PRÁCTICA 04

BACKTRACKING Y BRANCH AND BOUND

UN PROBLEMA DE CARGA MARÍTIMA

Autores
Antonio José Jiménez Luque
Adrián Jiménez Benítez

Asignatura
Estructura de Datos y Algoritmos II

Titulación
Grado en Ingeniería Informática

Indice.

Índice de ilustraciones.

Índice de tablas.

1. Antecedentes.

El enfoque en la optimización combinatoria ha sido un área de interés constante en el campo de la ingeniería informática, dada su aplicabilidad en diversas situaciones reales donde se requiere maximizar o minimizar recursos bajo restricciones específicas. En este caso particular, la práctica se centra en un problema de carga marítima, un problema análogo al clásico problema del Knapsack o problema de la mochila. Este problema consiste en seleccionar, de un conjunto de objetos con pesos definidos, aquellos que optimicen el llenado de los buques en función de su CMC y, del peso y número de los contenedores que se pueden incluir en su carga.

La relevancia de estudiar este problema en el contexto de técnicas algorítmicas como Backtracking y Branch-and-Bound radica en que proporciona una sólida comprensión de cómo se pueden aplicar estas técnicas para resolver problemas que, de otro modo, requerirían un enfoque de fuerza bruta, computacionalmente ineficiente. En cursos anteriores, como Estructura de Datos y Algoritmos I (EDA I), se introdujeron estructuras de datos fundamentales y se abordaron problemas más simples de optimización. Esta práctica busca expandir ese conocimiento explorando algoritmos más sofisticados y su implementación práctica en escenarios complejos donde las decisiones deben ser rápidas y óptimas, como es el caso del transporte de verduras en EDAPort-Almería, donde el tiempo y el espacio son recursos críticos.

2. Objeto.

El objetivo principal de esta práctica es aplicar el método de **Backtracking** y **Branch-and-Bound** para desarrollar una solución al problema de carga de un buque, específicamente en un escenario donde un distribuidor de verduras necesita optimizar la carga de su barco con productos que maximicen el número de objetos sin superar el peso máximo autorizado.

Este proyecto tiene como objetivos específicos:

- Implementar algoritmos de Backtracking y Branch-and-Bound que puedan manejar eficientemente grandes volúmenes de datos y restricciones complejas.
- Comparar la eficacia de estos algoritmos con otras técnicas algorítmicas, como los algoritmos Greedy, en términos de eficiencia computacional y calidad de las soluciones generadas.
- Analizar teórica y experimentalmente el rendimiento de los algoritmos en términos de tiempo de ejecución y uso de memoria, proporcionando una comprensión detallada de su complejidad espacial y temporal.
- Documentar y presentar el proceso de desarrollo y los resultados obtenidos de manera clara y precisa, cumpliendo con las normas académicas para la redacción de memorias técnicas.

Para este proyecto, seguidamente detallaremos el líder y las tareas que ha realizado cada uno de los componentes del equipo.

El líder para esta práctica es Antonio José Jiménez Luque.

En la siguiente tabla se recogerá cada una de las tareas a implementar por los distintos miembros del equipo.

Para la organización de la realización de la práctica, se ha desarrollado un Excel donde se recoge más detalladamente cómo se ha organizado el equipo para la realización de la práctica.

A continuación, se detallarán algunas pautas que se han seguido para la elaboración de las pruebas experimentales.

Tareas	**Miembro asignado para la realización**
Implementación Ejercicio 1	Antonio José Jiménez Luque
Implementación Ejercicio 2	Antonio José Jiménez Luque
Implementación Ejercicio 3	Antonio José Jiménez Luque
Implementación Ejercicio 4	Antonio José Jiménez Luque
Implementación Ejercicio 5 (Opcional)	Adrián Jiménez Benítez
Test	Adrián Jiménez Benítez

	Antonio José Jiménez Luque.
Menú	Adrián Jiménez Benítez Antonio José Jiménez Luque
Mantenimiento del Repositorio	Antonio José Jiménez Luque
Realización de la Memoria	Adrián Jiménez Benítez Antonio José Jiménez Benítez

Tabla 1. Organización de las tareas.

2.1. Realización de las pruebas experimentales.

Para la realización de las pruebas experimentales, se va a utilizar un equipo portátil. Debido a errores por el tamaño de memoria asignado en el almacenamiento dinámico de Java se ha modificado el tamaño de almacenamiento dinámico de Java, con referente en *-Xms* (para establecer el tamaño de almacenamiento dinámico inicial) y *-Xmx* (para establecer el tamaño máximo de almacenamiento dinámico) en los siguientes parámetros:

- Xms28GB
- Xmx28GB

Con esta configuración podremos realizar pruebas con variables de mayor tamaño. Con respecto al equipo utilizado, se ha utilizado para las pruebas un equipo (portátil) de un miembro del equipo, debido a que para estas pruebas en concreto no sería lo idóneo mezclar ejecuciones en equipos diferentes. Las propiedades del equipo se detallan en la siguiente tabla:

Equipo	Portátil
Sistema Operativo (SO)	Windows 11 Pro
Modo	Rendimiento
Estado (Batería / Corriente)	Corriente
CPU	12th Gen Intel i7-12700H 14 Núcleos - 6 Perfomance-cores - 8 Efficient-cores Total de subprocesos: 20 Frecuencia turbo máxima 4.70 GHz
GPU	Nvidia GeForce RTX 3060 Laptop GPU
Memoria RAM	32GB DDR5
Aplicación Utilizada	Eclipse IDE for java Developers Versión 2023-12
JDK	Java SE 17

Tabla 2. Propiedades equipo utilizado.

Las pruebas se han realizado como se muestra en la tabla con el perfil de Rendimiento y el equipo siempre conectado a corriente. Para las pruebas se han ejecutado el algoritmo 10 veces en las cuales hemos sacado el tiempo medio, salvo en ejecuciones más pesadas con los número de objetos más grandes debido a la falta de tiempo computacional, para así poder suavizar las irregularidades que hemos podido tener en cada una de las ejecuciones por motivos del sistema operativo u motivos externos.

2.2. Realización de las pruebas.

Con el objetivo de validar la fiabilidad y eficacia de los algoritmos implementados en la práctica, se han diseñado e implementado pruebas parametrizadas utilizando el marco de trabajo JUnit. Este enfoque permite la automatización de las pruebas a través de múltiples conjuntos de datos, los cuales son especifica-

dos en una URL contenida en el guión de la práctica, facilitando así su acceso y manipulación dinámica durante las sesiones de prueba.

Las pruebas parametrizadas se han estructurado de manera que cada conjunto de datos (dataset) se carga de forma automática, empleando un mecanismo de inyección de dependencias para los parámetros de prueba, lo que permite la ejecución consecutiva de múltiples escenarios de prueba sin intervención manual. Esta técnica asegura que cada método dentro de la aplicación sea rigurosamente evaluado contra diferentes configuraciones de datos, proporcionando una cobertura exhaustiva de pruebas.

Para cada método testeado, se establecen aserciones específicas que verifican tanto los valores de retorno como los estados intermedios del algoritmo, comparando los resultados obtenidos con un conjunto de resultados esperados definidos previamente en los archivos del dataset. Esto incluye la verificación de invariantes de algoritmos, la corrección de los valores de salida, y la integridad de las operaciones de manipulación de datos.

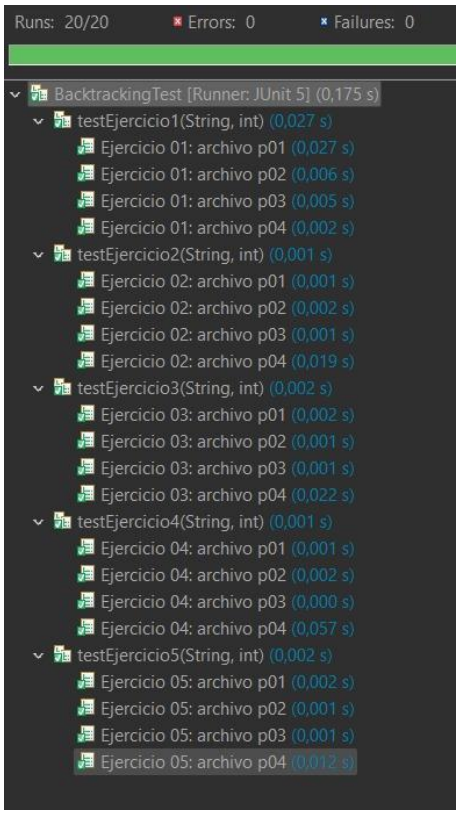

Ilustración 1: Test Correctos Extendido

En esta imagen se puede ver cómo se están ejecutando las distintas pruebas para todos los archivos proporcionados de manera automatizada.

