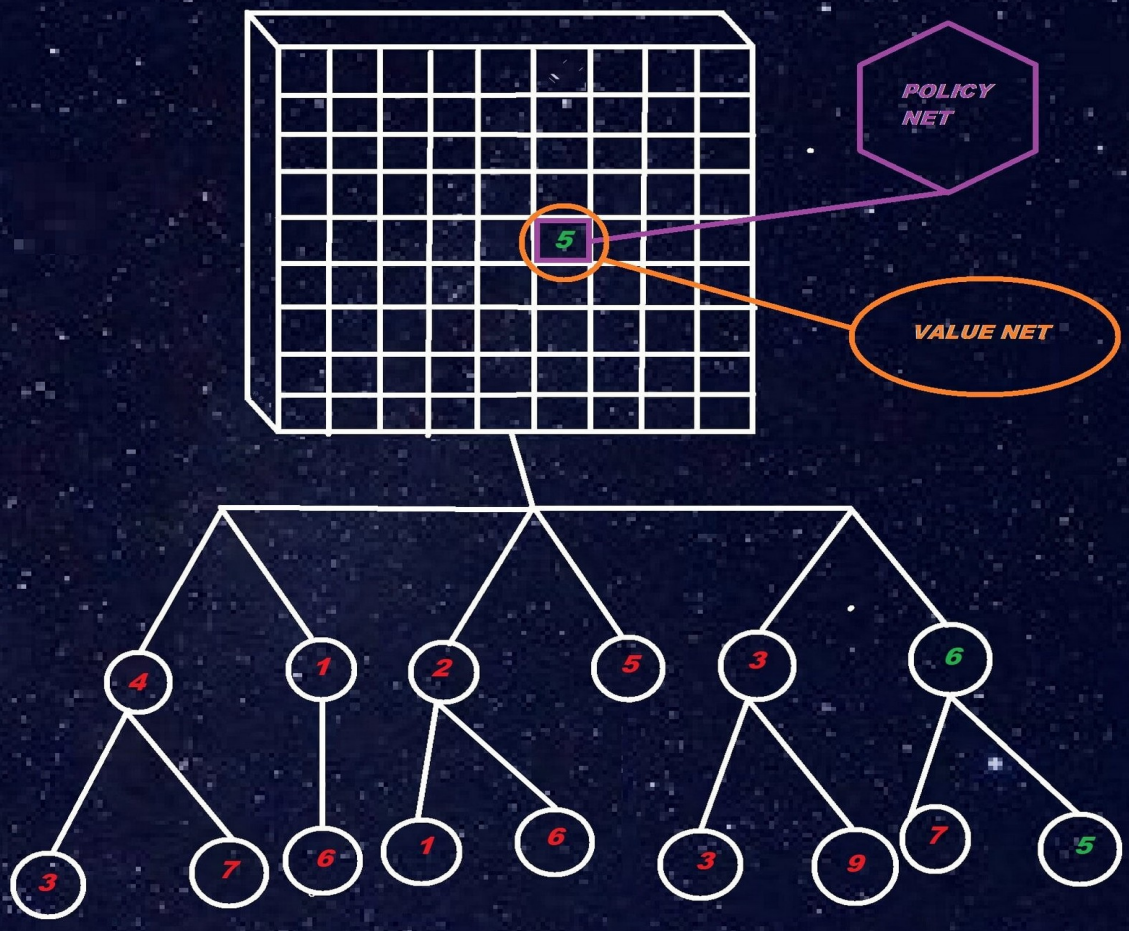
SUDOKU KILLER SOLVER



JAIME CASTILLO UCLÉS

-Prólogo:

A priori, podemos ver como un sudoku killer es mucho más complejo que un sudoku convencional, pues en el sudoku convencional para encontrar una única solución deben darnos muchos más números que ocupan más casillas. No obstante, a pesar de esta dificultad podemos considerar que tener una condición más, como es la suma que deben cumplir los números de ciertas casillas, puede ser algo que juege a nuestro favor. Nos puede ayudar a filtrar una solución más rápido.

