FLOW PLANNING

ASSIGNMENT 2

COSC364-19S1 INTERNET TECHNOLOGY AND ENGINEERING

Will Cowper

ID: 81163265

Contribution: 50%

Jesse Sheehan

ID: 53366509

Contribution: 50%

May 29, 2019

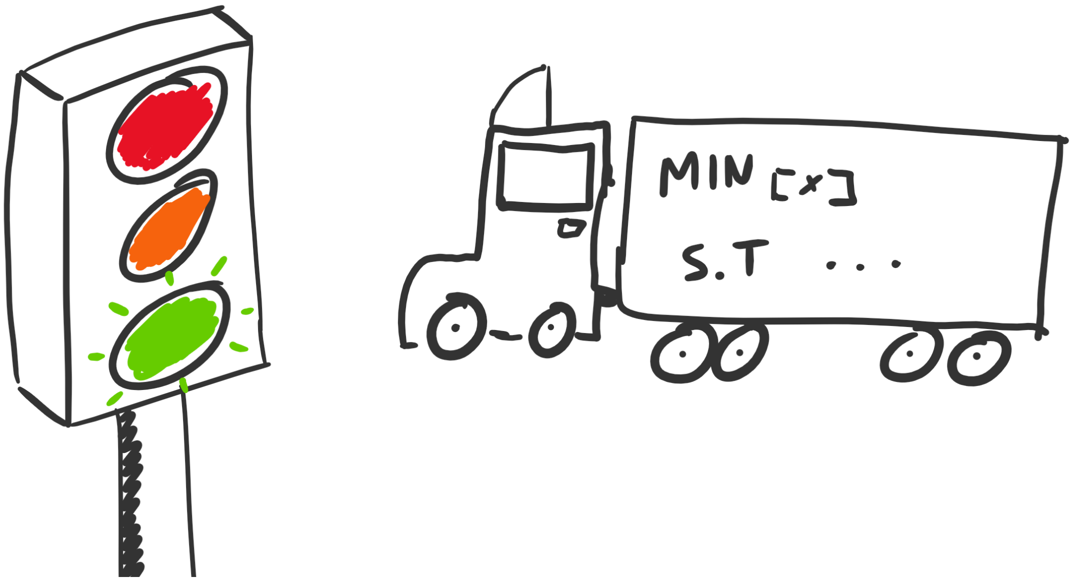

Figure 1: An artist's impression of a traffic problem outside of the Jack Erskine building (J. P. Sheehan, May 2019).

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	53366509
Signature:	[illegible]	[illegible]
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.

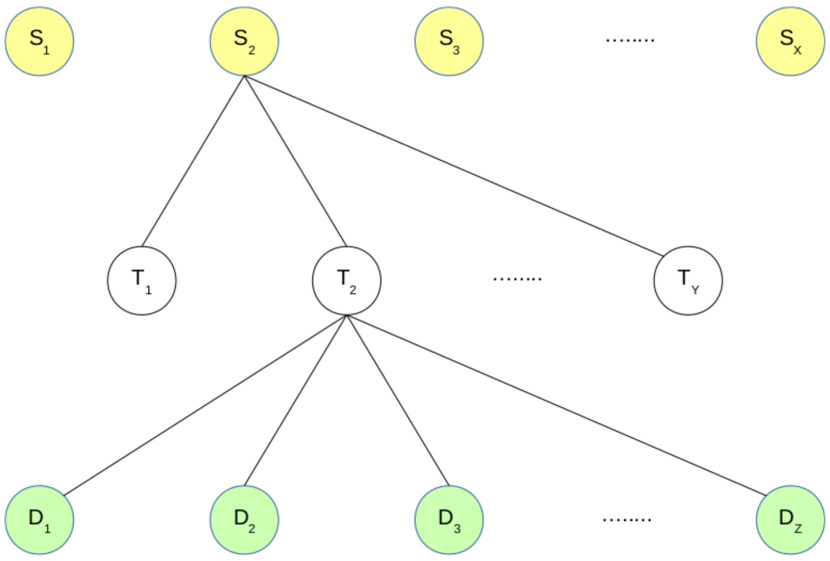

Figure 2: An example network (A. Willig, April 2019).

