Proyecto final: Manual de Usuario

Omar Eduardo Roa Quintero
Fernando Vargas Montero

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2017

Índice

1. Introducción

1.1. Método Simplex

El método Simplex es un procedimiento iterativo que permite mejorar la solución de la función objetivo en cada paso. El proceso concluye cuando no es posible continuar mejorando dicho valor, es decir, se ha alcanzado la solución óptima (el mayor o menor valor posible, según el caso, para el que se satisfacen todas las restricciones).

1.2. Como funciona

Partiendo del valor de la función objetivo en un punto cualquiera, el procedimiento consiste en buscar otro punto que mejore el valor anterior. Dichos puntos son los vértices del polígono (o poliedro o polícoro, si el número de variables es mayor de 2) que constituye la región determinada por las restricciones a las que se encuentra sujeto el problema (llamada región factible). La búsqueda se realiza mediante desplazamientos por las aristas del polígono, desde el vértice actual hasta uno adyacente que mejore el valor de la función objetivo. Siempre que exista región factible, como su número de vértices y de aristas es finito, será posible encontrar la solución.

El método Simplex se basa en la siguiente propiedad: si la función objetivo Z no toma su valor máximo en el vértice A, entonces existe una arista que parte de A y a lo largo de la cual el valor de Z aumenta.[1]

1.3. Descripción general

Se diseñó y realizó un lenguaje para facilitar y agilizar el uso del método Simplex. Cuenta con todas las entradas necesarias para realizar su calculo, además con operaciones matemáticas.

2. Descripción del modelo del lenguaje

El algoritmo es programado usando caracteres en formato de texto plano usando los caracteres del conjunto ASCII. Los caracteres válidos en el algoritmo son los siguientes:

- caracteres alfabéticos:

  ```
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  a b c d e f g h i j k l m n o p q r s t u v w x y z
  ```

- caracteres numéricos:

  ```
  0 1 2 3 4 5 6 7 8 9
  ```

- caracteres especiales:

  ```
  ! & ( ) * , - . : ; < = > / % + [ ] ^ { | }
  ```

- caracteres de espacios en blanco:

  ```
  espacio nuevaLinea tabulación
  ```

Dentro de los literales y comentarios cualquier caracter ASCII (excepto caracteres de control) son válidos.
Los caracteres de espacios en blanco no son significantes. Pueden ser usados libremente entre unidades léxicas para mejorar la legibilidad del codigo. También son usadas para separar unidades léxicas unas de otras si no hay otra forma de hacerlo. Las unidades léxicas del lenguaje son presentadas a continuación.

2.1. Nombres de variables

Un nombre de variable consiste en letras, inclusive de una sola letra como mínimo. Todos los nombres de variables son distintos (distingue mayúsculas y minúsculas).

Ejemplos:

```
v
valor
VALOR
```

Los nombres de variables son usados para identificar variables (números, parámetros).

2.2. Nombres de variables de funciones

Un nombre de variable consiste en caracteres alfanuméricos, el primero debe ser alfabético. Todos los nombres de variables son distintos (distingue mayúsculas y minúsculas).

Ejemplos:

```
variable123
Estatura_promedio
VALOR
```

Los nombres de variables son usados para identificar variables (números, parámetros).

2.3. Números

Un número tiene la forma *nnEsxx*, donde *nn* es un número con punto decimal opcional, *s* es el signo + o -, *xx* es un exponente decimal. La letra *E* también puede ser *e*.

Ejemplos:

```
123
3.14159
123.456e7
```

Los números son usados para representar cantidades numéricas.

2.4. Palabras Reservadas

Una palabra reservada es una secuencia de caracteres alfabéticos y posiblemente un caracter especial.
Todas las palabras reservadas corresponden a una categoría: palabras no reservadas, las cuales pueden ser usadas como nombres de variables, y palabras reservadas, las cuales son reconocidas por el contexto y por eso no pueden ser usadas como nombres de variables.
Las palabras reservadas son las siguientes:

```
imprimir    imp    maximizar  max    minimizar  min  funcion  fun
restringir  restr  resolver   resol  res        con  sen      cos
tan         exp    ln
```

Todas las palabras reservadas tienen un significado específico, el cual será explicado en las construcciones sintácticas correspondientes, donde las palabras reservadas son usadas.

2.5. Delimitadores y operadores

Un delimitador es o un caracter especial simple o una secuencia de dos caracteres especiales como se muestran a continuación:

```
( ) * , - < <= = == > >= / % +
```

Si el delimitador consiste de dos caracteres, no debe haber espacio entre los caracteres.
Todos los delimitadores tienen un significado específico, el cual será explicado en las correspondientes construcciones sintácticas, donde los delimitadores son usados.

