REPRESENTACIÓN Y SIMPLIFICACIÓN DE SUDOKUS EN PROLOG

Práctica 1: Conocimiento y Razonamiento Automatizado

(Adaptado para la asignatura de Algorítmica y Complejidad, Curso 2024-25)

Marcos Fuster Peña
Cristian Márquez García
Daniel de la Fuente Sánchez
José Manuel Martín Calzada

Resumen

Esta práctica aborda el manejo de listas en Prolog aplicadas a la representación y simplificación de Sudokus. Se desarrolla un modelo basado en listas lineales para gestionar casillas y posibles valores, implementando cuatro reglas iterativas de simplificación que permiten reducir progresivamente las opciones en el tablero. Además, se han incorporado mejoras como una interfaz gráfica interactiva, métricas para evaluar rendimiento, y detección automática de conflictos, logrando así una aplicación robusta y eficiente para explorar técnicas avanzadas de resolución lógica en Prolog.

Contenido

Introducción

El estudio de los sudokus ha sido un caso de interés en el ámbito de la resolución de problemas mediante programación lógica.

Un sudoku consiste en una cuadrícula de 9x9 dividida en subcuadros de 3x3, donde cada celda debe contener un número del 1 al 9 sin que se repita en la misma fila, columna o subcuadro. Para representar este problema en Prolog, se emplea una estructura basada en listas, donde cada casilla del tablero es un elemento de una lista de longitud 81.

En este contexto, la manipulación de listas resulta esencial para modelar el tablero y gestionar la información de cada casilla. La solución de un sudoku puede abordarse mediante la generación de posibles valores para cada casilla vacía y la aplicación de reglas que reduzcan progresivamente el espacio de búsqueda. Estas reglas permiten simplificar el problema eliminando opciones imposibles con base en los valores existentes en filas, columnas y subcuadros.

A medida que las reglas se aplican iterativamente, se produce una reducción en las posibilidades de cada casilla, facilitando la resolución del tablero. El análisis de la interacción entre estas reglas y su impacto en la simplificación del sudoku ofrece una perspectiva interesante sobre la eficiencia de los algoritmos basados en restricciones.

Además, explorar diferentes estrategias de reducción permite comprender mejor cómo la programación lógica puede emplearse para estructurar y resolver problemas de esta naturaleza.

Objetivos

El objetivo principal de esta práctica es desarrollar habilidades en la manipulación de listas en Prolog, utilizando la representación y simplificación de sudokus como caso de estudio. A través de esta implementación, se busca que el estudiante comprenda cómo modelar problemas complejos mediante estructuras lineales y aplicar reglas iterativas para la reducción de posibilidades en cada casilla del tablero.

Los objetivos específicos de la práctica incluyen:

1. **Resolver sudokus mediante un algoritmo en Prolog.**
    a. Modelar un sudoku en forma de lista y mostrar su estado actual por pantalla.
    b. Valorar los posibles valores que puede tomar cada casilla, empleando estos mismos para la resolución del sudoku.
    c. Aplicar reglas lógicas para resolver el Sudoku. Las reglas especificadas y que se emplearán son las siguientes:
        i. **Regla 0**: Asignar valores a casillas con solo una opción disponible.
        ii. **Regla 1**: Identificar números únicos en una fila, columna o subcuadro.
        iii. **Regla 2**: Identificar pares exclusivos y no valorarlos en otras casillas.
        iv. **Regla 3**: Identificar tríos exclusivos y no valorarlos en otras casillas.

2. **Analizar el funcionamiento del programa.**
    a. Generar una seria de sudokus de prueba para medir la capcidad de los algoritmos y las reglas lógicas.

b. Valorar si los sudokus se pueden resolver con las reglas y algoritmos que se tengan y, en su debido caso, valorar si se pueden mejorar o generar nuevas.
c. Captar el estado del sudoku, así como de su lista de posibilidades, iteración por iteración. Pudiendo moverse entre estados.

3. **Implementar mejoras más allá del enunciado descrito.**
   a. Implementar una interfaz gráfica personalizada para utilizar el programa y observar su funcionamiento por un medio más accesible y sencillo de usar.
   b. Exponer métricas para evaluar el rendimiento de cada regla y algoritmo empleado. Realizar medidas de tiempo, de iteraciones, de recursos empleados, etcétera.
   c. Asegurar que el funcionamiento del programa es resiliente y fiable mediante la detección de conflictos y un gestor de excepciones o errores.

Desarrollo.

Descripción de la Representación del Sudoku (Lista de 81 Casillas)

El Sudoku se representa en Prolog mediante una lista de 81 elementos, donde cada casilla del tablero 9x9 corresponde a una posición en la lista. Los números del 1 al 9 indican valores ya definidos, mientras que `'.'` representa casillas vacías. La disposición sigue un orden fila por fila, lo que facilita el acceso a cualquier celda mediante su índice en la lista.

A continuación, se muestra un ejemplo de la representación utilizada en Prolog:

```
sudoku([
    '.','.', 9 ,   6 ,'.','.',  '.', 1 ,'.',
     8 ,'.','.',  '.','.', 1 ,  '.', 9 ,'.',
     7 ,'.','.',  '.','.','.',  '.','.', 8 ,

    '.', 3 ,'.',  '.', 6 ,'.',  '.','.','.',
    '.', 4 ,'.',   1 ,'.', 9 ,  '.','.', 5 ,
     9 ,'.','.',  '.','.','.',  '.','.','.',

    '.', 8 ,'.',   9 ,'.','.',   5 , 4 ,'.',
     6 ,'.','.',   7 , 1 ,'.',  '.','.', 3 ,
    '.','.', 5 ,  '.', 8 , 4 ,  '.','.', 9
]).
```

Este modelo permite acceder a cada casilla mediante su índice en la lista, lo que facilita la implementación de reglas para la generación de posibilidades y la simplificación del Sudoku.

Generación de posibilidades para cada casilla vacía.

La generación de posibilidades consta de una serie de funciones que pertenecen al bloque principal o *main* del código.

El código se encarga de determinar, para cada casilla vacía de un Sudoku, la lista de números (del 1 al 9) que podrían ocuparla, descartando aquellos ya presentes en su fila, columna y cuadrante. Esto se realiza a través de las siguientes etapas:

Función Principal: La función principal es posibles, que, partiendo del Sudoku, genera una lista de posibilidades para cada celda. Para ello, llama a la función presentes que, por cada casilla, reúne los números que ya están en su entorno.