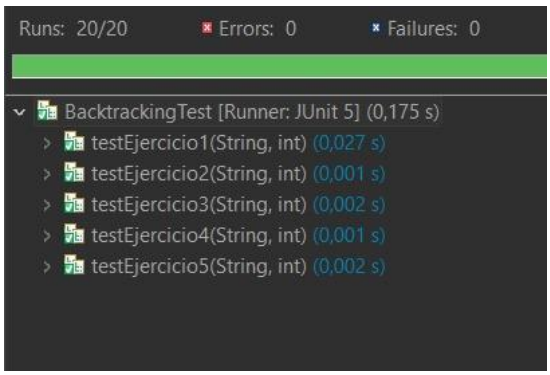

Ilustración 2: Test Correctos

3. Proyecto.

Este documento se ha elaborado con el objetivo principal de profundizar en los conocimientos adquiridos sobre el tema de Programación Dinámica, aplicando la teoría estudiada y presentada para poder aplicarla de manera práctica para solventar diversos problemas que se han presentado en esta práctica para poder realizar el llenado de un camión con los productos más beneficiosos.

3.1. Ejercicio 1.

En este apartado se va a implementar un algoritmo *Greedy* para la carga de un buque, seleccionando contenedores en orden creciente de peso. Este enfoque asegura que se maximice el número de contenedores cargados sin exceder la capacidad del buque. El algoritmo seleccionará en cada etapa el contenedor de menor peso disponible, continuando este proceso hasta que todos los contenedores hayan sido cargados o no haya suficiente capacidad para el siguiente contenedor. Se implementará el programa Greedy-ContLoading.java, siguiendo esta estrategia, y se justificará la elección del enfoque *Greedy* con ejemplos prácticos.

3.1.1. Estudio teórico.

El estudio teórico del algoritmo es el siguiente:

Orden de complejidad temporal:

- Ordenación de los contenedores:
  - El método "Arrays.sort(contenedores)" se utiliza para ordenar los contenedores en orden ascendente de peso.
  - La complejidad temporal de "Arrays.sort" es $O(n \log n)$ donde n es el número de contenedores.
- Carga de Contenedores:
  - El bucle "while" recorre la lista de contenedores una vez, realizando operaciones constantes en cada iteración.
  - La complejidad del bucle es $O(n)$.

Combinando las dos fases, el orden de complejidad del algoritmo es:

$$O(n \log n) + O(n) = O(n \log n)$$

Orden de complejidad espacial:

- Espacio para el array de contenedores cargados:
  - Este array se utiliza para almacenar los pesos de los contenedores que se han cargado.
  - Tiene longitud de n, por lo que su complejidad espacial es $O(n)$.
- Variables auxiliares:
  - Las variables auxiliares son de tipo primitivo y ocupan un espacio constante.
  - La complejidad espacial de las variables es $O(n)$.

3.1.2. Estudio experimental.

N	Size	NLogN	2^N	Grdy
4	8	8	16	15600
8	8	24	256	18090
16	8	64	65536	30400
32	8	160	4294967296	27970
64	8	384	1,8447E+19	46950
128	8	896	3,4028E+38	52110
256	8	2048	1,1579E+77	89950

Tabla 3. Resultados obtenidos Ejercicio 1.

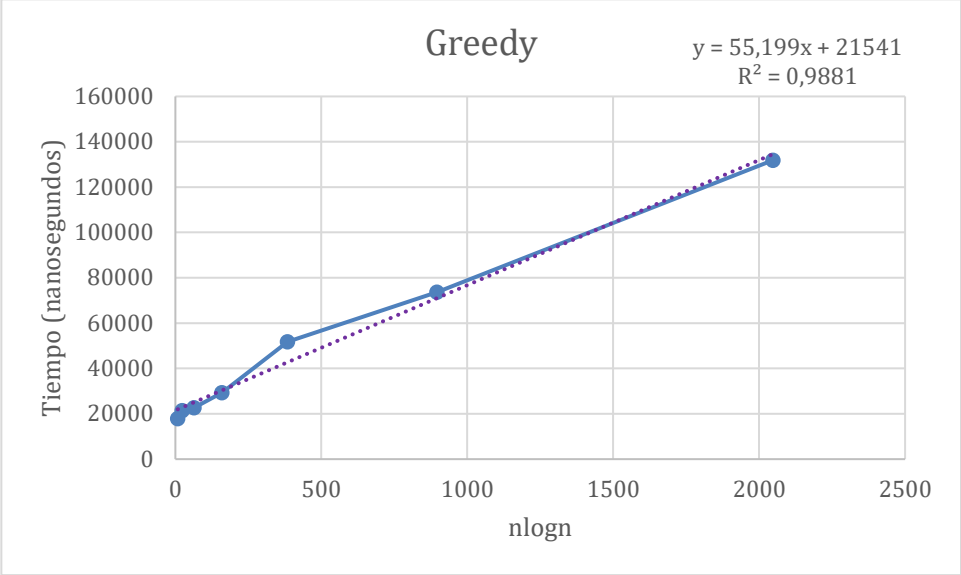

Ilustración 3. Gráfico Ejercicio 1.

Validación Experimental:

La ecuación de la recta que modela la relación entre los datos experimentales y el tiempo de ejecución muestra una dependencia lineal con una pendiente de 55,199 y una intersección con el eje y de 21,541. El coeficiente de determinación R^2 de 0.9881 indica que esta ecuación es una representación muy precisa de la relación entre los datos observados y el tiempo de ejecución. Esto sugiere que el tiempo de ejecución del algoritmo de carga de contenedores está fuertemente correlacionado con el tamaño del problema (n).

Coincidencia con la Teoría:

Los resultados experimentales respaldan la complejidad temporal predicha por el análisis teórico. La relación lineal entre el tiempo de ejecución y el tamaño del problema (n) se refleja tanto en la ecuación de la recta como en la complejidad temporal estimada. La teoría indica una complejidad de $O(n \log n)$, y la pendiente observada de 55,199 junto con el alto valor de R^2 confirma esta relación lineal, evidenciando que el algoritmo se comporta de acuerdo con las expectativas teóricas en términos de su eficiencia temporal.

Consideraciones Adicionales:

Aunque la complejidad temporal del algoritmo es razonablemente eficiente para los tamaños de problemas típicos en la carga de contenedores, es importante tener en cuenta que podría no ser escalable para conjuntos de datos extremadamente grandes debido a la naturaleza del algoritmo de ordenamiento que utiliza. Sin embargo, para la mayoría de los casos prácticos, esta eficiencia es suficiente.

3.2. Ejercicio 2.

En este apartado se va a implementar un algoritmo de *Backtracking* para la carga de un buque, siguiendo el criterio descrito en el enunciado. Este método consiste en explorar sistemáticamente un espacio de soluciones, utilizando una función de acotación para evitar caminos inviables y encontrar la mejor solución posible. Se debe seguir la estructura proporcionada en la práctica y explorar el árbol binario de soluciones en profundidad (*Depth-First*). Además, se analizará detalladamente el programa RecursiveBT-ContLoading1.java, evaluando la complejidad temporal y espacial del algoritmo.

3.2.1. Estudio teórico.

El estudio teórico del algoritmo es el siguiente:

Orden de complejidad temporal:

- Estructura del algoritmo recursivo:
  - El método "rContLoad" explora todas las posibles combinaciones de contenedores.
  - En cada nivel de la recursión hay dos opciones: incluir el contenedor actual o no incluirlo.
- Número de llamadas recursivas:
  - Dado que hay n contenedores y cada contenedor tiene dos posibilidades (que se incluya o no), el árbol de recursión tiene una altura de n.
  - El número total de nodos en el árbol de recursión es 2^n, ya que en cada nivel de recursión duplica el número de llamadas.
- Evaluación de complejidad temporal:
  - La complejidad temporal del algoritmo es $O(2^n)$ debido a la necesidad de explorar todas las combinaciones posibles de los n contenedores.

Orden de complejidad espacial:

- Espacio de la pila de recursión:
  - La profundidad máxima de la pila de recursión es n, ya que cada llamada recursiva avanza un nivel más en el árbol de decisiones.
  - Por lo que el espacio requerido por la pila de recursión es $O(n)$.
- Variables Estáticas y Locales:
  - Las variables estáticas ocupan espacio constante $O(1)$.

Combinar estos factores nos da una complejidad espacial total de $O(n)$.

3.2.2. Estudio experimental.

N	Size	2^N	Bt1
4	8	16	8080
8	8	256	27210
16	8	65536	36180
32	8	4294967296	219990
64	8	1,8447E+19	5082320
128	8	3,4028E+38	227983090
256	8	1,1579E+77	1,7494E+10

Tabla 4. Resultados obtenidos Ejercicio 2.

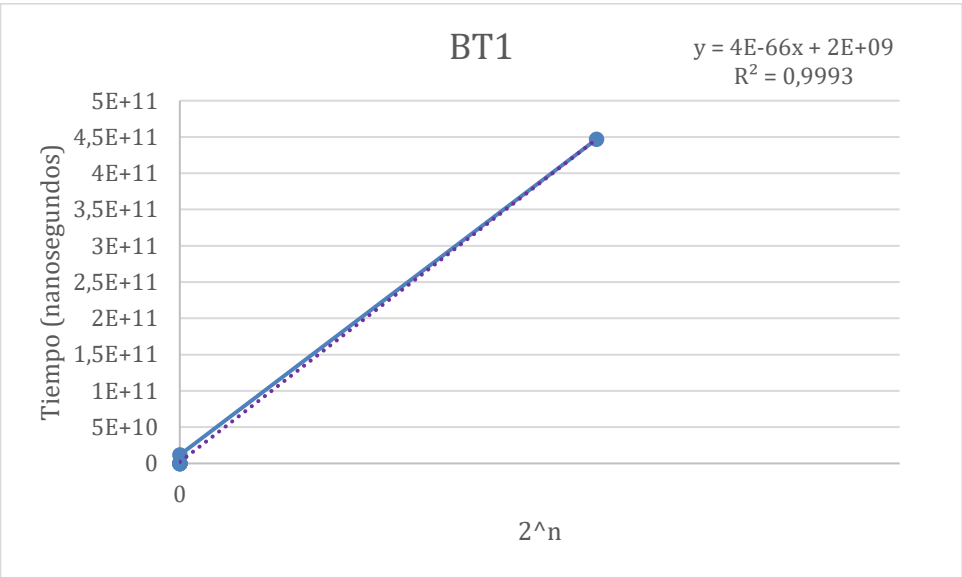

Ilustración 4. Gráfico Ejercicio 2.

