

FLOW PLANNING

ASSIGNMENT 2

COSC364-19S1 INTERNET TECHNOLOGY AND ENGINEERING

Will Cowper

ID: 81163265

Contribution: 50%

Jesse Sheehan

ID: 53366509

Contribution: 50%

May 30, 2019

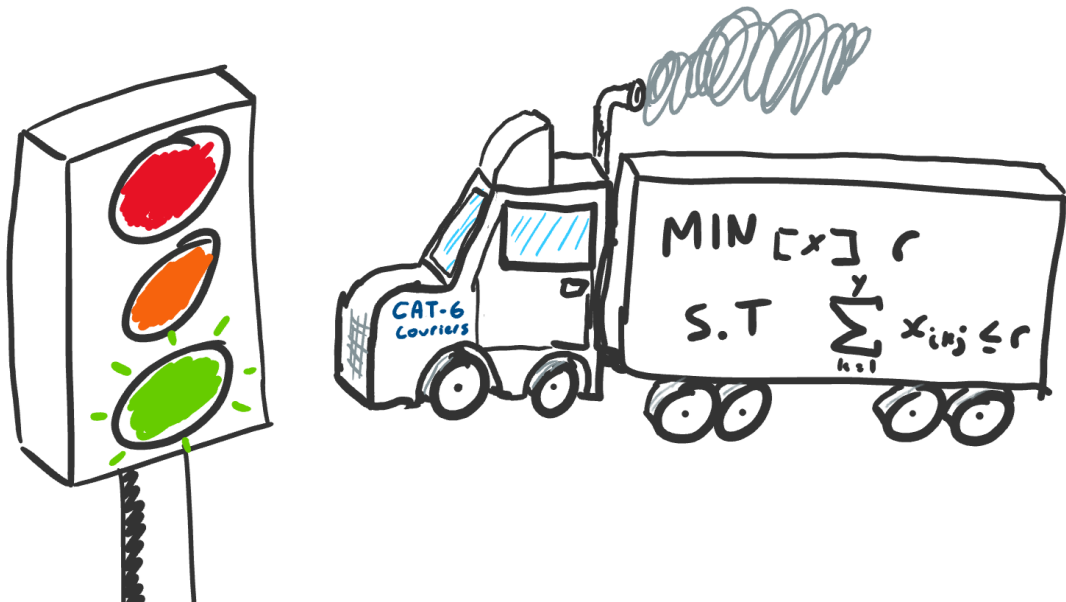


Figure 1: Traffic problems are not unique to computer networks.

Plagiarism Declaration

This form needs to accompany your COSC 364 assignment submission.

I understand that plagiarism means taking someone else's work (text, program code, ideas, concepts) and presenting them as my own, without proper attribution. Taking someone else's work can include verbatim copying of text, figures/images, or program code, or it can refer to the extensive use of someone else's original ideas, algorithms or concepts.

I hereby declare that:

- My assignment is my own original work. I have not reproduced or modified code, figures/images, or writings of others without proper attribution. I have not used original ideas and concepts of others and presented them as my own.
- I have not allowed others to copy or modify my own code, figures/images, or writings. I have not allowed others to use original ideas and concepts of mine and present them as their own.
- I accept that plagiarism can lead to consequences, which can include partial or total loss of marks, no grade being awarded and other serious consequences, including notification of the University Proctor.

Name:

Will Cowper Jesse Sheehan

Student ID:

81163265 53366500

Signature:

Date:

29-5-19 29/5/19

1 Problem Description

Given a network (figure 2) 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.

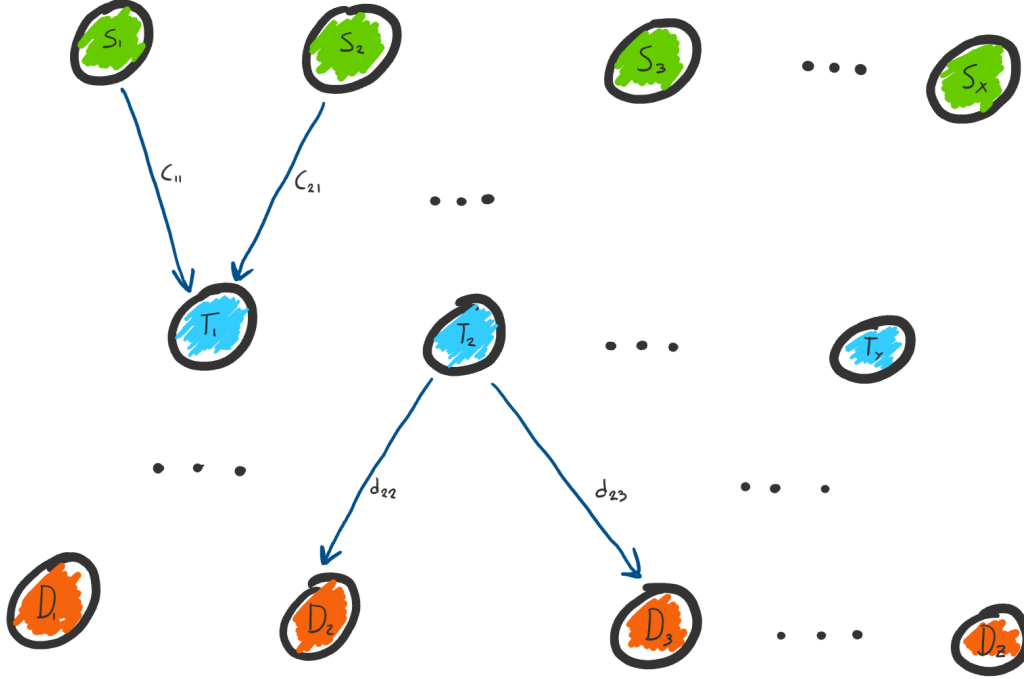


Figure 2: An example network showing nodes S_i , T_k and D_j and links c_{ik} and d_{kj} , $i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\}$.