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.

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 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.

2.1 Objective Function

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

2.2 Constraints

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

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

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

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

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

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

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

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

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

2.3 Non-Negativity Constraints

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

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

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

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

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.

... your table ...

Table 1: insert caption here, yo!

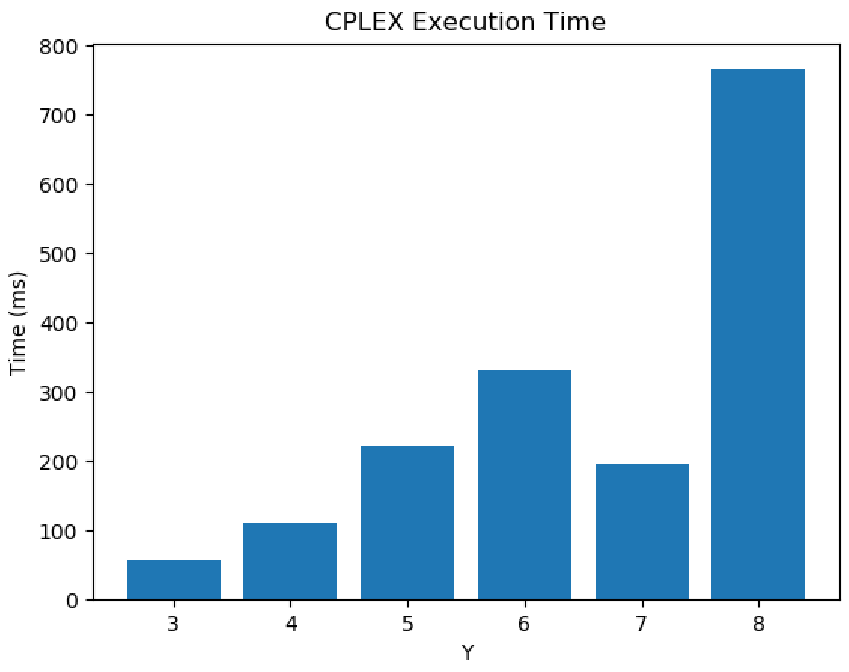

Figure 3: The time taken to execute the LP file in CPLEX for each network.

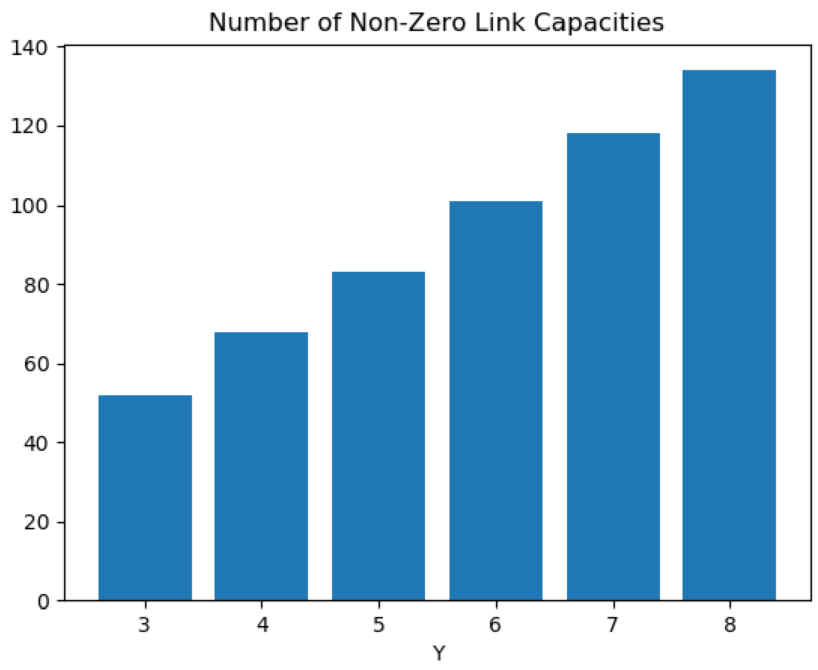

Figure 4: The number of non-zero link capacities in each network.

