ESCUELA DE POSGRADO

MAESTRÍA EN CIENCIA DE

DATOS

ENSAYO

RESOLUCIÓN DE UN SUDOKU CON BACKTRACKING

INTEGRANTES

LUNA ACOSTUPA JHOEEL EDDYE

GUEVARA VARGAS VICTOR MANUEL

ORCID - 0000-0002-7538-6222

NEIRA CHALAN JAVIER EDUARDO

SOTO CHAMBILLA ALONSO GABRIEL

VILCA PANTIGOSO KAROL BERLIZ

DOCENTE

ACOSTA TICSE DEISY LIZBETH

LIMA, PERÚ

2025

INDICE

I. INTRODUCCIÓN

La resolución de problemas lógicos mediante algoritmos computacionales es un componente esencial de la informática moderna, particularmente en áreas como la inteligencia artificial, la ciencia de datos y la optimización. Uno de los algoritmos más versátiles para este tipo de retos es el backtracking o retroceso, una técnica de búsqueda sistemática que explora soluciones posibles y retrocede cuando detecta una opción no válida [1]. En este contexto, el Sudoku representa un excelente caso de estudio para entender y aplicar este enfoque.

El Sudoku es un juego de lógica numérica que consiste en completar una cuadrícula de 9x9 celdas, dividida en subcuadros de 3x3, donde cada fila, columna y subcuadro deben contener los números del 1 al 9 sin repetirse. A pesar de sus reglas simples, resolver un Sudoku puede volverse computacionalmente complejo, especialmente cuando el tablero tiene pocas pistas iniciales. Este tipo de reto encaja dentro de los problemas de satisfacción de restricciones (CSP, por sus siglas en inglés) [2].

El presente ensayo se centra en la implementación práctica del algoritmo de backtracking para resolver automáticamente Sudokus utilizando Python [5]. El código desarrollado recorre el tablero buscando celdas vacías, prueba números del 1 al 9 verificando que no violen las reglas del juego (ni en fila, ni en columna, ni en el subcuadro 3x3), y continúa de forma recursiva hasta completar el tablero o detectar que debe retroceder (backtrack).

La implementación del algoritmo se estructura en torno a tres funciones fundamentales, cada una de las cuales cumple un rol específico en la resolución del Sudoku:

buscar_celda_vacia(tablero): Esta función se encarga de recorrer el tablero en búsqueda de la primera celda vacía, identificada por el valor 0. Su propósito es localizar dinámicamente el próximo punto donde debe intentarse una asignación de número, actuando como el punto de entrada para la recursión.

es_valido(tablero,número,posicion): Antes de colocar un número en una celda determinada, esta función verifica si la asignación cumple con las reglas del Sudoku. Evalúa que el número no esté repetido en la fila, columna ni en el subcuadro 3x3 correspondiente a la posición seleccionada.

resuelve_sudoku(tablero): Es la función principal que implementa la técnica de backtracking. Utiliza recursividad para probar números posibles en las celdas vacías, y retrocede (backtrack) cuando una elección conduce a un conflicto, reiniciando la búsqueda desde el estado anterior hasta encontrar una solución válida o determinar que el tablero es irresoluble.

Este ensayo tiene como propósito analizar paso a paso dicha implementación, explicar su funcionamiento desde la lógica del código Python, y reflexionar sobre sus ventajas, limitaciones y oportunidades de mejora. Además, se discutirá su rendimiento algorítmico y se plantean ideas para optimizar su eficiencia.

Estudiar el backtracking en este contexto no solo permite resolver Sudokus automáticamente, sino también entender cómo abordar otros problemas similares donde se requiere encontrar soluciones bajo restricciones específicas. Casos como planificación de horarios, configuración de sistemas complejos o incluso ciertos algoritmos de inteligencia artificial para juegos utilizan esta misma lógica, lo que hace que su comprensión sea muy valiosa más allá del entorno académico.

Este ensayo se divide en tres secciones:

1. Una revisión teórica del algoritmo de backtracking y su lógica de funcionamiento.
2. Una explicación detallada del código utilizado para resolver un Sudoku mediante Python.
3. Un análisis del rendimiento del algoritmo, con propuestas de mejora y posibles aplicaciones futuras.