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.

2 Problem Formulation

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 i th source node.
- T_k is the k th transit node.
- D_j is the j th 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.

Python, BASH and PowerShell scripts (section 4.1) were used to automate the process (figure 3) of generating the LP files, analysing the CPLEX data, and producing graphs.

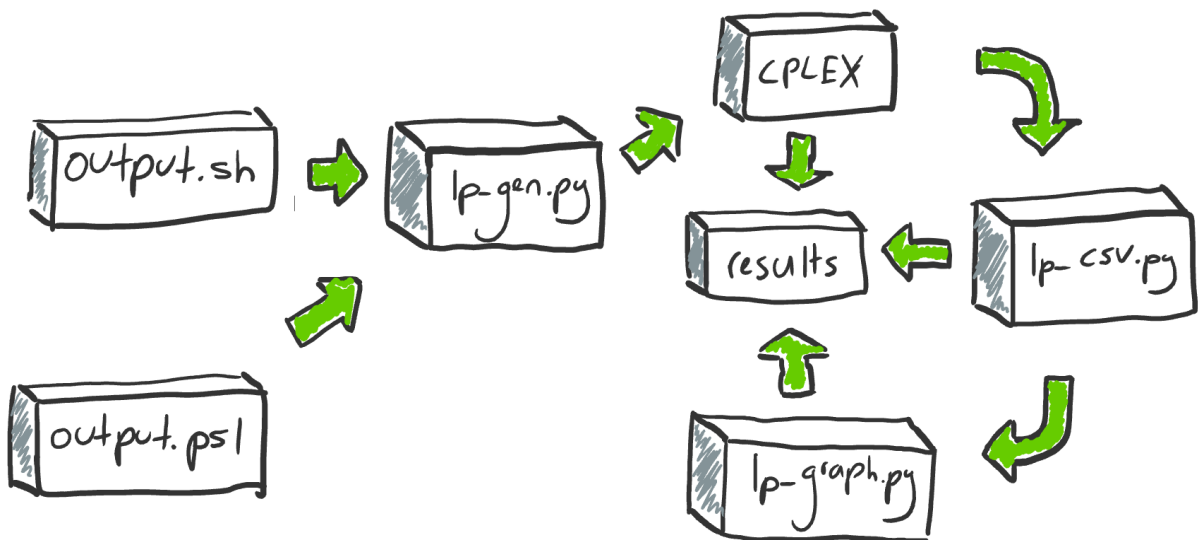


Figure 3: A graphical interpretation of the script execution used.

2.1 Objective Function

$$\text{minimize}_{[x,c,d,r]} r \quad (1)$$

2.2 Constraints

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

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

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

$$\sum_{k=1}^Y x_{ikj} = l_k \quad i \in \{1, \dots, X\}, j \in \{1, \dots, Z\} \quad (5)$$

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

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

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

$$u_{ikj} \in \{0, 1\} \quad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\} \quad (9)$$

$$h_{ij} = 2i + j \quad i \in \{1, \dots, X\}, j \in \{1, \dots, Z\} \quad (10)$$

2.3 Non-Negativity Constraints

$$r \geq 0 \quad (11)$$

$$x_{ikj} \geq 0 \quad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\} \quad (12)$$

$$c_{ik} \geq 0 \quad i \in \{1, \dots, X\}, k \in \{1, \dots, Y\} \quad (13)$$

$$d_{kj} \geq 0 \quad k \in \{1, \dots, Y\}, j \in \{1, \dots, Z\} \quad (14)$$

3 Results

LP files were generated with parameters $X = Z = 9, Y \in \{3, 4, 5, 6, 7, 8\}$. These were then processed with CPLEX, recording the time taken to solve each problem. Important data points were extracted from the CPLEX output and are listed in table 1.

Table 1: The raw data as extracted and processed from the CPLEX output.

Y	Time (ms)	Links	Load Spread	Max. c_{ik}	Max. d_{kj}
3	81.6132	52	0.0	94.500000	73.500000
4	117.3323	68	0.5	8.500000	9.500000
5	285.8704	83	0.0	9.500000	9.500000
6	443.1351	101	0.0	83.000000	9.500000
7	223.422	118	1.5	9.500000	9.500000
8	987.7775	134	0.5	9.500000	9.500000

An analysis of these results confirms many assumptions that were made about the problem. The number of non-zero link capacities increases linearly (figure 5), the transit node load spread is very close to (if not exactly) zero for most networks (figure 6), and the amount of time taken to solve the problem increases (almost) non-linearly as the number of transit nodes increases (figure 4). It is important to note that the load on the transit nodes was only able to be balanced in three of the networks, however the load spread on the remaining three networks is very low. CPLEX has done its best to equalise the loads on each transit node but could not balance them in some cases due to the demand flows and the number of nodes.

