

UNIVERSIDAD FRANCISCO DE VITORIA

**ESCUELA POLITÉCNICA SUPERIOR**

**GRADO EN INGENIERÍA INFORMÁTICA**

INGENIERÍA DEL CONOCIMIENTO

**PRÁCTICA SUDOKU**

Diego Viñals Lage, Javier Garrido Cobo

Alejandro Quiroz Coscollano, Daniel Ojeda Velasco

Pablo Quétin de la Vega, Ignacio Tejero Ruiz

Índice de Contenidos

Contenido

[1. Caso 1: Búsqueda No Informada 2](#_Toc166178130)

[1.1. Búsqueda en Anchura 2](#_Toc166178131)

[1.1.1. Inicialización 2](#_Toc166178132)

[1.1.2. Proceso de Búsqueda 2](#_Toc166178133)

[1.1.3. Generación de Nuevos Estados 3](#_Toc166178134)

[1.1.4. Expansión y Retroceso 3](#_Toc166178135)

[1.1.5. Condiciones de Terminación 3](#_Toc166178136)

[1.2. Búsqueda en Profundidad 3](#_Toc166178137)

[1.2.1. Selección de la Celda 4](#_Toc166178138)

[1.2.2. Generación de Asignaciones 4](#_Toc166178139)

[1.2.3. Asignación y Avance 5](#_Toc166178140)

[1.2.4. Recursión 5](#_Toc166178141)

[1.2.5. Condición de Parada 6](#_Toc166178142)

[1.3. Búsqueda de Coste Mínimo 6](#_Toc166178143)

[1.3.1. Inicialización 7](#_Toc166178144)

[1.3.2. Expansión de Nodos 7](#_Toc166178145)

[1.3.3. Generación de Estados Hijos 7](#_Toc166178146)

[1.3.4. Cálculo de Costes 7](#_Toc166178147)

[1.3.5. Actualización del Coste Acumulado 8](#_Toc166178148)

[1.3.6. Repetición y Terminación 8](#_Toc166178149)

[1.4. Análisis Eficiencia caso 1 8](#_Toc166178150)

[2. Caso 2: Búsqueda con Información 9](#_Toc166178151)

[2.1. Búsqueda Avara 9](#_Toc166178152)

[2.1.1. Heurística 9](#_Toc166178153)

[2.1.2. Orden de exploración 9](#_Toc166178154)

[2.1.3. Asignación y Actualización 10](#_Toc166178155)

[2.1.4. Condición de parada 11](#_Toc166178156)

[2.2. Búsqueda A\* 12](#_Toc166178157)

[2.2.1. Función de coste y Heurística 12](#_Toc166178158)

[2.2.2. Orden de exploración 13](#_Toc166178159)

[2.2.3. Asignación y Actualización 14](#_Toc166178160)

[2.2.4. Condición de Parada 14](#_Toc166178161)

[3. Caso 3: Búsqueda con Adversario 16](#_Toc166178162)

[3.1. Minimax 16](#_Toc166178163)

[3.1.1. Representación del Problema 16](#_Toc166178164)

[3.1.2. Función de Utilidad 16](#_Toc166178165)

[3.1.3. Definición de Jugadores 16](#_Toc166178166)

[3.1.4. Expansión del Árbol de Juego 17](#_Toc166178167)

[3.1.5. Minimización y Maximización 17](#_Toc166178168)

[3.1.6. Poda Alfa-Beta 17](#_Toc166178169)

[3.1.7. Turnos Alternados y Condición de Parada 17](#_Toc166178170)

[3.1.8. Implementación 17](#_Toc166178171)

[4. Caso 4: Búsqueda con Restricciones 18](#_Toc166178172)

[4.1. Variables 18](#_Toc166178173)

[4.2. Dominio 18](#_Toc166178174)

[4.3. Restricciones 18](#_Toc166178175)

[4.4. Búsqueda con Restricciones y revisión 19](#_Toc166178176)

[4.5. Búsqueda Heurística 19](#_Toc166178177)

[4.6. Heurística del valor mínimo 20](#_Toc166178178)

[4.7. Grado heurístico 20](#_Toc166178179)

[4.8. Valor menos restringido 20](#_Toc166178180)

[5. Bibliografía 22](#_Toc166178181)

Índice de Ilustraciones

[Ilustración 1. Búsqueda con restricciones 19](#_Toc166178182)

# Caso 1: Búsqueda No Informada

## Búsqueda en Anchura

La búsqueda en anchura es un algoritmo de recorrido de grafos que explora sistemáticamente todos los nodos de un grafo nivel por nivel, comenzando desde un nodo raíz dado. Este método se utiliza ampliamente en ciencias de la computación para realizar búsquedas sin información en estructuras de datos como grafos y árboles. [1]

### Inicialización

Estado Inicial: El tablero de Sudoku comienza con un estado inicial donde algunas celdas ya están llenas según las pistas proporcionadas y otras están vacías. Este es el punto de partida para el algoritmo. La correcta selección y verificación del estado inicial son cruciales porque cualquier error en este paso puede llevar a la exploración innecesaria de estados inválidos que no conducirán a una solución.

Cola de Estados: Utilizamos una estructura de cola para gestionar los estados durante la búsqueda. Inicialmente, esta cola contiene solo el estado inicial del tablero. La gestión de esta cola es fundamental ya que asegura que todos los estados a un mismo nivel de profundidad sean explorados antes de pasar a niveles más profundos. Esta estructura apoya la naturaleza expansiva de búsqueda en anchura, permitiendo una exploración sistemática sin retrocesos.

### Proceso de Búsqueda

Extraer Estado: En cada iteración del algoritmo, se extrae el estado en el frente de la cola. Este estado es el punto de partida para la exploración actual.

Verificación de Solución: Cada estado extraído se verifica para determinar si es una solución completa del Sudoku, es decir, todas las celdas están llenas y cumplen con las restricciones de no repetir números en las filas, columnas y cuadrantes 3x3. Esta verificación es crítica, ya que encontrar una solución válida termina el proceso de búsqueda inmediatamente.

