Proyecto Final Sudoku.

OMAR IBRAHIM MARTINEZ TRIGO

Ingeniería en Datos e Inteligencia Organizacional.

Técnicas Algorítmicas.

Emmanuel Morales Saavedra

1. Justificación de la Técnica seleccionada

De las tres técnicas implementadas, la Programación Dinámica es la más efectiva para resolver el Sudoku. Esto por su capacidad para estructurar el espacio de búsqueda de manera eficiente, lo que minimiza las decisiones erróneas y permite un enfoque más dirigido hacia la solución.

Su complejidad computacional, aunque sigue siendo dependiente del tablero inicial, es menor en promedio que la de Divide y Vencerás y Voraz. Además, su facilidad de implementación y adaptabilidad a variantes del Sudoku la convierten en una opción versátil y robusta.

2. Un análisis de la complejidad computacional.

El algoritmo de programación dinámica implementado para resolver el Sudoku combina backtracking con una heurística basada en la selección dinámica de la celda con menos posibilidades válidas.

Para cada celda seleccionada, el algoritmo intenta colocar uno de los números válidos en la celda y continúa recursivamente con la siguiente celda. Si una configuración no es válida, retrocede.

Para cada celda vacía en el tablero, el algoritmo genera una lista de números posibles verificando si cada uno de los números del 1 al 9 cumple las reglas del Sudoku.

Con la heurística: La complejidad práctica se reduce, ya que elegir la celda con menos posibilidades minimiza las ramificaciones. Sin embargo, la complejidad asintótica sigue siendo en el peor caso.

Cada vez que el algoritmo intenta colocar un número en una celda, verifica si la configuración es válida. Esto implica revisar la fila, columna y subcuadro de la celda:

1. Complejidad por celda: O(27) 9 elementos por fila, columna y subcuadro.
2. Para n celdas vacías, la verificación completa podría realizarse hasta 9^n veces en el peor caso, lo que contribuye al término O(n × 27 × 9^n) en el peor escenario.

- En el peor de los casos: El algoritmo tiene una complejidad aproximadamente O(9^n), donde nnn es el número de celdas vacías. Esto se debe a que, en el peor escenario, cada celda vacía podría requerir probar los 9 valores posibles.
- En el mejor de los casos: La heurística de selección de la celda con menos opciones válidas reduce significativamente las ramificaciones, acortando la ejecución en muchos casos prácticos.

3. Resultados Obtenidos.

```
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Comparación de los tiempos de ejecución:

```
Comparación de Tiempos:
Dibide y Venceras: 849.1656 segundos
Dinámica: 6.0167 segundos
Voraz: 15.3982 segundos
```

Comparaciones con otras técnicas

Técnica de Fuerza bruta: Esta técnica prueba todas las configuraciones posibles del Sudoku hasta encontrar una solución válida, algo parecido al backtraking la diferencia es que el el backtraking prueba las combinaciones y si no funciona retrocede.

Aunque se ve sencillo, el enfoque de fuerza bruta es ineficiente para tableros con muchas celdas vacías, por lo que no halla ninguna manera para reducir las ramificaciones del espacio de búsqueda. En comparación, la programación dinámica y las otras técnicas optimizan este proceso significativamente.

Deep Learning: Utiliza algoritmos de aprendizaje profundo para aprender patrones y resolver Sudokus de la forma más exacta posible.
Esto requiere conjuntos de datos masivos para entrenamiento y son excesivos para resolver un problema definido y estructurado como el Sudoku. Esto no garantiza una solución exacta sin ajustes.

Local Search: Empieza con una solución sencilla y realiza modificaciones locales para mejorarla, por ejemplo, intercambiando números dentro de bloques para cumplir con las restricciones.
Los algoritmos de búsqueda local son los más adecuados para problemas de optimización o para encontrar soluciones aproximadas. En el Sudoku, donde se busca una solución exacta, este enfoque puede quedar atrapado en mínimos locales y no garantizar una buena corrección.

Conclusión

En este análisis implementamos tres técnicas para resolver un Sudoku: Divide y Vencerás, Programación Dinámica, y Algoritmos Voraces. De estas, la Programación Dinámica resultó ser la más efectiva debido a su capacidad para optimizar el espacio de búsqueda al manejar restricciones locales y globales, lo que garantiza soluciones exactas con una complejidad razonable. Aunque Divide y Vencerás mostró una resolución progresiva y estructurada, su mayor consumo de tiempo y recursos lo hizo menos eficiente, mientras que el enfoque voraz, aunque rápido, puede fallar al no considerar el estado global del problema. Este ejercicio destacó la importancia de elegir técnicas adecuadas según las características del problema, equilibrando eficiencia, exactitud y facilidad de implementación.