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.

1.1 Objective Function

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

1.2 Demand Constraints

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

1.3 Capacity Constraints

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

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

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

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

1.4 Non-Negativity Constraints

$$r \geq 0 \tag{7}$$

$$x_{ijk} \geq 0 \qquad i \in \{1,\dots,X\}, k \in \{1,\dots,Y\}, j \in \{1,\dots,Z\} \tag{8}$$

2 Results

3 Appendix

3.1 Source Code

3.1.1 src/__main__.py

```python
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__])))


def print_usage():
    print('Usage: {0} <x> <y> <z>'.format(sys.argv[0]))


def get_problem_parameters():
    """ Returns a tuple containing the x, y and z parameters. """
    try:
        x = int(sys.argv[1])
        y = int(sys.argv[2])
        z = int(sys.argv[3])
    except:
        print_usage()
        exit(-1)

    if x <= 0:
        print("Error: x must be strictly positive")
        exit(-1)

    if y < 0:
        print("Error: y must be strictly positive")
        exit(-1)

    if z <= 0:
        print("Error: z must be strictly positive")
        exit(-1)

    return x, y, z


def save_lp_file(filename, data):
    try:
        f = open(filename, 'w')
        f.write(data)
        f.close()
    except:
        print("Error: could not save file '{0}'".format(filename))
        exit(-1)


def main():
    print_version()
    if len(sys.argv) != 4:
        print_usage()
        exit(-1)
    else:
        x, y, z = get_problem_parameters()
        data = generate_lp_file(x, y, z)
        filename = get_lp_filename(x, y, z)
        save_lp_file(filename, data)
        print("Success: saved as '{0}'".format(filename))
        run_cplex(filename)


if __name__ == "__main__":
    main()
```

../src/__main__.py

3.1.2 src/lp_utils.py

```
import functools
import subprocess
```

```python

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}" "optimize" "display solution variables -"'.format(filename))


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]


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

../src/lp_utils.py

3.1.3 src/lp_gen.py

```
from lp_utils import perms, concat

template = """\
\\ COSC-364 Assignment 2, LP Output File

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(2 * 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 \
        [' + '.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])]


def get_transit_load_constraints(s, t, d):
    """ Returns the list of transit load constraints. """
    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]

def get_objective_function_load_constraints(s, t, d):
    """ Returns the list of objective function load constraints. """
    return []
    #return [' + '.join(["X_{0}{1}{2}".format(i, k, j) for (i, j) in perms([s, d])]) +
    #        ' - r <= 0' for k in t]

def get_binary_and_decision_variable_constraints(s, t, d):
    """ Returns the binary and decision variable constraints. """
    return []


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


def get_binary_variables(s, t, d):
    """ Returns a list of binary variables. """
    return ["U_{0}{1}{2}".format(i, k, j) for (i, k, j) in perms([s, t, d])]


def get_non_negativity_constraints(s, t, d):
    """ Returns a list of non-negativity constraints. """
    return ["X_{0} >= 0".format(subscript) for subscript in concat(perms([s, t, d]))]


def generate_lp_file(x, y, z):
    """ Returns the LP file contents as per the project specification. """
    s, t, d = get_nodes(x, y, z)

    demand_constraints = '\n\t'.join(get_demand_constraints(s, t, d))
    source_transit_capacity_constraints = '\n\t'.join(
        get_source_transit_capacity_constraints(s, t, d))
    transit_destination_capacity_constraints = '\n\t'.join(
        get_transit_destination_capacity_constraints(s, t, d))
    non_negativity_constraints = '\n\t'.join(get_non_negativity_constraints(
        s, t, d))
    objective_function_load_constraints = '\n\t'.join(get_objective_function_load_constraints(s, t, d))
    transit_load_constraints = '\n\t'.join(
        get_transit_load_constraints(s, t, d))
    binary_and_decision_constraints = '\n\t'.join(get_binary_and_decision_variable_constraints(s, t, d))
    binary_variable_constraints = '\n\t'.join(get_binary_constraints(s, t, d))
    binary_variables = '\n\t'.join(get_binary_variables(s, t, d))

    return template.format(
        demand_constraints,
        source_transit_capacity_constraints,
        transit_destination_capacity_constraints,
        objective_function_load_constraints,
        transit_load_constraints,
        binary_and_decision_constraints,
        binary_variable_constraints,
        non_negativity_constraints,
        binary_variables)
```

../src/lp_gen.py

3.2 Generated LP File

3.2.1 problem_3_2_4.lp

3.3 Plagiarism Declaration