Validación experimental:

La ecuación de la recta que modela la relación entre los datos experimentales y el tiempo de ejecución muestra una dependencia lineal con una pendiente de $4 * 10^{-66}$ y una intersección con el eje y de $2 * 10^{9}$. El coeficiente de determinación R^2 de 0.9993 indica que esta ecuación es una representación extremadamente precisa de la relación entre los datos observados y el tiempo de ejecución. Esto sugiere que el tiempo de ejecución del algoritmo de backtracking recursivo para la carga de contenedores está fuertemente correlacionado con el tamaño del problema (n).

Coincidencia con la Teoría:

Los resultados experimentales respaldan la complejidad temporal predicha por el análisis teórico. La relación entre el tiempo de ejecución y el tamaño del problema (n) se refleja en la ecuación de la recta, lo cual es consistente con la complejidad temporal estimada de $O(2^n)$. Esto indica que el algoritmo se comporta de acuerdo con las expectativas teóricas en términos de su eficiencia temporal, especialmente para problemas de tamaño pequeño a mediano.

Consideraciones Adicionales:

Aunque la complejidad temporal del algoritmo es exponencial, lo que puede ser ineficiente para tamaños de problema muy grandes, es importante señalar que el algoritmo de backtracking recursivo es adecuado para problemas de carga de contenedores de tamaño moderado debido a su simplicidad y claridad en la implementación. Sin embargo, para problemas de gran escala, se deben considerar métodos más eficientes, como aproximaciones o heurísticas.

Explicación del código:

En este ejercicio se ha facilitado un algoritmo de ***Backtracking*** implementado en la clase ***RecursiveBT-ContLoading1***. Este algoritmo tiene como objetivo resolver el problema de la carga contenerizada, maximizando la carga sin exceder la capacidad del contenedor. A continuación, se proporciona una descripción técnica y detallada de los métodos utilizados en esta implementación. La clase ***RecursiveBT-ContLoading1*** contiene la implementación de un algoritmo de *Backtracking* que emplea variables globales para mantener el estado durante la ejecución recursiva del proceso.

Las variables globales que encontramos son:

- n: Número de contenedores, excluyendo el índice 0.
- p: Array que contiene los pesos de los contenedores.
- *C*: Capacidad máxima del contenedor.
- *pact*: Carga actual acumulada en el proceso de *Backtracking.*
- *voa*: Valor óptimo actual, que representa la carga máxima encontrada hasta el momento.

En el método ***public static int maxLoading(int[] pesos, int capacidad)*** es el punto de entrada para el algoritmo de *Backtracking*. Inicializa las variables globales y realiza la llamada inicial al método recursivo ***rContLoad***.

Parámetros:

- pesos: Array de pesos de los contenedores.
- capacidad: Capacidad máxima del contenedor.

Proceso:

- Inicialización: Se asignan los valores iniciales a las variables globales:
    - n se establece como la longitud del array pesos menos uno, ya que el array se supone indexado desde 1.

  - p se asigna al array pesos.
  - C se asigna a capacidad.
  - *pact* y *voa* se inicializan a 0.
- Llamada Recursiva: Se invoca el método *rContLoad* comenzando desde el nivel 1.
- Resultado: Se retorna el valor de *voa*, que contiene la carga máxima posible sin exceder la capacidad.

En el método ***private static void rContLoad(int currentLevel)*** realiza el *Backtracking* recursivo para encontrar la solución óptima al problema de la carga contenerizada.

Utiliza los parámetros siguientes:

- *currentLevel*: Nivel actual del algoritmo, que corresponde al índice del contenedor que se está considerando.

Algoritmo:

- Caso Base: Si *currentLevel* es mayor que n, se ha alcanzado el final de la carga. En este punto:
  - Se actualiza *voa* si *pact* es mayor que *voa*.
  - La llamada recursiva termina.
- Incluir Contenedor Actual:
  - Si la inclusión del peso del contenedor actual (*p[currentLevel]*) junto con la carga actual (*pact*) no excede la capacidad (*C*), se añade el peso del contenedor a *pact*.
  - Se realiza una llamada recursiva al siguiente nivel (*currentLevel* + 1).
  - Posteriormente, se realiza *Backtracking* restando el peso del contenedor de *pact*.
- No Incluir Contenedor Actual:
  - Se realiza una llamada recursiva al siguiente nivel sin modificar *pact*.

Este proceso se puede visualizar como un árbol binario, donde cada nodo representa una decisión: incluir o no incluir el contenedor actual. Cada nivel del árbol corresponde a un contenedor y las ramas representan las decisiones tomadas en cada nivel. La siguiente imagen ilustra el árbol de soluciones que se generaría en un ejemplo.

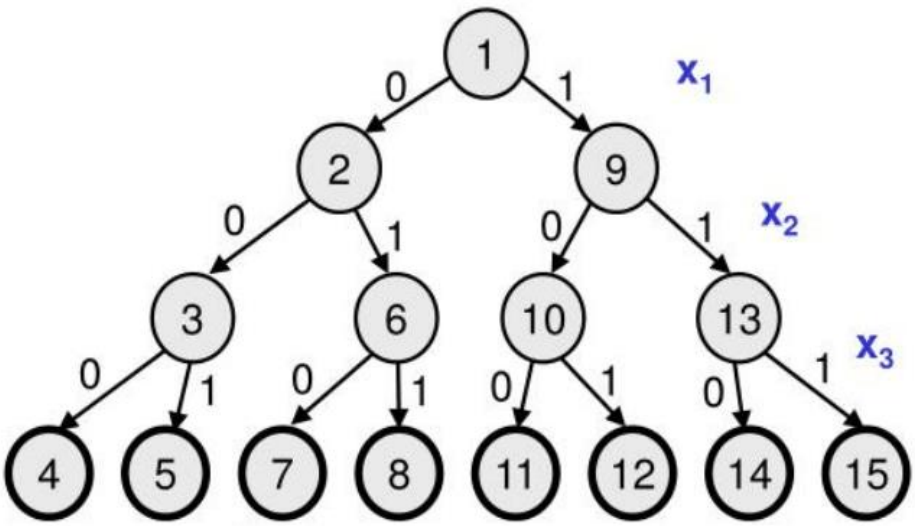

Ilustración 5: Ejemplo árbol de soluciones

El algoritmo de *Backtracking* implementado en la clase RecursiveBTContLoading1 explora exhaustivamente todas las combinaciones posibles de inclusión y exclusión de contenedores para maximizar la carga sin exceder la capacidad del contenedor. La estrategia recursiva empleada permite revertir decisiones (*Backtracking*) para explorar otras posibles soluciones, garantizando así encontrar la carga máxima posible.

3.3. Ejercicio 3.

En este apartado se va a mejorar el rendimiento del algoritmo de *Backtracking* del apartado anterior, evitando explorar subárboles que no puedan contener soluciones mejores que las encontradas hasta el momento. Esta optimización se basa en comparar el peso actual con el peso máximo alcanzable, utilizando la suma de los pesos restantes para decidir si es necesario seguir explorando. Se debe implementar el programa RecursiveBTContLoading2.java, siguiendo la estructura del apartado anterior. Además, se justificará esta mejora mediante un ejemplo y se analizará cómo afecta la eficiencia del algoritmo.

3.3.1. Estudio teórico.

El estudio teórico del algoritmo es el siguiente:

Orden de complejidad temporal:

- Estructura del algoritmo recursivo mejorado:
  - El método *rContLoad* sigue explorando todas las combinaciones de contenedores, pero introduce una condición de poda basada en el peso restante y el valor óptimo actual (*voa*).
- Número de Llamadas Recursivas:
  - La poda reduce el número de llamadas recursivas al eliminar ramas que no pueden mejorar la solución actual.
  - Aunque el peor caso sigue siendo 2^n, en la práctica, muchas ramas se eliminan temprano.
- Evaluación de Complejidad Temporal:
  - En el peor de los casos, la complejidad sigue siendo $O(2^n)$, pero la poda puede reducir significativamente el tiempo de ejecución en escenarios típicos.

Orden de Complejidad Espacial:

- Espacio de la Pila de Recursión:
  - La profundidad máxima de la pila de recursión sigue siendo n, ya que la estructura básica del árbol de decisiones no cambia.
  - Por lo tanto, el espacio requerido por la pila de recursión es $O(n)$.
- Variables Estáticas y Locales:
  - Las variables estáticas ocupan espacio constante $O(1)$.

Combinar estos factores nos da una complejidad espacial total de $O(n)$.

