IIC2133 - T02

Leonardo Olivares

15/10/18

1 Análisis de Complejidad

El algoritmo de Backtracking utilizado toma como variables las celdas del tablero a resolver. Por cada celda, el dominio abarca tres posibilidades. Estas posibilidades son que la celda contenga un polo positivo, negativo o que se encuentre vacío. Por esta razón, es razonable pensar que mientras mayor sea la cantidad de celdas, mayor es el tiempo que tarda el algoritmo en probar las combinaciones posibles.

A continuación se presenta un gráfico en donde se compara la cantidad de celdas de un tablero (calculado a partir de sus dimensiones) y el tiempo de ejecución que toma el algoritmo en determinar una solución posible (en caso de que tenga).

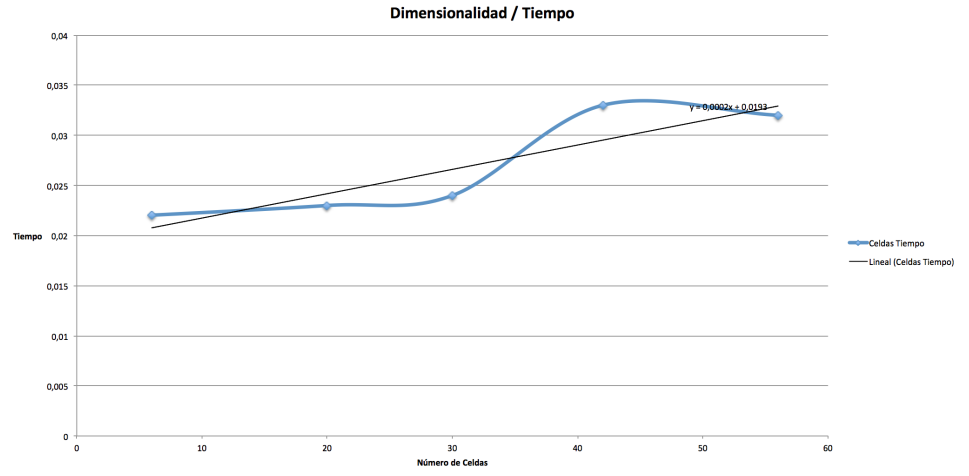

Figure 1: Celdas (Altura*Ancho) vs Tiempo (seg)

A partir del gráfico anterior, se puede observar que efectivamente, a medida que la dimensionalidad del tablero aumenta, mayor es el tiempo que tarda en encontrar una soluci'on. La línea negra que se muestra, es una línea de tendencia, que parece ajustarse a los datos. Por lo tanto, se podría decir que mientras mayor es la dimensionalidad, el tiempo aumenta de forma lineal.

Sin embargo, esta afirmación no es del todo correcta. El tiempo de ejecución del algoritmo no es principalmente afectado por la cantidad de celdas. A pesar de que la dimensionalidad es un factor importante, podemos contar con un tablero de gran tamaño y ninguna restriccin, lo que permitiría que el algoritmo terminara inmediatamente. Esto ocurre debido a que al no tener ninguna restricción de cargas en filas y columnas, no es necesario realizar ninguna asignación de variables. Por lo tanto, mientras mayor sea la cantidad de asignaciones que se deben realizar para llegar a la solución, mayor es el tiempo que debería tardar el algoritmo.

En el mejor de los casos, al resolver el tablero, se realizan la cantidad de asignaciones exactas para llegar a la solución. El problema ocurre cuando el algoritmo se ve en la necesidad de deshacer asignaciones, y

regresar a estados anteriores. Es por esto que, medir la cantidad de veces que se deshacen asignaciones puede ser más relevante, al momento de realizar un análisis de complejidad.

A continuación se presenta un gráfico en donde se compara la cantidad de veces que se deshacen asignaciones en un tablero y el tiempo que tarda en llegar a una solución.