Recorrido y Cálculo de Índices: La función presentes recorre recursivamente las 81 casillas usando un contador. Calcula:

- El **índice de fila** (división entera del contador).
- El **índice de columna** (módulo del contador).
- El índice de cuadrante mediante la fórmula: ((contador // 27) * 3 + (contador mod 9) // 3)
- **Extracción de Números en Unidades:** Se emplean las funciones auxiliares fila, columna y cuadrante para extraer los números presentes en la fila, columna y bloque, respectivamente. Luego, la función unir combina estos números, eliminando duplicados y ordenándolos, con la ayuda de eliminar_repetidos.
- **Generación de Posibilidades para Cada Casilla:** Finalmente, usando maplist junto con la función auxiliar posibles_aux, se generan las listas de números que no están presentes en la unión de la fila, columna y cuadrante, resultando en las posibilidades para cada casilla vacía.

En resumen, el proceso analiza sistemáticamente cada casilla del Sudoku y, a partir de los números ya fijados en sus unidades adyacentes, determina cuáles números podrían ocupar cada casilla vacía.

Aplicación de las reglas de simplificación (Reglas 0, 1, 2 y 3).

Regla 0:

La regla 0 es aquella que escribe los números en su correspondiente casilla cuando dicho número es la única opción posible para la casilla. Por ejemplo, si la única opción válida para una casilla es 1, la regla escribe un 1 en esa casilla.

La regla llama a una regla auxiliar recursiva. Esta emplea un contador, navega por las 81 celdas del sudoku y evalúa la lista de posibilidades de cada posición. Si la lista de posibilidades para una celda tiene un solo valor, escribe ese valor en la misma posición, pero en el sudoku. Realizando una escritura.

El caso base que finaliza la recursividad es aquel en el que el contador llega a 81, indicando que se han recorrido todas las casillas. En este momento se guarda el sudoku resultante de aplicar la regla invirtiendo un acumulador que va guardando el sudoku según se modifica.

Regla 1:

La Regla 1 se basa en el principio de que, si en una unidad del Sudoku (fila, columna o bloque) un número aparece como posibilidad en una única casilla, ese número debe fijarse en esa celda y eliminarse de las posibilidades de las demás celdas de la misma unidad.

Flujo General: La función principal regla1(P, NewP) muestra el estado inicial, procesa todas las unidades y presenta el estado actualizado. Se recorre sistemáticamente el tablero procesando filas, columnas y bloques mediante process_all_units/2.

Procesamiento de Unidades: Cada unidad se procesa generando una lista de índices (usando row_indices/1, column_indices/1 o block_indices/1) y aplicando process_unit/3. Esta función:

- Recolecta las posibilidades de cada celda no fijada con collect_numbers_in_unit/3 y get_possible_numbers/3.
- Identifica los números que aparecen únicamente una vez en la unidad usando find_unique_numbers/2.
- Fija el número único en la casilla correspondiente mediante apply_unique_changes/4.

Fijación y Propagación: Al fijar un número en una celda (con replace_possibility/4), se actualiza la lista de posibilidades y se elimina dicho número de las celdas relacionadas (fila, columna y bloque) usando get_related_indices/2 y remove_number_from_cell/4.

Regla 2:

La Regla 2 se encarga de identificar en un grupo (fila, columna o cuadrante) aquellas celdas que contienen exactamente dos posibilidades y que se repiten en dos celdas (pares desnudos), para luego eliminar esos dos números del resto de las celdas del mismo grupo.

Para ello, se utiliza:

- **regla2/2:** Es la función principal que intenta aplicar la regla sobre filas, columnas y cuadrantes. Llama a **parejas_filas/2**, **parejas_columnas/2** y **parejas_cuadrantes/2** para procesar cada tipo de agrupación.
- **parejas_filas/2:** Divide la lista de 81 elementos en 9 filas usando **split_filas/2**, y luego aplica **procesar_listas/2** a cada fila.
- **procesar_listas/2:** Para cada grupo (fila, columna o cuadrante), busca pares desnudos usando **encontrar_todos_los_pares/2**. Si se detectan, procede a eliminar los números involucrados del resto de celdas mediante **eliminar_elementos_de_pares/3**, que a su vez utiliza **eliminar_si_no_es_par/4** para preservar las celdas que ya son pares válidos.
- **parejas_columnas/2:** Aplica el mismo proceso que en las filas, pero primero transpone la lista (con **transponer/2**) para tratar las columnas como filas, y luego vuelve a transponer el resultado.
- **parejas_cuadrantes/2:** Divide el sudoku en cuadrantes mediante **split_cuadrantes/2** (que, a su vez, usa **split_row/4** para agrupar los segmentos de las filas), aplica **procesar_listas/2** a cada cuadrante y finalmente reensambla la lista de 81 elementos.

Además, **count_occurrences/3** se emplea en **encontrar_todos_los_pares/2** para contar cuántas veces aparece un par específico en el grupo.

Regla 3:

La Regla 3 identifica en un grupo (fila, columna o cuadrante) aquellas tres celdas cuya lista de posibilidades contiene exactamente los mismos 3 números (tripletas desnudas). Al encontrar estas tripletas, se eliminan esos 3 números de las posibilidades de las demás celdas del mismo grupo, ya que se tiene la certeza de que esos números solo pueden estar en las celdas detectadas.

Para ello se emplean las siguientes funciones:

- **regla3/2:** Función principal que intenta aplicar la regla en filas, columnas y cuadrantes. Llama secuencialmente a **tripletas_filas/2**, **tripletas_columnas/2** y **tripletas_cuadrantes/2** para procesar cada tipo de agrupación.
- **tripletas_filas/2:** Utiliza **split_filas/2** para dividir la lista de posibilidades en 9 filas y luego procesa cada fila con **procesar_listas_3/2**.
- **procesar_listas_3/2:** Recorre cada grupo (fila, columna o cuadrante) y, si la fila no está vacía, busca tripletas con **encontrar_todas_las_tripletas/2**. Si se hallan tripletas, procede a eliminarlas del resto del grupo usando **eliminar_elementos_de_tripletas/3**, que a su vez emplea **eliminar_si_no_es_tripleta/4** para actualizar cada celda.
- **encontrar_todas_las_tripletas/2:** Mediante la primitiva **findall**, recoge todas las listas de tamaño 3 (ordenadas) que se repiten exactamente tres veces en el grupo, utilizando **count_occurrences_3/3** para el conteo, y luego elimina duplicados con **list_to_set/2**.
- **tripletas_columnas/2:** Aplica el proceso de las filas a las columnas; para ello transpone la lista (con **transponer/2**), procesa las filas resultantes y vuelve a transponer el resultado.
- **tripletas_cuadrantes/2:** Convierte las filas en cuadrantes usando **split_cuadrantes/2**, procesa cada cuadrante con **procesar_listas_3/2** y reensambla la lista de 81 elementos, similar al proceso de la Regla 2.