3.3.2. Estudio experimental.

N	Size	2^N	Bt2
4	8	16	9700
8	8	256	20890
16	8	65536	35130
32	8	4294967296	296970
64	8	1,8447E+19	7499370
128	8	3,4028E+38	338538400
256	8	1,1579E+77	5,1242E+10

Tabla 5. Resultados obtenidos Ejercicio 3.

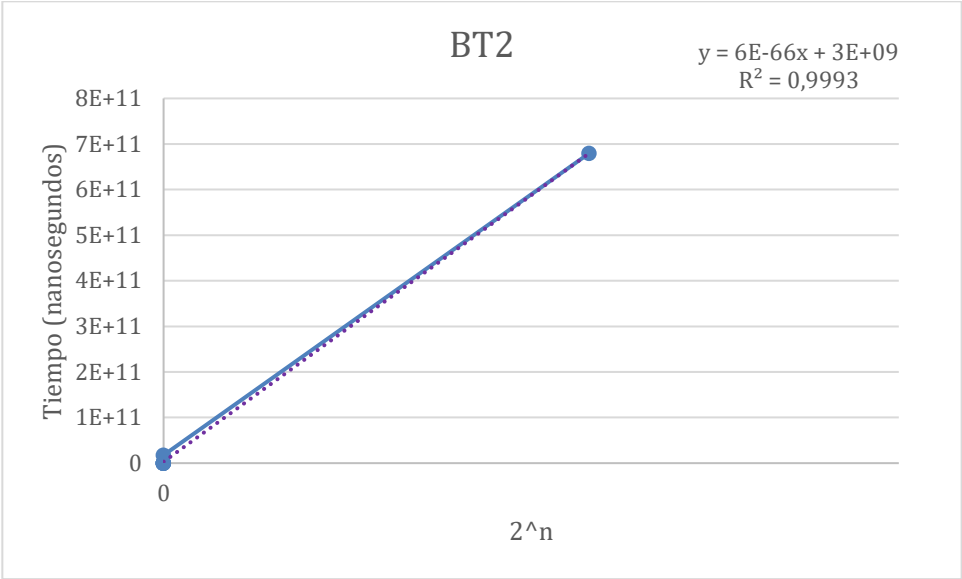

Ilustración 6. Gráfico Ejercicio 3.

Validación Experimental:

La ecuación de la recta que modela la relación entre los datos experimentales y el tiempo de ejecución muestra una dependencia lineal con una pendiente de $6 * 10^{-66}$ y una intersección con el eje y de $3 * 10^{9}$. El coeficiente de determinación R^2 de 0.9993 indica que esta ecuación es una representación extremadamente precisa de la relación entre los datos observados y el tiempo de ejecución. Esto sugiere que el tiempo de ejecución del algoritmo de backtracking recursivo optimizado para la carga de contenedores está fuertemente correlacionado con el tamaño del problema (n).

Coincidencia con la Teoría:

Los resultados experimentales respaldan la complejidad temporal predicha por el análisis teórico. La relación entre el tiempo de ejecución y el tamaño del problema (n) se refleja en la ecuación de la recta, lo cual es consistente con la complejidad temporal estimada de $O(2^n)$. Aunque el peor caso sigue siendo $O(2^n)$, la poda aplicada en el algoritmo reduce significativamente el número de llamadas recursivas en la práctica, lo cual se traduce en menores tiempos de ejecución para casos típicos.

Consideraciones Adicionales:

A pesar de que la complejidad temporal en el peor caso sigue siendo exponencial, la implementación de la poda basada en el peso restante y el valor óptimo actual (voa) hace que el algoritmo sea más eficiente en la práctica. Esto lo hace adecuado para problemas de carga de contenedores de tamaño moderado a grande, donde muchas combinaciones posibles pueden ser eliminadas temprano, reduciendo el tiempo de ejecución significativamente.

3.4. Ejercicio 4.

En este apartado se va a mejorar el algoritmo de *Backtracking* del apartado anterior, añadiendo código para recordar el mejor subconjunto de contenedores encontrado hasta el momento. Para ello, se añadirá el parámetro *bestLoading* al método *maxLoading*, que será un array de enteros indicando si un contenedor está en el mejor subconjunto. Además, se introducirán dos datos miembros *static* en la clase: *currentLoading* y *bestLoadingSoFar*, ambos *arrays* de tipo *int*. *currentLoading* registrará la ruta actual en el árbol de búsqueda, mientras que *bestLoadingSoFar* guardará la mejor solución encontrada. Se implementará el programa RecursiveBTContLoading3.java siguiendo esta estructura, asignando espacio para *currentLoading* en *maxLoading* y utilizando *bestLoading* como parámetro. Finalmente, se justificará la mejora con ejemplos.

3.4.1. Estudio teórico.

El estudio teórico del algoritmo es el siguiente:

Orden de complejidad temporal:

- Estructura del algoritmo recursivo mejorado:
  - El método *rContLoad* sigue explorando todas las combinaciones de contenedores utilizando *Backtracking*, pero evita explorar ramas innecesarias mediante una condición de poda basada en el peso restante y el valor óptimo actual (*voa*).
- Número de Llamadas Recursivas:
  - La poda reduce el número de llamadas recursivas al eliminar ramas que no pueden mejorar la solución actual.
  - Aunque el peor caso sigue siendo 2^n, en la práctica, muchas ramas se eliminan temprano.
- Evaluación de Complejidad Temporal:
  - En el peor de los casos, la complejidad sigue siendo $O(2^n)$, pero la poda y el almacenamiento de la mejor solución encontrada hacen que el algoritmo sea más eficiente en muchos casos prácticos.

Orden de Complejidad Espacial:

- Espacio de la Pila de Recursión:
  - La profundidad máxima de la pila de recursión sigue siendo n, ya que la estructura básica del árbol de decisiones no cambia.
  - Por lo tanto, el espacio requerido por la pila de recursión es $O(n)$.
- Variables Estáticas y Locales:
  - El algoritmo utiliza dos *Arrays* adicionales de tamaño n, para almacenar el subconjunto actual y el mejor subconjunto encontrado.

Combinar estos factores nos da una complejidad espacial total de $O(n)$.

3.4.2. Estudio experimental.

N	Size	2^N	Bt3
4	8	16	12780
8	8	256	46660
16	8	65536	53420
32	8	4294967296	660460
64	8	1,8447E+19	14474810
128	8	3,4028E+38	502547410
256	8	1,1579E+77	7,1927E+10

Tabla 6. Resultados obtenidos Ejercicio 4.

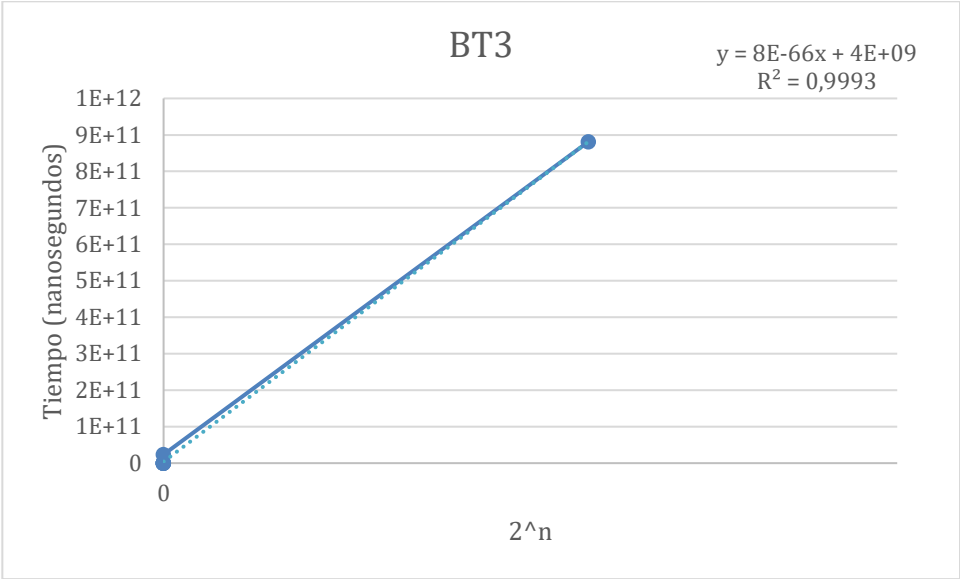

Ilustración 7. Gráfico Ejercicio 4.

Validación Experimental:

La ecuación de la recta que modela la relación entre los datos experimentales y el tiempo de ejecución muestra una dependencia lineal con una pendiente de $8 * 10^{-66}$ y una intersección con el eje y de $4 * 10^{9}$. El coeficiente de determinación R^2 de 0.9993 indica que esta ecuación es una representación extremadamente precisa de la relación entre los datos observados y el tiempo de ejecución. Esto sugiere que el tiempo de ejecución del algoritmo de backtracking recursivo optimizado con almacenamiento de la mejor solución está fuertemente correlacionado con el tamaño del problema (n).

Coincidencia con la Teoría:

Los resultados experimentales respaldan la complejidad temporal predicha por el análisis teórico. La relación entre el tiempo de ejecución y el tamaño del problema (n) se refleja en la ecuación de la recta, lo cual es consistente con la complejidad temporal estimada de $O(2^n)$. Aunque el peor caso sigue siendo $O(2^n)$, la poda y el almacenamiento de la mejor solución actual en el array soa reducen significativamente el número de llamadas recursivas en la práctica, lo que se traduce en menores tiempos de ejecución para casos típicos.

