Modelos de Optimización
Laboratorio 7: Uso de software para solución numérica de problemas de Optimización Lineal Discreta

Osmany Pérez Rodríguez
Enrique Martínez González
Carmen Irene Cabrera Rodríguez
Grupo C412

1 Python - MIP

Los paquetes de Python-MIP constituyen una herramienta para el modelado y solución de Problemas de Programación Lineal Enteros Mixtos (*MIPs - Mixed-Integer Linear Programming Problems*) en Python. Contiene diferentes solucionadores para realizar la optimización, entre ellos: CBC (*COIN-OR Branch-and-Cut*), altamente configurable, que fue el utilizado para hallar la solución de los ejercicios propuestos. Se empleó la clase **Model**, que permite la declaración de las variables, las restricciones y la función objetivo de manera muy sencilla y cómoda. El método **optimize** es el que, finalmente computa el algoritmo que resuelve el sistema y permite, de forma opcional, setear el máximo tiempo de ejecución del algoritmo, la máxima cantidad de soluciones a encontrar, entre otros. Este método retorna el resultado del cómputo a través del *status*, que puede ser:

- OPTIMAL: La búsqueda fue concluida y una solución óptima fue encontrada.
- FEASIBLE: Existe una solución factible que fue encontrado, pero no hubo tiempo de probar su optimalidad.
- NO_SOLUTION_FOUND: No se encontró ninguna solución.
- INFEASIBLE: No existe solución factible para el modelo.
- INT_INFEASIBLE: La relajación del problema lineal tiene solución factible pero no existe solución entera que sea factible.
- UNBOUNDED: Una o más variables que aparecen en la función objetivo no están incluidas en las restricciones vinculantes y el valor objetivo óptimo es infinito.
- ERROR: Si ocurrió algún error durante la optimización.

Puede encontrar más información acerca de esta biblioteca a través del siguiente enlace.

2 Solución de los problemas propuestos

2.1 Ejercicio 1

El algoritmo empleado fue el siguiente:

```python
def solve():
    m = Model(sense=MAXIMIZE, solver_name=CBC)

    # Variables
    x1 = m.add_var(var_type=INTEGER, lb=0)
    x2 = m.add_var(var_type=INTEGER, lb=0)

    # Constraints
    m += x1 + 3 * x2 <= 5
    m += 2 * x1 + x2 <= 6

    # Objective function
    m.objective = -2 * x1 + 3 * x2

    m.optimize()
```

Figure 1: Código completo aquí

.

Los resultados obtenidos fueron:

```
# OptimizationStatus.OPTIMAL
# Solution to the objective function: 3.0
# x1: 0.0
# x2: 1.0
# Execution time: 0.012865543365478516
```

La solución encontrada coincide con la reportada en la solución de la clase práctica, ya sea por el método Primal Todo Entero o el de Gomory I. El valor de la función objetivo se devuelve con el signo opuesto en la CP puesto que maximizar la función objetivo es equivalente a hallar el mínimo de esta función multiplicado por -1.

2.2 Ejercicio 2

El algoritmo empleado fue el siguiente:

```python
def solve():
    m = Model(solver_name=CBC)

    # Variables
    x1 = m.add_var(var_type=INTEGER, lb=0)
    x2 = m.add_var(var_type=INTEGER, lb=0)

    # Constraints
    m += 2 * x1 + 2 * x2 == 3
    m += x1 + 2 * x2 <= 2

    # Objective function
    m.objective = x1 + 2 * x2

    m.optimize()
```

Figure 2: Código completo aquí

.

Los resultados obtenidos fueron:

```
# OptimizationStatus.INFEASIBLE
# Execution time: 0.011461734771728516
```

En la CP se puede apreciar que tanto al aplicar el método Primal Todo Entero, como el de Gomory I se determina que no existe solución factible del problema. De igual forma, el estatus que retorna el método optimize: INFEASIBLE implica que el problema no tiene solución factible. Note que al tratarse de variables enteras la primera restricción del problema no tiene sentido; por tanto era de esperar este resultado.

2.3 Ejercicio 3

El algoritmo empleado fue el siguiente:

```python
def solve():
    m = Model(solver_name=CBC)

    # Variables
    x1 = m.add_var(var_type=INTEGER, lb=0)
    x2 = m.add_var(var_type=INTEGER, lb=0)

    # Constraints
    m += 2 * x1 + 3 * x2 == 6
    m += 2 * x1 + 9 * x2 <= 6

    # Objective function
    m.objective = x1 + x2

    m.optimize()
```

Figure 3: Código completo aquí

.

Los resultados obtenidos fueron:

```
# OptimizationStatus.OPTIMAL
# Solution to the objective function: 3.0
# x1: 3.0
# x2: 0.0
# Execution time: 0.013741016387939453
```

La solución obtenida coincide con la reportada en la clase práctica.

2.4 Ejercicio 4

El algoritmo empleado fue el siguiente:

```python
def solve():
    m = Model(sense=MAXIMIZE, solver_name=CBC)

    # Variables
    x1 = m.add_var(var_type=INTEGER, lb=0)
    x2 = m.add_var(var_type=INTEGER, lb=0)
    x3 = m.add_var(var_type=INTEGER, lb=0)

    # Constraints
    m += 3 * x1 + 2 * x2 <= 10
    m += x1 + 4 * x2 <= 11
    m += 3 * x1 + 3 * x2 + x3 == 13

    # Objective function
    m.objective = 4 * x1 + 5 * x2 + x3

    m.optimize()
```

Figure 4: Código completo aquí

.

Los resultados obtenidos fueron:

```
# OptimizationStatus.OPTIMAL
# Solution to the objective function: 19.0
# x1: 2.0
# x2: 2.0
# x3: 1.0
# Execution time: 0.015565633773803711
```

La solución obtenida coincide con la reportada en la clase práctica.

2.5 Ejercicio 6

El algoritmo empleado fue el siguiente:

```python
def solve():
    m = Model(sense=MAXIMIZE, solver_name=CBC)

    # Variables
    x1 = m.add_var(var_type=INTEGER, lb=0, ub=4)
    x2 = m.add_var(var_type=INTEGER, lb=0)

    # Constraints
    m += 2 * x1 + x2 <= 10
    m += -x1 + x2 <= 5

    # Objective function
    m.objective = x1 + 2 * x2

    m.optimize()
```

Figure 5: Código completo aquí

.

Los resultados obtenidos fueron:

```
# OptimizationStatus.OPTIMAL
# Solution to the objective function: 14.0
# x1: 2.0
# x2: 6.0
# Execution time: 0.015373945236206055
```

La solución obtenida coincide con la reportada en la clase práctica.

2.6 Ejercicio 7

El algoritmo empleado fue el siguiente:

```python
def solve():
    m = Model(solver_name=CBC)

    # Variables
    x1 = m.add_var(var_type=INTEGER, lb=0)
    x2 = m.add_var(var_type=INTEGER, lb=0)
    x3 = m.add_var(var_type=INTEGER, lb=0)

    # Constraints
    m += 2 * x1 - 3 * x2  + 3 * x3 <= 4
    m += 4 * x1 + x2  + x3 <= 8
    m +=  3 * x1 + 3 * x2 + x3 == 13

    # Objective function
    m.objective = -2 * x1 - x2 - x3

    m.optimize()
```

Figure 6: Código completo aquí

.

Los resultados obtenidos fueron:

```
# OptimizationStatus.OPTIMAL
# Solution to the objective function: -7.0
# x1: 0.0
# x2: 3.0
# x3: 4.0
# Execution time: 0.020112514495849 61
```

La solución obtenida coincide con la reportada en la clase práctica.