Facultat d'Informàtica de Barcelona

Conjunts de problemes 12 i 49
Informe sobre la creación de un solver de Simplex

Simplex Primal

Alumnes :
Álvarez Aragonés, Rubén
53324691L
Mejía Rota, César Elías
47245330H

Tutors :
Linares, Mª Paz

March 30, 2024

Contents

1 Descripción del algorítmo

Para la implementación del algoritmo Símplex Primal se ha seguido el proceso matemático de cálculo que hemos visto en clase.

El algoritmo Símplex es un método de optimización que se utiliza para resolver problemas de programación lineal.

Se basa en la idea de que cualquier problema de programación lineal puede ser transformado en una forma estándar, y que la solución óptima de un problema en esta forma puede ser encontrada en los vértices de la región factible.

Hemos optado por optimizar el algoritmo de forma que no se realicen cálculos de inversas de matrices, ya que es un proceso costoso computacionalmente. Para ello, hemos utilizado la técnica de actualización de inversas de matrices, que consiste en actualizar la inversa de la matriz de restricciones en cada iteración del algoritmo. Además, hemos hecho las implementaciones correspondientes en el programa para que no sea necesario calcular la primera inversa.

El algoritmo Símplex generado se divide, como hemos comentado, en dos fases: la fase I y la fase II. La fase I se encarga de encontrar una solución básica factible ayudándose de un problema artificial, en el que se añaden variables artificiales a cada una de las restricciones y se substituye la función objetivo a minimizar por la suma de dichas variables. De igual manera, estas variables artificiales son las que empiezan dentro de la base de la solución a este problema artificial. Cuando encontremos la solución óptima de dicho problema, habremos encontrado una SBF (solución básica factible) del problema original y podremos empezar con la fase II. En la fase II repetimos el mismo algoritmo usado previamente en la Fase I para encontrar la solución óptima del problema original, partiendo con la SBF encontrada en la fase I.

Como último detalle, nos hemos ayudado de la Regla de Bland para evitar ciclos infinitos debido a la posible degeneración del problema.

2 Estructura del programa

2.1 Lectura de datos

2.1.1 Fichero de texto

Para la lectura de datos hemos optado por la extracción de los mismos desde un fichero de texto generado por nosotros manualmente. En este fichero de texto se encuentran los datos del problema de programación lineal a resolver.

En el fichero de texto encontramos en primer lugar el nombre del problema, seguido de la función objetivo a minimizar, los coeficientes de las restricciones del problema, sus términos independientes y finalmente el valor óptimo del problema junto a las variables básicas en ese óptimo (estas dos últimos valores no se utilizan en nuestro programa, pero pueden resultar útiles para comprobar su funcionamiento).

Para indagar más en la estructura del fichero de texto, podemos ver un ejemplo del mismo en los archivos adjuntos al informe (datos.txt).

2.1.2 Clase Lector

Para la lectura de datos hemos creado una clase llamada lector que se encarga de leer los datos del fichero de texto. Los datos se almacenan en un diccionario cuyas claves son los nombres de los problemas y cuyos valores son instáncias de un objeto (problema) que contiene toda la información que nos será necesaria para realizar el algorítmo de optimización.

Hemos escogido esta implementación para poder tener una estructura de datos más clara y poder acceder a los datos de forma más sencilla. Esto nos facilita mucho la visualización de los datos en caso de que queramos hacer pruebas con diferentes problemas y nos brinda una gran versatilidad a la hora de trabajar con nuevas implementaciones sobre el programa.

2.2 El Algoritmo

El algoritmo Símplex que hemos codificado se encuentra dentro de su propia clase (class Simplex). Hemos decidido utilizar la programación orientada a objetos debido a la utilidad que representa tener una clase de la que poder generar diversas instancias para hacer pruebas.

Guardamos los datos del problema sin alterar para poder usarlos más tarde durante la ejecución al inicializar la instancia de la clase.

2.2.1 El método 'iter'

La función 'iter' es la parte principal de nuestro código y se encarga de hacer los cálculos del algoritmo siguiendo los pasos que hemos aprendido en clase. Hemos optado para la implementación por sobrescribir los datos del problema artificial y del problema original en las variables de la función 'iter' (self.iter_X) para ahorrar memoria y simplificar la codificación.

En este método iteramos sobre el hecho de que exista un coste reducido negativo, puesto que en el momento en que no haya coste reducido negativo habremos llegado a la solución óptima del problema.

2.2.2 Los métodos 'genenrar_PA' y 'generar_sol'

Como se ha mencionado antes, en estas dos funciones escribimos los datos sobre las variables de forma 'self.iter_X' para que sean utilizadas en los cálculos tanto de la Fase I como de la Fase II.