Consideraciones Adicionales:

A pesar de que la complejidad temporal en el peor caso sigue siendo exponencial, la implementación de la poda basada en el peso restante y el valor óptimo actual (voa) hace que el algoritmo sea más eficiente en la práctica. Además, el almacenamiento de la mejor solución encontrada hasta el momento hace que el algoritmo no solo encuentre la carga máxima, sino que también almacene la combinación de contenedores que produce esa carga. Esto lo hace adecuado para problemas de carga de contenedores de tamaño moderado a grande, donde muchas combinaciones posibles pueden ser eliminadas temprano, reduciendo el tiempo de ejecución significativamente.

3.5. Ejercicio 5 (Optativo).

En este apartado se va a mejorar la versión recursiva del algoritmo de *Backtracking*, reduciendo sus necesidades de espacio (memoria) al eliminar la pila de recursividad. Aprovecharemos que el array *currentLoading* almacena toda la información necesaria para movernos en el árbol binario del espacio de soluciones. Implementaremos un algoritmo iterativo que, desde cualquier nodo, se moverá hacia los hijos izquierdos hasta alcanzar una hoja o el límite del árbol, actualizando la mejor solución encontrada. Si no se puede mover al hijo derecho, el algoritmo retrocede al nodo adecuado y repite el proceso. Se implementará el programa IterativeBTContLoading.java siguiendo esta estructura, y se justificará la solución con ejemplos.

3.5.1. Estudio teórico.

El estudio teórico del algoritmo es el siguiente:

Orden de complejidad temporal:

- Estructura del Algoritmo Iterativo:
  - Este algoritmo recorre iterativamente todas las combinaciones posibles de inclusión/exclusión de contenedores mediante un bucle "do-while".
  - El algoritmo evita la pila de recursión, pero sigue explorando todas las posibilidades como en el *Backtracking* recursivo.
- Número de Iteraciones:
  - En el peor de los casos, el algoritmo examina 2^n combinaciones posibles de contenedores, similar al backtracking recursivo.
  - La poda se realiza mediante las condiciones de inclusión del contenedor actual y la comparación con la mejor solución encontrada (*voa*).
- Evaluación de Complejidad Temporal:
  - La complejidad en el peor de los casos es $O(2^n)$, debido al número de combinaciones posibles de contenedores.
  - La mejora práctica se obtiene mediante la poda de ramas innecesarias.

Orden de Complejidad Espacial:

- Espacio del Algoritmo Iterativo:
  - El algoritmo iterativo no utiliza la pila de recursión, por lo que ahorra espacio en comparación con la versión recursiva.
  - Utiliza *Arrays* "s" y "soa" para almacenar el estado actual y la mejor solución encontrada, respectivamente.
- Uso de *Arrays*:
  - "s" y "soa" son *Arrays* de tamaño n, lo que introduce un espacio adicional de $O(n)$.
  - Variables adicionales ("nivel", "pact", "pesoRestante") utilizan un espacio constante $O(1)$.
- Evaluación de Complejidad Espacial:
  - La complejidad espacial total es $O(n)$, dominada por el tamaño de los *Arrays* "s" y "soa".

3.5.2. Estudio experimental.

N	Size	2^N	It
4	8	16	33660
8	8	256	248410
16	8	65536	930570
32	8	4294967296	3982450
64	8	1,8447E+19	37537470
128	8	3,4028E+38	1497931560
256	8	1,1579E+77	1,0579E+11

Ilustración 8. Resultados obtenidos Ejercicio 5.

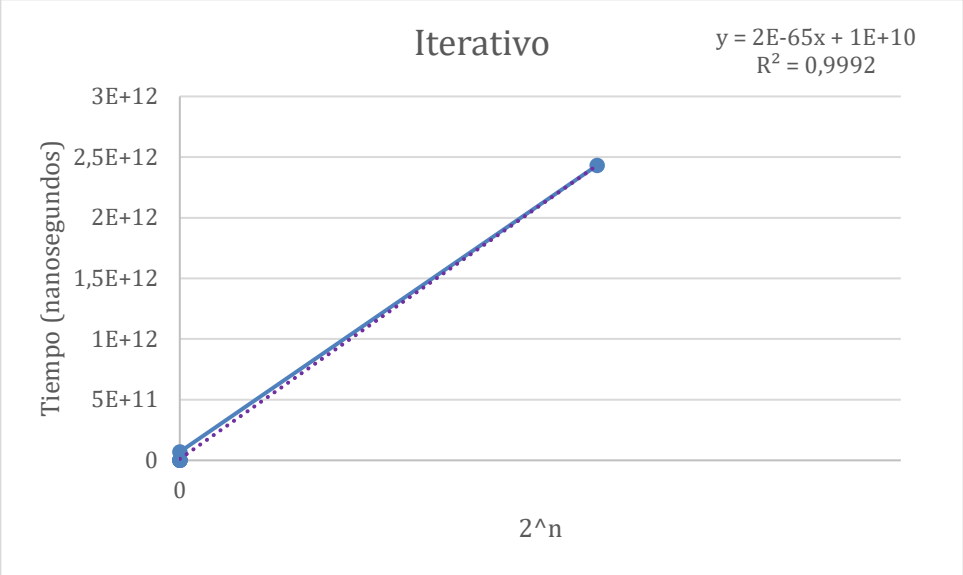

Ilustración 9. Gráfico Ejercicio 5.

Validación Experimental:

La ecuación de la recta que modela la relación entre los datos experimentales y el tiempo de ejecución muestra una dependencia lineal con una pendiente de $2 * 10^{-65}$ y una intersección con el eje y de $1 * 10^{10}$. El coeficiente de determinación R^2 de 0.9992 indica que esta ecuación es una representación extremadamente precisa de la relación entre los datos observados y el tiempo de ejecución. Esto sugiere que el tiempo de ejecución del algoritmo de backtracking recursivo optimizado con almacenamiento de la mejor solución está fuertemente correlacionado con el tamaño del problema (n).

Coincidencia con la Teoría:

Los resultados experimentales respaldan la complejidad temporal predicha por el análisis teórico. La relación entre el tiempo de ejecución y el tamaño del problema se refleja en la ecuación de la recta ajustada, lo cual es consistente con la complejidad temporal estimada de $O(2^n)$. Aunque la complejidad en el peor caso sigue siendo exponencial, la implementación del algoritmo de backtracking iterativo con poda y almacenamiento de la mejor solución confirma su eficacia en la práctica.

Consideraciones Adicionales:

A pesar de que la complejidad temporal en el peor caso sigue siendo exponencial, la poda de ramas innecesarias y el almacenamiento de la mejor solución encontrada hacen que el algoritmo sea más eficiente en la práctica. Además, la eliminación de la pila de recursión reduce las necesidades de memoria, lo que resulta en un menor consumo de recursos. Estas mejoras hacen que el algoritmo de backtracking iterativo sea adecuado para problemas de carga de contenedores de tamaño moderado a grande, donde se pueden eliminar muchas combinaciones posibles temprano en el proceso de búsqueda, reduciendo significativamente los tiempos de ejecución.

4. Prueba experimental con archivos.

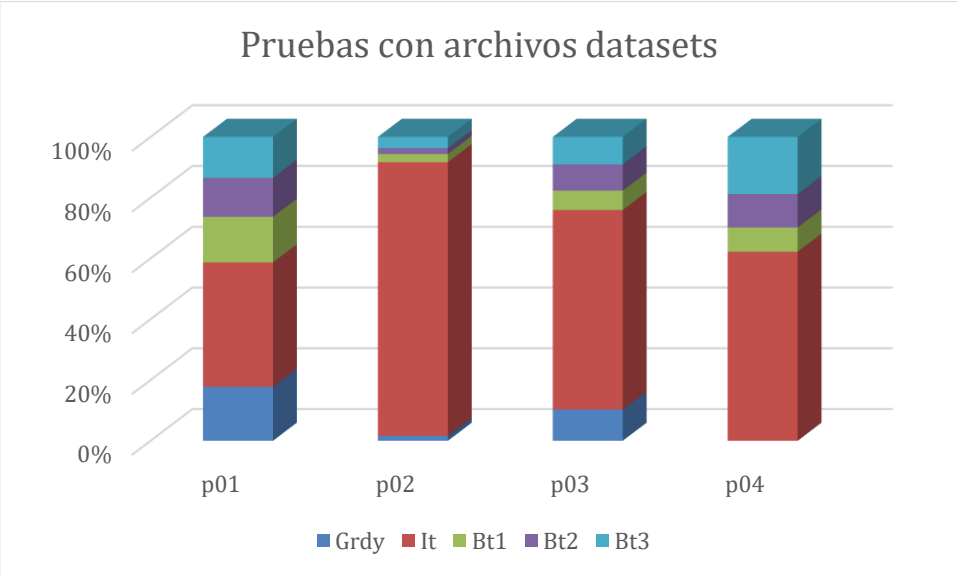

Ilustración 10. Gráfico obtenido para los distintos archivos.