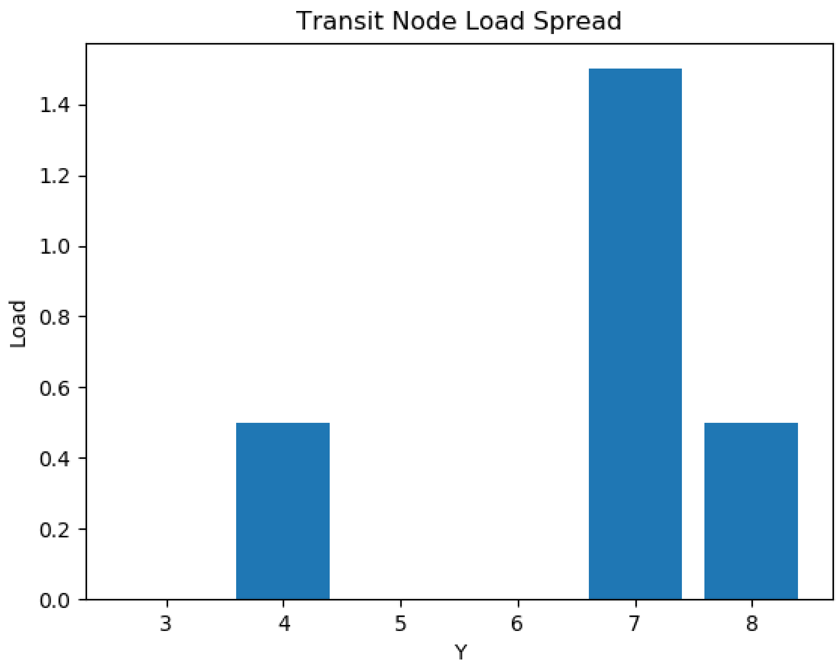

Figure 5: The amount of spread in the load for all transit nodes in each network.

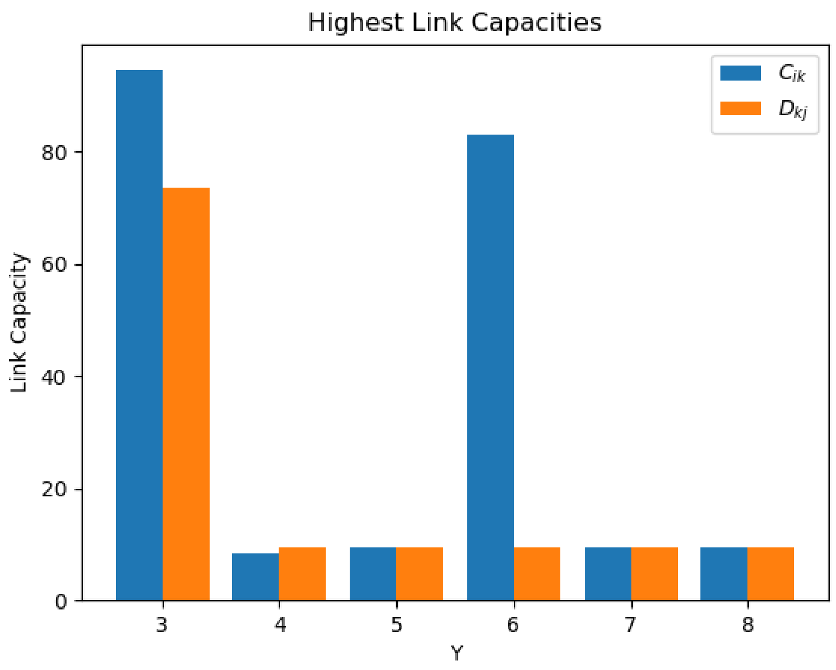

Figure 6: The highest capacity links for each network. Both the C_{ik} and D_{kj} links are listed.