Identificar Celda Vacía: Si el estado actual no es una solución, se busca la primera celda vacía. Esta es una operación clave porque determina dónde se realizarán las próximas inserciones de números, y por ende, cómo se expandirá el árbol de búsqueda. [2]

### Generación de Nuevos Estados

Generar Posibilidades: Para cada celda vacía identificada, se generan todos los estados posibles asignando números del 1 al 9 que no violan las reglas del Sudoku. Este paso es donde la mayor parte del trabajo computacional tiene lugar, ya que cada número posible puede llevar a un nuevo estado del tablero.

Añadir a la Cola: Cada nuevo estado generado se añade al final de la cola. Este método asegura que los estados se exploran en orden de su generación y mantiene la integridad de la exploración de nivel por nivel de búsqueda en anchura.

### Expansión y Retroceso

Expansión: Este proceso de generar nuevos estados y añadirlos a la cola se repite iterativamente, lo que permite al algoritmo explorar todas las posibles configuraciones del tablero de Sudoku a un cierto nivel antes de moverse a niveles más profundos. Esta expansión sistemática es la esencia de búsqueda en anchura.

Retroceso: A diferencia de otras técnicas como la búsqueda en profundidad, en búsqueda en anchura no se necesita gestionar explícitamente el retroceso ya que la propia estructura de la cola asegura que una vez que un estado es explorado y no lleva a una solución, simplemente se procede al siguiente estado en la cola.

### Condiciones de Terminación

Cola Vacía: La exploración termina cuando la cola se vacía, lo cual indica que todos los posibles estados han sido explorados sin encontrar una solución. Esta situación solo ocurre si el Sudoku no tiene solución bajo las condiciones iniciales dadas.

La búsqueda en anchura es exhaustiva y garantiza que si hay una solución, la encontrará. Sin embargo, puede usar mucha memoria, especialmente para Sudokus de alta dificultad con muchas celdas vacías al inicio. Por esto, aunque es teóricamente completa, en la práctica, la eficiencia de búsqueda en anchura puede ser limitada por los recursos del sistema y la naturaleza del Sudoku que se esté resolviendo.

## Búsqueda en Profundidad

La búsqueda en profundidad prioriza la exploración hacia las profundidades del árbol de búsqueda antes de volver a explorar los hermanos de un nodo. Este método usa una pila (ya sea implícita mediante llamadas recursivas o explícita) para mantener un registro de los nodos a visitar.

Una característica clave de la búsqueda en profundidad es el backtracking (retroceso). Cuando el algoritmo llega a un nodo sin hijos no explorados o donde todos los caminos posibles ya han sido evaluados sin éxito, retrocede a su nodo padre para explorar otros caminos posibles. Esto continúa hasta que todos los caminos han sido explorados o se encuentra una solución.

Para evitar ciclos infinitos y la exploración de los mismos nodos, la búsqueda en profundidad marca los nodos visitados. Esta marca es esencial en grafos donde algunos nodos pueden ser alcanzados desde múltiples rutas.

En el Sudoku, la búsqueda en profundidad intentará llenar el tablero de manera recursiva, probando números del 1 al 9 en cada celda vacía y avanzando. Si en algún punto se determina que no hay número válido para la siguiente celda vacía, el algoritmo retrocederá a la celda anterior (backtracking) y probará con el siguiente número posible. Este proceso se repite hasta que se llena todo el tablero correctamente o se determina que no hay soluciones posibles con las elecciones hechas anteriormente.

### Selección de la Celda

La selección de la celda es un paso crucial en el algoritmo de búsqueda en profundidad para resolver Sudoku. El proceso de selección de celdas determina el orden en el que se explorarán las celdas vacías del tablero, lo que puede tener un impacto significativo en la eficiencia y efectividad del algoritmo para encontrar una solución.

El método que se va a seguir para la selección de celdas es un orden secuencial comenzando desde la esquina superior izquierda del tablero y moviéndose de izquierda a derecha y de arriba abajo. Este método es intuitivo y fácil de implementar. La idea es tratar de llenar cada celda vacía encontrada en este orden con números del 1 al 9 que cumplan con las reglas del Sudoku.

Al ser un algoritmo de búsqueda no informada, la búsqueda en profundidad no establece ningún tipo de prioridad a la hora de seleccionar celdas. En lugar de ello, simplemente sigue el proceso de explorar sistemáticamente desde el punto de inicio, avanzando lo más profundamente posible en cada rama antes de retroceder.

### Generación de Asignaciones

En el proceso de resolución del Sudoku mediante técnicas de búsqueda no informada, como la búsqueda en profundidad, no se dispone de información adicional sobre el estado del juego más allá de lo observado en el tablero actual. Por lo tanto, la asignación de números a las celdas vacías debe adherirse estrictamente a las reglas fundamentales del Sudoku para avanzar hacia una solución válida.

* **Regla de la Fila:** Antes de asignar un número a una celda, se debe verificar que este número no está ya presente en la misma fila. Si, por ejemplo, se desea colocar el número '1' en una celda vacía, se debe primero asegurar que no haya otro '1' en ninguna de las celdas de esa fila. Esto garantiza que cada número del 1 al 9 aparezca solo una vez por fila.
* **Regla de la Columna**: De manera similar a la fila, el número que se desea asignar no debe aparecer en la misma columna. Esto implica que antes de colocar un número en una celda, se revisa toda la columna para confirmar que el número no se repite. Por ejemplo, si un '1' ya está colocado en cualquier parte de la columna, no se puede asignar un '1' a otra celda de esa misma columna.
* **Regla del Bloque de 3x3**: El tablero de Sudoku está dividido en nueve bloques de 3x3. La celda en la que se desea colocar un número también debe cumplir la condición de que este número no esté presente en el bloque de 3x3 al que pertenece la celda. Esto significa que cada bloque debe contener todos los números del 1 al 9 sin repetirse dentro del bloque.

### Asignación y Avance

El proceso de Asignación y Avance es fundamental para avanzar de manera sistemática y lógica a través del tablero. Este proceso no solo implica llenar las celdas con números, sino también hacerlo de una manera que respete las reglas del juego y prepare el terreno para las subsiguientes asignaciones.

En este algoritmo, los números se intentan colocar de forma ascendente (del 1 al 9) en las celdas vacías. Este método sistemático asegura que cada opción se evalúa de manera uniforme y en orden, lo cual facilita la implementación del algoritmo y el seguimiento del proceso de resolución.