II. DESARROLLO

2.1. Definición del Algoritmo Backtracking

Backtracking (retroceso) es una estrategia algorítmica de búsqueda que se utiliza para resolver problemas de toma de decisiones, como combinatoria, búsqueda en espacios de soluciones y resolución de restricciones [1]. La idea central es explorar todas las posibles combinaciones de forma sistemática mediante un proceso de:

1. Construcción progresiva de soluciones parciales.
2. Verificación de restricciones en cada etapa.
3. Retroceso (backtrack) cuando una solución parcial no puede conducir a una solución válida completa.

Se basa en recursión y consiste en tomar una decisión, seguir adelante con ella, y si más adelante se detecta que no es viable, deshacer (retroceder) esa decisión y probar otra alternativa.

Aplicaciones comunes:

- Resolver laberintos
- Sudoku
- Problemas de las N reinas
- Generación de combinaciones o permutaciones
- Problemas de mochila o subconjuntos

2.2 Aplicación a Sudoku: Regla y Validación

El Sudoku es un problema clásico de restricción y búsqueda, donde el objetivo es rellenar una cuadrícula de 9×9 con dígitos del 1 al 9, cumpliendo estas tres reglas fundamentales:

1. Regla de fila: Cada número del 1 al 9 debe aparecer solo una vez por fila.
2. Regla de columna: Cada número del 1 al 9 debe aparecer solo una vez por columna.
3. Regla de subcuadro 3×3: Cada número del 1 al 9 debe aparecer solo una vez por subcuadro 3×3

2.2.1 Validación paso a paso

Antes de colocar un número, el algoritmo hace lo siguiente:

1. Revisa si el número ya está en la misma fila.
2. Revisa si el número ya está en la misma columna.
3. Revisa si el número ya está en el mismo bloque 3×3.

Si pasa estas validaciones, se coloca el número y se avanza a la siguiente celda vacía.Si no hay opciones válidas, se retrocede (backtrack) a la celda anterior para probar otro número.

2.3. Implementación del Algoritmo en Python

Se presenta a continuación un ejemplo de código que resuelve un Sudoku usando backtracking:

Link de repositorio: https://github.com/LunaJhoeel/backt

2.3.1 Contexto del Problema

La planificación de horarios de personal sanitario constituye un problema de optimización combinatoria NP-completo, donde se debe satisfacer simultáneamente:

- Requisitos de cobertura mínima por turno
- Disponibilidad individual del personal
- Restricciones laborales y de bienestar
- Distribución equitativa de la carga de trabajo

Alcance del Análisis

El presente estudio examina la implementación de un algoritmo de backtracking con propagación de restricciones, evaluando su eficiencia computacional y calidad de solución para un caso real de planificación hospitalaria.

2.3.2 Descripción del Problema

a. Parámetros del Sistema

- **Personal disponible**: 5 enfermeros (Ana, Luisa, Marta, Cris, Eva)
- **Período de planificación**: 5 días laborales (lunes a viernes)
- **Turnos por día**: 3 turnos (Mañana: M, Tarde: T, Noche: N)
- **Espacio de soluciones**: 3^25 = 847,288,609,443 configuraciones teóricas

b. Restricciones Implementadas

Restricciones Duras (Hard Constraints)

1. **Cobertura mínima**: 2 enfermeros en turno mañana, 1 en tarde y 1 en noche
2. **Disponibilidad personal**: Cada enfermero tiene horarios preestablecidos de disponibilidad
3. **Límite semanal**: Máximo 5 turnos por enfermero por semana
4. **Límite nocturno**: Máximo 2 turnos nocturnos por enfermero por semana
5. **Días consecutivos**: Máximo 4 días consecutivos de trabajo

Restricciones Blandas (Soft Constraints)

- Distribución equitativa de turnos entre el personal disponible

2.3.3 Metodología Algorítmica

a. Arquitectura del Algoritmo

El sistema implementa un **algoritmo de backtracking cronológico** con las siguientes características:

PROCEDIMIENTO BacktrackingPlanificacion:

1. Inicializar estado vacío

2. Para cada (día, turno) en orden cronológico:

a. Para cada enfermero disponible:

- Verificar restricciones

- Si válido: asignar y continuar recursivamente

- Si inválido: probar siguiente enfermero

b. Si ningún enfermero válido: retroceder

3. Si todas las asignaciones completas: retornar solución

b. Estrategia de Búsqueda

- **Orden de variables**: Cronológico (Lun-M, Lun-T, Lun-N, ..., Vie-N)
- **Orden de valores**: Secuencial por enfermero
- **Propagación**: Verificación inmediata de restricciones tras cada asignación
- **Criterio de poda**: Eliminación temprana de ramas inviables

c. Estructura de Datos

EstadoPlan {

plan: Dict[Día][Turno] → List[Enfermero]

turnos_asignados: Dict[Enfermero] → Int

noches_asignadas: Dict[Enfermero] → Int

dias_consecutivos: Dict[Enfermero] → Int

}

2.3.4. Análisis de Resultados

a. Métricas de Rendimiento

- **Tiempo de ejecución**: 0.714527 segundos
- **Recursiones totales**: 2,796 llamadas
- **Retrocesos**: 2,775 operaciones
- **Tasa de éxito**: 100% (solución encontrada)
- **Eficiencia**: 99.25% de retrocesos (indicativo de espacio de búsqueda complejo)

b. Solución Óptima Encontrada

Día	Mañana (M)	Tarde (T)	Noche (N)
Lun	Luisa, Eva	Ana	Marta
Mar	Ana, Luisa	Eva	Cris
Mie	Marta, Ana	Cris	Eva
Jue	Luisa, Marta	Ana	Cris
Vie	Marta, Eva	Cris	Luisa

c. Distribución de Carga de Trabajo

Enfermero	Turnos Totales	Turnos Noche	Utilización
Ana	5	0	100%
Luisa	5	1	100%
Marta	5	1	100%
Cris	5	2	100%
Eva	5	1	100%

2.4 Comparación de Algoritmos para la Resolución de Sudoku

En este documento, se analizan varios algoritmos utilizados para resolver rompecabezas de Sudoku, comparando sus ventajas y desventajas. Uno de los algoritmos más populares es el algoritmo de Backtracking, el cual se analiza a continuación en comparación con otros enfoques comunes como: algoritmos de fuerza bruta, algoritmos de recocido simulado y búsqueda en profundidad [2].

2.4.1. Algoritmo de Backtracking

El algoritmo de Backtracking es una técnica de búsqueda en la que se construye una solución paso a paso y se deshace de los pasos previos si se detecta que la solución es inviable [3]. Este algoritmo es ideal para resolver el Sudoku, ya que puede explorar todas las posibles combinaciones de manera eficiente, retrocediendo cuando se encuentra con una solución inválida.

Ventajas del Backtracking:

- Explora todas las soluciones posibles: El Backtracking garantiza encontrar la solución, ya que explora todas las combinaciones posibles hasta encontrar la correcta.
- Adaptabilidad a diferentes tamaños de Sudoku: Puede ser fácilmente adaptado a diferentes tamaños de tablas y configuraciones de Sudoku.
- Simplicidad en la implementación:Aunque el algoritmo puede ser costoso en términos de tiempo de ejecución, su implementación es relativamente simple y directa.

Desventajas del Backtracking:

- Ineficiencia en tableros grandes: En casos de tableros grandes de Sudoku, el algoritmo puede ser lento debido a que verifica todas las posibles combinaciones.
- Requiere de recursión: El uso de recursión puede hacer que el algoritmo consuma una cantidad significativa de memoria, especialmente con tableros complejos.

2.4.2 Algoritmos Similares al Backtracking

2.4.2.1 Algoritmo de Fuerza Bruta

El algoritmo de fuerza bruta intenta resolver el Sudoku probando todas las combinaciones posibles de forma exhaustiva sin ninguna optimización. Este método es similar al Backtracking, pero no retrocede ni optimiza los intentos, simplemente prueba cada combinación de manera secuencial.