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.

```
import inspect
import functools
import sys
import os.path

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

# Change these variables to alter the behaviour of the LP file generator
PATH_SPLIT = 2


def DEMAND_FLOW(i, j): return 2 * i + j


TEMPLATE = """\
\\ {}, LP Output File
\\ Written by {}
\\ 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
"""

# DEFINE SOME UTILITY FUNCTIONS


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 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('return')+7:]


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

# DEFINE SOME FUNCTIONS SPECIFIC TO THE PROBLEM
```

```

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(title, authors, 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(
        title,
        authors,
        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)


# DEFINE SOME HELPERS FOR GETTING THE THING RUNNING

def print_version():
    print('{0} by {1}'.format(__TITLE__, get_author_string()))


def print_usage():
    print('Usage: {0} <x> <y> <z> [output directory]'.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 x >= 10:
        print("Error: x must be less than ten")
        exit(-1)

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

    if y >= 10:
        print("Error: y must be less than ten")
        exit(-1)

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

    if z >= 10:
        print("Error: z must be less than ten")
        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 get_author_string():
    return ', '.join(
        ["{0} ({1})".format(name, sid) for (name, sid) in __AUTHORS__])
```

```

def main():
    print_version()
    if len(sys.argv) != 4 and len(sys.argv) != 5:
        print_usage()
        exit(-1)
    else:
        output_dir = '.'
        if len(sys.argv) == 5:
            output_dir = sys.argv[4]

        x, y, z = get_problem_parameters()
        data = generate_lp_file(__TITLE__, get_author_string(), x, y, z)
        filename = os.path.join(output_dir, get_lp_filename(x, y, z))
        save_lp_file(filename, data)
        print("Success: saved as '{0}'".format(filename))


if __name__ == "__main__":
    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.

```
import csv
import sys
import os.path


def csvWrite(data):
    with open(sys.argv[2], 'a', newline='') as csvFile:
        writer = csv.writer(csvFile)
        writer.writerow(data)


def floatmap(enumerable):
    return list(map(lambda x: float(x), enumerable))


def openFile(Y):
    with open(os.path.join(sys.argv[1], '{0}.txt'.format(Y)), 'r') as
   in_file:
        stripped = [line.strip() for line in in_file.readlines()]
        lines = [line for line in stripped if line]
        data = []
        # Y
        data.append(Y)
        # elapsed time
        data.append(max(parseFile("elapsed_", lines)))
        # no of non-zero c and d links
        data.append(len(parseFile("c_", lines)) + len(parseFile("d_", lines
   )))
        # transit load spread (largest_transit_node_load -
   smallest_transit_node_load)
```

```
        data.append(max(floatmap(parseFile("l_", lines))) -
                    min(floatmap(parseFile("l_", lines))))
        # highest cap c network
        data.append(max(parseFile("c_", lines)))
        # highest cap d network
        data.append(max(parseFile("d_", lines)))
        csvWrite(data)


'''Returns a list of all values that start with the given string'''


def parseFile(string, lines):
    values = []
    for line in lines:
        if line.startswith(string):
            values.append(line.split()[1])

    return values


if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: {0} <input directory> <csv file>".format(sys.argv[0])
    )
        exit(-1)

    # delete the CSV, otherwise we will append to it
    os.unlink(sys.argv[2])

    openFile(3)
    openFile(4)
    openFile(5)
    openFile(6)
    openFile(7)
    openFile(8)

    print("Saved CSV data to '{}'".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.

```
import csv
import sys
import os.path
import numpy as np

try:
    import matplotlib.pyplot as plt
except:
    print("Error: could not load 'matplotlib'. Install with 'pip install
   matplotlib' and then try again.")
    exit(-1)