Antes de que un número sea definitivamente asignado a una celda, se verifica que su colocación sea válida según las tres reglas principales del Sudoku: no debe repetirse en la fila, la columna ni el bloque de 3x3 donde se encuentra la celda. Esta evaluación se realiza cada vez que se considera un número para una celda específica. Si el número satisface todas las condiciones necesarias, se coloca en la celda.

Una vez asignado un número a una celda, el algoritmo avanza a la siguiente celda vacía en el tablero, generalmente siguiendo un orden de izquierda a derecha y de arriba abajo. Este enfoque lineal simplifica el proceso de búsqueda al mantener un patrón claro y ordenado de progresión.

Si en algún punto se encuentra que no hay números válidos para una celda dada (una situación de "callejón sin salida"), el algoritmo debe retroceder: se deshace la última asignación y se prueba con la siguiente opción numérica en la celda anterior. Este proceso de retroceso es esencial para explorar todas las configuraciones posibles del tablero sin omitir soluciones potenciales.

### Recursión

Este enfoque permite al algoritmo explorar sistemáticamente todas las posibles combinaciones de números en el tablero, asegurando que se examine cada configuración potencial antes de llegar a una solución definitiva o concluir que no existe solución.

Cuando se asigna un número a una celda en el Sudoku utilizando búsqueda de profundidad, la recursión permite continuar avanzando hacia la siguiente celda vacía, repitiendo el mismo proceso de selección y asignación de números. Este enfoque asegura que el algoritmo profundice en una dirección (una secuencia de asignaciones) hasta que se encuentre con un impedimento (una celda que no puede ser llenada sin violar las reglas del Sudoku).

El backtracking se activa cuando el proceso recursivo llega a una situación donde no hay números válidos que asignar a la celda actual sin infringir las reglas. Esto puede ocurrir porque las asignaciones previas han creado un conflicto que solo se hace evidente más adelante en el proceso.

Al detectar un bloqueo, el algoritmo retrocede, es decir, revierte la última asignación (el último número colocado que aún permite otras posibilidades) y prueba con la siguiente opción numérica disponible para esa celda. Este proceso de deshacer y rehacer se repite cuantas veces sea necesario hasta encontrar una ruta que permita avanzar de nuevo o hasta que se agoten todas las posibilidades.

### Condición de Parada

#### Solución Completa

La condición de parada más directa y deseada ocurre cuando todas las celdas del tablero de Sudoku están llenas y cada asignación cumple con las reglas del juego (no hay repeticiones en filas, columnas o bloques de 3x3).

Esta condición se verifica después de cada asignación. Si una celda es llenada y resulta ser la última celda vacía del tablero, el algoritmo realiza una verificación final de todo el tablero para asegurar que no existen violaciones de las reglas. Si todo está correcto, el algoritmo termina y devuelve el tablero como una solución válida.

#### Solución Incompleta

Esta condición de parada ocurre cuando se determina que no hay soluciones posibles desde el estado actual del tablero. Esto sucede si una celda no tiene ningún número válido que asignar después de haber probado todas las opciones posibles, indicando un error en las asignaciones anteriores o la imposibilidad de resolver el tablero con las decisiones tomadas hasta ese momento.

Cada vez que se asigna un número a una celda, se verifica si es válido. Si no hay números válidos disponibles y la función de backtracking ha regresado a esta celda, significa que todas las rutas exploradas desde las decisiones anteriores han llevado a un callejón sin salida. En este caso, el algoritmo retrocede aún más o, si se está en la celda inicial y todas las posibles asignaciones han sido probadas sin éxito, el algoritmo concluye que no hay solución bajo las condiciones dadas.

## Búsqueda de Coste Mínimo

La búsqueda de costo mínimo es un enfoque algorítmico utilizado en inteligencia artificial y en la resolución de problemas en general, donde el objetivo es encontrar la solución óptima a un problema minimizando un determinado costo o función de costo. En este contexto, el "costo" puede representar diferentes métricas según el problema, como distancia, tiempo, recursos utilizados, entre otros.

En términos generales, la búsqueda de costo mínimo implica explorar el espacio de búsqueda del problema, evaluando y comparando los costos asociados con diferentes estados o acciones posibles. El objetivo es encontrar la secuencia de acciones que minimice el costo total para alcanzar la solución deseada.

Este enfoque se utiliza en una amplia variedad de aplicaciones, desde la planificación de rutas en logística hasta la resolución de problemas en juegos como el Sudoku. Algoritmos comunes que implementan este enfoque incluyen el algoritmo de Dijkstra para encontrar el camino más corto en grafos, el algoritmo A\* para búsqueda heurística en grafos, y en el contexto que mencionaste, adaptaciones de algoritmos de búsqueda de costo mínimo para resolver el Sudoku.

### Inicialización

* **Configuración del tablero**: El algoritmo comienza con el tablero inicial de Sudoku, que puede tener algunas celdas ya llenas y otras vacías.
* **Cola de prioridades**: Se utiliza una cola de prioridades para gestionar los estados del tablero. Cada estado en la cola tiene asociado un coste acumulado desde el estado inicial. Este coste representa el "peso" de llegar a ese estado, considerando las acciones (asignaciones de números) realizadas hasta el momento.
* **Función de comparación**: En la cola, los estados son ordenados de acuerdo con el coste acumulado. El estado con el menor coste acumulado es siempre el próximo a ser expandido, asegurando así que el algoritmo siga el camino de menor coste en cada paso.

### Expansión de Nodos

* **Selección del nodo**: En cada iteración, el estado con el menor coste acumulado es extraído de la cola de prioridades. Este estado representa la configuración actual del tablero de Sudoku.
* **Exploración de opciones**: Desde este estado, el algoritmo explora todas las posibles continuaciones, es decir, todas las celdas vacías que pueden ser llenadas siguiendo las reglas del Sudoku.