Con este conjunto de funciones, la Regla 3 reduce las posibilidades eliminando los números asociados a las tripletas desnudas de las celdas restantes, ayudando a simplificar progresivamente la solución del sudoku.

Mejoras

Una vez ella la funcionalidad completa en Prolog, tanto el archivo principal como las distintas reglas, hemos añadido los siguientes elementos y aplicado estas mejoras:

Interfaz ASCII

De cara a probar y evaluar el código, hemos escrito una simple función `mostrar_sudoku.` Su cometido es imprimir el estado actual de sudoku fila por fila y con un formato atractivo.

Para ello utiliza una función llamada **mostrar filas**. Para hacerlo esta función utiliza las primitivas append, write y lenght para constuir las listas de longitud 9 y mostarlas por pantalla. Hace uso de un contador para las filas que lleva impresas, imprimiendo una separación cada 3 y parando a la 9ª.

Aparte utiliza una función llamada **mostrar grupo** que separa las filas en 3 cuadrantes y las otorga el formato deseado. Finalmente las une para que se muestren en orden. Dentro de cada fila, agrupa los elementos de 3 en 3, separándolos con una | o un espacio si es del mismo grupo.

Conexión entre Prolog y Python

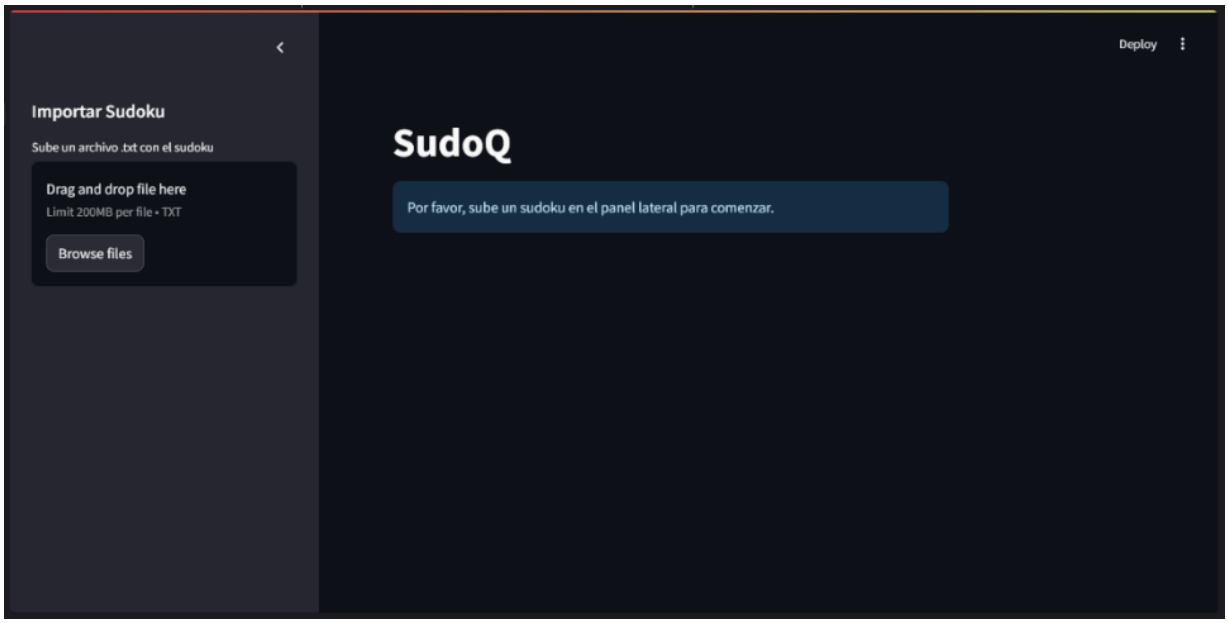
En nuestra aplicación, utilizamos la librería pyswip para interactuar con Prolog desde Python. Esta biblioteca actúa como una interfaz que nos permite ejecutar código Prolog de manera similar a como lo haríamos en la consola swipl. Gracias a pyswip, podemos realizar consultas y ejecutar reglas definidas en Prolog de forma dinámica.

Para optimizar esta interacción, hemos desarrollado un objeto llamado PrologConnector, el cual centraliza todas las operaciones necesarias para trabajar con Prolog. A través de este objeto, gestionamos la ejecución de reglas, la lectura de sudokus, la validación de movimientos y cualquier otra consulta

relevante. Esto nos permite mantener un código más organizado y facilitar la comunicación entre Python y Prolog en nuestra aplicación.

Aplicación Web interactiva

Otra mejora es la implementación de una interfaz gráfica interactiva por medio de una aplicación web que facilita al usuario la interacción con el sistema mediante un navegador.



Para este propósito hemos utilizado la librería de Python streamlit, la cual suele ser utilizada para aplicaciones de machine learning por su facilidad de programar, se presenta como una herramienta muy potente e intuitiva en nuestro caso de uso. Destaca su facilidad de uso con la instanciación de variable y el uso de variables de sesión que se guardan a lo largo de la ejecución del programa. Se ejecuta con streamlit run my_app.py puesto que no está desplegada, y la interfaz se renderiza en un navegador.

La aplicación comienza con una pantalla donde se muestra el nombre del programa y se espera a que se introduzca un fichero de sudoku en el rectángulo de la izquierda.