The most obvious feature of the results is the data for the $Y = 7$ network. It appears to be an outlier as it takes less time to solve than the $Y = 5$ network and has the highest transit node load spread by a factor of 3. **Explain this behaviour!!!**

The second odd feature of the data is the highest link capacities of the $Y = 3$ and $Y = 6$ networks (figure 7). **Also explain this behaviour!!!**

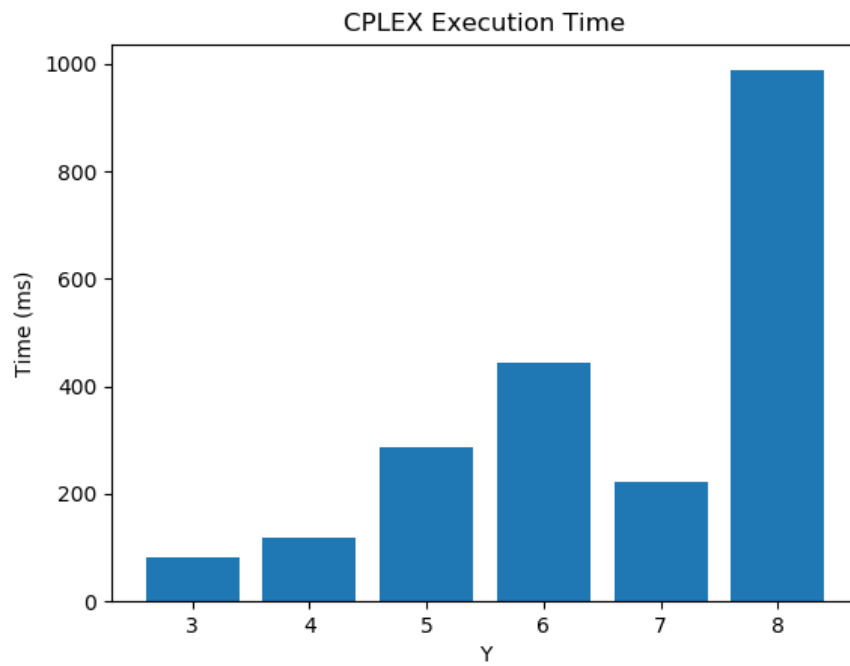


Figure 4: The time taken to execute the LP file in CPLEX for each network.

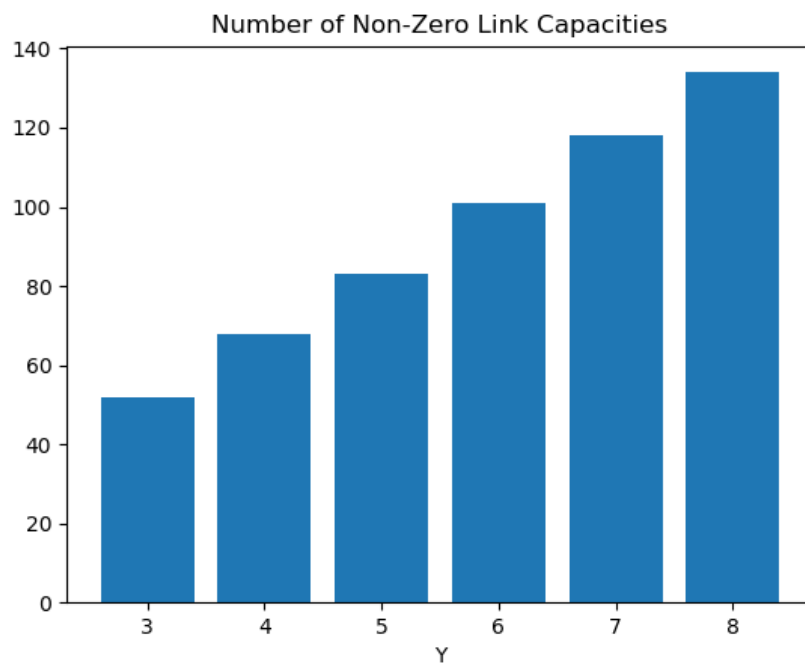


Figure 5: The number of non-zero link capacities in each network.

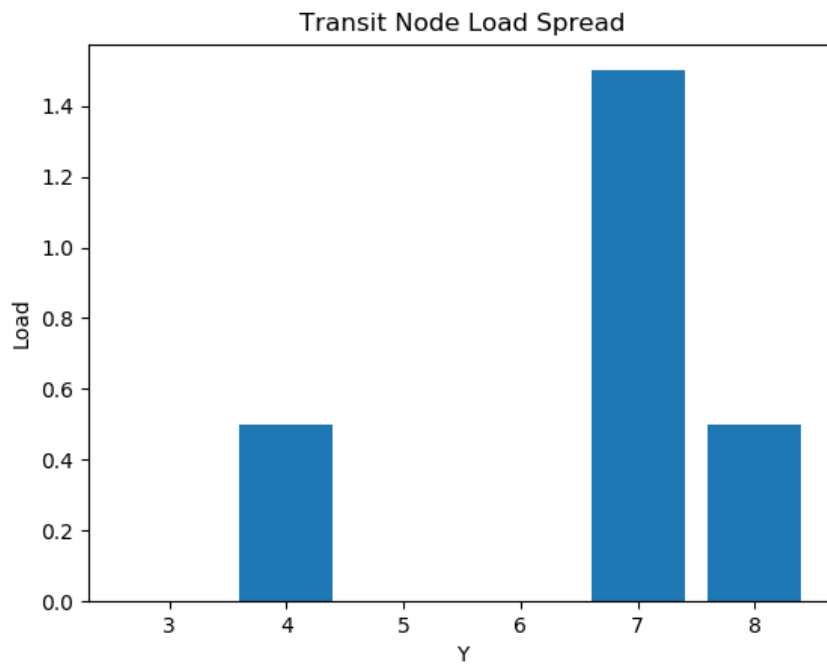


Figure 6: The amount of spread in the load for all transit nodes in each network.