En el problema planteado la dificultad central para contruir un algoritmo lo más eficiente posible está en cómo ordenar la información de forma que el programa pueda construir la solución sin necesidad de pararse a procesar demasiadas opciones.

Para la resolución de este problema utilizaremos la estructura de datos, árbol de poda y de ramificación. La justificación del uso de esta estructura, aún sabiendo que no la hemos visto en clase, es que creemos que es la mejor para resolverlo. Nuestra inspiración a la hora de crear un algoritmo que resuelva el sudoku killer es imitar la buena forma que un humano puede tener a la hora de resolverlo, con las ventajas de potencia de cálculo que tiene una máquina, por supuesto. Ya que el sudoku killer a pesar de todo es un juego pensado por y para humanos. Bajo nuestro juicio, la estructura de datos árbol, se adapta muy bien a las decisiones de descarte y elección que podría tener un humano al resolverlo.

-Algoritmo básico, TDA Sudoku Killer:

1.Inicialización.

Se implementará una matriz 9x9 que representará el tablero, trabajaremos sobre el.

Se inicializa con unos valores de entrada de las sumas de cada grupo y de casillas, y el valor inicial de algunas casillas si nos la dan.

- Función CuanticValue.
Asigna posibles valores a cada casilla vacía solo teniendo en cuenta el valor de la suma correspondiente.

2. Búsqueda en árbol con ayuda de redes de selección de casilla y del primer valor.

El siguiente paso es buscar ya en nuestra estructura. Para hacerlo de una forma más eficiente usamos dos funciones complementarias.

-Función PolicyNet.
Selecciona por cuales casillas empezar a buscar. Estimando mayor éxito en algunas. Se basa en propiedades aritméticas teniendo en cuenta el número de posibles valores asignado a cada casilla por CuanticValue y las casillas resueltas cerca, de forma que resulte más fácil obtener nuevos números en el árbol.

- Función ValueNet.
Selecciona algunos numeros de los posibles filtrados por CuanticValue antes que otros para realizar la búsqueda en el árbol. Se basa en propiedades aritméticas para estimar sus decisiones teniendo en cuenta las sumas y el número de casillas de cada suma. (A sumas más bajas asignará posibles valores de CuanticValue más bajos para que el árbol empiece por ellos).

-Búsqueda en árbol:
Usamos nuestra estructrura para buscar de entre las combinaciones posibles, empezando por los valores indicados por la función ValueNet y desde las casilla que indique PolicyNet, los resultados solución.

- El arbol.

-Funcionamiento:

Nuestro árbol de poda y ramificación seguirá una estrategia FIFO de exploración en árbol a su vez basandose en celdas enlazadas para gestionar los nodos vivos, es decir los escenarios por los cuales existe y se debe filtrar la solución.

-Propiedades:

Las propiedades que tendrá el árbol son las propiedades básicas y genéricas de un árbol gestionado por un vector de punteros. Podar, injertar, copiar, revisar si contiene un nodo especifico...

Después de toda esta implementación, el objetivo que se persigue es esquivar el mayor número de posibles valores de las casillas asignado por CuanticValue, para reducir la búsqueda en el árbol.

¿Cómo podría funcionar un programa principal que use este TDA?

En primer lugar, se persigue reducir los posibles valores de cada casilla a uno. Para ello se irán quitando nodos del árbol y se pasarán a la función descartando para quitarlos del tablero. Si se encuentra la solución antes de recorrer el árbol entero se usará el método SoluciónEncontrada, una versión hermana de descartando pero que está diseñada para resolver este evento de encontrar la solución.
El programa principal también será el que se encargará de saber cuándo el tablero esté con un solo elemento en cada casilla, es decir, resuelto para devolver la solución.