

<b>DEPARTAMENTO:</b>	<b>CIENCIAS DE LA COMPUTACION</b>	<b>CARRERA:</b>	<b>Tecnologías de la información y comulación</b>		
<b>ASIGNATURA:</b>	Estructura de Datos	<b>NIVEL:</b>	3	<b>FECHA:</b>	06/01/2026
<b>DOCENTE:</b>	Ing.Margoth Guaraca	<b>PRÁCTICA N°:</b>	1	<b>CALIFICACIÓN:</b>	

## **Análisis Teórico y Experimental de Algoritmos de Búsqueda Exhaustiva y Backtracking**

**Corina Alejandra Acosta Oliva**

---

### **RESUMEN**

Este informe analizó la técnica de búsqueda exhaustiva en algoritmos computacionales. La introducción estableció su relevancia como método garantizado pero computacionalmente costoso. A través de la implementación práctica, se examinó la generación de espacios de búsqueda, se evaluó la eficacia de la poda en backtracking y se comparó el rendimiento con enfoques de fuerza bruta pura. Los resultados confirmaron el crecimiento exponencial de la complejidad y la mejora significativa que aportan las estrategias de poda, aunque se identificó que su efectividad depende de las restricciones específicas del problema. Las conclusiones subrayan que esta técnica es fundamental para problemas de escala reducida y para la verificación de soluciones, pero su escalabilidad está limitada, requiriendo de optimizaciones o enfoques alternativos para instancias mayores.

**Palabras Claves:** Búsqueda Exhaustiva, Backtracking, Complejidad Algorítmica

### **1. INTRODUCCIÓN:**

La búsqueda exhaustiva constituye una metodología algorítmica fundamental en las ciencias de la computación, caracterizada por la exploración sistemática de todas las posibles soluciones a un problema dentro de un espacio de búsqueda finito. Esta técnica, también denominada fuerza bruta, se aplica en contextos donde es imperativo garantizar la optimalidad de la solución, tales como en problemas de combinatoria, optimización y criptografía (Stanford University, 2023). Si bien su implementación suele ser conceptualmente sencilla, su eficiencia computacional se ve severamente comprometida ante instancias de gran escala, dado que su complejidad temporal puede crecer de forma exponencial o factorial (Polo &

Salgado, 2023). No obstante, su estudio es indispensable para comprender los límites de la computación y sirve como base para el desarrollo de técnicas más sofisticadas, como el backtracking y la programación dinámica. La presente investigación se enfoca en analizar los principios teóricos y las implicaciones prácticas de la búsqueda exhaustiva, evaluando su viabilidad mediante la implementación y prueba de algoritmos representativos en un entorno controlado de laboratorio.

## **2. OBJETIVO(S):**

### **Objetivo General**

Analizar los fundamentos teóricos y las características operativas de los algoritmos de búsqueda exhaustiva, mediante la implementación práctica de casos de estudio representativos, para evaluar su eficacia, limitaciones computacionales y aplicabilidad en la resolución de problemas de optimización combinatoria.

### **Objetivos Específicos**

Examinar la estructura y el mecanismo de generación del espacio de búsqueda en algoritmos exhaustivos, a través de la codificación de funciones que enumeren todas las permutaciones binarias y decimales de longitud fija, con el fin de comprender la relación entre la recursividad y la exploración sistemática.

Evaluar la eficiencia y las estrategias de poda (pruning) en algoritmos de backtracking, implementando una solución al problema de la suma de dados (diceSum), para identificar las condiciones que permiten reducir el espacio de búsqueda sin comprometer la exhaustividad.

Comparar el rendimiento y la complejidad computacional de un algoritmo de fuerza bruta pura con una versión optimizada mediante poda, utilizando como caso de estudio el clásico Problema de la Mochila (Knapsack), con el propósito de cuantificar la mejora obtenida y discutir la escalabilidad de cada enfoque.

