Catalog of Software and Data Files

That Accompany the Book:

Integer Linear Programming in Computational and Systems Biology

by Dan Gusfield

Published by Cambridge University Press, 2019

In order to use these programs, you will need to have Python 2.7 and Perl installed on your computer. You probably already have those installed, but if not, go to

https://legacy.python.org/download/
for Python 2.7
and to
https://www.perl.org/get.html
for Perl.

These programs were written for execution on a command line in a terminal window. Python is often executed in an "integrated development environment" called "IDLE", but I recommend staying away from it – call the programs on a command line in a terminal window, and if you want to modify them, use a vanilla text editor (not a word processor, such as MS Word).

1 Cleanres.pl

Before discussing software producing concrete ILP formulations, the following Perl program is useful for all of the programs and problems:

cleanres.pl

Program *cleanres* processes a Gurobi result file, which always has the extension ".sol", and removes any variables that have value 0. In many problems, the number of variables that have been given non-zero values is a small fraction of all the variables, so the result of program *cleanres* helps the user see the important variables with much less work. Call the program on a command line in a terminal window with:

```
Perl cleanres.pl file.sol
```

where *file.sol* is the name of Gurobi-produced result file. The cleaned results will be in a new file whose name is the same as "file.sol", but with the prefix "c". For example, if the Gurobi results are in file *mysolution.sol* then the cleaned results will be in file *cmysolution.sol*.[1]

2 Software for Chapter 2: Biological Networks, Graphs, and High-Density Subgraphs

1. *CLgraphgen.py*

Python program to build a random undirected graph and the concrete ILP formulation to find a maximum clique in the graph. Also, output the adjacency matrix of the graph.

Call on a command line in a terminal window as:

```
python CLgraphgen.py
```

2. *graphgen.py*

Python program to build and output the adjacency matrix of a random undirected graph.

Call on a command line in a terminal window as:

```
python graphgen.py
```

3. *CLgraphfile.py*

This Python program reads a file containing an adjacency matrix for a graph, and creates the concrete ILP for the maximum clique problem, for that graph. The program assumes that the adjacency matrix of the graph has no spaces between entries.

Call on a command line in a terminal window as:

```
python CLgraphfile.py adjacency-matrix-file ILP-file.lp
```

Example Data: *yeastAdjMatrix* - this is the adjacency matrix for the 209-node yeast PPI network It can be used with several of the programs listed here.

Another data file: *example-graph* is a 60 node graph.

4. *inverseclique.py*

This Python program reads a file describing a graph, and the size, denoted k, of the largest clique in the graph (previously determined); and the size, d, of the required increment. The program creates the concrete ILP for the

[1]You can think of the added "c" as an abbreviation for "clean", or as a shorthand for the word "see". Yes, that is a bad pun.

inverse Near-clique problem, for that graph. The ILP, when executed, finds the minimum number of new edges that have to be added so that the resulting graph has a clique of size $k + d$.

Call on a command line in a terminal window as:

```
python inverseclique.py graph-file ILP-file.lp k d
```

Example Data: *example-graph*

which has a clique of size $k = 8$.

5. *CL2graphfile.py*

This Python program reads a file containing the adjacency matrix of a graph, and creates the concrete ILP for the maximum clique problem, finding two cliques. One is a largest clique, K, and the other is a largest clique that shares no nodes with K.

Call on a command line in a terminal window as:

```
python CL2graphfile.py adjacency-matrix-file ILP-file.lp
```

Example data: *example-graph*

3 Software for Chapter 3: Maximum Character Compatibility in Phylogenetics

1. *charcliquefast.pl*

This Perl program creates the concrete ILP formulation to solve the Maximum Character-Clique problem, which also solves the Minimum Character Removal (MCR) problem. Call the program as:

```
perl charcliquefast.pl data-file
```

Example data file: *char-data*

The ILP formulation will be in a file with the same name as the data-file, but with the ".lp" extension. The solution to the ILP formulation sets variable Xi to 1 to mean that column i is part of a largest clique of pairwise compatible columns. So, the columns associated with the variables with value 0 in the optimal solution, form a solution to the MCR problem.

4 Software for Chapter 4: Near Cliques, Dense Subgraphs, and Motifs in Biological Networks

1. *Nclique.py*

This Python program reads a file containing an adjacency matrix of a graph, and creates the concrete ILP for the maximum Near-clique problem, for that graph. The definition of a near-clique in this program is a subset of nodes that is either a true clique, or would be a true clique with the addition of one more edge.