3. Expresiones

Una expresión es una regla para calcular un valor. En el algoritmo las expresiones son usadas como constituyentes de ciertas sentencias.
En las expresiones generales consiste en operandos y operadores.
Dependiendo del tipo del valor resultante todas las expresiones caen dentro de una de las siguientes categorías:

- expresiones numéricas;
- expresiones simbólicas;

3.1. Expresiones numéricas

Una expresión numéricas es una regla de computación para un valor numérico simple representado como un número de punto flotante.
La expresión numérica primaria puede ser un número, un parámetro u otra expresión numérica encerrada en paréntesis.
Ejemplos:

```
1.23
j
tiempo
```

Más expresiones numéricas conteniendo dos o más expresiones numéricas primarias pueden ser construidas usando ciertos operadores aritméticos.
Ejemplos:

```
j+1
```

3.1.1. Referencia de funciones

En el lenguaje existen las siguientes funciones predefinidas las cuales pueden ser usadas en expresiones numéricas:

Función	Descripción
`sen(x)`	Seno de `x`
`cos(x)`	Coseno de `x`
`tan(x)`	Tangente de `x`
`exp(x)`	Exponencial de `x`
`ln(x)`	Logaritmo natural de `x`

Los argumentos de todas las funciones predefinidas deben ser expresiones numéricas .
El valor resultante de una expresión numérica, la cual es una referencia de función, es el resultado de aplicar la función a su(s) argumento(s).
Nota: las funciones trigonométricas usan números radianes.

3.1.2. Expresiones con paréntesis

Cualquier expresión numérica puede ser encerrada en paréntesis que sintácticamente lo hacen una expresión numérica primaria.
Los paréntesis pueden ser usados en expresiones numéricas, como en el álgebra, para especificar el orden deseado del cual las operaciones se van a desarrollar. Donde los paréntesis son usados, la expresión dentro de los paréntesis es evaluada antes de que el valor resultante sea usado.
El valor resultante de las expresiones con paréntesis es la misma que el valor de la expresión encerrada dentro de los paréntesis.

3.1.3. Operadores aritméticos

En el lenguaje existen los siguientes operadores aritméticos, los cuales pueden ser usados en expresiones numéricas:

Operación	Descripción
`n + m`	Suma
`n - m`	Resta
`n * m`	Producto
`n / m`	División
`n ** m`	Potenciación

donde `n` y `m` son expresiones numéricas.
Si la expresión incluye más que un operador aritmético, todos los operadores son aplicados de izquierda a derecha de acuerdo a la jerarquía de operadores (mirar a continuación) con la única excepción de que los operadores de exponenciación son aplicados de derecha a izquierda.
El valor resultante de la expresión, la cual contiene operadores aritméticos, es el resultado de aplicar los operadores a sus operandos.

3.1.4. Jerarquía de operaciones

La siguiente lista muestra la jerarquía de los operadores en expresiones numéricas:

Operación	Jerarquía
Evaluación de funciones	1º
Exponenciación	2º
Suma y resta unarias	3º
Multiplicación y división	4º
Suma y resta	5º

La jerarquía es usada para determinar cuál de dos operaciones consecutivas es aplicada primero. Si el primero operador es mayor o igual al segundo, el primer operador es aplicado. Si no, el segundo operador es comparado con el tercero, etc. Cuando el fin de la expresión es alcanzado, todas las operaciones restantes son ejecutadas en el orden inverso.

3.1.5. Expresiones numéricas

El valor resultante de la expresión lógica simple, la cual es una expresión numérica, es verdadero, si el valor resultante de la expresión no es cero. En otro caso el valor resultante de la expresión numérica es falsa.

3.1.6. Operadores relacionales

En el lenguaje existen los siguientes operadores lógicos, los cuales pueden ser usados en expresiones lógicas:

Operación	Descripción
`x < y`	Menor que
`x <= y`	Menor o igual que
`x >= y`	Mayor o igual que
`x > y`	Mayor que

donde `x` e `y` son expresiones numéricas o simbólicas.
Como los operadores relacionales listados anteriormente tienen su significado matemático convencional. El valor resultante es verdadero, si la relación correspondiente es satisfecha por sus operandos, en otro caso falso. (Notar que los valores simbólicos son ordenados lexicográficamente, y cualquier valor numérico precede cualquier valor simbólico.