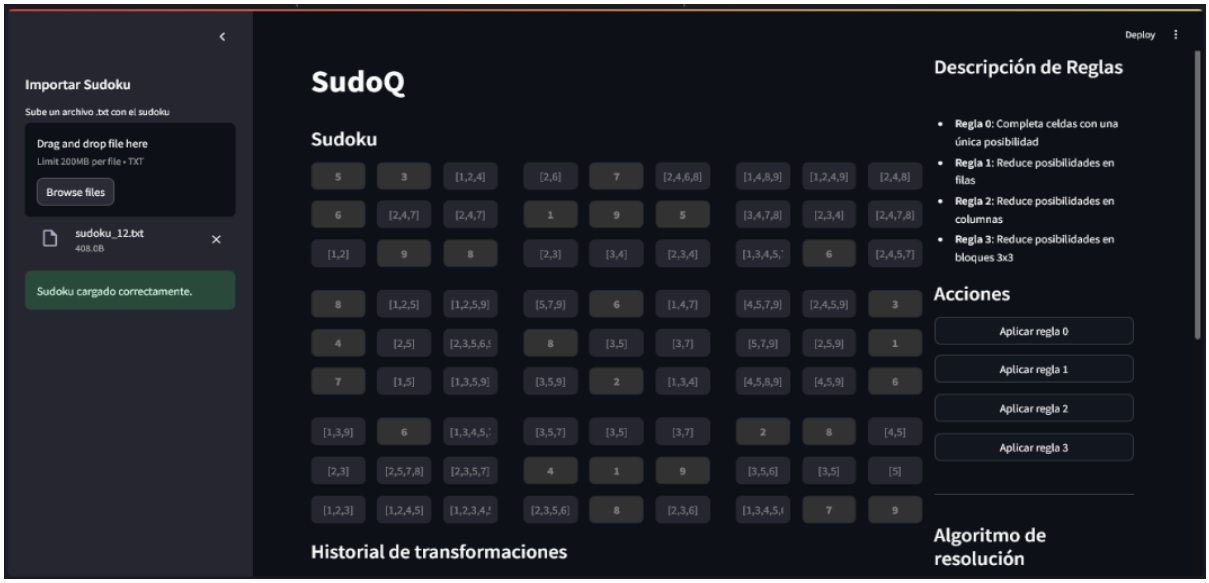
Nuestro programa trabaja con sudokus guardados en archivos de texto, con el mismo formato con el que se definían previamente en `sudokus.pl`, solo que ahorrándonos la regla que lo rodeaba.

```
[
    5 , 3 ,'.',  '.', 7 ,'.',  '.','.','.',
    6 ,'.','.',   1 , 9 , 5 ,  '.','.','.',
  '.', 9 , 8 ,  '.','.','.',  '.', 6 ,'.',

    8 ,'.','.',  '.', 6 ,'.',  '.','.', 3,
    4 ,'.','.',   8 ,'.','.',  '.','.', 1,
    7 ,'.','.',  '.', 2 ,'.',  '.','.', 6,

  '.', 6 ,'.',  '.','.','.',   2 , 8 ,'.',
  '.','.','.',   4 , 1 , 9 ,  '.','.','.',
  '.','.','.',  '.', 8 ,'.',  '.', 7 , 9
]
```

Una vez subido, se carga tanto la representación del sudoku como las distintas herramientas que la interfaz ofrece:



La representación del sudoku está formada por recuadros de texto editables y no editables, pues las casillas que ya han sido rellenadas no pueden ser modificadas. Por el contrario, las casillas libres muestran los números que pueden encontrarse en ella, y puede también el usuario introducir un número en una celda cualquiera, tras lo cual se actualizarán las posibilidades.

Se pueden apreciar a la derecha las reglas que se pueden aplicar, junto a sus descripciones.

Otra parte fundamental de la interfaz, como podemos ver, es el tener un historial de todas las acciones realizadas hasta el momento, tanto reglas aplicadas como entradas de usuario. Se

Obtención de métricas

Como extensión a la interfaz gráfica, se ha implementado la recolección de métricas acerca de la resolución del sudoku. Existen cuatro diferentes:

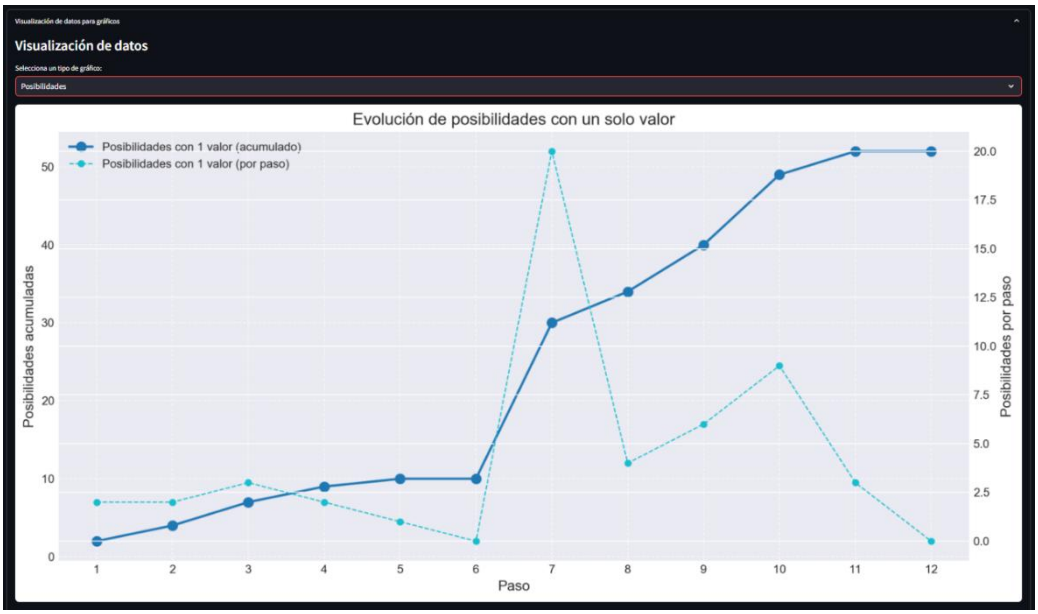
- **Posibilidades por cada paso:** una posibilidad equivale a que, dentro de la lista de valores posibles para una casilla, aplicar una regla deje esta lista con un único valor. Las

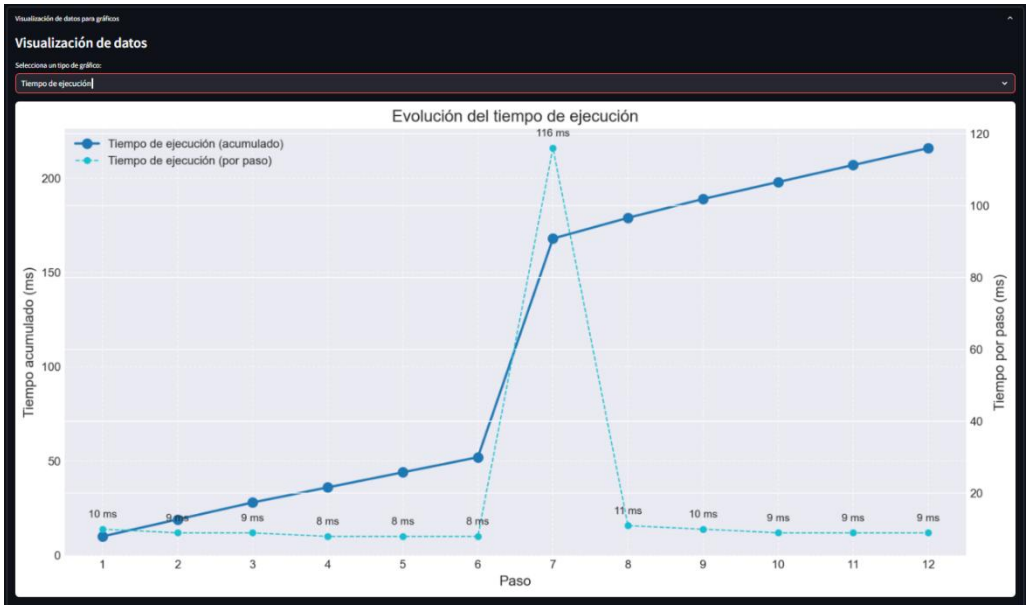
posibilidades son generadas por las reglas de reducción y son consumidas por las reglas de resolución.