Call on a command line in a terminal window as:

```
python Nclique.py graph-file ILP-file.lp
```

Example data: *example-graph*

2. *HDNclique.py*

This Python program reads a file with the adjacency matrix of a graph, and creates the concrete ILP for the maximum High-density Near-clique problem, for that graph.

Call on a command line in a terminal window as:

```
python HDNclique.py graph-file ILP-file.lp fraction
```

where "fraction" is the maximum allowed fraction of original edges all the edges in the created clique that are added.

Example data: *example-graph*

3. *largest-dense.pl*

Perl program to create the ILP formulation to find a largest subgraph (number of nodes) that has density at least that of the input value.

Call on a command line in a terminal file as:

```
perl largest-dense.pl graph-file-name density
```

——————————-

5 Software for Chapter 5: The Maximum Parsimony Problem

1. *Aparsimony.pl*

Perl program to create a concrete ILP formulation for the Maximum Parsimony Problem (MPP). Aparsimony.pl builds a full hypercube of dimension equal to the length of the sequences. Hence it is limited to sequences of lengths under twenty SNPs. This is not as restrictive as it seems, because many real biological data sets with several hundred SNP sites have a huge number of duplicated columns and after those are removed, the number of remaining columns may be under twenty.

Call on a command line in a terminal window as:

```
perl Aparsimony.pl sequence-file-name length-of-sequences
```

Example Data: *bookpars*

This program has a feature (or some would call it a bug). It misbehaves if there are any duplicate rows. Also it misbehaves if there is an all-zero row.

2. *BBparsimony.pl*

Perl program to create a concrete ILP formulation for the Maximum Parsimony Problem (MPP), but instead of working on a full hypercube, it first builds the Buneman graph from the data. Then it finds the Maximum Parsimony Solution on the Buneman graph. That solution is guaranteed to be a solution for the full hypercube, and for real data, the empirical observation is that the size of the Buneman graph is extremely small compared to the size of the corresponding full hypercube. Hence, it is able to handle cleaned data that is much larger than the program *Aparsimony* can handle.

Call *BBparsimony.pl* with the same arguments as for *Aparsimony.pl.*

Example Data: *chumanY*. This dataset has 40 sites, and so it would be impossible for *Aparsimony.pl.*

Unlike *Aparsimony.pl*, this program assumes that there is an all-zero row. Also, it misbehaves if there are any duplicate rows or columns. Program *BBparsimony* was written after the book was finished, and is not mentioned in the book. However, the paper by S. Sridhar et al., who first used a Buneman graph for MPP, was discussed in Chapter 5.

3. *cleandata.pl*

Perl program that takes in data intended for BBparsimony.pl. It removes duplicate rows and columns, and then transforms the data so that the first row in the input becomes an all-zero row. This involves transforming all 1s to 0s, and all 0s to 1s in each column that contains a 1 in the first row. The optimal solution to the MPP has the same value before and after the transformation, and an optimal solution to the transformed problem can be back-transformed to explicitly create an optimal solution in terms of the original input.

6 Software for Chapter 6: RNA Folding

1. *first-rna.py*

Python program to generate the inequalities for the ILP formulation for the simple RNA folding problem, discussed in Chapter 6, Section 6.1.1.

Call on a command line in a terminal window as:

```
python first-rna.py
```

2. *randomrna.pl*

Perl program to create a random RNA string. This asks the user for the desired length length of a random RNA string and then generates a random RNA string of that length, where the frequencies of each nucleotide agree with known averages over many RNA sequences. Such sequences are not necessarily biologically realistic, but provide a sensible starting point for empirical exploration of computer programs. The result is put into a file called "randomstring".

3. *fourth-rnaf.py*

Python program to generate the inequalities for the ILP formulation for the RNA folding problem. This program only generates a P(i,j) variable if nucleotides i and j are complementary, and farther apart than a minimum distance between paired nucleotides in the RNA string.

Also, this ILP formulation has inequalities to determine how many stacked quartets (not stacked pairs) there are in the computed fold, and the number of stacked quartets is included in the objective function. The variable Q(p,q) gets set to 1 if and only if (p,q) first pair in a stacked quartet. So, for the innermost stacked pair (x,y) in a stack, the variable Q(x,y) gets set to 0.

Call this program on a command line as:

```
python fourth-rnaf.py name-of-RNA-file LP-output-file.lp min-distance-between-paired-nucleotides
```

7 Software for Chapter 7: Protein Problems

1. *HPb.pl*

Perl program to create the concrete ILP for the HP Prototein problem with an input string of length n, on an n-by-n grid.

