NOMBRE: Diego Fluxá

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructura de Datos y Algoritmos — 2' 2020

Informe Tarea 3

1 Modelación alternativa

Una modelación alternativa a la propuesta en el enunciado podría ser, en vez de definir las variables como todas las celdas vacías, centrarse en aquellas celdas vacías que están al alcance de un nodo azul inicial (con grado asignado). De esta manera, se podrá recorrer una sola vez la matriz del puzzle para encontrar los nodos azules iniciales y luego ir recorriendo estos en el algoritmo de backtracking en lugar de todas las celdas.

Además, al realizar esto, no será necesario que las celdas tomen color rojo, si no que habrá que probar las combinaciones posibles de nodos vecinos (nodos azules conectados al nodo inicial) y, luego de que todos los nodos iniciales alcancen su grado designado (sin ver a través de los grises), simplemente se tendrá que recorrer la matriz y reemplazar todos los 0 (gris) por -1 (rojo) para obtener una solución al problema.

Por último, de esta manera también se podrán ordenar los nodos iniciales en una estructura de datos de manera que el proceso sea más eficiente. Por ejemplo, se podrían ordenar de menor a mayor grado, dado que es a menor grado, será más probable encontrar la combinación de nodos vecinos en una iteración.

Entonces, al modelar de este modo, las variables no serían todas las celdas vacías si no que las combinaciones de celdas alcanzables por los nodos azules iniciales, el dominio sería solo azul dado que no sería necesaria colocar los rojos si no que solo habría que cambiar la combinación, y finalmente las restricciones será que para ninguno de estos nodos iniciales se supere su grado inicial, pues siempre será igual, a menos que se coloque una combinación inválida que afecte a otro nodo.

2 Mejoras

Para esta sección, utilizaremos la modelación propuesta en el enunciado para proponer 2 mejoras a esta. Además, es importante mencionar que los datos empíricos variarán solo en la mejora propuesta, pero aún así poseen otras mejoras al modelo propuesto para apreciar mejor el efecto.

Las mejoras propuestas corresponden a:

1. (PODA) Al momento de validar si un nodo asignado es válido, revisar si dentro de esa columna o fila ocurre que un nodo azul con grado determinado (uno de los nodos azules iniciales del puzzle), es incapaz de llegar a su grafo. Esto ocurrirá cuando un nodo azul "ve" un cantidad de nodos azules y grises menor a su grado. De suceder esto, determinaremos que no es válido y, por lo tanto, se eliminará esta rama del árbol de posibilidades. Esta condición se suma a la condición base que otorga validez a un punto, la cual corresponde a que no exista un nodo en el tablero que pueda "ver" a un número de nodos azules mayor a su grado.

Esta mejora será muy útil para evitar recorrer caminos que sabemos que no llegarán a un resultado factible lo antes posible. Específicamente, en esta poda, se encuentra un nodo que nunca llegará a su grado, por lo que hay que dejar de recorrer esa rama inmediatamente.

A continuación, se mostrará una tabla con la velocidad (promedio de 3 ejecuciones) de dos algoritmos que se diferencian únicamente en la implementación de esta mejora, al ser ejecutados con distintos tests seleccionados para poder observar claramente la diferencia en el desempeño.

Test	Sin mejora	Con mejora
test_1.txt (debug)	0m0.013s	0m0.012s
test_6.txt (debug)	0m0.012s	0m0.013s
test_1.txt (easy)	0m0.015s	0m0.014s
test_6.txt (easy)	0m0.014s	0m0.014s
test_1.txt (normal)	>32m59.320s	0m0.020s
test_6.txt (normal)	7m8.521s	0m0.013s

De esta tabla, podemos notar que para tests fáciles, como los debug y easy, la implementación de esta mejora no provoca gran diferencia en el tiempo de ejecución (Probablemente porque se utilizan heurísticas al comienzo del programa para rellenar aquellos nodos que desde un principio podemos estar seguros de su color y, al ser pequeños, estos tests quedarán casi listos). Sin embargo, para los tests normales podemos ver una gran diferencia en el tiempo de ejecución, llegando a ser más de 65.964,58 veces más rápido en promedio que sin utilizar esta poda (promedio entre tests 1 y 6 normal).

2. (HEURÍSTICA) Nuestra segunda mejora propuesta corresponderá a una heurística, es decir, una "estrategia" para ver que nodo elegir. Esta heurística corresponde a analizar si en el tablero actual existe un nodo que vea tantos nodos azules y grises como su grado. Es decir, que su máximo grado final posible (con el tablero actual) corresponda a su grado asignado. En caso de ocurrir esto, todos los nodos grises que vea tendrán que colorearse azules.

   Esta mejora nos será de utilidad ya que nos evitará recorrer la posibilidad de que estos nodos grises tomen el valor rojo, dado que obligatoriamente serán azules.

   Al igual que para la mejora anterior, se mostrará una tabla con la velocidad (promedio de 3 ejecuciones) de dos algoritmos que se diferencian únicamente en la implementación de esta mejora, al ser ejecutados con distintos tests seleccionados para poder observar claramente la diferencia en el desempeño.

Test	Sin mejora	Con mejora
test_1.txt (hard)	1m17.870s	0m43.429s
test_2.txt (hard)	0m0.108s	0m0.054s
test_3.txt (hard)	1m17.540s	0m17.849s
test_5.txt (hard)	0m0.495s	0m0.117s
test_6.txt (hard)	0m51.188s	0m1.646s

   Aquí, podemos ver que la implementación de esta heurística presenta una clara mejora en la velocidad del algoritmo, yendo desde 1.79 veces más rápido (test_1) hasta 39 veces más rápido (test_6).