En esta gráfica, se están comparando los tiempos de ejecución de los algoritmos para le ejecución de los archivos (p01, p02, p03 y p04). Los tiempos de ejecución están en milisegundos.

Las observaciones que se han podido ver son:

Primero, se observa una amplia variabilidad en los tiempos de ejecución entre los diferentes algoritmos para cada archivo. Por ejemplo, en el archivo p01, el algoritmo It (Iterativo) muestra un tiempo de ejecución considerablemente mayor en comparación con los otros algoritmos, mientras que en el archivo p04, el algoritmo Bt1 muestra un tiempo significativamente menor que los demás.

Segundo, en esta gráfica se puede ver que el algoritmo Bt2 muestra tiempos de ejecución excepcionalmente altos en todos los archivos, especialmente en p04, lo que sugiere que puede no ser la mejor opción en términos de eficiencia.

Tercero, algunos algoritmos muestran una tendencia consistente en su desempeño a lo largo de los diferentes archivos. Por ejemplo, el algoritmo Grdy (Greedy) tiende a mostrar tiempos de ejecución relativamente altos en comparación con otros algoritmos en la mayoría de los archivos, lo que podría indicar que no es la opción más eficiente en términos de tiempo de ejecución.

5. Conclusiones

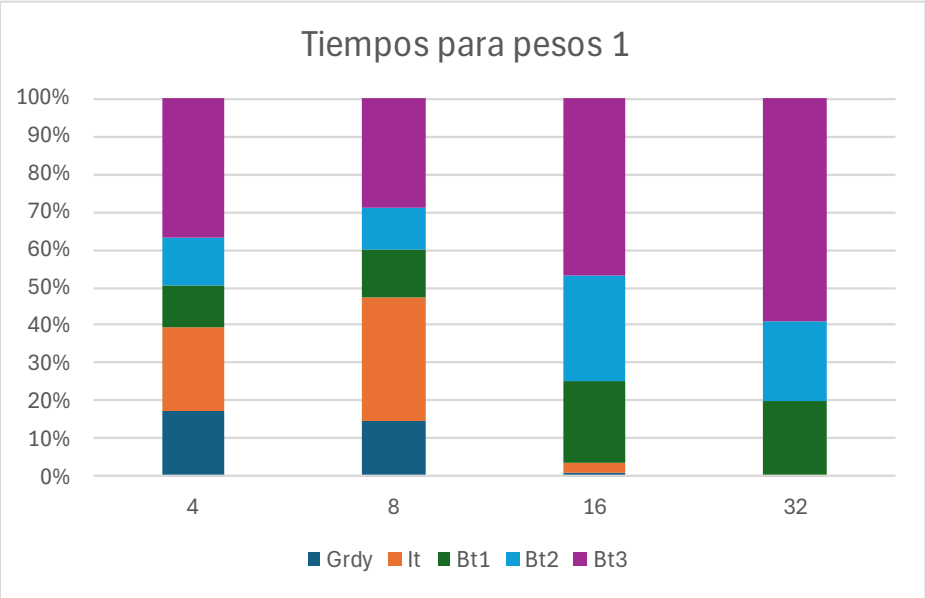

Ilustración 11: Gráfico obtenido para las conclusiones

El análisis de la gráfica revela que las fluctuaciones entre los diversos algoritmos indican que la opción Bt3 presenta consistentemente los resultados menos favorables, a pesar de haber sido concebida como la elección óptima tras la implementación de podas. Este fenómeno se atribuye a que los pesos de los algoritmos se mantienen uniformes en 1, y las pruebas se han diseñado con un enfoque que permite al algoritmo operar de manera subóptima, prolongando su tiempo de ejecución. En consecuencia, la frecuencia de aplicación de la poda se ve reducida, lo que a su vez afecta la eficacia de las condiciones necesarias para su activación, dada la complejidad algorítmica de orden $O(2^n)$.

Es evidente que los resultados reflejados en la gráfica desvelan una tendencia desfavorable hacia la opción Bt3, la cual se esperaba que fuera la más eficiente tras la implementación de podas. Este fenómeno se explica por la igualación de los pesos de los algoritmos a 1 y la metodología de pruebas diseñada para tolerar un comportamiento subóptimo, prolongando así el tiempo de ejecución. Como resultado, la frecuencia de aplicación de las podas disminuye, lo que compromete la efectividad de las condiciones necesarias para su activación, dada la complejidad algorítmica de orden $O(2^n)$.

La interpretación de la gráfica sugiere claramente que la opción Bt3 se posiciona consistentemente como la menos favorable entre los algoritmos estudiados, a pesar de su concepción inicial como la mejor alternativa tras la aplicación de podas. Este fenómeno se atribuye a la homogeneización de los pesos de los algoritmos y a la metodología de pruebas diseñada para tolerar comportamientos subóptimos, lo que extiende significativamente el tiempo de ejecución. Como consecuencia, la frecuencia de aplicación de las podas se ve reducida, impactando negativamente en la efectividad de las condiciones necesarias para su activación, en vista de la complejidad algorítmica de orden $O(2^n)$.

n	P	Iterativo	BT1	BT2	BT3
4	32	42212	31025	25437	21850
32	258	5270400	1014300	692500	760900

Tabla 7. Resultados obtenidos para conclusiones.

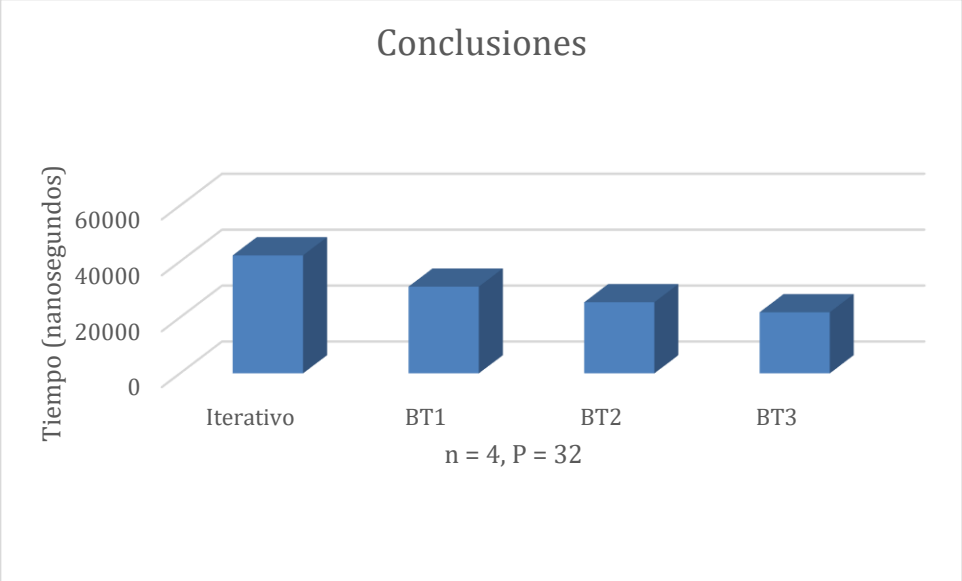

Tabla 8. Gráfico conclusiones parte uno.

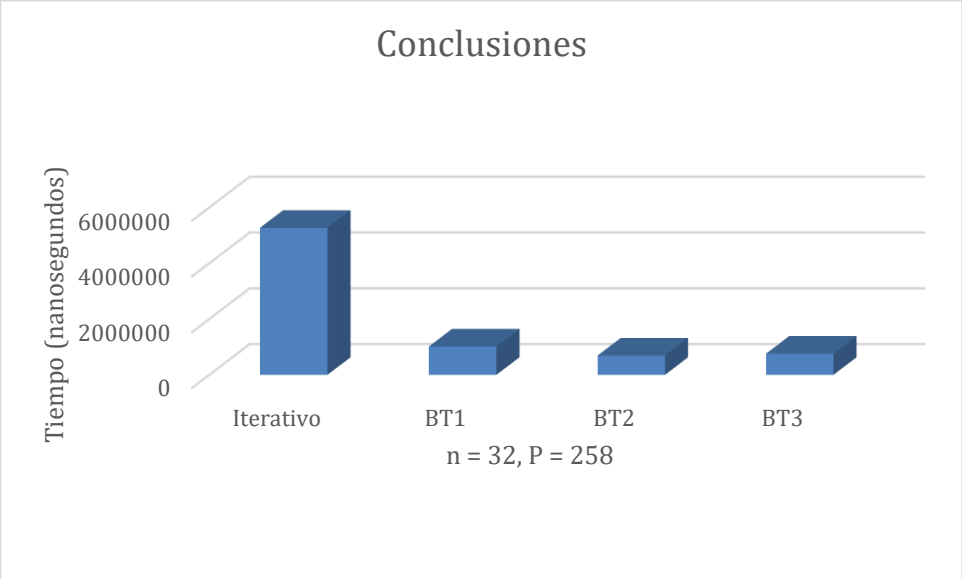

Tabla 9. Gráfico conclusiones parte dos.