Call *HPb* on a command line in a terminal window as:

```
perl HPb.pl HPstring
```

where *HPstring* is a file containing a binary string.

2. *HPb1.pl*

Perl program for the HP Prototein problem with an input string of length n, on an n/4-by-n/4 grid. Run *HPb1.pl* the same way as *HPb.pl*

3. *HPb1mid.pl*

Perl program for the HP Prototein problem with an input string of length n, on an n/4-by-n/4 grid, where the first and second characters in the string are placed on the midpoint and a neighbor of the midpoint, on the grid. This produces ILP formulations that solve faster than *HPb1.pl* for compact proteins. But for proteins that don't fold very compactly, the ILP generated might be infeasible. Generally, if the generated ILP is infeasible, it is unlikely that the input sequence comes from a real or a globular protein.

Run *HPb1mid.pl* the same way as *HPb.pl*

4. *HPb1-3D.py*

Python program to create concrete ILP formulations for the 3-D prototein problem. This runs with Python 2.7. This program was written by Jonathan Kim.

Call *HPb1-3D.py*, on a command line in a terminal window as:

```
python HPb1-3D.py HPstring file.lp
```

where *HPstring* is a file containing a binary string, and *file.lp* is the name of the .lp file holding the ILP formulation.

8 Software for Chapter 8: The Tanglegram Problem

1. tangILP.pl

Perl program to create concrete ILP formulations for the tanglegram problem. The input file has three lines. The first is the number of leaves in the input trees. The second and third lines describe the input trees in *Newick* format.

Call the program as:

perl tangILP.pl file-name

A test case for the program is in file: *tangtest*

9 Software and Data for Chapter 9: The Traveling Salesman Problem in Genomics

9.1 TSP problems

1. *assignment.pl*

This Perl program creates a concrete ILP formulation for a concrete instance of the *Assignment* problem.

Execute on a command line in a terminal window:

```
perl assignment.pl edge-weight-file max-weight-for-a-real-edge
```

where the "edge-weight-file" holds an n-by-n matrix of the weights for the edges, and "max-weight-for-a-real-edge" is a number such that any number on an edge larger than it in the matrix is taken as an edge that does not exist.

Data: *p43dist*

which is the edge-weight matrix for the TSPLIB problem p43.

2. *M1graphTSPfile.pl*

This creates a concrete GG formulation for an instance of the TS *tour* problem.

Call on a command line in a terminal window with:

```
perl M1graphTSPfile.pl edge-weight-file max-weight-for-a-real-edge
```

The program handles asymmetric data - as long as for every pair of nodes (i,j) both (i,j) and (j,i) are allowed, i.e., with a cost below the threshold, or both are disallowed.

3. *MgraphTSPfile.pl*

This creates a concrete GG formulation for an instance of the TS *path* problem.

Call on a command line in a terminal window with:

```
perl MgraphTSPfile.pl edge-weight-file max-weight-for-a-real-edge
```

4. *WgraphTSPfile.pl*

This creates a concrete MTZ formulation for an instance of the TS *path* problem.

Call on a command line in a terminal window with:

perl WgraphTSPfile.pl edge-weight-file max-weight-for-a-real-edge

4. *FgraphTSPfile4.pl*

This creates a concrete FGG formulation for an instance of the TS *tour* problem.

Call on a command line in a terminal window:

perl FgraphTSPfile4.pl edge-weight-file number-of-nodes max-weight-for-a-real-edge
where "edge-weight-file" and "max-weight-for-a-real-edge" are the same, but note that this program asks for the number of nodes in the graph as the second argument.

5. *tspjulia.py*

This Python program modifies the Gurobi program *tsp.py*, to read in a user-supplied edge-distance matrix.

This implements and runs the DJF formulation for solving an instance of the TSP problem, via a separation strategy. This program is just a small modification of the Gurobi-written program *tsp.py*. The Gurobi program generates a random problem instance using Euclidean distances between points on a plane, while *tspjulia.py* allows input from a file of any TSP problem instance. This allows more robust testing of the DJF solution via a separation strategy. This modification was made by Julia Matsieva.

The ILP generated is only for undirected graphs, i.e., the cost for using an edge is the same in both directions. If your argument is a distance matrix, run the program with -d.

Examples of how to run the program:

a) python tspjulia.py saharaData

the data contains the longitude and latitude coordinates of each city in the saharaData. The program computes the pairwise Euclidean distances for input to the TSP problem.

b) python tspjulia.py -d outD

outD is a symmetric distance matrix for n cities.

c) python tspjulia.py drilling280

drilling280 holds the (x,y) coordinates for 280 grid points on a circuit board.