### Generación de Estados Hijos

* **Búsqueda de celdas vacías**: El algoritmo identifica la próxima celda vacía en el tablero. Esta selección puede seguir un patrón específico, como de izquierda a derecha y de arriba abajo, para mantener el orden y la coherencia.
* **Asignaciones válidas**: Para la celda seleccionada, se generan todos los estados posibles asignando números que no violen las reglas del Sudoku (números no repetidos en la fila, columna y subcuadrícula).
* **Creación de nuevos estados**: Cada asignación válida resulta en un nuevo estado del tablero, que es potencialmente un hijo del estado actual en el árbol de búsqueda.

### Cálculo de Costes

* **Determinación de costes**: El coste de moverse a un nuevo estado (hijo) se determina por el número de opciones válidas (hijos) que tenía el estado padre. Esto refleja la idea de que menos opciones implica decisiones más críticas o restrictivas, lo cual podría considerarse de mayor "coste" en términos de complejidad o dificultad.
* **Incentivo para estados menos complejos**: Al utilizar este esquema de costes, el algoritmo tiende a favorecer los estados que presentan menos opciones, posiblemente conduciendo a una solución más rápidamente.

### Actualización del Coste Acumulado

* **Acumulación de costes**: Cada vez que un estado hijo es creado, su coste acumulado es calculado como la suma del coste acumulado del estado padre más el coste asociado con la transición al estado hijo.
* **Mantenimiento de la cola**: Los nuevos estados son insertados en la cola de prioridades con su respectivo coste acumulado. Esto asegura que el estado con el menor coste siempre sea explorado primero.

### Repetición y Terminación

* **Iteración continua**: El proceso de extracción de estados de la cola, expansión y generación de estados hijos, y calculación de costes se repite.
* **Condiciones de terminación**: El algoritmo termina cuando se encuentra una configuración del tablero que es una solución válida (todas las celdas están llenas correctamente según las reglas del Sudoku) o cuando la cola de prioridades se vacía, lo cual indicaría que, bajo las decisiones tomadas y las reglas definidas, no existe solución posible para el Sudoku dado.

## Análisis Eficiencia caso 1

Búsqueda en Anchura es exhaustivo y consume mucha memoria, pues mantiene todos los nodos de un nivel antes de avanzar al siguiente.

Búsqueda en profundidad tiene un coste menor en memoria pero puede quedar atrapado en caminos sin solución, especialmente en tableros grandes o complicados.

Búsqueda de Coste Mínimo puede ser eficiente en términos de encontrar la solución óptima, pero su rendimiento depende de la función de coste.

Como conclusión para Sudokus, búsqueda en profundidad podría ser más práctico en la mayoría de los casos debido a su menor uso de memoria y capacidad para retroceder eficientemente. Búsqueda en anchura puede ser útil para tableros más pequeños o menos celdas vacías.

# Caso 2: Búsqueda con Información

## Búsqueda Avara

### Heurística

#### Minimizar las Celdas Vacías

Este aspecto de la heurística se centra en cuánto progreso se ha hecho hacia completar el tablero. La lógica es simple: [3]

* Conteo de Celdas Vacías: Simplemente cuenta el número de espacios que no tienen asignado un número todavía.
* Impacto en la Heurística: Menos celdas vacías significa un valor de heurística más bajo, lo que indica un estado más prometedor o deseable.

#### Maximizar los Números Correctamente Colocados

Este es un criterio crucial porque no solo considera llenar las celdas, sino llenarlas correctamente según las reglas del Sudoku:

* Verificación de Reglas del Sudoku: Para cada número colocado, verifica que no viole las reglas de Sudoku en su fila, columna y subcuadrícula (3x3).
* Cálculo de Violaciones:
  + Filas: Por cada fila, verifica la presencia de números repetidos.
  + Columnas: Igual que las filas, pero verifica cada columna.
  + Subcuadrículas: Divide el tablero en 9 subcuadrículas de 3x3 y verifica los números repetidos en cada una.
  + Impacto en la Heurística: Menos violaciones de reglas aumenta la "corrección" de un tablero, reduciendo así el valor de la heurística.

### Orden de exploración

En la búsqueda avara, la elección del próximo estado a explorar se basa exclusivamente en una heurística que estima cuán cerca está cada estado de la solución final. Esta heurística no sólo guía la búsqueda hacia los estados más prometedores, sino que también intenta minimizar el número de pasos innecesarios al ignorar aquellos que parecen menos prometedores según la evaluación heurística.

Para el Sudoku, la orden de exploración se puede detallar en los siguientes pasos:

#### Evaluación Inicial:

Al comenzar, cada configuración del tablero se evalúa usando la heurística diseñada.

#### Selección de Estado:

* Utilizando una estructura de datos como una cola de prioridad, donde los estados con menor valor heurístico tienen prioridad, seleccionamos el estado que parece estar más cerca de la solución completa y correcta.
* Esto significa que el estado con el menor número de celdas vacías y violaciones (es decir, el menor valor heurístico) se selecciona para la exploración.

#### Exploración y Expansión:

* Desde el estado seleccionado, generamos nuevos estados posibles llenando una celda vacía con un número que no infrinja las reglas del Sudoku en esa fila, columna o subcuadrícula.
* Cada nuevo estado generado se evalúa inmediatamente con la misma heurística y se añade a la cola de prioridad.

#### Iteración Continua:

* El proceso se repite: se extrae el estado con la mejor heurística de la cola, se explora, y se generan nuevos estados hasta que se resuelve el Sudoku o se determina que no hay solución viable.

### Asignación y Actualización

El proceso de Asignación y Actualización en la búsqueda avara para resolver Sudoku es una parte crítica del algoritmo, ya que es aquí donde efectivamente se modifican los estados del tablero en búsqueda de la solución. Veamos este proceso con más detalle, explicando cómo se realiza la asignación de valores a las celdas y cómo se actualiza el estado del tablero, asegurando que cada paso se acerca más a una solución válida del Sudoku.

#### Asignación:

Selección de la Celda: Identifica una celda vacía en el tablero para la asignación. La selección de la celda puede basarse en varias estrategias, como elegir la celda con menos opciones válidas (enfoque de "mínimos valores restantes") o la que está más restringida por las celdas ya llenadas (enfoque de "grado").