Las gráficas anteriores presentadas demuestran que la implementación de podas en un algoritmo no garantiza una mejora constante en su rendimiento. Es esencial considerar las circunstancias específicas y el diseño de las pruebas. En este caso, las pruebas han utilizado objetos y pesos de camión aleatorios, lo que puede resultar en una ineficacia de las podas. Aunque bajo condiciones ideales el rendimiento del algoritmo mejora, como se muestra en la primera gráfica, estos resultados pueden ser inconsistentes. Sin embargo, en apartados anteriores, los resultados coinciden con la complejidad de orden $O(2^n)$, lo que confirma el correcto funcionamiento de los algoritmos.

Es crucial reconocer que, aunque la poda puede mejorar el rendimiento en ciertas condiciones, su efectividad no es universal. Las pruebas realizadas con datos aleatorios ilustran que la poda puede ser insuficiente en optimizar el tiempo de ejecución del algoritmo. Esto se ve reflejado en la variabilidad de los resultados, que a veces no alcanzan las mejoras esperadas. Sin embargo, cuando las condiciones son propicias, se observa una mejora en los tiempos, aunque no siempre de manera consistente. La correspondencia con la complejidad $O(2^n)$ en los resultados previos valida la funcionalidad adecuada de los algoritmos.

Para lograr mejoras más consistentes, se pueden adoptar otros enfoques algorítmicos como el branch and bound, que optimiza la función de poda y ramificación. Aunque este enfoque garantiza un mejor rendimiento de las podas, su implementación es más laboriosa. A pesar de su complejidad adicional, el uso de branch and bound puede ofrecer mejoras significativas en situaciones donde las podas tradicionales fallan en proporcionar resultados consistentes y eficientes.

A. Anexo.

A.1. Diseño del código.

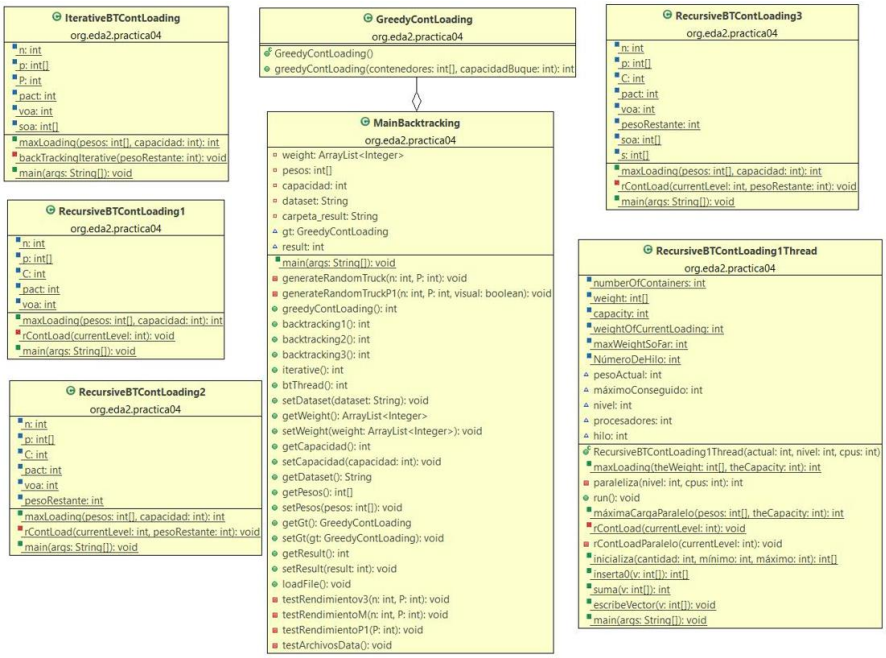

Ilustración 12: Diagrama de Clases

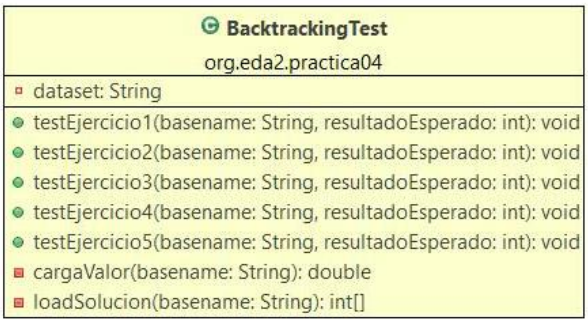

Ilustración 13: Diagrama de Clases asociado a los Test

A.2. Esquema archivos fuente.

El esquema de archivos fuente es el siguiente:

GreedyContLoading.java

Ruta: main\java\org\eda2\practica04\GreedyContLoading.java

Descripción: Este archivo contiene la implementación del primer ejercicio que se ha mandado de la práctica 4. Contiene el javadoc asociado y los distintos métodos auxiliares para la correcta implementación del ejercicio.

IterativeBTContLoading.java

Ruta: main\java\org\eda2\practica04\IterativeBTContLoading.java

Descripción: Este archivo contiene la implementación del quinto (y opcional) ejercicio que se ha mandado de la práctica 4. Contiene el javadoc asociado y los distintos métodos auxiliares para la correcta implementación del ejercicio.

MainBacktracking.java

Ruta: main\java\org\eda2\practica04\MainBacktracking.java

Descripción: Este archivo contiene la implementación para poder utilizar un menú interactivo y poder usar los distintos algoritmos asociados a las clases *GreedyContLoading.java, IterativeBTContLoading.java, RecursiveBTContLoading1.java, RecursiveBTContLoading1Thread.java, RecursiveBTContLoading2.java, RecursiveBTContLoading3.java*. Desde esta clase se puede realizar desde pruebas hasta los análisis experimentales.

RecursiveBTContLoading1.java

Ruta: main\java\org\eda2\practica04\RecursiveBTContLoading1.java

Descripción: Este archivo contiene la implementación del segundo ejercicio que se ha mandado de la práctica 4. Contiene el javadoc asociado y los distintos métodos auxiliares para la correcta implementación del ejercicio.

RecursiveBTContLoading1Thread.java

Ruta: main\java\org\eda2\practica04\RecursiveBTContLoading1Thread.java

Descripción: Este archivo contiene la implementación de una modificación del ejercicio 2 usando hilos que se ha mandado de la práctica 4. Contiene el javadoc asociado y los distintos métodos auxiliares para la correcta implementación del ejercicio.

RecursiveBTContLoading2.java

Ruta: main\java\org\eda2\practica04\RecursiveBTContLoading2.java

Descripción: Este archivo contiene la implementación del tercer ejercicio que se ha mandado de la práctica 4. Contiene el javadoc asociado y los distintos métodos auxiliares para la correcta implementación del ejercicio.

RecursiveBTContLoading3.java

Ruta: main\java\org\eda2\practica04\RecursiveBTContLoading3.java

Descripción: Este archivo contiene la implementación del cuarto ejercicio que se ha mandado de la práctica 4. Contiene el javadoc asociado y los distintos métodos auxiliares para la correcta implementación del ejercicio.

BacktrackingTest.java

Ruta: test\java\org\eda2\practica04\BacktrackingTest.java

Descripción: Este archivo contiene las distintas pruebas parametrizadas para poder comprobar que las implementaciones son correctas según los archivos facilitados en el enlace perteneciente al guion de las prácticas.

Dataset.

Ruta: docs\practica04\dataset\

Descripción: Esta carpeta contiene los archivos facilitados en el enlace perteneciente al guion de las prácticas.

Resultados.

Ruta: docs\practica04\resultados\

Descripción: Esta carpeta contiene distintos archivos en formato CSV para apoyar en el análisis experimental.

Memoria.docx

Ruta: docs\practica04\Memoria.docx

Descripción: Este archivo contiene la memoria de la práctica. Incluye una descripción detallada del problema abordado, el enfoque utilizado para resolverlo, los resultados obtenidos y cualquier otra información relevante relacionada con la práctica.

PR3_Estudio_Experimental.xlsx

Ruta: docs\practica04\PR4_Estudio_Experimental.xlsx

Descripción: Este archivo Excel alberga los cálculos y datos recopilados durante el estudio experimental realizado como parte de la práctica. Puede incluir tablas, gráficos u otros elementos visuales que representen los resultados de las pruebas realizadas.

PR3_Tareas_a_Repartir.xlsx

Ruta: docs\practica04\PR4_Tareas_a_Repartir.xlsx

Descripción: Este archivo Excel contiene la distribución de tareas asignadas a cada uno de los compañeros que trabajan en la práctica. Incluye una lista de tareas específicas y las personas responsables de completar cada tarea.

A.3. Resultados pruebas experimentales.

Se presentan los resultados de las pruebas experimentales.

A.3.1. Primera prueba experimental.

