COSC-364 FLOW PLANNING ASSIGNMENT

Will Cowper
ID: 81163265
wgc22@uclive.ac.nz
Contribution: 50%

Jesse Sheehan
ID: 53366509
jps111@uclive.ac.nz
Contribution: 50%

May 29, 2019

1 Problem Formulation

Notation:

- X is the number of source nodes.
- Y is the number of transit nodes.
- Z is the number of destination nodes.
- S_i is the ith source node.
- T_k is the kth transit node.
- D_j is the jth destination node.
- h_{ij} is the demand flow between S_i and D_j. This is equal to $2i + j$.
- c_{ik} is the link capacity between S_i and T_k.
- d_{kj} is the link capacity between T_k and D_j.
- x_{ikj} is the decision variable associated with the...
- u_{ikj} is the binary decision variable associated with the... These are required because h_{ij} must be split across exactly two transit nodes.
- l_k is the load on T_k.

Note: Due to the limitations of the LP file format, many of the following equations must be rearranged for use in CPLEX. Most notably, there cannot be any variables on the right hand side of an equality or inequality.

1.1 Objective Function

$$\text{minimize}_{[r]} \tag{1}$$

1.2 Demand Constraints

$$\sum_{k=1}^{Y} x_{ikj} = 2i + j \qquad\qquad i \in \{1, \ldots, X\}, j \in \{1, \ldots, Z\} \tag{2}$$

1.3 Capacity Constraints

$$\sum_{j=1}^{Z} x_{ikj} = c_{ik} \qquad i \in \{1,\ldots,X\}, k \in \{1,\ldots,Y\} \tag{3}$$

$$\sum_{i=1}^{X} x_{ikj} = d_{kj} \qquad k \in \{1,\ldots,Y\}, j \in \{1,\ldots,Z\} \tag{4}$$

$$\sum_{k=1}^{Y} x_{ikj} = l_{k} \qquad i \in \{1,\ldots,X\}, j \in \{1,\ldots,Z\} \tag{5}$$

$$\sum_{k=1}^{Y} u_{ikj} = 2 \qquad i \in \{1,\ldots,X\}, j \in \{1,\ldots,Z\} \tag{6}$$

$$u_{ikj} \in \{0,1\} \qquad i \in \{1,\ldots,X\}, k \in \{1,\ldots,Y\}, j \in \{1,\ldots,Z\} \tag{7}$$

$$\sum_{i=1}^{X} c_{ij} \leq r \qquad j \in \{1,\ldots,Z\} \tag{8}$$

1.4 Non-Negativity Constraints

$$r \geq 0 \tag{9}$$

$$x_{ijk} \geq 0 \qquad i \in \{1,\ldots,X\}, k \in \{1,\ldots,Y\}, j \in \{1,\ldots,Z\} \tag{10}$$

2 Results

3 Appendix

3.1 Source Code

3.1.1 src/__main__.py

```
import sys

from lp_gen import generate_lp_file
from lp_utils import get_lp_filename, run_cplex

__TITLE__ = "COSC-364 Assignment 2"
__AUTHORS__ = [("Will Cowper", "81163265"), ("Jesse Sheehan", "53366509")]


def print_version():
    print('{0} by {1}'.format(__TITLE__, ', '.join(
        ["{0} ({1})".format(name, sid) for (name, sid) in __AUTHORS__])))
```

```
14
15  def print_usage():
16      print('Usage: {0} <x> <y> <z>'.format(sys.argv[0]))
17
18
19  def get_problem_parameters():
20      """ Returns a tuple containing the x, y and z parameters. """
21      try:
22          x = int(sys.argv[1])
23          y = int(sys.argv[2])
24          z = int(sys.argv[3])
25      except:
26          print_usage()
27          exit(-1)
28
29      if x <= 0:
30          print("Error: x must be strictly positive")
31          exit(-1)
32
33      if y < 0:
34          print("Error: y must be strictly positive")
35          exit(-1)
36
37      if z <= 0:
38          print("Error: z must be strictly positive")
39          exit(-1)
40
41      return x, y, z
42
43
44  def save_lp_file(filename, data):
45      try:
46          f = open(filename, 'w')
47          f.write(data)
48          f.close()
49      except:
50          print("Error: could not save file '{0}'".format(filename))
51          exit(-1)
52
53
54  def main():
55      print_version()
56      if len(sys.argv) != 4:
57          print_usage()
58          exit(-1)
59      else:
60          x, y, z = get_problem_parameters()
61          data = generate_lp_file(x, y, z)
62          filename = get_lp_filename(x, y, z)
63          save_lp_file(filename, data)
64          print("Success: saved as '{0}'".format(filename))
65          run_cplex(filename)
66
67
68  if __name__ == "__main__":
69      main()
```