* Generación de Opciones Válidas: Para la celda seleccionada, genera una lista de posibles números que se pueden colocar en esa posición sin violar las reglas del Sudoku. Esto implica asegurarse de que el número no esté ya presente en la misma fila, columna o subcuadrícula de 3x3.
* Evaluación Heurística: Evalúa cada posible número utilizando la heurística definida. Por ejemplo, podrías reevaluar el tablero completo con cada posible número insertado para ver cuál reduce más la heurística, indicando un acercamiento más prometedor hacia la solución.
* Asignación del Número: Elige el número que resulte en el valor heurístico más bajo y asigna este número a la celda vacía. Esta decisión se basa en la expectativa de que el número elegido es el que más contribuye a avanzar hacia una solución completa y correcta.

#### Actualización del estado del tablero:

* Modificación del Tablero: Una vez seleccionado el número, el tablero se actualiza modificando la celda vacía para contener el nuevo número. Esto cambia el estado del tablero, que será usado para la siguiente iteración del algoritmo.
* Reevaluación: Con cada nuevo número asignado, el tablero debe ser reevaluado para asegurar que no se introducen nuevas violaciones de las reglas y que la heurística sigue siendo válida. Esto puede implicar recalcular las heurísticas para cada celda restante o solo para aquellas que son afectadas directamente por la última asignación (por ejemplo, en la misma fila, columna o subcuadrícula).
* Preparación para la Próxima Iteración: Una vez actualizado el tablero, el algoritmo se prepara para la siguiente ronda de selección y asignación, utilizando el nuevo estado del tablero como base para futuras evaluaciones y expansiones.

### Condición de parada

La Condición de Parada en la búsqueda avara es crucial para garantizar que el algoritmo se detenga de manera eficiente una vez que se ha alcanzado un objetivo o se ha determinado que es inalcanzable bajo las condiciones dadas. En el contexto del Sudoku, esta condición de parada necesita ser cuidadosamente implementada para maximizar la efectividad del algoritmo y evitar el desperdicio de recursos computacionales. Veamos cómo se puede estructurar esta condición de parada en detalle. [4]

#### Evaluación del Estado del Tablero

* Tablero Completo: Verificación de Completitud: La condición más directa de parada es verificar si todas las celdas del tablero están llenas. Si no quedan celdas vacías, es un indicativo de que potencialmente se ha alcanzado una solución.
* Validación del Tablero: No basta con que el tablero esté lleno; cada número colocado debe cumplir con las reglas del Sudoku (no repetirse en su fila, columna o subcuadrícula). Por lo tanto, cada vez que el tablero se llena, es necesario realizar una validación completa para asegurar que la solución es correcta.

#### Determinación de la Inviabilidad:

* Detección de Conflictos Insolubles: Durante la búsqueda, puede llegar un punto donde, a pesar de que el tablero no está completo, no hay números válidos que se puedan colocar en las celdas vacías sin violar las reglas del Sudoku. Esto puede ser resultado de una mala elección anterior que no se detectó inmediatamente.
* Retroceso y Reevaluación: Si se detectan tales conflictos, el algoritmo debe tener la capacidad de retroceder (deshacer las últimas asignaciones) y probar diferentes alternativas. Si todas las alternativas posibles han sido exploradas y el conflicto persiste, entonces se concluye que no hay solución válida bajo el estado actual.

#### Implementación de la Condición de Parada

* Chequeos en Cada Paso: Cada vez que se asigna un número a una celda, se debe verificar si esta asignación ha llevado el tablero a un estado completo y válido o si ha resultado en un estado donde no se puede progresar más.
* Control de Flujo del Algoritmo: La lógica del algoritmo debe gestionar adecuadamente el flujo de la búsqueda para poder detenerse o retroceder según lo necesite. Esto requiere una gestión cuidadosa de los estados almacenados y posiblemente el uso de una pila o cola para controlar el orden de exploración y retroceso.

## Búsqueda A\*

### Función de coste y Heurística

#### Función de coste

La función de coste en A\* para Sudoku se define generalmente como el número de movimientos realizados desde el estado inicial. Cada movimiento, que consiste en asignar un número a una celda vacía, incrementa en una unidad. Aquí están los detalles de cómo se implementa y se calcula.

* Incremento por Movimientos:
* Cada vez que se coloca un número en una celda vacía, se incrementa el contador de para ese estado particular.
* Esto refleja el camino recorrido desde el estado inicial hasta el estado actual, proporcionando una medida de cuánto esfuerzo se ha invertido en llegar a este punto.
* Utilidad en la Búsqueda:
* Esta medida ayuda a evitar ciclos y a no retroceder, ya que cualquier estado que requiera más movimientos será menos preferible bajo la misma heurística.

#### Función Heurística

 la función heurística es más compleja y crítica, ya que estima el costo restante para alcanzar el objetivo final desde el estado actual. Para Sudoku, una heurística efectiva podría ser la cantidad de celdas vacías, ya que esto proporciona una estimación directa del número mínimo de movimientos restantes. Sin embargo, puede ser beneficioso considerar otros factores para hacer la heurística más informativa:

* Cantidad de Celdas Vacías: Simplemente contar las celdas vacías ofrece una vista optimista y directa de cuánto trabajo queda por hacer.
* Complejidad de las Celdas Vacías: No todas las celdas vacías son igualmente fáciles de llenar. Algunas pueden tener muchas restricciones debido a su posición en filas, columnas y subcuadrículas ya parcialmente llenas.

Una posible mejora a la heurística podría incluir un factor que pondera las celdas vacías por la dificultad de llenarlas, basado en el número de opciones válidas restantes para cada celda.

* Admisibilidad y Consistencia:
* Admisibilidad: La heurística nunca debe sobreestimar el costo de completar el objetivo. En el caso de Sudoku, contar las celdas vacías garantiza esta propiedad porque en el mejor de los casos, cada celda vacía requerirá al menos un movimiento para ser llenada.
* Consistencia: Para A\*, la heurística es consistente si el costo estimado de llegar al objetivo desde n no es mayor que el costo de un paso desde n a un sucesor m más el costo estimado de llegar al objetivo desde m.

### Orden de exploración

Para comprender más profundamente la Orden de Exploración en el algoritmo A\* aplicado al Sudoku, es esencial destacar cómo la suma de las funciones g(n) y h(n) influye en la elección y expansión de los estados durante la búsqueda. Esta metodología asegura que se prioricen aquellos caminos que aparentemente se acercan más a una solución óptima.