N	Size	Grdy	It	Bt1	Bt2	Bt3	Tht
2	8	198300	293900	218900	185000	169200	1819200
2	8	201700	297500	220000	186200	170600	2933600
2	8	204600	301300	221100	187300	172100	3858800
2	8	207500	306000	222300	188800	174100	5048900
2	8	209500	309300	223200	189800	175300	5845100
2	8	211700	312600	224200	197400	178400	6537300
2	8	213800	315600	225200	198300	179700	7200300
2	8	217200	320300	226200	199400	181000	7943400
2	8	219200	323600	227100	200400	182300	10798200
2	8	221400	327200	228100	201400	183800	11600900
4	8	2400	5600	1300	1700	2200	1753100
4	8	4700	10900	2600	3200	4200	3359500

4	8	7500	18400	4300	5600	7400	4905100
4	8	9600	23800	5600	7100	9500	6593200
4	8	12900	29700	7200	8800	11900	8609600
4	8	15600	35300	8500	10400	13900	10215000
4	8	21200	42700	10200	12300	16000	12290800
4	8	24000	48600	11400	13900	18200	13894200
4	8	27700	56900	13900	15800	20600	15643600
4	8	30400	64700	15800	18200	23900	17007300
8	8	2700	30200	12000	8500	27700	1815600
8	8	7300	101000	22100	14000	34600	3678500
8	8	10900	133300	23400	15900	38500	5757600
8	8	14300	199800	25700	17900	41900	8515900
8	8	16800	240500	27000	20700	44600	9837400
8	8	20600	276100	28400	22300	49100	11157400
8	8	22800	311900	29800	23900	52400	12719300
8	8	25800	358000	31700	26400	56400	14015400
8	8	28400	395400	35100	28300	58800	15519800
8	8	31300	437900	36900	31000	62600	17122100
16	8	5500	223400	5700	7700	13100	1698500
16	8	10400	382200	10200	12400	21300	3208100
16	8	14200	530600	14800	17000	29900	4752600
16	8	17200	672100	18800	21400	38000	6339600
16	8	34800	862200	41500	26600	48500	8083600
16	8	37800	1013100	44800	30700	56400	9498300
16	8	40500	1168300	53500	49800	69200	11055200
16	8	43800	1325200	55600	54400	77100	12610400
16	8	48100	1493400	57500	64300	85500	14201100
16	8	51700	1635200	59400	67000	95200	15724900
32	8	7200	1953800	40600	54900	280600	1621700
32	8	12500	2748600	76000	107000	379900	3200000
32	8	17300	3154800	117000	160500	461000	4934700
32	8	20900	3524000	154000	214300	541300	6743300
32	8	24900	3899200	194700	267800	621200	8318600
32	8	31000	4197000	242100	321900	702200	9650900
32	8	34500	4622300	282300	375700	783100	11123500
32	8	39000	4928700	323700	435800	864300	12618600
32	8	44600	5236900	364600	489200	945000	13876600
32	8	47800	5559200	404900	542600	1026000	15247900
64	8	19600	7112500	952300	1421000	2090300	2062100
64	8	24600	14204300	1907700	2838600	7949600	4456200
64	8	29300	20630300	2880400	4290500	10018300	6774300
64	8	35300	27771500	3814500	5636600	11980900	9012800
64	8	47000	34393900	4714100	6940400	14171700	11693800
64	8	51600	41051800	5531200	8205800	16045200	14567900
64	8	58500	47827200	6424300	9525300	17861800	17476600
64	8	63200	54381700	7370200	10794400	19700300	19807900

64	8	68200	60780700	8186600	12028300	21560300	22266200
64	8	72200	67220800	9041900	13312800	23369700	25076500
128	8	22400	273758600	39771100	64410700	90514800	25141300
128	8	28600	544159600	79962400	123054400	180881900	53793500
128	8	33900	812949100	122942500	185971300	275201400	75441200
128	8	41100	1086154600	163548600	247194200	362976200	98558700
128	8	50300	1356970900	203760600	309156700	455466300	120725300
128	8	55900	1632077500	244066900	368547500	543313700	142707300
128	8	61900	1908056300	284309200	428933400	641379100	165672300
128	8	69300	2177779900	328568500	491403100	733584300	191935600
128	8	75500	2460181100	385437700	552586600	824001900	216574300
128	8	82200	2727228000	427463400	614126100	918154500	239230600
256	8	89950	1,0579E+11	17494413500	51242465600	71926703500	20586662400

Tabla 10. Resultados primera prueba.

A.3.2. Segunda prueba experimental.

N	Size	Grdy	It	Bt1	Bt2	Bt3	Tht
2	16	202800	286300	227200	179400	198500	2418700
2	16	205400	291500	228300	180800	200200	4348600
2	16	207500	294600	229100	182000	201600	5668400
2	16	210300	299200	232600	183800	205800	6672600
2	16	212300	302400	233400	184900	207100	7956000
2	16	215900	306100	234300	186000	208400	8844500
2	16	220700	310700	236300	187200	209600	9659900
2	16	222400	313800	237100	188100	210700	10409900
2	16	224100	316400	237800	189000	211800	16685200
2	16	226700	319600	238600	190100	213100	18881600
4	16	2900	5900	1400	1700	2200	3100200
4	16	7600	14400	4400	3600	5700	4732600
4	16	9800	20100	5900	5100	7800	7001200
4	16	13800	26200	7100	6700	9800	8930900
4	16	17400	36900	11800	9700	14100	12038700
4	16	21000	45300	13800	12200	17300	13709200
4	16	23400	52600	15100	13700	19600	15133200
4	16	25500	58100	16300	15200	21600	16846400
4	16	27500	63500	17500	16800	23600	18491800
4	16	29400	68700	18700	18300	25500	19842100
8	16	3300	18100	13100	5500	28300	2147400
8	16	8200	65500	24600	8500	34400	3637900
8	16	12700	85900	25600	9700	37900	5048800
8	16	15900	112900	27100	11600	40500	6577100
8	16	18000	128900	28000	12600	41800	8074800
8	16	20000	151100	30000	14000	43600	9494700
8	16	23500	171100	31000	15300	45400	10666500
8	16	34200	192400	31900	16500	48100	15708300
8	16	38100	218600	33400	18500	50900	17513600
8	16	40400	234900	34300	19800	52500	19440300

16	16	4800	270400	5800	6600	12200	2064500
16	16	8400	547300	11800	13000	24000	3697400
16	16	12100	822900	18200	19100	36100	5558800
16	16	15300	1128100	38800	29900	48700	7443500
16	16	18600	1423300	45900	36400	73800	9399600
16	16	23200	1708100	56400	43600	86000	11543000
16	16	27000	1989600	59000	51300	98000	13510600
16	16	32300	2291700	62200	59100	110800	16564300
16	16	37800	2600200	64900	66400	123100	18044800
16	16	46800	2906900	67800	73400	135600	19636700
32	16	7300	222100	10000	20200	19400	1509500
32	16	13100	435800	19300	33500	38100	3644100
32	16	17700	640100	29400	48200	57600	5313900
32	16	23400	797300	39400	62500	76900	6866500
32	16	28900	955100	49500	76800	96600	9533300
32	16	32600	1049100	59200	95700	120200	10992100
32	16	36300	1135800	71000	109400	138800	12254900
32	16	40900	1229700	83900	124800	158800	13921900
32	16	43900	1308100	93900	139200	178600	15417900
32	16	49600	1381500	103300	152700	197000	16798500
64	16	22800	2989800	390800	582700	838200	2137000
64	16	30200	5700000	766800	1168000	1675700	5156800
64	16	35300	8840100	1229600	1851600	2617900	8382800
64	16	41700	12571900	1805400	2608200	3662700	12821800
64	16	47700	16028800	2306400	3381200	4679700	19647200
64	16	53400	19190800	2688100	3980800	5607200	22278600
64	16	61700	22331700	3172900	4676600	6597700	26472600
64	16	68800	25425500	3657000	5410100	7645500	29757400
64	16	74400	28755000	4090400	6088300	8727800	33936700
64	16	81500	32482200	4682500	6925300	9796200	37596300
128	16	30200	7468912400	1185425400	1741812700	2881556400	832097600
128	16	37900	21728074800	3801171800	5586906100	7848893600	1714475000
128	16	49700	35743387100	6263514100	9381614600	13006655500	2496552100
128	16	55200	49921410600	8828640100	13176222200	18042092900	3325964300
128	16	65900	64094074900	11319189900	17013608000	22967237700	4298117200
128	16	74000	78065902100	13922338900	20777970500	28014526400	4954362500
128	16	95600	91921185000	16368006000	24620527000	32974422500	5735590700
128	16	103200	1,05893E+11	17784917200	26335528100	35383279900	6119990200
128	16	109100	1,13267E+11	18884606300	28035343400	37687604500	6498515000
128	16	116300	1,20553E+11	19956892300	29768935000	40050004200	6895410100
256	16	131900	2,43257E+12	4,46978E+11	6,7981E+11	8,81249E+11	1,8109E+11

Tabla 11. Resultados segunda prueba.

A.3.3. Tercera prueba experimental.

N	Size	NlogN	2^N	Grdy	It	Bt1	Bt2	Bt3
4	64	8	16	3860	4800	2440	2890	8070
8	64	24	256	5380	12460	4660	4180	10780
16	64	64	65536	7560	28630	240360	299180	511370
32	64	160	4294967296	11540	84800	10466314400	11301193100	31433796100

Tabla 12. Resultados tercera prueba, pesos 1.

A.3.4. Cuarta prueba experimental.

Archivo	Grdy	It	Bt1	Bt2	Bt3
p01	34960	80590	29530	25290	26440
p02	5090	282820	8430	6010	11630
p03	4550	29110	2820	3830	4010
p04	8650	209868070	26916690	37134450	63421770

Tabla 13. Resultados prueba con archivos.

B. Bibliografía

Temario Asignatura. (2024). Almeria.

Backtracking Algorithm - GeeksforGeeks

PARTE II: ALGORÍTMICA Tema 5. Backtracking.. (um.es)

Pages 16 to 54 of 5 Exploración de grafos (ugr.es)