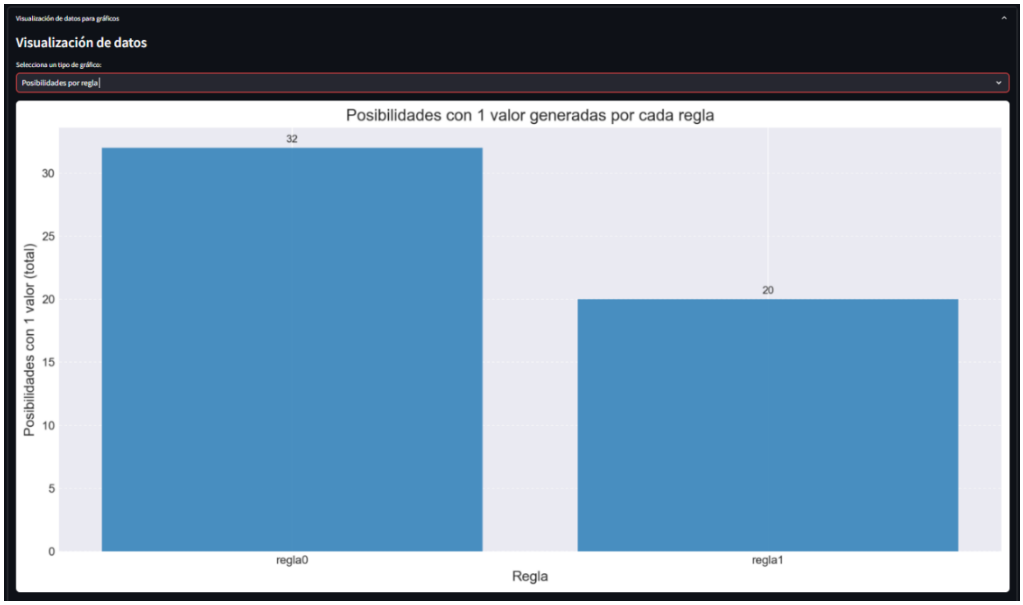
- **Crecimiento del tiempo de ejecución:** mide el tiempo efectivo que consume la ejecución del algoritmo. Se mide el tiempo acumulado por cada paso, de forma que siempre se encuentra en crecimiento.
- **Posibilidades que ha generado cada regla:** separa las posibilidades generadas por la regla que la ha generado, en vez de por el paso donde se ha generado.
- **Eficiencia de cada regla:** la eficiencia mide el tiempo en ejecución por recursos consumidos, en este caso será la porción del tiempo de ejecución total que ha tomado cada regla. Permite valorar la calidad del diseño de cada regla, así como su complejidad.

Estas métricas son medidas por la interfaz gráfica en cada ejecución y se pueden visualizar por la sección visualización de datos al fondo de la página. Pudiendo seleccionarse la opción deseada.

A continuación, se presentan capturas de como se ve esta mejora:





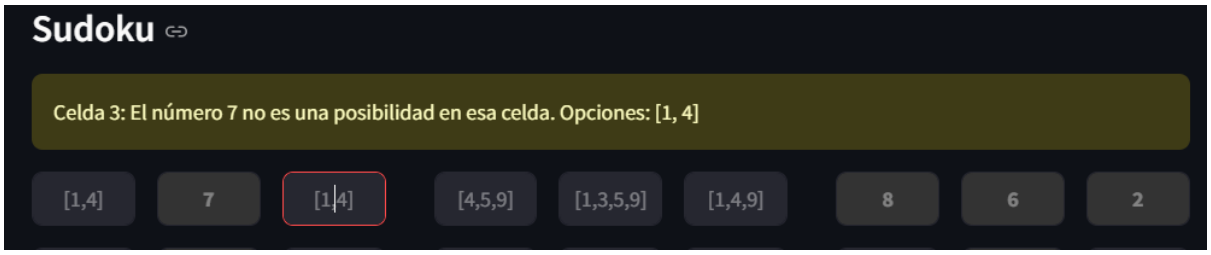


Detección de conflictos

La última mejora implementada realiza la detección de conflictos entre casillas. Esta es una función que es capaz de localizar dos valores iguales dentro de una misma fila, columna o cuadrante.

Dado que la interfaz permite la inserción manual de número en el sudoku, también detecta si un número insertado es una posibilidad para la celda o no.

Si se detecta un conflicto se vuelve al último estado correcto y se lanza un mensaje de advertencia en amarillo:



Algoritmos de resolución

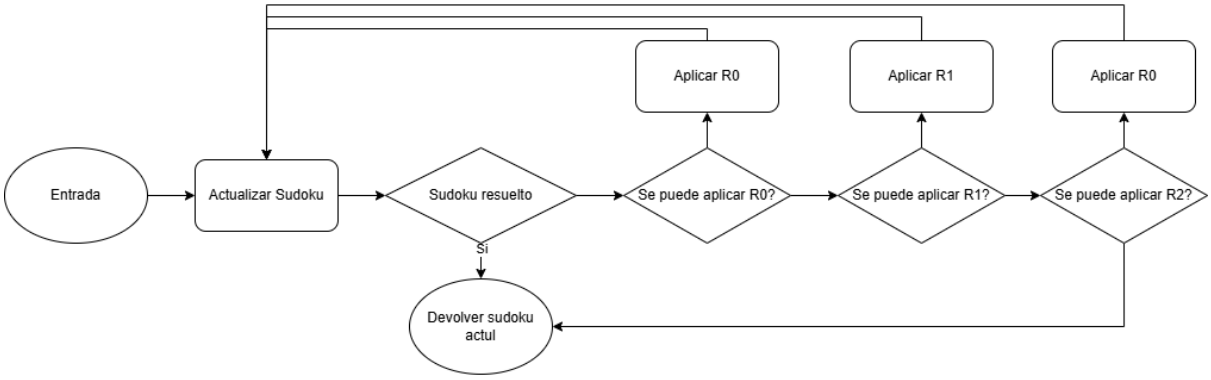
Llamamos a un algoritmo de resolución a la aplicación ordenada de un número definido de reglas según un criterio ordenado sobre un sudoku dado. La finalidad de estos algoritmos serán la resolución completa de los sudokus, pero debido a las limitaciones de resolución de las reglas, cabe la posibilidad de que el algoritmo resuelva parcialmente el sudoku. De esta manera, el número definido de reglas que se aplican será dado por la imposibilidad de aplicar ninguna regla que posee el algoritmo.