En 'generar_PA', generamos las matrices propias del problema artificial. Para ello convertimos 'c' en una array de 0 seguidas de 1. La cantidad de 0 es n (número de variables) y la cantidad de 1 es m (número de restricciones). Con esto convertimos la función objetivo en el sumatorio de las variables artificiales. Justo después añadimos dichas variables artificiales a 'A', concatenando la matriz original 'A' con la identidad de tamaño m y dejamos b tal y como esta, puesto que no se ve modificada al crear el problema artificial. De la misma forma, creamos todo el resto de matrices y arrays que utilizaremos durante la ejecución.

En 'generar_sol' lo que hacemos es devolver todos los valores a los del problema inicial, manteniendo la base encontrada (Beta) para no perder la SBF (solución básica factible) encontrada en la Fase I.

2.2.3 Actualización de la inversa

Para implementar la actualización de la inversa hemos utilizado el pseudocódigo que se nos ha proporcionado en clase. Hemos creado un método llamado 'actualizar_inversa' que se encarga de hacer los cálculos necesarios para actualizar, evitar el costo computacional que representa el cálculo de la inversa de una matriz que, en ciertos problemas, puede ser muy grande.

2.2.4 Método de aplicación de la Regla de Bland

El método de Bland está implementado en la función bland_argmin, que se encarga de devolver el índice de la variable básica/no básica que entra/sale de la base. Para hacer esto mismo, simplemente generamos un array de las variables con valor mínimo y luego las recorremos para encontrar la que tiene menor índice.

Esto nos garantiza que no caeremos en ciclos infinitos debido a la degeneración del problema.

2.2.5 Excepciones

Lo último por hacer sobre el algoritmo a nivel funcional es encontrar cuando un problema es no acotado o simplemente no tiene solución.

Para comprobar si el problema tiene solución miramos, primeramente, que todas las variables tengan valor positivo, puesto que es una de las restricciones del problema en forma estándar. Esto sucede en nuestro programa si cualquier elemento de la array xB tiene valor negativo, ya que la array xB contiene el valor de las variables. El otro caso a estudiar es cuando la solución óptima de Fase I tiene alguna variable artificial dentro de la base óptima (Beta). Para comprobar esto, miraremos al salir de la Fase I si ha quedado alguna de las variables artificiales dentro de la base (todas han empezado dentro). Al mirar el valor de 'z' (función objetivo) podemos saber esto, pues si 'z'es otra cosa que no sea 0 sabremos que alguna variable de la base está dándole valor a la función, y como la función objetivo solo contiene las variables artificiales, sabremos que ay al menos una variable artificial en la base.

Por último, comprobaremos si el problema es acotado. Para esto, tenemos que comprobar que existe una dirección básica negativa siempre que exista un coste reducido negativo. Como nuestro bucle itera sobre el hecho de que existe, como mínimo, un coste reducido negativo. Si entramos al bucle quiere decir que existe dicho coste reducido negativo, y, si después de calcular las direcciones básicas son todas positivas, sabremos que el problema es no acotado.

2.3 Ejecución del código

El código se ejecuta desde el archivo 'ejecutar.py'. Al ejecutar, genera las soluciones de los problemas especificados en la lista 'Problemas' y las escribe en cada una en un archivo de texto con el número de problema como nombre del archivo, y las guarda en una carpeta creada con el nombre especificado en la variable 'Carpeta'.

3 Soluciones

Para consultar las soluciones obtenidas, es necesario ver los archivos contenidos en la carpeta soluciones, incluida dentro de este mismo archivo comprimido.

4 Conclusiones

Para concluir este informe, podemos decir que hemos implementado un algoritmo Símplex Primal que funciona correctamente y que cumple con los requisitos que se nos han pedido. Hemos optado por una implementación que no requiere el cálculo de inversas de matrices, lo que nos ha permitido optimizar el algoritmo y hacerlo más eficiente computacionalmente.

4.1 Posibles mejoras y ampliaciones

Como posibles mejoras del programa, podríamos implementar un sistema de lectura que sea capaz de leer ficheros más complejos (por ejemplo, el fichero de texto que se nos proporciona en la entrega de la práctica). También podríamos mejorar la entendibilidad del código, ya que puede resultar algo confuso para alguien ajeno a la creación del mismo. Por último, creemos que hemos utilizado muchos atributos para diferentes clases que realmente no son necesarios para la ejecución del código y que simplemente nos ayudan a la hora de gestionar diferentes implementaciones.

Como posibles ampliaciones del programa, queda abierta la posibilidad de implementar el algoritmo de Símplex Dual, así como el algoritmo de Branch and Bound para que resuelva problemas de programación lineal enteros.