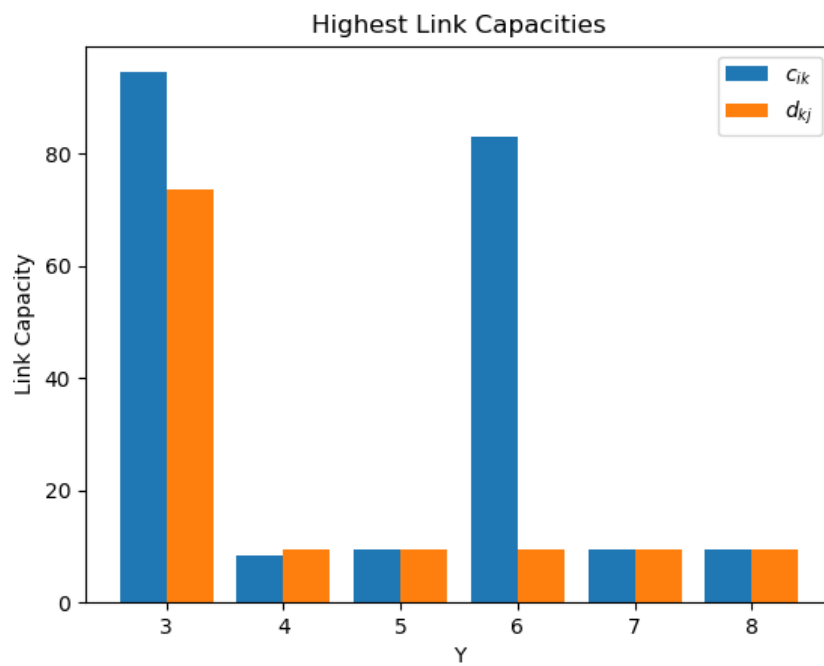


Figure 7: The highest capacity links for each network. Both the C_{ik} and D_{kj} links are listed.

4 Appendix

4.1 Source Code

4.1.1 src/lp_gen.py

This script is responsible for producing a valid LP file from the given command line parameters.

```

1 # lp_gen.py
2 #
3 # COSC364 Assignment 2
4 # 30/05/2019
5 # Written by Will Cowper, Jesse Sheehan
6
7 import inspect
8 import functools
9 import sys
10 import os.path
11
12 __TITLE__ = "COSC-364 Assignment 2 LP Generator"
13 __AUTHORS__ = [("Will Cowper", "81163265"), ("Jesse Sheehan", "53366509")]
14
15 # Change these variables to alter the behaviour of the LP file generator
16 PATH_SPLIT = 2
17
18 def DEMANDFLOW(i, j): return 2 * i + j
19
20
21
22 TEMPLATE = """\
23 \ \ {}, LP Output File
24 \ \ Written by {}
25 \ \ Parameters: X={}, Y={}, Z={}, Split={}, Demand={}
26
27 MINIMIZE
28 \tr
29
30 SUBJECT TO
31
32 \t\ \ DEMAND CONSTRAINTS
33 \t{}
34
35 \t\ \ CAPACITY CONSTRAINTS FOR LINKS BETWEEN SOURCE AND TRANSIT NODES
36 \t{}
37
38 \t\ \ CAPACITY CONSTRAINTS FOR LINKS BETWEEN TRANSIT AND DESTINATION NODES
39 \t{}
40
41 \t\ \ OBJECTIVE FUNCTION LOAD CONSTRAINTS
42 \t{}
43
44 \t\ \ TRANSIT NODE LOAD CONSTRAINTS
45 \t{}
46
47 \t\ \ BINARY VARIABLE AND DECISION VARIABLE CONSTRAINTS
48 \t{}

```

```

50 | \t\\ BINARY VARIABLE CONSTRAINTS (ONLY 2 ACTIVE TRANSIT NODES)
    | \t{}
52 |
    | BOUNDS
54 |
    | \t\\ NON-NEGATIVITY CONSTRAINTS
56 | \tr >= 0
    | \t{}
58 |
    | BIN
60 |
    | \t\\ BINARY VARIABLES
62 | \t{}
64 | END
    | """
66 |
    | # DEFINE SOME UTILITY FUNCTIONS
68 |
70 | def get_lp_filename(x, y, z):
    | """ Returns the filename that the LP data should be saved to. """
72 |     return "problem_{0}_{1}_{2}.lp".format(x, y, z)
74 |
76 | 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']
    | """
82 |     if ord(first) > ord(last):
    |         raise ValueError("last must come after first")
84 |
    |     else:
86 |         return [chr(i) for i in range(ord(first), ord(last) + 1)]
88 |
90 | 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]]
    | """
100 |     return [obj for _ in range(n)]
102 |
104 | def perms(lists):
    | """ Returns all the permutations of the elements.
    | >>> perms([])
    | []
106 |