La nomenclatura que sigue estos algoritmos es la siguiente:

- Todo algoritmo comienza por “R”, lo cual hace referencia a la palabra “Resolver”, puesto que esta es la finalidad del algoritmo.
- Posteriormente a esa “R” observaremos una serie de números separados por “-”, el número indica la regla que se aplicará y en su orden de ejecución se respetará el orden en el que aparecen.

- La manera general en la que se aplican las reglas es recursiva, de esta manera que se seleccionará la regla que este en una posición menor, solo en caso de que se pueda.
- Si no se puede

Por ejemplo, si contamos con el algoritmo R0-1-2, el algoritmo operará priorizando siempre la regla0, en caso de que no sea posible su ejecución se tratará aplicar una vez la regla1, si esta posibilita aplicar la regla0 se volverá a aplicar esta misma, pero en caso de que ninguna de estas dos pueda, se aplicará la 2, si finalmente no se puede se devolverá el sudoku como esté, pudiendo estar resuelto o no. Se muestra un diagrama de flujo para comprender la aplicación de las reglas:



Como ya hemos explicado con anterioridad las reglas se pueden dividir en dos grupos:

- Reglas de reducción: Cuando se aplican estas reglas únicamente se reducen o no (en caso de que no sea aplicable) las posibilidades dentro del sudoku. Es decir, las reglas 1, 2 y 3.
- Reglas de resolución: Es aquella regla que tiene la capacidad de transformar una casilla vacía en una llena, quiere decir que puede colocar un número fijo. Cabe destacar que este tipo también puede reducir las posibilidades, puesto que rellenar una casilla puede disminuir los posibles números en casillas afectadas. Esta regla es la 0

Otro dato importante que aclarar es que por esta lógica todo algoritmo necesita la regla 0 para resolver un sudoku, puesto que sin él no es posible resolver ninguna de las casillas. Una vez conocidos estos conceptos explicamos los tipos de algoritmos que presentamos.

Algoritmos simples

Estos algoritmos principalmente están hechos para probar cada una de las reglas por separado, de esta manera somos capaces de ver si solos con una regla en especifico somos capaces de resolver todo el sudoku. Así pues, todos los algoritmos comenzarán por la regla en cuestión que se quiere probar y posteriormente la regla 0, necesaria para la resolución del sudoku.

Por lo que los algoritmos propuestos son: R0, R1-0, R2-0 y R3-0

Algoritmos Resolución-Reducción

Como indica el nombre de este conjunto de algoritmos, la lógica será aplicar resoluciones (regla 0) todo lo que se pueda. Cuando no se pueda más se recurrirán a reglas de reducción, de esta manera se habilitará a la regla 0 a poder ser usada de vuelta. De esta manera se minimizan el número de reducciones.

Nosotros usaremos los algoritmos R0-1-2-3 y R0-2-3-1 que demuestran ser las más eficientes.

Algoritmos Reducción-Resolución

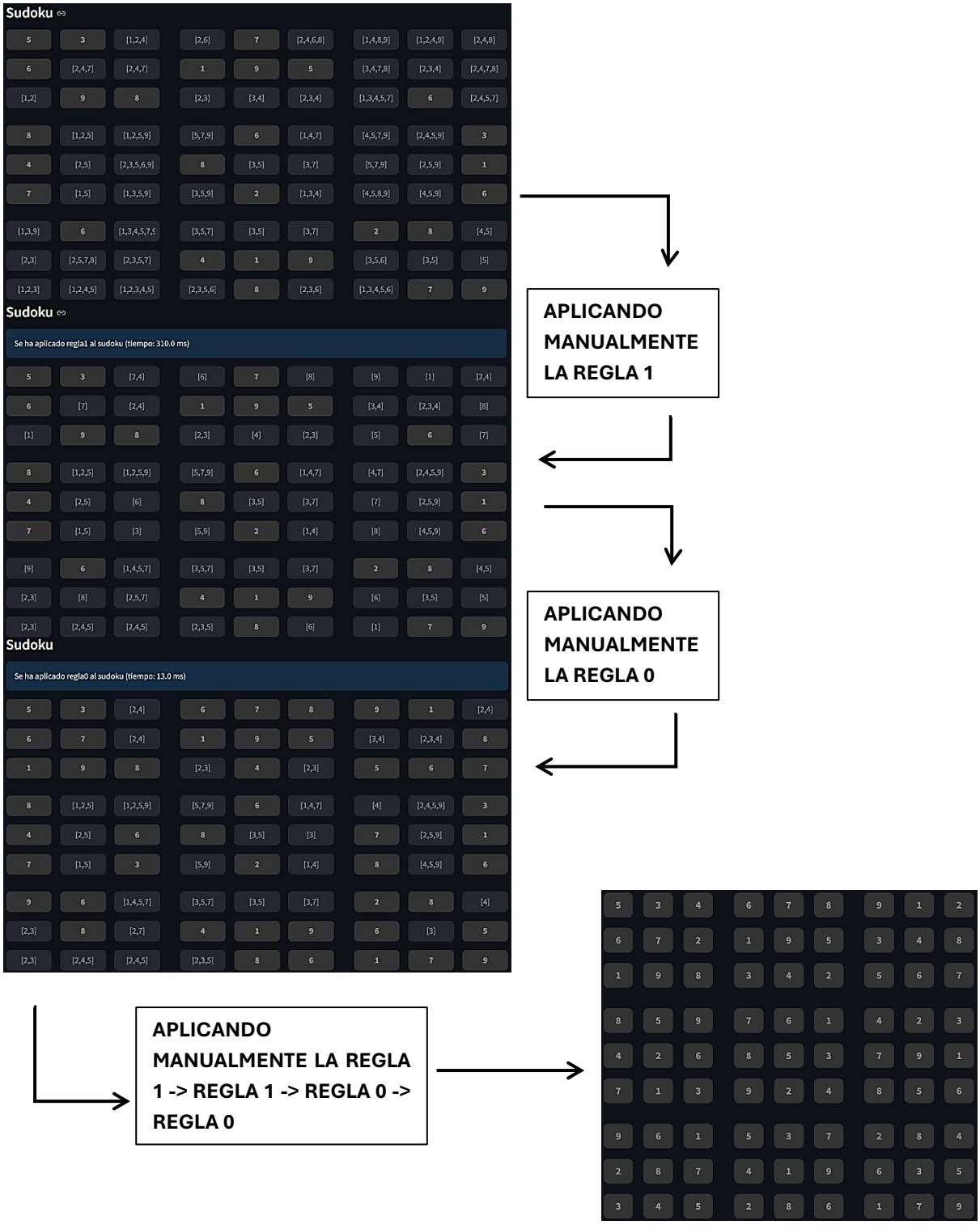
En este tipo se tratará de reducir al máximo el número de posibilidades de las casillas. Una vez que no se pueda reducir más el sudoku, aplicaremos una regla resolutiva que solucione todo lo preparado

anteriormente. A diferencia de la anterior, tratamos de reducir el número de resoluciones, por lo que cuando se aplique la regla 0 se resolverán más casillas a la vez. Usaremos los algoritmos: R1-2-3-0 y R2-3-1-0.