3.1.7. Expresiones con paréntesis

Cualquier expresión lógica puede ser encerrada en paréntesis que sintácticamente lo hacen una expresión lógica simple.
Los paréntesis pueden ser usados en expresiones lógicas, como en el álgebra para especificar el orden deseado en el cual los operadores se aplicarán. Donde los paréntesis son usados, la expresión dentro de los paréntesis es evaluada antes de que el valor resultante es usado.
El valor resultante de la expresión con paréntesis es la misma que el valor de la expresión encerrada dentro de los paréntesis.

4. Sentencias

Las sentencias son unidades básicas en la descripción del método.

4.1. Sentencia min, minimizar, max ,maximizar

```
min:
    fun ( param, ..., param ) = funcionMatematica
restringir:
    funciones
        .
        .
        .
con:
    expresionMatematica
        .
        .
        .
res
```

Donde
`param, ..., param` son parámetros opcionales de entrada, el lenguaje tambien soporta el uso de variables y expresiones matematicas complejas definidas por el usuario, para esto deben ser introdusidas dentro de parentesis redonddos"(expresion)".

Ejemplos:

```
min:
    fun (a,b,c)= -(2**2)a-4b+ 10.01 + (c)
restringir:
    2a+2b +10 <= 15.00001
    6a+3b <= 1
    5a+10b <= 10
con:
    a >= 0
    b >= 0
    c >= 0
res

max:
    fun (n,b,c)= 2n+3b-(5/(a-z))c
restringir:
    n+b+c = 7
    2n-5b+c >= 10
con:
```

```
    n >= 0
    b >= 0
    c >= 0
res
```

4.2. Declaración de variables

```
nombre = valor
```

`nombre` es un nombre de variable;
`valor` puede ser una expresión numérica.

Ejemplos:

```
a = 12
z= 12**2
c = 0.00001
```

5. Requerimientos

A continuación se explica el procedimiento de instalación junto con los requisitos del sistema para poder ejecutar el algoritmo correctamente.

5.1. Software

Para la correcta ejecución de los algoritmos sobre el lenguaje desarrollado, es necesaria la instalación del siguiente software, junto con un sistema operativo, ya sea Windows o alguna distribución de Linux:

- Python version 2.x o 3.x.
- Librería AntLR version 4.x

5.2. Hardware

Los requerimientos de hardware están dados por el sistema operativo y por el entorno de desarrollo de Java.

6. Instalación

6.1. Linux

6.1.1. Instalación de Python (Arch Linux)

Para instalar Python v3.x se ejecuta:

```
yaourt install python
```

Para instalar Python v2.x se ejecuta:

```
yaourt install python2
```

6.1.2. Instaalcion de ANTLR (Arch Linux)

La librería de ANTLR se puede instalar con el siguiente comando:

```
yaourt install antlr4
```

6.1.3. Configuración adicional

Para poder hacer uso de las librerias de ANTLR en Python se hace necesario ejecutar como super-usuario:
Para instalar Python v3.x se ejecuta:

```
sudo pip install antlr4-python3-runtime==4.x
```

Para instalar Python 2.x se ejecuta:

```
sudo pip2 install antlr4-python2-runtime==4.x
```

La x indica el numero de verision del paquete de *antlr4* instalado en el sistema operativo.

7. Ejecución

Para poder hacer uso de la herramienta, es necesario el uso de una consola (o un terminal) del sistema operativo para ejecutar el archivi python *simplex.py* contenida en el directorio del lenguaje.
Para ejecutar un código previamente almacenado en un archivo, ejecutando el siguiente comando:

```
python simplex.py <archivo>
```

8. Ejemplos

```
a = 12
z = sen(4)
c = 0.00001
imprimir(a / z )
min:
    fun (a,b,c)= -(2**2)a-4b+ 10.01 + (c)
restringir:
    2a+2b +10 <= 15.00001
    6a+3b <= 1
    5a+10b <= 10
con:
    a >= 0
    b >= 0
    c >= 0
res
max:
    fun (n,b,c)= 2n+3b-(5/(a-z))c
restringir:
    n+b+c = 7
    2n-5b+c >= 10
con:
    n >= 0
    b >= 0
    c >= 0
res
imp(a)
imp(n)
imp(b)
imp(c)
imp(F)
maximizar:
    fun (a,b,c)= 315a+110b+50c
restringir:
    15a+2b+c <= 200
    7.5a+3b+c >= 150
    5a+2b+c = 120
con:
    a >= 0
    b >= 0
    c >= 0
res
minimizar:
    fun (a,b,c)= 315a+110b+50c
restringir:
    15a+2b+c <= 200
    7.5a+3b+c >= 150
    5a+2b+c = 120
con:
    a >= 0
    b >= 0
    c >= 0
res
```

Referencias

[1] Teoría del método Simplex. http://www.phpsimplex.com/teoria_metodo_simplex.htm