#### Cálculo de f(n)=g(n)+h(n)

* g(n) Representa el coste acumulado para llegar al estado actual desde el estado inicial. En el caso del Sudoku, esto se traduce en el número de celdas que se han llenado correctamente.
* h(n) Estima el coste mínimo para completar el objetivo desde el estado actual. Por ejemplo, podría ser el número de celdas vacías, lo cual implica que cada celda vacía requerirá al menos un movimiento adicional para ser llenada.

#### Utilización de una Cola de Prioridad

* Los estados se mantienen en una cola de prioridad donde el estado con el menor valor de f(n) tiene la prioridad más alta. Esto permite que el algoritmo A\* extraiga siempre el estado más prometedor para la siguiente exploración.

#### Proceso de Selección del Estado

* En cada paso del algoritmo, se selecciona y se extrae de la cola el estado con el menor f(n)Esto asegura que los esfuerzos de búsqueda se centren en los caminos que tienen el potencial más alto para alcanzar una solución eficiente y rápida, balanceando el progreso hecho (a través de g(n)y el trabajo estimado que queda (a través de h(n)).

#### Expansión de Estados

* Después de seleccionar un estado, A\* procede a generar estados sucesores, los cuales representan las posibles formas de llenar una celda vacía más en el tablero de Sudoku. Cada sucesor tiene su propio valor de g(n) actualizado (incrementado en uno desde el estado actual) y un h(n)recalculado, dependiendo de las celdas vacías que quedan y posiblemente otros factores del tablero.

### Asignación y Actualización

La fase de Asignación y Actualización en el algoritmo A\* para resolver Sudoku es crucial y compleja, dado que en ella no solo se modifica el estado del tablero sino que también se recalcula la evaluación de ese nuevo estado, que influirá en las decisiones subsiguientes del algoritmo. Vamos a desglosar y profundizar aún más en cómo se llevan a cabo estas operaciones de manera eficiente y efectiva.

#### Asignación de Valor

* Selección de la Celda y Número:
  + Celda Objetivo: Identifica cuál celda vacía es la más estratégica para llenar. Esto puede basarse en la heurística como mínimo número de opciones válidas (técnica de "menos valores restantes"), o la celda que, una vez llenada, permite la mayor reducción en h(n).
  + Número a Asignar: De las opciones válidas para esa celda (números que no violen las reglas de Sudoku para esa fila, columna, y subcuadrícula), el número seleccionado debería ser aquel que, se espera, minimice más h(n) en el próximo paso. Esto puede requerir una evaluación rápida de cada opción.
* Asignación: Coloca el número seleccionado en la celda. Esta acción cambia el estado del tablero y afecta directamente g(n) al incrementar el número de movimientos realizados.

#### Actualización del Estado del Tablero

Recálculo de g(n): Incremento: g(n)se incrementa en uno cada vez que se llena una celda, reflejando el costo del camino tomado hasta el momento.

* Recálculo de h(n):
  + Reducción de Celdas Vacías: Con cada número colocado correctamente, h(n) se recalcula. Si simplemente contamos celdas vacías, h(n) se reduciría en uno.
  + Evaluación de Complejidad: Para una heurística más avanzada, se debería reevaluar las restricciones de las celdas relacionadas. Por ejemplo, si una celda en una subcuadrícula crítica se llena correctamente, esto podría simplificar las opciones para las celdas restantes de esa subcuadrícula, reduciendo potencialmente h(n)de manera más significativa.
* Recálculo de f(n): Suma de Costos: f(n) se recalcula como g(n)+h(n). Este es el valor que determinará la prioridad del estado en la cola de prioridad del algoritmo A\*, influenciando directamente el orden en que se explorarán los estados futuros.

### Condición de Parada

La Condición de Parada en el algoritmo A\* para resolver Sudoku es una pieza clave para garantizar que el algoritmo concluya de manera eficiente y efectiva. Esta condición no solo indica cuándo el algoritmo ha encontrado una solución o debe detenerse porque no hay solución posible, sino que también ayuda a gestionar los recursos computacionales al evitar una búsqueda innecesaria y prolongada. Profundicemos en cómo se implementan y manejan estas condiciones de parada.

#### Condición de Parada: Tablero Completo y Válido

* Verificación de Compleción: El algoritmo debe verificar continuamente si todas las celdas del tablero están llenas. Esta es la primera parte de la condición de parada: un tablero completo.
* Validación del Tablero:
  + No es suficiente con que el tablero esté completo; cada número colocado debe cumplir con las reglas del Sudoku:
    - No debe haber números repetidos en ninguna fila.
    - No debe haber números repetidos en ninguna columna.
    - No debe haber números repetidos en ninguna subcuadrícula de 3x3.
  + Esta validación se realiza mediante una revisión sistemática de todas las filas, columnas y subcuadrículas para asegurar que se respeten las reglas.
* Detención del Algoritmo:
  + Si se encuentra que el tablero está completo y válido, el algoritmo se detiene y retorna este tablero como la solución final.

#### Condición de Parada: Determinación de No Solución

* Exploración de Todos los Estados Posibles: A\* explora sistemáticamente los caminos más prometedores basados en la función f(n)=g(n)+h(n). Si, después de agotar todos los caminos posibles (es decir, la cola de prioridad está vacía), no se ha encontrado una solución válida, se debe considerar la posibilidad de que el Sudoku no tenga solución bajo la configuración inicial dada o las decisiones tomadas durante la búsqueda.
* Agotamiento de Estados:
  + Cuando la cola de prioridad se vacía y no se han generado más estados para explorar, se concluye que no hay soluciones posibles. Esto puede ocurrir por varias razones:
    - Un error en la asignación inicial de números que hizo imposible cualquier solución válida.
    - Decisiones subóptimas en las asignaciones de números debido a una heurística que no logró capturar la complejidad de algunas situaciones.