Resultados

Análisis de la evolución de ejemplos concretos.

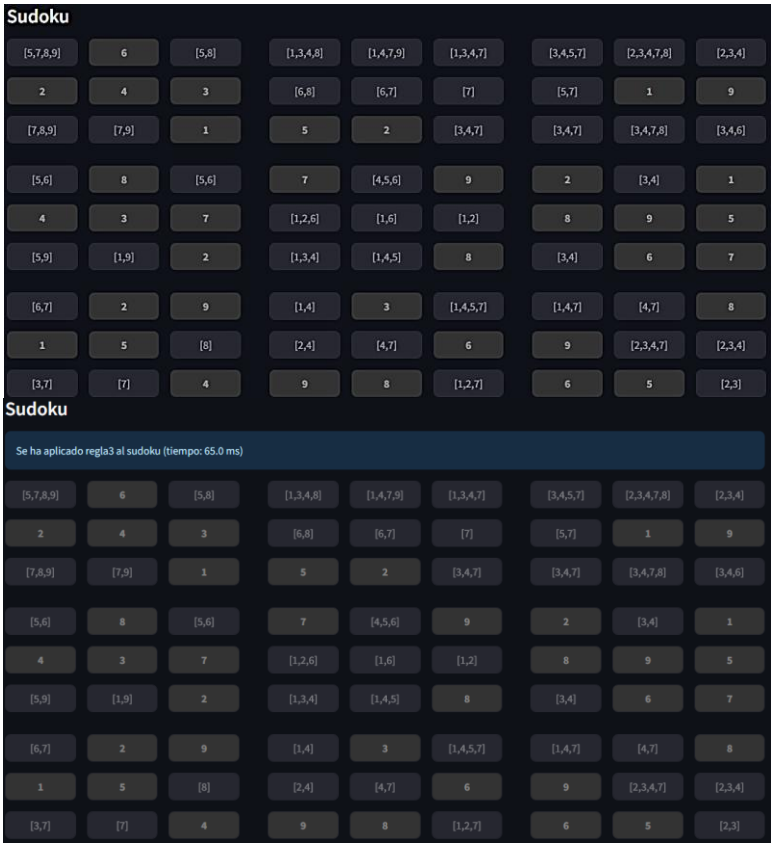
Ejemplo de uso de R0 y R1.



Ejemplo de uso de R2.

APLICANDO MANUALMENTE LA REGLA 2

Ejemplo de uso de R3.



APLICANDO MANUALMENTE LA REGLA 3

Métricas

Una vez presentados los algoritmos de resolución de algoritmos será necesario medir su rendimiento, para ello proponemos una serie de métricas que nos ayuden a determinar cuál es el algoritmo óptimo en determinados sudokus, así como dar la capacidad de explicar a porque es o no el mejor. En estas métricas mediremos la eficiencia y eficacia de cada uno de los algoritmos, en base a sudokus resueltos y casillas rellenadas, así como el número de reglas que emplean para obtener esos resultados:

Análisis de las métricas

Para evaluar el rendimiento de cada algoritmo, se han considerado cuatro parámetros principales:

1. **Sudokus resueltos**: Indica cuántos sudokus han sido completados exitosamente con cada algoritmo.
2. **Celdas rellenadas**: Representa el número total de casillas que fueron determinadas en cada conjunto de pruebas.
3. **Reglas aplicadas**: Número de veces que se ha ejecutado alguna de las reglas de simplificación para resolver los sudokus.
4. **Categorías de dificultad**: Se presentan métricas separadas para Sudokus fáciles, medios y difíciles, además de un resumen general.

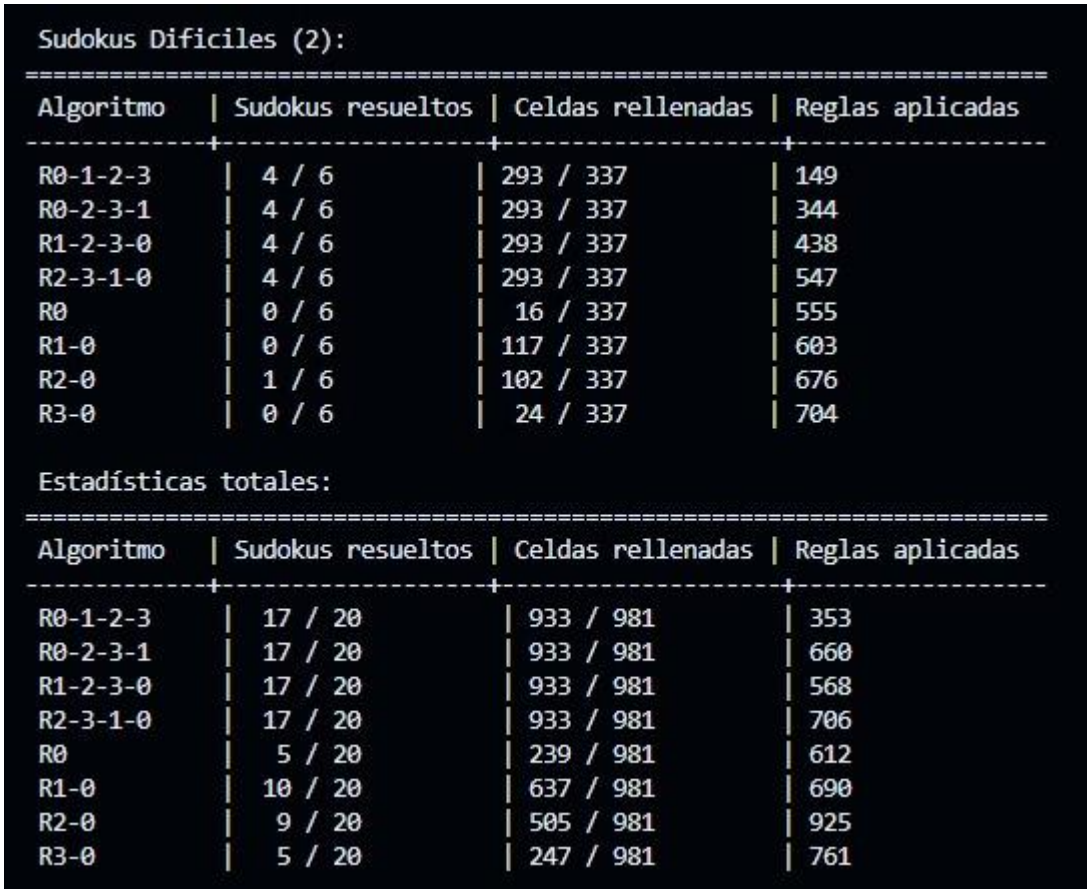
Resultados obtenidos

Sudokus Fáciles (0):