```

```

108 >>> perms([[ 'a', 'b', 'c ']])
110 [[ ('a',), ('b',), ('c',)]]
>>> perms([[ 'a', 'b', 'c '], [ 'x', 'y', 'z ']])
110 [[ ('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'x'), ('b', 'y'), ('b', 'z'),
, ('c', 'x'), ('c', 'y'), ('c', 'z')]
"""
112 if len(lists) == 0:
    return []
114
116 elif (len(lists) == 1):
    return [(x,) for x in lists[0]]
118
120 else:
    return [(x,) + y for x in lists[0] for y in perms(lists[1:])]
122
def concat(permutations):
    """ Returns the permutations concatenated as strings.
124 >>> concat(perms([[ 'a', 'b', 'c ']]))
    [ 'a', 'b', 'c ']
126 >>> concat(perms([[ 'a', 'b', 'c '], [ 'x', 'y', 'z ']]))
    [ 'ax', 'ay', 'az', 'bx', 'by', 'bz', 'cx', 'cy', 'cz ']
128 """
    return [functools.reduce(lambda x, y: x + str(y), p, '') for p in
permutations]
130
132 def get_function_source(fn):
    src = inspect.getsource(fn)
134 return src[str(src).index('return')+7:]
136
138 def get_lines(strings):
    return '\n\t'.join(strings)
140
# DEFINE SOME FUNCTIONS SPECIFIC TO THE PROBLEM
142
144 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))
148 return s, t, d
150
152 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(DEMANDFLOW(i, j))
154 for (i, j) in perms([s, d])]
156
158 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 \

```

```

160         [ ' + '.join(["x-{}{}{}2".format(i, k, j) for j in d]) +
162             ' - c-{}{}1 = 0'.format(i, k) for (i, k) in perms([s, t])]

164 def get_transit_destination_capacity_constraints(s, t, d):
165     """ Returns a list of capacity constraints for the links between the
166     transit and destination nodes. """
167     return \
168         [ ' + '.join(["x-{}{}{}2".format(i, k, j) for i in s]) +
169             ' - d-{}{}1 = 0'.format(k, j) for (k, j) in perms([t, d])]

170 def get_transit_load_constraints(s, t, d):
171     """ Returns the list of transit load constraints. """
172     return [ ' + '.join(["x-{}{}{}2".format(i, k, j) for (i, j) in perms([
173         s, d])]) +
174             ' - l-{} = 0'.format(k) for k in t]

176 def get_objective_function_load_constraints(s, t, d):
177     """ Returns the list of objective function load constraints. """
178     return [ ' + '.join(["c-{}{}1".format(i, j) for i in s]) +
179             ' - r <= 0' for j in d]

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

186

188 def get_binary_constraints(s, t, d):
189     """ Returns a list of binary variable constraints. """
190     return [ ' + '.join(["u-{}{}{}2".format(i, k, j) for k in t]) + ' = {}
191         '.format(PATH_SPLIT)
192             for (i, j) in perms([s, d])]

194 def get_binary_variables(s, t, d):
195     """ Returns a list of binary variables. """
196     return ["u-{}{}{}2".format(i, k, j) for (i, k, j) in perms([s, t, d])
197         ]

198

200 def get_non_negativity_constraints(s, t, d):
201     """ Returns a list of non-negativity constraints. """
202     return ["x-{}{}{}2 >= 0".format(i, k, j) for (i, k, j) in perms([s, t,
203         d])]) + ["c-{}{}1 >= 0".format(i, k) for (i, k) in perms([s, t])] + ["
204         d-{}{}1 >= 0".format(k, j) for (k, j) in perms([t, d])]

206

208 def generate_lp_file(title, authors, x, y, z):
209     """ Returns the LP file contents as per the project specification. """
210     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(
212     get_transit_destination_capacity_constraints(s, t, d))
non_negativity_constraints = get_lines(get_non_negativity_constraints(
214     s, t, d))
objective_function_load_constraints = get_lines(
216     get_objective_function_load_constraints(s, t, d))
transit_load_constraints = get_lines(
218     get_transit_load_constraints(s, t, d))
binary_and_decision_constraints = get_lines(
220     get_binary_and_decision_variable_constraints(s, t, d))
binary_variable_constraints = get_lines(get_binary_constraints(s, t, d)
222 )
binary_variables = get_lines(get_binary_variables(s, t, d))

224 return TEMPLATE.format(
    title ,
226     authors ,
    x,
228     y,
    z,
230     PATH.SPLIT,
    get_function_source(DEMANDFLOW) ,
232     demand_constraints ,
    source_transit_capacity_constraints ,
234     transit_destination_capacity_constraints ,
    objective_function_load_constraints ,
236     transit_load_constraints ,
    binary_and_decision_constraints ,
238     binary_variable_constraints ,
    non_negativity_constraints ,
240     binary_variables)

242 # DEFINE SOME HELPERS FOR GETTING THE THING RUNNING
244
246 def print_version():
    print( '{0} by {1}'.format(--TITLE--, get_author_string()))

248
250 def print_usage():
    print( 'Usage: {0} <x> <y> <z> [output directory]'.format(sys.argv[0]))

252
254 def get_problem_parameters():
    """ Returns a tuple containing the x, y and z parameters. """
    try:
256         x = int(sys.argv[1])
258         y = int(sys.argv[2])
260         z = int(sys.argv[3])
    except:
262         print_usage()
264         exit(-1)

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

    if x >= 10:

```

```
268     print("Error: x must be less than ten")
269     exit(-1)
270
271 if y <= 0:
272     print("Error: y must be strictly positive")
273     exit(-1)
274
275 if y >= 10:
276     print("Error: y must be less than ten")
277     exit(-1)
278
279 if z <= 0:
280     print("Error: z must be strictly positive")
281     exit(-1)
282
283 if z >= 10:
284     print("Error: z must be less than ten")
285     exit(-1)
286
287 return x, y, z
288
289 def save_lp_file(filename, data):
290     try:
291         f = open(filename, 'w')
292         f.write(data)
293         f.close()
294     except:
295         print("Error: could not save file '{0}'".format(filename))
296         exit(-1)
297
298
299 def get_author_string():
300     return ', '.join(
301         ["{0} ({1})".format(name, sid) for (name, sid) in _AUTHORS_])
302
303
304 def main():
305     print_version()
306     if len(sys.argv) != 4 and len(sys.argv) != 5:
307         print_usage()
308         exit(-1)
309     else:
310         output_dir = '.'
311         if len(sys.argv) == 5:
312             output_dir = sys.argv[4]
313
314         x, y, z = get_problem_parameters()
315         data = generate_lp_file(_TITLE_, get_author_string(), x, y, z)
316         filename = os.path.join(output_dir, get_lp_filename(x, y, z))
317         save_lp_file(filename, data)
318         print("Success: saved as '{0}'".format(filename))
319
320
321 if __name__ == "__main__":
322     main()
```

../src/lp_gen.py

4.1.2 src/lp_csv.py

This script is responsible for converting the output of the CPLEX log files into a single CSV file for further processing.

```

1 # lp_csv.py
2 #
3 # COSC364 Assignment 2
4 # 30/05/2019
5 # Written by Will Cowper, Jesse Sheehan
6
7 import csv
8 import sys
9 import os.path
10
11
12 def csvWrite(data):
13     with open(sys.argv[2], 'a', newline='') as csvFile:
14         writer = csv.writer(csvFile)
15         writer.writerow(data)
16
17
18 def floatmap(enumerable):
19     return list(map(lambda x: float(x), enumerable))
20
21
22 def openFile(Y):
23     with open(os.path.join(sys.argv[1], '{0}.txt'.format(Y)), 'r') as
24         in_file:
25         stripped = [line.strip() for line in in_file.readlines()]
26         lines = [line for line in stripped if line]
27         data = []
28         # Y
29         data.append(Y)
30         # elapsed time
31         data.append(max(parseFile("elapsed_", lines)))
32         # no of non-zero c and d links
33         data.append(len(parseFile("c_", lines)) + len(parseFile("d_", lines
34         )))
35         # transit load spread (largest-transit-node-load -
36         # smallest-transit-node-load)
37         data.append(max(floatmap(parseFile("l_", lines))) -
38                     min(floatmap(parseFile("l_", lines))))
39         # highest cap c network
40         data.append(max(parseFile("c_", lines)))
41         # highest cap d network
42         data.append(max(parseFile("d_", lines)))
43         csvWrite(data)
44
45
46 '''Returns a list of all values that start with the given string'''
47
48 def parseFile(string, lines):
49     values = []
50     for line in lines:
51         if line.startswith(string):
52             values.append(line.split()[1])

```

```

52     return values
54
55 if __name__ == "__main__":
56     if len(sys.argv) != 3:
57         print("Usage: {0} <input directory> <csv file>".format(sys.argv[0])
58     )
59     exit(-1)
60
61     # delete the CSV, otherwise we will append to it
62     os.unlink(sys.argv[2])
63
64     openFile(3)
65     openFile(4)
66     openFile(5)
67     openFile(6)
68     openFile(7)
69     openFile(8)
70
71     print("Saved CSV data to '{0}'".format(sys.argv[2]))

```

../src/lp-csv.py

4.1.3 src/lp_graph.py

This script is responsible for reading the CSV file and producing several graphs.

```

# lp-graph.py
2 #
3 # COSC364 Assignment 2
4 # 30/05/2019
5 # Written by Will Cowper, Jesse Sheehan
6
7 import csv
8 import sys
9 import os.path
10
11 try:
12     import numpy as np
13 except:
14     print("Error: could not load 'numpy'. Install with 'pip install numpy'
15     and then try again.")
16     exit(-1)
17
18 try:
19     import matplotlib.pyplot as plt
20 except:
21     print("Error: could not load 'matplotlib'. Install with 'pip install
22     matplotlib' and then try again.")
23     exit(-1)
24
25 def get_data(data, key):
26     return list(map(lambda d: d[key], data))
27
28