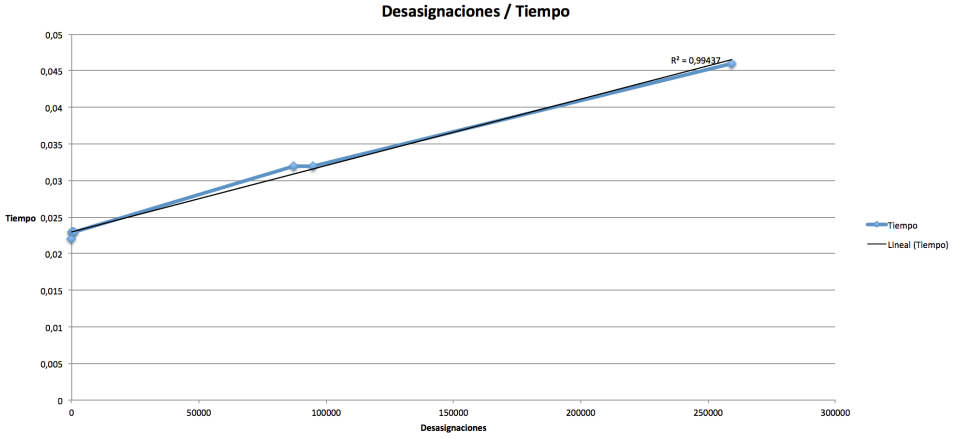

Figure 2: Desasignaciones vs Tiempo (seg)

A partir del gráfico podemos observar los resultados esperados. La cantidad de desasignaciones que se realizan se relaciona linealmente con el tiempo de ejecución del algoritmo de backtracking. La linea de tendencia lineal, nos indica que tiene un R^2 de 0.994, lo cual significa que este comportamiento lineal explica la varianza de los datos de manera bastante cercano a lo esperado.

Podemos concluir que entre la dimensionalidad del tablero y la cantidad de veces que se deshacen asignaciones, esta última es un factor más importante para determinar el tiempo que tardará el algoritmo en culminar. Un algoritmo de backtracking eficiente será aquel que logre reducir significativamente la cantidad de desasignaciones que se realizan, evitando regresar a estados anteriores. Para lograr esto, se pueden realizar distintos tipos de optimizaciones, como cambiar el órden en que se asignan las variables. En la siguiente sección, se procederá a plantear diversas técnicas de optimización para el problema planteado.

2 Optimizaciones del Algoritmo

Para mejorar el desempeño del algoritmo, existen tcnicas de backtracking que permiten reducir la cantidad de permutaciones a probar, y por lo tanto a su vez, reducen la cantidad de desasignaciones que se realizan. A continuación, se mencionan los tres principales métodos de optimización y se plantea una manera de aplicarlos en el problema.

- **Poda**: Una posible poda en este algoritmo (que fue implementada en el código) es revisar que se cumplan los requisitos cada vez que pasamos de una fila a otra. Es decir, al realizar una asignación en la última celda de una fila, se verifica que esa fila tenga los polos positivos y negativos que requiere. En algunos casos, la fila tiene imanes asignados pero no los suficientes, por lo tanto, no tiene sentido continuar asignando en filas posteriores, ya que sabemos que esa fila no cumple todos los requisitos. Esta misma técnica se puede implementar al pasar de una columna a otra. Con esta poda, el algoritmo detecta más rápido que una combinación de variables no funcionará.

- **Heurística**: Una posible heurística para el problema es considerar el órden en que se intentan las asignaciones en cada celda. Por ejemplo, si nos encontramos en una celda, con mayor cantidad de restricciones positivas en su fila y columna, que negativas, sería más adecuado comenzar intentando

asignar un polo positivo en dicha celda. De esta manera, aumentamos las posibilidades de realizar asignaciones correctas desde el inicio, evitando que ocurran algunas desasignaciones.

- **Propagación**: Para implementar la técnica de propagación, una opción sería verificar en cada asignación si la fila o columna cumple las restricciones. En caso de que cumpla, automáticamente se asignan celdas vacías al resto de las celdas en dicha fila o columna. De esta manera, se ahorran algunos pasos, debido a que evitamos realizar una evaluación particular en una celda que tiene solo una posible asignación. Para esta implementación, es necesario guardar un stack con todas las asignaciones que se realizaron en un paso, debido a que si es necesario deshacer asignaciones, en ese paso se debe regresar al estado en que se encontraba el tablero, es decir sin ninguna de las asignaciones que se propagaron.