos frontera | Estados generados |
| BFS | 13198 | 224010 | 0 | 224010 |
| DFS | 9757 | 174500 | 63 | 174563 |
| HIDFS (n²/2) | 13298 | 224679 | 2903 | 227582 |
| Greedy H1 | 12747 | 203483 | 50 | 203533 |
| Greedy H2 | 13204 | 205303 | 52 | 205355 |
| Greedy H3 | 13373 | 206337 | 58 | 206395 |
| AStar H1 | 1807 | 54186 | 23 | 54209 |
| AStar H2 | 1475 | 45922 | 31 | 45953 |
| AStar H3 | 1540 | 45922 | 31 | 45953 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QuadOp4x4 (2) | Tiempo | Nodos exp. | Nodos frontera | Estados generados |
| BFS | 296 | 4977 | 0 | 4977 |
| DFS | 164 | 2022 | 24 | 2046 |
| HIDFS (n²/2) | 347 | 6507 | 194 | 6701 |
| Greedy H1 | 136 | 1013 | 27 | 1040 |
| Greedy H2 | 130 | 1013 | 29 | 1042 |
| Greedy H3 | 140 | 1046 | 28 | 1074 |
| AStar H1 | 214 | 3547 | 10 | 3557 |
| AStar H2 | 209 | 3550 | 10 | 3560 |
| AStar H3 | 214 | 3505 | 13 | 3518 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QuadOp4x4 (3) | Tiempo [ms] | Nodos exp. | Nodos frontera | Estados generados |
| BFS | 1454 | 45474 | 0 | 45474 |
| DFS | 640 | 19537 | 50 | 19587 |
| HIDFS (n²/2) | 1612 | 54881 | 2685 | 57566 |
| Greedy H1 | 810 | 23326 | 49 | 23375 |
| Greedy H2 | 788 | 23409 | 50 | 23459 |
| Greedy H3 | 805 | 23062 | 51 | 23113 |
| AStar H1 | 904 | 28800 | 18 | 28818 |
| AStar H2 | 898 | 28771 | 21 | 28792 |
| AStar H3 | 906 | 28394 | 27 | 28421 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| QuadOp3x3 (5) | Tiempo [ms] | Nodos exp. | Nodos frontera | Estados generados |
| BFS | 206 | 2103 | 0 | 2103 |
| DFS | 112 | 559 | 45 | 604 |
| HIDFS (n²/2) | 82 | 277 | 513 | 790 |
| Greedy H1 | 111 | 471 | 38 | 509 |
| Greedy H2 | 109 | 471 | 40 | 511 |
| Greedy H3 | 111 | 471 | 38 | 509 |
| AStar H1 | 41 | 123 | 27 | 150 |
| AStar H2 | 32 | 92 | 27 | 119 |
| AStar H3 | 25 | 66 | 32 | 98 |