Ventajas de la Fuerza Bruta:

- Solución garantizada: Al igual que el Backtracking, el algoritmo de fuerza bruta garantiza encontrar una solución, si es que existe.

Desventajas de la Fuerza Bruta:

- Ineficiencia extrema: Debido a que no hay retroceso ni optimización, la fuerza bruta es extremadamente ineficiente, especialmente en tableros grandes.
- Gran consumo de tiempo y recursos: Este método es impráctico para resolver el Sudoku más complejo debido al enorme número de combinaciones a evaluar.

2.4.2.2 Algoritmo de Recocido Simulado

El algoritmo de recocido simulado es un algoritmo de optimización probabilística que simula el proceso físico de enfriamiento de un metal. Este algoritmo puede ser utilizado para resolver Sudoku, pero su enfoque es estocástico y no garantiza encontrar la solución exacta, sino una aproximación [6].

Ventajas del Recocido Simulado:

- Velocidad: El recocido simulado puede ser más rápido que Backtracking y fuerza bruta, especialmente en grandes tableros de Sudoku.

Desventajas del Recocido Simulado:

- No garantiza una solución exacta: El recorrido simulado no garantiza encontrar la solución exacta, sino que puede converger a una aproximación.
- Dependencia de parámetros: La efectividad del algoritmo depende de la correcta configuración de sus parámetros, lo que puede complicar la implementación.

2.4.2.3 Algoritmo de Búsqueda en Profundidad

La búsqueda en profundidad explora un camino en su totalidad antes de retroceder y explorar otros caminos. Este algoritmo es similar al Backtracking pero no realiza retrocesos inteligentes como el Backtracking.

Ventajas de la Búsqueda en Profundidad:

- Implementación simple: La búsqueda en profundidad es relativamente fácil de implementar y entender.

Desventajas de la Búsqueda en Profundidad:

- No garantiza la solución: Al igual que el algoritmo de recocido simulado, la búsqueda en profundidad no garantiza encontrar una solución óptima.
- Posible ciclo infinito: La búsqueda en profundidad puede quedar atrapada en ciclos infinitos sin una estrategia de retroceso eficiente.

III. CONCLUSIONES

- El estudio y la implementación del algoritmo de backtracking para la resolución de Sudokus permiten comprender de manera práctica cómo funciona una estrategia de búsqueda sistemática bajo restricciones. A través del análisis del código en Python, se evidenció que este enfoque es capaz de encontrar la solución exacta explorando todas las posibilidades, pero optimizando el proceso mediante retrocesos cuando se detecta una ruta inviable.
- El backtracking presenta como principales ventajas su fiabilidad (garantiza encontrar la solución si existe) y su adaptabilidad a diferentes tamaños y configuraciones del problema. Sin embargo, también se identificaron limitaciones, principalmente relacionadas con el tiempo de ejecución y el consumo de memoria en casos de tableros grandes o con pocas pistas iniciales.

IV. REFERENCIAS

[1] M. T. Goodrich, R. Tamassia y M. H. Goldwasser, *Data Structures and Algorithms in Python*. Wiley, 2013.

[2] W.-M. Lee, *Sudoku Programming with C*. Apress, 2006.

[3] arXiv, "A Study Of Sudoku Solving Algorithms: Backtracking and Heuristic," 2025. . Disponible en: https://arxiv.org/pdf/2507.09708.pdf

[4] dpublication, "Efficient Data Structures for Solving Sudoku Puzzles," 2021. . Disponible en: https://www.dpublication.com/wp-content/uploads/2021/11/01-2670.pdf

[5] AirTribe, "Mastering Sudoku with Recursion & Backtracking," 2022.]. Disponible en: https://www.airtribe.live/dsa-sheet/resource/sudoku-solver

[6] arXiv, “Techniques for Solving Sudoku Puzzles,” 2012. Disponible en: https://arxiv.org/abs/1203.2295

[7] River Publishers, “Comparative study on Sudoku using Backtracking algorithm,” 2019. Disponible en: https://www.riverpublishers.com/downloadchapter.php?file=RP_9788770229647C32.pdf