Algoritmo	Sudokus resueltos	Celdas rellenadas	Reglas aplicadas
R0-1-2-3	4 / 4	139 / 139	25
R0-2-3-1	4 / 4	139 / 139	25
R1-2-3-0	4 / 4	139 / 139	12
R2-3-1-0	4 / 4	139 / 139	23
R0	4 / 4	139 / 139	25
R1-0	4 / 4	139 / 139	12
R2-0	4 / 4	139 / 139	40
R3-0	4 / 4	139 / 139	25

Sudokus Medios (1):

Algoritmo	Sudokus resueltos	Celdas rellenadas	Reglas aplicadas
R0-1-2-3	4 / 4	215 / 215	71
R0-2-3-1	4 / 4	215 / 215	108
R1-2-3-0	4 / 4	215 / 215	51
R2-3-1-0	4 / 4	215 / 215	53
R0	1 / 4	68 / 215	23
R1-0	2 / 4	133 / 215	27
R2-0	2 / 4	122 / 215	77
R3-0	1 / 4	68 / 215	23



Sudokus Dificiles (2):

Algoritmo	Sudokus resueltos	Celdas rellenadas	Reglas aplicadas
R0-1-2-3	4 / 6	293 / 337	149
R0-2-3-1	4 / 6	293 / 337	344
R1-2-3-0	4 / 6	293 / 337	438
R2-3-1-0	4 / 6	293 / 337	547
R0	0 / 6	16 / 337	555
R1-0	0 / 6	117 / 337	603
R2-0	1 / 6	102 / 337	676
R3-0	0 / 6	24 / 337	704

Estadísticas totales:

Algoritmo	Sudokus resueltos	Celdas rellenadas	Reglas aplicadas
R0-1-2-3	17 / 20	933 / 981	353
R0-2-3-1	17 / 20	933 / 981	660
R1-2-3-0	17 / 20	933 / 981	568
R2-3-1-0	17 / 20	933 / 981	706
R0	5 / 20	239 / 981	612
R1-0	10 / 20	637 / 981	690
R2-0	9 / 20	505 / 981	925
R3-0	5 / 20	247 / 981	761

Comparación de algoritmos

Sudokus Fáciles:

En esta categoría, todos los algoritmos logran resolver los 4 sudokus planteados, rellenando las 139 casillas a rellenar totales en cada caso. La única diferencia sobre la que podemos incidir entonces es el número de reglas aplicadas:

Las reglas que tienen la regla 0 como primera regla a aplicar, así como R3-0 (ya que no llega a aplicar en este caso la 3), tienen el mismo número de reglas aplicadas porque todos estos sudokus podían resolverse con solo la regla 0, y ese es el proceso que han seguido. Esto ocurre porque siempre que hay cambios se vuelve al principio de la lista de reglas a aplicar, y la regla 0 siempre a podido efectuar cambios.

El algoritmo con peor actuación ha sido el R2-0, podemos suponer que por reducciones innecesarias que se han hecho con la regla 2

El claro ganador es R1-2-3-0, ya que presumiblemente efectúa una serie de reducciones hasta llegar a un estado con todas o casi todas las celdas a rellenar con una sola posibilidad, por lo que la regla 0 cuenta como aplicada una sola iteración, y rellena la totalidad del sudoku en una pasada. Este algoritmo es el que mejor desempeño tiene para sudokus fáciles.

Sudokus Medios:

En esta sección ya empezamos a ver algoritmos que no son capaces de resolver los sudokus en su totalidad.

R0 y R3-0 carecen de la regla 1, y apenas son capaces de resolver un sudoku. R1-0 y R2-0 les siguen de cerca con 1 resolución más. Ya aparentan no ser suficientes para problemas más exigentes.

R0-1-2-3 destaca por la gran cantidad de reglas aplicadas, sospechamos que por empezar por la regla 0 e ir haciendo pequeñas escrituras siempre que puede.

Mientras, R1-2-3-0 y R2-3-1-0, claros favoritos, dotan de una menor cantidad de reglas aplicadas por la misma razón que en los fáciles, por tener la regla 0 al final

Sudokus Difíciles:

En esta categoría, se aprecia más la diferencia entre los algoritmos completos y parciales.

Sorprendentemente, el algoritmo R0-1-2-3 parece ser el que mejor desempeño tiene para casos más difíciles, ya que para una mayor cantidad de celdas vacías y extensas posibilidades, puede ser que el resto haga un gran número de reducciones innecesarias antes de escribir lo que se podría haber escrito desde el principio.

Después apreciamos el total de sudokus, sumando las anteriores estadísticas más unos cuantos sudokus más para sumar al muestreo.

Conclusiones

Como conclusiones podemos afirmar que, para casos más simples y fáciles, el algoritmo R1-2-3-0 es el más eficiente y rápido, mientras que, para casos más complejos, R0-1-2-3 ataja mejor el problema. En todo caso, siempre es preferible un algoritmo que haga uso de todas las reglas a nuestra disposición.

Bibliografía:

SWI-Prolog. (2025). *Prolog Environment*. https://www.swi-prolog.org/

SWI-Prolog Documentation. (2025). *Manual de SWI-Prolog*. https://www.swi-prolog.org/pldoc/doc_for?object=manual

Sudoku.com. (2025). *Solver y creador de sudokus online*. https://sudoku.com/

The PySwip Project. (2025). Interfaz *Python - SWI-Prolog*. https://github.com/yuce/pyswip https://pypi.org/project/pyswip/

Wielemaker, J. (2025). *SWI-Prolog Reference Manual*. https://www.swi-prolog.org/pldoc/refman/

Streamlit. (2025). *Documentación de Streamlit*. https://docs.streamlit.io/

Streamlit Community. (2025). *Componentes de Streamlit*. https://streamlit.io/components

Covington, M. A., Bagnara, R., O'Keefe, R. A., Wielemaker, J., & Price, S. (2012). Coding guidelines for Prolog. *Theory and Practice of Logic Programming*, 12(6), 889-927.

Crook, J. F. (2009). A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles. *Notices of the American Mathematical Society*, 56(4), 460-468.