# Caso 3: Búsqueda con Adversario

## Minimax

El algoritmo Minimax es un método de búsqueda utilizado en juegos de adversario para encontrar la mejor jugada posible para un jugador, asumiendo que el oponente también juega de manera óptima. Funciona explorando de manera recursiva un árbol de juego que representa todas las posibles secuencias de movimientos para ambos jugadores, hasta alcanzar un estado terminal del juego. [5]

### Representación del Problema

En este contexto, el tablero de Sudoku se transforma en un campo de batalla donde cada jugador tiene turnos alternos para colocar números, respetando las reglas estándar de Sudoku. Cada estado del tablero refleja una configuración particular donde algunos números ya están colocados, y cada movimiento altera este estado. Los estados son dinámicos y cambian después de cada jugada de un jugador.

### Función de Utilidad

La función de utilidad mide directamente cómo cada movimiento afecta las posibilidades de cada jugador de ganar el juego. Una forma efectiva de hacer esto podría ser:

Para el jugador Max (Impares): Sumar puntos por cada número impar colocado correctamente en el tablero, asegurando que no se violen las reglas del Sudoku.

Para el jugador Min (Pares): Sumar puntos por cada número par colocado, también asegurándose de adherirse a las reglas.

La utilidad de un estado podría calcularse como la diferencia entre los puntos de ambos jugadores, proporcionando una medida clara de quién está en ventaja en cualquier momento del juego.

### Definición de Jugadores

Jugador Max (Impares): Este jugador busca maximizar la diferencia en la puntuación colocando números impares en posiciones estratégicas que no solo sumen puntos sino que también limiten las opciones del oponente.

Jugador Min (Pares): Al igual que Max, Min intenta minimizar la diferencia de puntos, enfocándose en colocar números pares de manera estratégica para bloquear a Max y sumar puntos.

### Expansión del Árbol de Juego

El árbol de juego se expande generando todos los posibles movimientos válidos desde el estado actual del tablero para el jugador en turno. Cada movimiento lleva a un nuevo estado del tablero, que se convierte en un nodo en el árbol de juego. Esta expansión continúa alternativamente entre los jugadores hasta que se alcanza una condición de terminación.

### Minimización y Maximización

* Maximizar: En su turno, el jugador Max evaluará todos los movimientos posibles utilizando la función de utilidad y seleccionará el que maximice su ventaja.
* Minimizar: De manera similar, el jugador Min evaluará sus opciones y elegirá el movimiento que resulte en la menor ventaja posible para Max, efectivamente minimizando la utilidad.

### Poda Alfa-Beta

Este paso es crucial para mejorar la eficiencia del algoritmo Minimax. La poda alfa-beta elimina ramas del árbol de búsqueda que, según la evaluación actual, no influirán en la decisión final, reduciendo así el número total de estados evaluados.

Alfa representa el mejor valor que el maximizador puede garantizar, mientras que beta representa el mejor valor que el minimizador puede garantizar. Si el valor calculado de un movimiento es peor que el valor actual de alfa o beta, esa rama se puede podar. [6]

### Turnos Alternados y Condición de Parada

Los turnos de los jugadores se alternan con cada movimiento, manteniendo el flujo del juego. La partida termina cuando se llenan todas las celdas del tablero, o cuando se alcanza una profundidad predeterminada en el árbol de búsqueda para limitar el tiempo de computación. La condición de parada ayuda a evitar que el juego entre en un estado de cálculo prolongado que podría no ser práctico en el contexto de un juego.

### Implementación

* No hay "Movimientos" de un Oponente: El Minimax evalúa los posibles movimientos de un oponente para decidir el mejor curso de acción. En Sudoku, no hay movimientos adversos; cada número colocado es un paso más hacia la solución sin influencia externa.
* Función de Utilidad: Minimax utiliza una función de utilidad para evaluar el estado del tablero desde la perspectiva de maximizar la ganancia y minimizar la pérdida. En Sudoku, la "utilidad" de un movimiento solo puede ser evaluada en términos de si ayuda a resolver el puzzle correctamente o no, y no en términos de impacto relativo a un oponente.

# Caso 4: Búsqueda con Restricciones

La búsqueda con restricciones es una estrategia de resolución de problemas que se enfoca en encontrar soluciones que cumplan con un conjunto predefinido de restricciones. A menudo se utiliza para problemas donde las soluciones deben satisfacer ciertas condiciones o reglas.

Una característica clave de la búsqueda con restricciones es su capacidad para manejar eficientemente restricciones complejas y grandes espacios de búsqueda. A medida que explora el espacio de búsqueda, el algoritmo utiliza técnicas de propagación de restricciones para reducir el espacio de búsqueda y dirigir la exploración hacia las áreas más prometedoras.

Para ello vamos a utilizar la búsqueda en profundidad la cual prioriza la exploración hacia las profundidades del espacio de búsqueda, añadiendo vuelta atrás. Sin embargo, en lugar de simplemente explorar todos los nodos, la búsqueda con restricciones se guía por las restricciones específicas del problema. [7]

## Variables

En el contexto de la búsqueda por restricciones en un Sudoku, las variables se refieren a las casillas del tablero que necesitan ser asignadas con un número del 1 al 9 para completar el rompecabezas. Estas casillas pueden ser tratadas como variables del problema. Una forma común de representarlas es asignar una variable a cada celda, que puede ser identificada por su fila y columna en el tablero, similar a cómo se nombran las casillas en el ajedrez.

Esta representación facilita la aplicación de restricciones y la manipulación de las variables durante el proceso de búsqueda por restricciones para resolver el Sudoku.

## Dominio

El dominio de cada variable se refiere al conjunto de números del 1 al 9 que pueden ser asignados a una casilla vacía del tablero. Es decir, para cada celda vacía, los posibles valores que puede contener son los números enteros del 1 al 9, ambos inclusive. Durante el proceso de búsqueda, estos dominios se irán reduciendo a medida que se apliquen restricciones y se asignen valores a las casillas del Sudoku, hasta que finalmente se encuentre una solución válida para el rompecabezas.

## Restricciones

Las restricciones se derivan directamente de las reglas fundamentales del juego. Estas reglas establecen que en cada fila, columna y subcuadrícula no puede haber números repetidos. Por lo tanto, las restricciones se aplican para garantizar que ningún número se repita en ninguna fila, columna o subcuadrícula del tablero, la cual nos va a ayudar durante el proceso de búsqueda, que utilizaremos para guiar la exploración del espacio de búsqueda y evitar soluciones que no cumplan las reglas del Sudoku.