```

```
def get_data(data, key):
    return list(map(lambda d: d[key], data))


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


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


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


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


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


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


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


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


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


def save_highest_capacity_links_plot(filename, data):
    """ 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
for y in 3 4 5 6 7 8
do
    python3 src/lp_gen.py 9 $y 9 lp_files
    start=$(date +%s%N)
    cplex -c "read lp_files/problem_9_${y}_9.lp" "optimize" "display
     solution variables -" > cplex_logs/$y.txt
    end=$(date +%s%N)
    duration=$(expr $end - $start)
    duration=$(expr $duration / 1000000)
    echo -e "\nelapsed_time: $duration ms" >> cplex_logs/$y.txt
done

python3 src/lp_csv.py cplex_logs cplex_data.csv
python3 src/lp_graph.py 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++) {
    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 -"}
    $ms = $perf.TotalMilliseconds
    [System.IO.File]::WriteAllLines("cplex_logs/$i.txt", $data + "`
     nelapsed_time: $ms ms")
}

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
\ Written by Will Cowper (81163265), Jesse Sheehan (53366509)
\ Parameters: X=3, Y=2, Z=4, Split=2, Demand=2 * i + j

```

```
MINIMIZE
   r

SUBJECT TO

   \ DEMAND CONSTRAINTS
   x_111 + x_121 = 3
   x_112 + x_122 = 4
   x_113 + x_123 = 5
   x_114 + x_124 = 6
   x_211 + x_221 = 5
   x_212 + x_222 = 6
   x_213 + x_223 = 7
   x_214 + x_224 = 8
   x_311 + x_321 = 7
   x_312 + x_322 = 8
   x_313 + x_323 = 9
   x_314 + x_324 = 10

   \ CAPACITY CONSTRAINTS FOR LINKS BETWEEN SOURCE AND TRANSIT NODES
   x_111 + x_112 + x_113 + x_114 - c_11 = 0
   x_121 + x_122 + x_123 + x_124 - c_12 = 0
   x_211 + x_212 + x_213 + x_214 - c_21 = 0
   x_221 + x_222 + x_223 + x_224 - c_22 = 0
   x_311 + x_312 + x_313 + x_314 - c_31 = 0
   x_321 + x_322 + x_323 + x_324 - c_32 = 0

   \ CAPACITY CONSTRAINTS FOR LINKS BETWEEN TRANSIT AND DESTINATION NODES
   x_111 + x_211 + x_311 - d_11 = 0
   x_112 + x_212 + x_312 - d_12 = 0
   x_113 + x_213 + x_313 - d_13 = 0
   x_114 + x_214 + x_314 - d_14 = 0
   x_121 + x_221 + x_321 - d_21 = 0
   x_122 + x_222 + x_322 - d_22 = 0
   x_123 + x_223 + x_323 - d_23 = 0
   x_124 + x_224 + x_324 - d_24 = 0

   \ OBJECTIVE FUNCTION LOAD CONSTRAINTS
   c_11 + c_21 + c_31 - r <= 0
   c_12 + c_22 + c_32 - r <= 0
   c_13 + c_23 + c_33 - r <= 0
   c_14 + c_24 + c_34 - r <= 0

   \ TRANSIT NODE LOAD CONSTRAINTS
   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
   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

   \ BINARY VARIABLE AND DECISION VARIABLE CONSTRAINTS
   2 x_111 - 3 u_111 = 0
   2 x_112 - 4 u_112 = 0
   2 x_113 - 5 u_113 = 0
   2 x_114 - 6 u_114 = 0
   2 x_121 - 3 u_121 = 0
   2 x_122 - 4 u_122 = 0
   2 x_123 - 5 u_123 = 0
   2 x_124 - 6 u_124 = 0
```

```
    2 x_211 - 5 u_211 = 0
    2 x_212 - 6 u_212 = 0
    2 x_213 - 7 u_213 = 0
    2 x_214 - 8 u_214 = 0
    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
    c_31 >= 0
    c_32 >= 0
    d_11 >= 0
    d_12 >= 0
    d_13 >= 0
    d_14 >= 0
    d_21 >= 0
    d_22 >= 0
    d_23 >= 0
    d_24 >= 0

BIN

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

END
```

../lp_files/problem_3_2_4.lp