The command:

python tspjulia.py -h

will bring up a help page.

9.2 Software for the Marker-Ordering Problem

1. *marker-pipeline.pl*

This pipeline, written in Perl, generates marker-probe data for the Karp et al. physical mapping problem discussed in Chapter 9. It generates several blocks of consecutive ones per row; and then generates the GG compact ILP formulation to solve it; calls Gurobi to optimize it; and then extracts and displays the results. Call as

perl marker-pipeline.pl new-data-file-name number-of-rows number-of-cols max-block-length

2. *markerTSP.pl*

Creates a dataset mimicking what one could get from clone-marker data with multiple fragments. It outputs the dataset, and both the distance matrix derived from the dataset and the distance matrix derived from the matrix with the columns permuted. This program is called insider of *marker-pipeline.pl*, but can also be used as a stand-alone program.

Call the program on a command line in a terminal window as:

perl markerTSP.pl number-of-rows number-of-cols max-block-length

3. *pmarkersolcheck.pl*

This Perl program processes the cX.sol file to extract the TSP solution and then relabels the permutation to conform to the original graph data. This variant is for use when the columns of the original graph data were permuted, to break up the intervals of consecutive ones. This program is called inside of marker-pipeline.pl

10 Software for Chapter 10: Molecular Sequence Analysis

1. *LCS.py*

This program was written by Jessica Au and Jessie Vuong as part of their class project in the class Integer Linear Programming in Computational Biology, Spring quarter 2017.

This program creates the ILP to solve the LCS problem for two strings

Call this program on command line as:

```
python LCS.py DNA.txt LP_outputfile.lp
```

where *DNA.txt* is the name of a plain text file containing a DNA sequence over the alphabet A,T,C,G, with at most 100 nucleotides. *LP_outputfile.lp* names an output file for concrete ILP formulation.

2. *LCS.pl* A Perl program that solves the LCS problem for two strings by Dynamic Programming (DP). It can be used to compare the results from the ILP approach to the LCS problem, and the relative computations times. Spoiler Alert: both approaches should give the same optimal solution value, but the DP program will generally be much faster.

11 Software for Chapter 13: Communities, Cuts and High-Density Subgraphs

1. *modular2.py*

A Python program for computing the modularity of a graph and the optimal partition used to compute modularity.

Data: *zacharymatrix.txt*

2. *kcutf.py*

A Python program to compute a minimum K-cut, given a graph and an initial list of function for some nodes.

Call the program in a terminal window on a command line as:

```
python kcutf.py graph-file function-file ilp-file.lp number-of-nodes
```

"graph-file" holds the adjacency matrix for the graph

"function-file" holds the functions assigned to the nodes whose functions are known

"ilp-file.lp" is the name of the file that will hold the concrete ILP formulation

"number-of-nodes" is the number of nodes in the input graph

The program *kcutf.py* can also be used to compute an (s, t) minimum cut by just specifying s and t as nodes with different functions, 1 and 2, with the functions of all other nodes unspecified.

Example data files: *zacharymatrix.txt, zachfuncs*

3. *4part-random.py*

A Python program that reads a file describing a graph. Then it randomly chooses nodes with probability 0.5 to assign functions to. The function assigned to a chosen nodes is chosen uniformly at random from four functions. Then it creates the concrete ILP for the resulting *max 4-cut* problem.

12 Software for Chapter 14: Corrupted Phylogenetic Data and Models

12.1 Missing data

1. *bbmisstest.pl*

This program generates the concrete ILP formulation to solve the IMCR problem. That is, given a binary matrix M with some missing values (each indicated by '2'), set the missing values (0 or 1), creating matrix M', in order to minimize the number of resulting incompatible pairs of sites in M'. Another way to express this is to set the missing values, creating the M' for which the solution value of problem MCR is minimized.

Call on a command line in a terminal window as:

```
perl bbmisstest.pl data-file ILP-file.lp
```

Example Data sets: *missing-data, missing-test*

12.2 Section 14.4: Persistent Phylogeny

1. *sperfreduce.pl*

This Perl program removes columns that are compatible with all other columns. We can prove that given a persistent tree for the remaining sites, the positions on the tree for the removed sites are easily determined. Removing those sites usually reduces the size of the problem significantly, and the resulting ILP is solved faster. Call as:

```
perl sperfreduce.pl data-matrix
```

The resulting reduced matrix is written to a file with the same name as the input data-matrix, but with the letter 'r' prepended to it. For example, if the input matrix is called 'dm', then the reduced matrix will be in file 'rdm'.

