TALLER N°3

Knapsack Problem

Taller de programación 1-2024

Fecha: 16 de Agosto

Autor: Sebastián Cassone

TALLER N°3

Knapsack Problem

Explicación breve del algoritmo

El algoritmo implementado busca resolver problemas de optimización lineal con restricciones utilizando el método de Branch and Bound. En esencia, este método divide el problema en subproblemas más pequeños (ramificación) y evalúa estos subproblemas para encontrar la mejor solución posible, mientras descarta soluciones que no cumplen con los criterios óptimos (podas). La idea básica del código es empezar resolviendo el problema original, luego ramificar sobre las soluciones parciales que no sean enteras, y aplicar el método de poda para mejorar la eficiencia.

Heurísticas o técnicas utilizadas

El método principal para resolver este problema es el Branch and Bound, que divide el problema en subproblemas más manejables y descarta aquellas ramas que no conducen a una solución óptima. Este proceso de eliminación, conocido como poda, se encarga de excluir subproblemas que no pueden mejorar la solución actual.

Se emplea una heurística greedy para obtener rápidamente una cota superior. Este enfoque se basa en seleccionar soluciones que ofrezcan la mayor ganancia inmediata, tomando en cuenta los costos y pesos de las variables, y proporciona una solución inicial útil para la poda.

La comparación de variables ayuda a decidir en qué variable ramificar, enfocándose en aquella cuya fracción se aleje más de los enteros.

Por último, se utiliza una cola de prioridad (priority_queue) para gestionar los subproblemas, organizándolos según su potencial para mejorar la solución actual.

Funcionamiento del programa

El código implementa un solucionador para problemas de optimización lineal utilizando el método de Branch and Bound. La clase Branch busca maximizar una función objetivo con restricciones lineales asociadas. El proceso general comienza con la recepción del problema de optimización lineal representado por una instancia de la clase Simplex. Este problema es inicialmente resuelto utilizando

un método greedy para obtener una cota superior inicial. La solución greedy se calcula mediante el método greedy.

Posteriormente, se establece una restricción adicional en el problema original basada en la solución greedy para mejorar la cota superior. Luego, se resuelve el problema original utilizando el método de Branch and Bound, comenzando con la cola de problemas (Live) que contiene el problema original. Durante la ejecución del algoritmo, se extrae el problema con la mejor cota (en la cima de la cola de prioridad) y se verifica su factibilidad. Si el problema actual es óptimo, se devuelve la solución encontrada. Si el problema no es óptimo, se identifica la variable con la parte fraccionaria más cercana a 0.5, es decir, la variable más alejada de un valor entero. Se ramifica el problema en dos subproblemas agregando restricciones para esta variable fraccionaria.

Los subproblemas se resuelven y se añaden a la cola de problemas si son factibles y sus valores de objetivo son mayores o iguales a la cota inferior del problema original. El algoritmo utiliza heurísticas, como el método greedy para calcular una cota superior inicial y la selección de variables para ramificación para optimizar la búsqueda de soluciones enteras. Si no se encuentra una solución factible, el programa informa al usuario; de lo contrario, se devuelve la solución óptima encontrada.

El método addBranchProblems tiene el propósito de agregar dos problemas a la cola de problemas con restricciones adicionales. Este método clona el problema actual, añade restricciones para limitar el valor de una variable fraccionaria, y resuelve los problemas resultantes. Los problemas factibles y con valores de objetivo válidos se añaden a la cola de prioridad Live.

Por último, en este algoritmo se utilizan técnicas de poda para descartar soluciones que no pueden mejorar la solución actual. El proceso continúa hasta que se encuentra una solución óptima o se determina que no hay solución.

Para entender con mayor claridad la solución, se escribe el siguiente psedocódigo.

1. Función solve (Simplex problema):
2. Resolver el problema original usando una heurística greedy.
3. Insertar la solución greedy como una restricción al problema original.
4. Resolver el problema original.
5. Mientras haya problemas en la cola:
    a. Extraer el problema de mayor prioridad.
    b. Verificar la factibilidad.
    c. Si es factible y óptimo, devolver la solución.

d. Si no es óptimo, ramificar en dos nuevos problemas y agregar a la cola si son factibles.

6. Si no se encuentra solución, devolver un mensaje indicando que no se encontró solución.

Aspectos de implementación y eficiencia

La utilización de una cola de prioridad para gestionar los subproblemas permite que el algoritmo se enfoque en los problemas más prometedores. La técnica de poda evita la exploración innecesaria de ramas que no pueden mejorar la solución.

El programa maneja la memoria a través de la clonación de problemas y la gestión de subproblemas en una cola de prioridad, lo que es eficiente para grandes espacios de soluciones.

El uso de la heurística greedy para obtener una cota superior inicial puede mejorar significativamente el rendimiento al reducir el número de subproblemas necesarios.

Ejecución del código

Para compilar y ejecutar el programa, se requiere tener instalado un compilador de C+ (como g++) y el utilitario Make.

Para compilar el programa, simplemente ejecute el comando `make` en su terminal desde el directorio raíz del repositorio. Esto generará tres ejecutables: `test_Branch`, `test_Simplex` y `main`.

Con respecto a los archivos, `test_Branch` y `test_Simplex` son ejecutables de los dos test unitarios de las dos clases del proyecto solo con la finalidad de comprobar que las clases y sus respectivos métodos funcionan. Por lo que se ejecutan sin necesidad de introducir parámetros por consola.

Finalmente, el archivo ejecutable principal del programa `main` en cualquier distribución Linux (también válido para los test unitarios) se realizaría con `./main` de esta forma se comenzaría a ejecutar el programa principal. Mostrará en la consola un menú con diferentes opciones, “1. Elegir archivo” y “2. Para salir”. Si se pulsa “1. Elegir archivo”” inmediatamente

pedirá el nombre del archivo, una vez introducido y pulsado `Enter` comenzará a ejecutarse mostrando posteriormente la solución de Knapsack Problem y el tiempo de ejecución.

Bibliografía

https://drive.google.com/file/d/1APgLie_zq6FOXj7ZeOb2ID0mVJTXdZvf/

https://ocw.mit.edu/courses/1-204-computer-algorithms-in-systems-engineering-spring-2010/resources/lecture-notes/