../src/__main__.py

3.1.2 src/lp_utils.py

```
import functools
import subprocess
import inspect


def get_lp_filename(x, y, z):
    """ Returns the filename that the LP data should be saved to. """
    return "problem_{0}_{1}_{2}.lp".format(x, y, z)


def run_cplex(filename):
    """ Runs cplex on the LP file. """
    subprocess.run(
        ['cplex', '-c', '"read {0}"'.format(filename), '"optimize"', '"
    display solution variables -"'])


def crange(first, last):
    """ Returns a list of characters between the two characters passed in (
    inclusive).
    >>> crange('A', 'C')
    ['A', 'B', 'C']
    >>> crange('A', 'A')
    ['A']
    """
    if ord(first) > ord(last):
        raise ValueError("last must come after first")

    else:
        return [chr(i) for i in range(ord(first), ord(last) + 1)]


def repeat(obj, n):
    """ Returns a list with obj repeated n times.
    >>> repeat(1, 1)
    [1]
    >>> repeat(42, 0)
    []
    >>> repeat(5, 4)
    [5, 5, 5, 5]
    >>> repeat([1, 2], 2)
    [[1, 2], [1, 2]]
    """
    return [obj for _ in range(n)]


def perms(lists):
    """ Returns all the permutations of the elements.
    >>> perms([])
    []
    >>> perms([['a', 'b', 'c']])
    [('a',), ('b',), ('c',)]
    >>> perms([['a', 'b', 'c'], ['x', 'y', 'z']])
    [('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'x'), ('b', 'y'), ('b', 'z')
    , ('c', 'x'), ('c', 'y'), ('c', 'z')]
    """
```

```
    if len(lists) == 0:
        return []

    elif (len(lists) == 1):
        return [(x,) for x in lists[0]]

    else:
        return [(x,) + y for x in lists[0] for y in perms(lists[1:])]


def concat(permutations):
    """ Returns the permutations concatenated as strings.
    >>> concat(perms([['a', 'b', 'c']]))
    ['a', 'b', 'c']
    >>> concat(perms([['a', 'b', 'c'], ['x', 'y', 'z']]))
    ['ax', 'ay', 'az', 'bx', 'by', 'bz', 'cx', 'cy', 'cz']
    """
    return [functools.reduce(lambda x, y: x + str(y), p, '') for p in
    permutations]

def get_function_source(fn):
    src = inspect.getsource(fn)
    return src[str(src).index(':')+2:]

def get_lines(strings):
    return '\n\t'.join(strings)


if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

../src/lp_utils.py

3.1.3 src/lp_gen.py

```
from lp_utils import perms, concat, get_lines, get_function_source

# Change these variables to alter the behaviour of the LP file generator
PATH_SPLIT = 2
DEMAND_FLOW = lambda i, j: 2 * i + j

TEMPLATE = """\
\\ COSC-364 Assignment 2, LP Output File
\\ Parameters: X={}, Y={}, Z={}, Split={}, Demand={}

MINIMIZE
\tr

SUBJECT TO

\t\\ DEMAND CONSTRAINTS
\t{}

\t\\ CAPACITY CONSTRAINTS FOR LINKS BETWEEN SOURCE AND TRANSIT NODES
\t{}
```

```
\t\\ CAPACITY CONSTRAINTS FOR LINKS BETWEEN TRANSIT AND DESTINATION NODES
\t{}

\t\\ OBJECTIVE FUNCTION LOAD CONSTRAINTS
\t{}

\t\\ TRANSIT NODE LOAD CONSTRAINTS
\t{}

\t\\ BINARY VARIABLE AND DECISION VARIABLE CONSTRAINTS
\t{}

\t\\ BINARY VARIABLE CONSTRAINTS (ONLY 2 ACTIVE TRANSIT NODES)
\t{}

BOUNDS

\t\\ NON-NEGATIVITY CONSTRAINTS
\tr >= 0
\t{}

BIN

\t\\ BINARY VARIABLES
\t{}

END
"""


