TRAFFIC PLANNING

ASSIGNMENT 2

COSC364-19S1 INTERNET TECHNOLOGY AND ENGINEERING

Will Cowper

ID: 81163265

Contribution: 50%

Jesse Sheehan

ID: 53366509

Contribution: 50%

May 29, 2019

1 Problem Formulation

Given a network (figure 1) with X source nodes, Y transit nodes and Z destination nodes, a program was designed to generate an LP file that could be used by CPLEX to determine certain network characteristics.

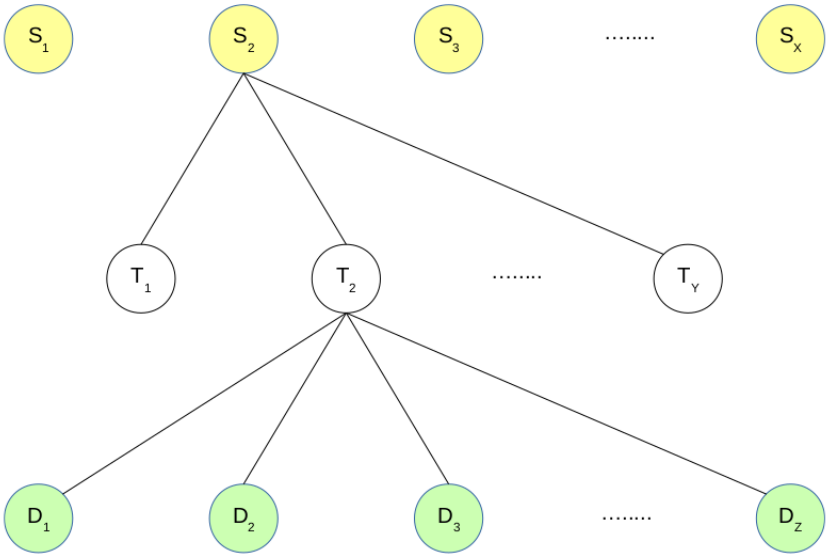

Figure 1: An example network (A. Willig, April 2019).

Traffic travelling from S_i to D_j must travel through exactly 2 transit nodes with a total demand volume of h_{ij} (equation 10). Furthermore, the load upon each transit node

must be balanced. This problem was solved with the use of binary variable constraints (equations 6, 7 and 9) and the minimisation of our objective function (equation 1). All normal non-negativity constraints were applied (equations 11, 12, 13 and 14).

The following network properties were solved for:

- The capacities of each link (equations 3 and 4).
- The load on each transit node (equation 5).
- The value of each flow (equations 2 and 8).

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 path S_i-T_k-D_j.
- u_{ikj} is the binary decision variable associated with x_{ikj}. These are required because h_{ij} must be split across exactly 2 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

$$\text{minimize}_{[x,c,d,r]}\ r \tag{1}$$

1.2 Constraints

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

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

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

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

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

$$x_{ikj} = \frac{u_{ikj} h_{ij}}{2} \qquad i \in \{1,\dots,X\}, k \in \{1,\dots,Y\}, j \in \{1,\dots,Z\} \tag{7}$$

$$\sum_{i=1}^{X} \sum_{j=1}^{Z} x_{ikj} \leq r \qquad k \in \{1,\dots,Y\} \tag{8}$$

$$u_{ikj} \in \{0,1\} \qquad i \in \{1,\dots,X\}, k \in \{1,\dots,Y\}, j \in \{1,\dots,Z\} \tag{9}$$

$$h_{ij} = 2i + j \qquad i \in \{1,\dots,X\}, j \in \{1,\dots,Z\} \tag{10}$$

1.3 Non-Negativity Constraints

$$r \geq 0 \tag{11}$$

$$x_{ikj} \geq 0 \qquad i \in \{1,\dots,X\}, k \in \{1,\dots,Y\}, j \in \{1,\dots,Z\} \tag{12}$$

$$c_{ik} \geq 0 \qquad i \in \{1,\dots,X\}, k \in \{1,\dots,Y\} \tag{13}$$

$$d_{kj} \geq 0 \qquad k \in \{1,\dots,Y\}, j \in \{1,\dots,Z\} \tag{14}$$

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__])))


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
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 \
        [' + '.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 [' + '.join(["c_{0}{1}".format(i, j) for i in s]) +
            ' - r <= 0' for j in d]

def get_binary_and_decision_variable_constraints(s, t, d):
    """ Returns the binary and decision variable constraints. """
    return ['{3} x_{0}{1}{2} - {4} u_{0}{1}{2} = 0'.format(i, k, j,
    PATH_SPLIT, DEMAND_FLOW(i, j)) for (i, k, j) in perms([s, t, d])]


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]) + ' = {}
    '.format(PATH_SPLIT)
            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}{1}{2} >= 0".format(i, k, j) for (i, k, j) in perms([s, t, d])] + ["c_{0}{1} >= 0".format(i, k) for (i, k) in perms([s, t])] + ["d_{0}{1} >= 0".format(k, j) for (k, j) in perms([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 = get_lines(get_demand_constraints(s, t, d))
    source_transit_capacity_constraints = get_lines(
        get_source_transit_capacity_constraints(s, t, d))
    transit_destination_capacity_constraints = get_lines(
        get_transit_destination_capacity_constraints(s, t, d))
    non_negativity_constraints = get_lines(get_non_negativity_constraints(
        s, t, d))
    objective_function_load_constraints = get_lines(
        get_objective_function_load_constraints(s, t, d))
    transit_load_constraints = get_lines(
        get_transit_load_constraints(s, t, d))
    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))
    binary_variables = get_lines(get_binary_variables(s, t, d))

    return TEMPLATE.format(
        x,
        y,
        z,
        PATH_SPLIT,
        get_function_source(DEMAND_FLOW),
        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