Podríamos representar estas:

* No hay valores iguales en la misma fila
* No hay valores iguales en la misma columna
* No hay valores iguales en la misma subcuadrícula

Texto

Descripción generada automáticamente

Ilustración 1. Búsqueda con restricciones

## Búsqueda con Restricciones y revisión

Para resolver el Sudoku utilizando una búsqueda con vuelta atrás, aplicamos un enfoque que nos permite explorar las diferentes asignaciones de valores a las celdas del tablero, retrocediendo cuando una asignación no es válida. Como ya hemos explicado en el apartado 2.2 de esta memoria, la búsqueda en profundidad prioriza la exploración hacia las profundidades del árbol de búsqueda antes de volver a explorar los hermanos de un nodo.

Este método usa una pila (ya sea implícita mediante llamadas recursivas o explícita) para mantener un registro de los nodos a visitar. A este algoritmo le añadimos la vuelta atrás, una característica crucial que nos permite deshacer asignaciones anteriores cuando se detecta que conducen a un resultado no válido, retrocediendo al nodo padre y explorando otras opciones posibles.

## Búsqueda Heurística

En la búsqueda de soluciones para el Sudoku la aplicación de estrategias efectivas, tanto en las restricciones como en la elección de heurísticas, es esencial para completar el tablero de manera precisa y eficiente. Dentro del marco de la búsqueda en profundidad con vuelta atrás, se emplean diversas heurísticas para guiar el proceso de exploración del espacio de búsqueda. Tres de las heurísticas más comunes son: la heurística del mínimo valor, la heurística del valor menos restringido y la heurística del máximo valor.

## Heurística del valor mínimo

La heurística del valor mínimo para resolver un Sudoku se basa en seleccionar la celda vacía con el menor número de posibles valores para asignar. Esta estrategia implica priorizar la exploración de las celdas que presentan menos opciones disponibles, lo que puede acelerar el proceso de resolución al reducir la cantidad de opciones a considerar en cada paso.

Esto es un problema muy común cuando estamos resolviendo un sudoku donde nos encontramos con una celda vacía que tiene dos posibles valores para asignar, y otra celda vacía que tiene tres o cuatro valores posibles valores, la heurística del mínimo valor priorizará la exploración de la primera celda, ya que presenta menos opciones para considerar. Este enfoque busca minimizar la complejidad de la búsqueda al reducir la cantidad de ramificaciones en el árbol de búsqueda.

## Grado heurístico

Después de haber descrito la heurística del mínimo valor para resolver un Sudoku, podemos considerar la opción contraria: la heurística del máximo valor. Esta estrategia implica seleccionar la celda vacía con el mayor número de posibles valores para asignar, es decir, la celda que tiene el dominio más grande. Sin embargo, esta opción es generalmente menos deseable en comparación con la heurística del mínimo valor debido a algunas razones:

* Mayor probabilidad de conflicto y retroceso: Cuantas más opciones haya para asignar en una celda, mayor será la probabilidad de que surjan conflictos con las restricciones del Sudoku. Esto puede llevar a más retrocesos en el algoritmo, ya que es más probable que se encuentren callejones sin salida que requieran deshacer varias asignaciones anteriores.
* Menor eficiencia en la exploración del espacio de búsqueda: Al elegir la celda con el dominio más grande, es menos probable que se exploren eficientemente las ramas del árbol de búsqueda. Esto se debe a que el algoritmo puede pasar más tiempo explorando opciones en una sola celda antes de retroceder, en lugar de explorar diversas posibilidades en múltiples celdas al mismo tiempo.

## Valor menos restringido

La heurística del valor menos restringido para resolver un Sudoku se basa en seleccionar el valor que genere la menor cantidad de restricciones futuras al ser asignado a una celda vacía. Esto significa que, entre los números del dominio (del 1 al 9) que pueden ser asignados a una celda vacía, se elige aquel que permita la mayor flexibilidad en términos de asignaciones futuras en el tablero.

Por ejemplo, si tenemos una celda vacía y dos valores posibles para asignar, el valor menos restringido sería aquel que tenga menos impacto en las celdas vecinas. Es decir, si asignar el valor "1" a una celda vacía solo restringe una celda vecina, mientras que asignar el valor "4" restringe tres celdas vecinas, entonces se seleccionaría el "1" como el valor menos restringido.

Esta heurística busca minimizar los conflictos y retrocesos durante el proceso de resolución del Sudoku, ya que prioriza los valores que permiten una mayor exploración del espacio de búsqueda respetando las reglas del juego.

# Bibliografía

|  |  |
| --- | --- |
| [1] | H. Franco, «Universidad Central,» 23 7 2022. [En línea]. Available: https://hpclab.ucentral.edu.co/~hfranco/int\_art/Session\_6\_UninformedSearch.pdf. |
| [2] | «BibliaDelProgramador,» [En línea]. Available: https://www.bibliadelprogramador.com/2014/04/algoritmos-de-busqueda-en-anchura-bfs-y.html. [Último acceso: 6 5 2024]. |
| [3] | W. J. V. Rodriguez, «Prezi,» 26 4 2016. [En línea]. Available: https://prezi.com/ljevojmrfb9a/algoritmo-de-busqueda-avara/. |
| [4] | M. Castor, «InforUVA,» [En línea]. Available: https://www.infor.uva.es/~calonso/IAI/Tema4-BusquedaInformada/BusquedaInformada09-00.pdf. [Último acceso: 5 5 2024]. |
| [5] | F. Sancho, «Universidad de Sevilla,» [En línea]. Available: https://www.cs.us.es/~fsancho/Blog/posts/Minimax.md. [Último acceso: 3 5 2024]. |
| [6] | J. R. Q. Sarmiento, «Universidad Palmas de Gran Canaria,» [En línea]. Available: https://accedacris.ulpgc.es/bitstream/10553/121232/1/183-602-1-PB.pdf. [Último acceso: 5 5 2024]. |
| [7] | J. Larrosa, 2023. [En línea]. Available: https://www.iiia.csic.es/~pedro/Tut-CAEPIA-03-Pedro.pdf. [Último acceso: 6 5 2024]. |