```



```

def get_time(data):
30     return get_data(data, "time")

32
def get_len_nonzero_links(data):
34     return get_data(data, "len_links")

36
def get_transit_load_spread(data):
38     return get_data(data, "load_spread")

40
def get_max_cap_c(data):
42     return get_data(data, "max_cap_c")

44
def get_max_cap_d(data):
46     return get_data(data, "max_cap_d")

48
def get_Y(data):
50     return get_data(data, "Y")

52
def save_execution_time_plot(filename, data):
54     """ Saves a plot of execution time. """
    plt.bar(get_Y(data), get_time(data))
56     plt.xlabel("Y")
    plt.ylabel("Time (ms)")
58     plt.title("CPLEX Execution Time")
    plt.savefig(filename)
60     plt.close()
    print("Saved '{}'".format(filename))

62
def save_num_nonzero_links_plot(filename, data):
64     """ Saves a plot of the number of non-zero links. """
    plt.bar(get_Y(data), get_len_nonzero_links(data))
66     plt.xlabel("Y")
    plt.ylabel("")
68     plt.title("Number of Non-Zero Link Capacities")
    plt.savefig(filename)
70     plt.close()
    print("Saved '{}'".format(filename))

72
def save_transit_load_spread_plot(filename, data):
74     """ Saves a plot of the transit load spread. """
    plt.bar(get_Y(data), get_transit_load_spread(data))
76     plt.xlabel("Y")
    plt.ylabel("Load")
78     plt.title("Transit Node Load Spread")
    plt.savefig(filename)
80     plt.close()
    print("Saved '{}'".format(filename))

82
def save_highest_capacity_links_plot(filename, data):
84
86

```

```

""" Saves a plot of the transit load spread. """
width = 0.4
Ys = np.array(get_Y(data))
cs = plt.bar(Ys, get_max_cap_c(data), width, label="$c_{ik}$")
ds = plt.bar(Ys + width, get_max_cap_d(data), width, label="$d_{kj}$")
plt.xticks(Ys + width / 2, map(lambda x: int(x), Ys))
plt.legend(handles=[cs, ds])
plt.xlabel("Y")
plt.ylabel("Link Capacity")
plt.title("Highest Link Capacities")
plt.savefig(filename)
plt.close()
print("Saved '{}'".format(filename))

def get_data_from_csv(csv_filename):
    """ Returns an array of dictionaries containing the CSV data. """
    with open(csv_filename, newline='') as csv_file:
        csv_reader = csv.DictReader(csv_file, fieldnames=[
            "Y", "time", "len_links", "load_spread",
            "max_cap_c", "max_cap_d"])
        rows = []
        for row in csv_reader:
            d = {}
            for key in row:
                d[key] = float(row[key])
            rows.append(d)
        return rows

def convert_csv_to_images(csv_filename, output_folder):
    """ Converts the data from the CSV into a set of graphs. """
    data = get_data_from_csv(csv_filename)
    base_filename = os.path.splitext(os.path.join(
        output_folder, os.path.basename(csv_filename)))[0]

    save_execution_time_plot(base_filename + "_time.png", data)
    save_num_nonzero_links_plot(base_filename + "_num_nonzero_links.png",
data)
    save_transit_load_spread_plot(
        base_filename + "_transit_load_spread.png", data)
    save_highest_capacity_links_plot(
        base_filename + "_highest_capacity_links.png", data)

def print_usage():
    print("Usage: {0} <csv file> <output folder>")

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print_usage()
        exit(-1)

    convert_csv_to_images(sys.argv[1], sys.argv[2])

```

../src/lp_graph.py

4.1.4 output.sh

This BASH script is responsible for executing the other scripts as well as timing and running CPLEX (under the Linux operating system).

```
#!/bin/bash
2 for y in 3 4 5 6 7 8
do
4     python3 src/lp-gen.py 9 $y 9 lp_files
      start=$(date +%s%N)
6     cplex -c "read lp_files/problem_9-{$y}_9.lp" "optimize" "display
      solution variables -" > cplex_logs/$y.txt
      end=$(date +%s%N)
8     duration=$(expr $end - $start)
      duration=$(expr $duration / 1000000)
10    echo -e "\nelapsed_time: $duration ms" >> cplex_logs/$y.txt
done
12
python3 src/lp_csv.py cplex_logs lp_files/cplex_data.csv
14 python3 src/lp_graph.py lp_files/cplex_data.csv graphs
```

../output.sh

4.1.5 output.ps1

This PowerShell script is responsible for executing the other scripts as well as timing and running CPLEX (under the Windows operating system).

```
For ($i=3; $i -le 8; $i++) {
2     python src/lp-gen.py 9 $i 9 lp_files
      $perf = Measure-Command -Expression {$data = cplex -c ("read lp_files/
      problem_9_" + $i + "_9.lp") "optimize" "display solution variables -"}
4     $ms = $perf.TotalMilliseconds
      [System.IO.File]::WriteAllLines("cplex_logs/$i.txt", $data + "
      elapsed_time: $ms ms")
6 }

8 python src/lp_csv.py cplex_logs lp_files/cplex_data.csv
python src/lp_graph.py lp_files/cplex_data.csv graphs
```