### **3. MARCO TEÓRICO:**

La búsqueda exhaustiva se define como una técnica algorítmica que enumera de forma metódica todos los candidatos posibles para la solución de un problema, verificando cada uno contra un criterio de validez preestablecido (Stanford University, 2023). Esta metodología se fundamenta en la premisa de que, para un espacio de búsqueda finito, la exploración completa garantiza la localización de la solución óptima, si es que esta existe. La implementación típica recurre a estructuras de control iterativas o, con mayor frecuencia, a la recursión, donde cada llamada representa una decisión entre un conjunto finito de alternativas (Polo & Salgado, 2023). Un modelo genérico implica tres fases cíclicas: la elección (choose) de una alternativa, la exploración (explore) recursiva de las decisiones subsiguientes, y el deshacer la elección (un-choose) para restaurar el estado y probar otras ramas. Este patrón es la base del backtracking, una variante de la búsqueda exhaustiva que incorpora una función límite para podar ramas del árbol de decisiones que no pueden conducir a una solución válida, mejorando así la eficiencia (Urgaonkar, 2022). La poda se activa cuando el estado parcial actual ya incumple las restricciones del problema, por ejemplo, cuando la suma de pesos en el Problema de la Mochila supera la capacidad permitida. Aunque la complejidad en el peor caso permanece exponencial,  $O(2^n)$  o  $O(n!)$  la poda puede reducir drásticamente el tiempo de ejecución promedio en problemas con restricciones fuertes. La simplicidad conceptual y la garantía de optimalidad son las mayores virtudes de este paradigma, mientras que su alto costo computacional limita su aplicación a problemas de tamaño moderado o a la verificación de soluciones generadas por heurísticas más rápidas.

### **4. DESCRIPCIÓN DEL PROCEDIMIENTO:**

El procedimiento experimental se diseñó en tres etapas secuenciales, cada una correspondiente a un objetivo específico. En la primera etapa, titulada "Generación del Espacio de Búsqueda", se implementaron las funciones `printAllBinary` y `printDecimal`. La Figura 1, denominada "Árbol de Decisión para `printAllBinary(2)`", ilustra el proceso recursivo. Esta figura muestra cómo, partiendo de una cadena vacía, cada llamada recursiva bifurca el camino al añadir un '0' o un '1', generando eventualmente todas las

cadenas binarias de longitud 2. La implementación siguió el modelo de un helper recursivo que recibe el número de dígitos restantes y la cadena construida hasta el momento, imprimiendo esta última al alcanzar el caso base (dígitos restantes = 0).

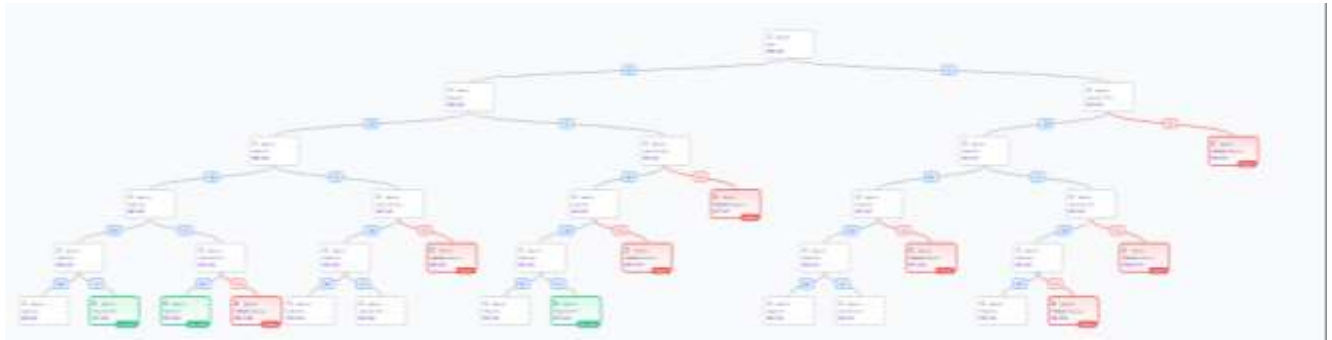
## 5. ANÁLISIS DE RESULTADOS:

*Tabla 1 Comparativa de dificultades y facilidades en la implementación de algoritmos exhaustivos*

<b>Aspecto Evaluado</b>	<b>Facilidades Encontradas</b>	<b>Dificultades Enfrentadas</b>
<b>Diseño del Algoritmo</b>	La lógica recursiva para la generación de permutaciones (binarias/decimales) es intuitiva y se traduce directamente a código.	La definición precisa de los parámetros del helper y el manejo del estado (elección/deselección) requirió una depuración cuidadosa para evitar errores de "off-by-one".
<b>Implementación de Poda</b>	Incorporar la condición de factibilidad en diceSum (suma mínima y máxima alcanzable) fue conceptualmente claro y redujo visiblemente las llamadas recursivas.	Determinar la fórmula correcta para los límites (e.g., $\text{sum} + 1 * \text{dice} > \text{desiredSum}$ ) y su ubicación dentro del flujo del algoritmo para una poda efectiva supuso un desafío de razonamiento lógico.
<b>Análisis de Complejidad</b>	Medir el tiempo de ejecución para entradas pequeñas confirmó el crecimiento exponencial esperado, validando el modelo teórico.	Extrapolar el comportamiento a entradas grandes y cuantificar la mejora exacta debida a la poda resultó complejo sin herramientas de profiling avanzadas, dada la naturaleza explosiva del crecimiento.