def get_nodes(x, y, z):
    """ Returns a tuple containing the source, transit and destination node ids as integers. """
    s = list(range(1, x + 1))
    t = list(range(1, y + 1))
    d = list(range(1, z + 1))
    return s, t, d


def get_demand_constraints(s, t, d):
    """ Returns a list of demand constraints. """
    return [' + '.join(["x_{0}{1}{2}".format(i, k, j) for k in t]) + ' = {0}'.format(DEMAND_FLOW(i, j))
            for (i, j) in perms([s, d])]


def get_source_transit_capacity_constraints(s, t, d):
    """ Returns a list of capacity constraints for the links between the source and transit nodes. """
    return \
        [' + '.join(["x_{0}{1}{2}".format(i, k, j) for j in d]) +
            ' - c_{0}{1} = 0'.format(i, k) for (i, k) in perms([s, t])]


def get_transit_destination_capacity_constraints(s, t, d):
    """ Returns a list of capacity constraints for the links between the transit and destination nodes. """
    return \
```

```
76          [' + '.join(["x_{0}{1}{2}".format(i, k, j) for i in s]) +
                ' - d_{0}{1} = 0'.format(k, j) for (k, j) in perms([t, d])]
78
80  def get_transit_load_constraints(s, t, d):
        """ Returns the list of transit load constraints. """
82      return [' + '.join(["x_{0}{1}{2}".format(i, k, j) for (i, j) in perms([
        s, d])]) +
                ' - l_{0} = 0'.format(k) for k in t]
84
    def get_objective_function_load_constraints(s, t, d):
86      """ Returns the list of objective function load constraints. """
        return [' + '.join(["c_{0}{1}".format(i, j) for i in s]) +
88              ' - r <= 0' for j in d]
90  def get_binary_and_decision_variable_constraints(s, t, d):
        """ Returns the binary and decision variable constraints. """
92      return []
94
    def get_binary_constraints(s, t, d):
96      """ Returns a list of binary variable constraints. """
        return [' + '.join(["u_{0}{1}{2}".format(i, k, j) for k in t]) + ' = {}
        '.format(PATH_SPLIT)
98              for (i, j) in perms([s, d])]
100
    def get_binary_variables(s, t, d):
102     """ Returns a list of binary variables. """
        return ["u_{0}{1}{2}".format(i, k, j) for (i, k, j) in perms([s, t, d])
        ]
104
106 def get_non_negativity_constraints(s, t, d):
        """ Returns a list of non-negativity constraints. """
108     return ["x_{0} >= 0".format(subscript) for subscript in concat(perms([s
        , t, d]))]
110 def generate_lp_file(x, y, z):
        """ Returns the LP file contents as per the project specification. """
112     s, t, d = get_nodes(x, y, z)

114     demand_constraints = get_lines(get_demand_constraints(s, t, d))
        source_transit_capacity_constraints = get_lines(
116         get_source_transit_capacity_constraints(s, t, d))
        transit_destination_capacity_constraints = get_lines(
118         get_transit_destination_capacity_constraints(s, t, d))
        non_negativity_constraints = get_lines(get_non_negativity_constraints(
120         s, t, d))
        objective_function_load_constraints = get_lines(
        get_objective_function_load_constraints(s, t, d))
122     transit_load_constraints = get_lines(
            get_transit_load_constraints(s, t, d))
124     binary_and_decision_constraints = get_lines(
        get_binary_and_decision_variable_constraints(s, t, d))
        binary_variable_constraints = get_lines(get_binary_constraints(s, t, d)
        )
126     binary_variables = get_lines(get_binary_variables(s, t, d))
```

```
128     return TEMPLATE.format(
            x,
130         y,
            z,
132         PATH_SPLIT,
            get_function_source(DEMAND_FLOW),
134         demand_constraints,
            source_transit_capacity_constraints,
136         transit_destination_capacity_constraints,
            objective_function_load_constraints,
138         transit_load_constraints,
            binary_and_decision_constraints,
140         binary_variable_constraints,
            non_negativity_constraints,
142         binary_variables)
```

../src/lp_gen.py

3.2 Generated LP File

3.2.1 problem_3_2_4.lp

3.3 Plagiarism Declaration