../output.ps1

4.2 Generated LP File

4.2.1 lp_files/problem.3.2.4.lp

```
\ COSC-364 Assignment 2 LP Generator, LP Output File
2 \ Written by Will Cowper (81163265), Jesse Sheehan (53366509)
  \ Parameters: X=3, Y=2, Z=4, Split=2, Demand=2 * i + j
4
6 MINIMIZE
  r
8
10 SUBJECT TO
```

```

\ DEMAND CONSTRAINTS
12 x_111 + x_121 = 3
13 x_112 + x_122 = 4
14 x_113 + x_123 = 5
15 x_114 + x_124 = 6
16 x_211 + x_221 = 5
17 x_212 + x_222 = 6
18 x_213 + x_223 = 7
19 x_214 + x_224 = 8
20 x_311 + x_321 = 7
21 x_312 + x_322 = 8
22 x_313 + x_323 = 9
23 x_314 + x_324 = 10
24
\ CAPACITY CONSTRAINTS FOR LINKS BETWEEN SOURCE AND TRANSIT NODES
26 x_111 + x_112 + x_113 + x_114 - c_11 = 0
27 x_121 + x_122 + x_123 + x_124 - c_12 = 0
28 x_211 + x_212 + x_213 + x_214 - c_21 = 0
29 x_221 + x_222 + x_223 + x_224 - c_22 = 0
30 x_311 + x_312 + x_313 + x_314 - c_31 = 0
31 x_321 + x_322 + x_323 + x_324 - c_32 = 0
32
\ CAPACITY CONSTRAINTS FOR LINKS BETWEEN TRANSIT AND DESTINATION NODES
34 x_111 + x_211 + x_311 - d_11 = 0
35 x_112 + x_212 + x_312 - d_12 = 0
36 x_113 + x_213 + x_313 - d_13 = 0
37 x_114 + x_214 + x_314 - d_14 = 0
38 x_121 + x_221 + x_321 - d_21 = 0
39 x_122 + x_222 + x_322 - d_22 = 0
40 x_123 + x_223 + x_323 - d_23 = 0
41 x_124 + x_224 + x_324 - d_24 = 0
42
\ OBJECTIVE FUNCTION LOAD CONSTRAINTS
44 c_11 + c_21 + c_31 - r <= 0
45 c_12 + c_22 + c_32 - r <= 0
46 c_13 + c_23 + c_33 - r <= 0
47 c_14 + c_24 + c_34 - r <= 0
48
\ TRANSIT NODE LOAD CONSTRAINTS
50 x_111 + x_112 + x_113 + x_114 + x_211 + x_212 + x_213 + x_214 + x_311 +
    x_312 + x_313 + x_314 - l_1 = 0
51 x_121 + x_122 + x_123 + x_124 + x_221 + x_222 + x_223 + x_224 + x_321 +
    x_322 + x_323 + x_324 - l_2 = 0
52
\ BINARY VARIABLE AND DECISION VARIABLE CONSTRAINTS
54 2 x_111 - 3 u_111 = 0
55 2 x_112 - 4 u_112 = 0
56 2 x_113 - 5 u_113 = 0
57 2 x_114 - 6 u_114 = 0
58 2 x_121 - 3 u_121 = 0
59 2 x_122 - 4 u_122 = 0
60 2 x_123 - 5 u_123 = 0
61 2 x_124 - 6 u_124 = 0
62 2 x_211 - 5 u_211 = 0
63 2 x_212 - 6 u_212 = 0
64 2 x_213 - 7 u_213 = 0
65 2 x_214 - 8 u_214 = 0
66 2 x_221 - 5 u_221 = 0

```

```

2 x_222 - 6 u_222 = 0
2 x_223 - 7 u_223 = 0
2 x_224 - 8 u_224 = 0
2 x_311 - 7 u_311 = 0
2 x_312 - 8 u_312 = 0
2 x_313 - 9 u_313 = 0
2 x_314 - 10 u_314 = 0
2 x_321 - 7 u_321 = 0
2 x_322 - 8 u_322 = 0
2 x_323 - 9 u_323 = 0
2 x_324 - 10 u_324 = 0

```

```

\ BINARY VARIABLE CONSTRAINTS (ONLY 2 ACTIVE TRANSIT NODES)

```

```

u_111 + u_121 = 2
u_112 + u_122 = 2
u_113 + u_123 = 2
u_114 + u_124 = 2
u_211 + u_221 = 2
u_212 + u_222 = 2
u_213 + u_223 = 2
u_214 + u_224 = 2
u_311 + u_321 = 2
u_312 + u_322 = 2
u_313 + u_323 = 2
u_314 + u_324 = 2

```

BOUNDS

```

\ NON-NEGATIVITY CONSTRAINTS

```

```

r >= 0
x_111 >= 0
x_112 >= 0
x_113 >= 0
x_114 >= 0
x_121 >= 0
x_122 >= 0
x_123 >= 0
x_124 >= 0
x_211 >= 0
x_212 >= 0
x_213 >= 0
x_214 >= 0
x_221 >= 0
x_222 >= 0
x_223 >= 0
x_224 >= 0
x_311 >= 0
x_312 >= 0
x_313 >= 0
x_314 >= 0
x_321 >= 0
x_322 >= 0
x_323 >= 0
x_324 >= 0
c_11 >= 0
c_12 >= 0
c_21 >= 0
c_22 >= 0

```

```
126 c_31 >= 0
128 c_32 >= 0
130 d_11 >= 0
132 d_12 >= 0
134 d_13 >= 0
136 d_14 >= 0
138 d_21 >= 0
140 d_22 >= 0
142 d_23 >= 0
144 d_24 >= 0
146
148 BIN
150
152 \ BINARY VARIABLES
154 u_111
156 u_112
158 u_113
160 u_114
162 u_121
164 u_122
166 u_123
168 u_124
170 u_211
172 u_212
174 u_213
176 u_214
178 u_221
180 u_222
182 u_223
184 u_224
186 u_311
188 u_312
190 u_313
192 u_314
194 u_321
196 u_322
198 u_323
200 u_324
202
204 END
```

../lp_files/problem_3_2_4.lp