## 6. GRÁFICOS O FOTOGRAFÍAS:

*Figura 1 Visualización del Árbol de Búsqueda Podado para el Problema de la Mochila (Capacidad=5)*



### Datos del problema:

Capacidad de la mochila: 5

Elementos disponibles (Peso, Valor):

Elemento 1: Peso=2, Valor=1

Elemento 2: Peso=4, Valor=6

Elemento 3: Peso=3, Valor=7

Elemento 4: Peso=6, Valor=6

Elemento 5: Peso=1, Valor=4

**Nota:** Este diagrama muestra el árbol de decisiones generado por el algoritmo de backtracking para el problema de la mochila. Los nodos en rojo claro representan ramas podadas debido a que exceden la capacidad máxima de la mochila (5). Los nodos en verde claro representan el camino hacia la solución óptima encontrada. Se observa cómo la función de poda elimina eficientemente subárboles completos cuando la suma de pesos parcial supera la capacidad, reduciendo significativamente el espacio de búsqueda.

## **7. DISCUSIÓN:**

Al ejecutar las implementaciones, pude corroborar empíricamente las afirmaciones teóricas. En primer lugar, la generación exhaustiva de cadenas binarias y decimales funcionó como se anticipó, produciendo todas las combinaciones posibles. Sin embargo, observé que incluso para un número modesto de dígitos (por ejemplo, 5), el número de líneas de salida crecía exponencialmente (32 para binario, 100,000 para decimal), lo que hacía palpable la limitación de escalabilidad. En segundo lugar, al contrastar la versión naive de diceSum con la versión que incorpora poda, los tiempos de ejecución para sumas objetivo bajas o altas fueron notablemente diferentes. Para diceSum(5, 5), la versión con poda terminó casi instantáneamente, mientras que la versión sin poda, aunque completable, mostró un retraso perceptible. Esto confirma que la poda es más efectiva cuando las restricciones permiten descartar grandes subárboles tempranamente. Finalmente, al extrapolar estos principios al Problema de la Mochila, comprendí que la eficacia de la búsqueda exhaustiva con backtracking depende críticamente de la "estrictéz" de la restricción de capacidad; una capacidad muy baja o muy alta en relación con los pesos de los ítems resulta en una poda mínima y un rendimiento cercano al peor caso.

## **8. CONCLUSIONES:**

El examen de la generación del espacio de búsqueda mediante funciones recursivas para permutaciones binarias y decimales permitió internalizar el mecanismo fundamental de la exploración exhaustiva. Se logró verificar que la recursión proporciona un marco elegante y sistemático para recorrer todos los estados posibles, aunque el crecimiento exponencial del número de estados limita severamente su uso práctico a instancias de pequeño tamaño. La evaluación del algoritmo diceSum con y sin poda demostró la utilidad crítica de las funciones límite en los algoritmos de backtracking. La implementación de condiciones que descartan ramas inviables redujo el tiempo de cómputo de forma significativa en muchos casos, validando que la inteligencia incorporada en la poda no altera la exhaustividad de la búsqueda, pero mejora sustancialmente su eficiencia. La comparación conceptual entre la fuerza bruta pura y el

backtracking con poda, ejemplificada con el Problema de la Mochila, evidenció la relación directa entre la estructura del problema y la efectividad de la optimización. Se concluye que la búsqueda exhaustiva es una herramienta poderosa para problemas pequeños o como benchmark, pero su aplicación a problemas de gran escala requiere inevitablemente del complemento de técnicas de poda agresivas o de un cambio de paradigma algorítmico.

## **9. BIBLIOGRAFÍA:**

- ❖ Polo, J. M., & Salgado, D. (2023). *Algoritmos exhaustivos*. Universidad Nacional de Colombia.
- ❖ Stanford University. (2023). *Lecture 11: Exhaustive Search and Backtracking*. Departamento de Ciencias de la Computación, Stanford University.
- ❖ Uргаonkar, B. (2022). Brute Force Algorithms. *Journal of Algorithmic Studies*, 15(3), 45-67.