Example data: *per-data*

2. *sprepare-persistent.pl*

This processes a binary matrix, read from a file, to create the file containing the extended matrix with 2s (representing ?s). That file is used next to create the ILP to test for a persistent phylogeny.

Call on a command line in a terminal window as:

```
perl sprepare-persistent.pl data-file
```

The data-file can hold any binary matrix. However, if the binary matrix has first been processed by *sperfreduce.pl*, then it will be smaller, and the ultimate concrete ILP formulation will be smaller and will solve faster. So, it is recommended that the data-file be one that is the output of *sperfreduce.pl.*

Example data: *rper-data*, which is the output of program *sperfreduce.pl* with input *per-data.*

3. *persistent.pl*

This creates the concrete ILP to solve the Persistent Phylogeny Problem for the binary matrix that is first input to *sperfreduce.pl.* That output is input to *sprepare-persistent.pl*, which outputs the matrix used by *persistent.pl.*

If there is a persistent phylogeny, then there will be an optimal for the ILP produced by persistent.pl, with value 0. If there is no persistent phylogeny, then the ILP will be infeasible.

Call on a command line in a terminal file as:

```
perl persistent.pl matrix-file
```

where *matrix-file* holds a matrix with values of 0, 1 and 2, created by the program *sprepare-persistent.pl.*

Example Data: *prper-data*

The output concrete ILP formulation will be in file "persistent.lp".

4. *spersistent.pl*

This is a Perl script that takes in the name of a file with a binary matrix and calls the programs *sperfreduce.pl, sprepare-persistent.pl* and *persistent.pl*, finally producing, in file *persistent.lp*, the concrete ILP formulation to solve the persistent phylogeny problem for the input matrix in the data-file called with *sperfreduce.pl.*

Call as:

perl sperfreduce.pl data-file

13 Software for Chapter 18: More String and Sequence Problems

13.1 LCS for three strings

1. *LCS_Mult.py*

This python program generates the ILP formulation to solve the LCS problem for MULTIPLE strings. Currently, it only works for THREE strings.

LCS_Mult.py was written by Jessica Au and Jessie Vuong as part of their class project in the class Integer Linear Programming in Computational Biology, Spring quarter 2017.

Call this program on command line as:

```
python LCS_Mult.py DNA.txt LP_outputfile.lp minDist num-strings
```

Input is a plain text file containing three DNA sequences, each with a max of 100 nucleotides.

minDist is the minimum distance between allowed matches. We can consider this as spaces/indels. Set this to 0 for the basic LCS problem.

num-strings is the number of strings input. Currently, use 3.

Output: *LP_outputfile.lp*

Example Data files: *test3seqs.txt, multiple_string.txt*

13.2 Software for the Sorting-by-Reversals Problem

1. *breversals.pl*

This program creates the compact ILP for sorting-by-reversals problem. The first permutation is assumed to be the identity permutation, i.e., the integers from 1 to n in natural order. So the input only specifies the second permutation, i.e., $P2$.

Every X variable has four parameters: level, origin of flow from level 1 (i.e., the integer), node at level, node at level+1. So this is a "flow out" variable.

The R variable has three parameters: left, right, level. So $R(i, j, k)$ means that there is a rotation between positions i and $j > i$ at level k.

Call the program on a command line in a terminal window as:

perl breversals.pl number file-name

where "number" is n, the number of integers in the permutation, and "file-name" is the name of the file where permutation P2 is written on one line. The ILP formulation is output to a file whose name begins with the letter 'R', followed by the input file-name, followed by '.lp'. So, the ILP formulation created will be in a file named *Rfile-name.lp*.

Example Data: *tobacco-lobelia.txt*

14 Software for Chapter 20: The MP-PPH Problem

1. *bbminpph.pl*

The Perl program *bbminpph.pl* takes in a matrix, $\mathcal{G}$, of n genotypes, and if $\mathcal{G}$ can be phased so that the $2n$ haplotypes have no incompatibilities (i.e., they can be derived on a perfect phylogeny), then the program will solve the MP-PPH problem for $\mathcal{G}$. If no perfect phylogeny solution is possible, then the ILP will be infeasible.

Call the program on a command line in a terminal window as:

perl bbminpph.pl genotype-matrix output-file.lp

where "genotype-matrix" is the file name where the input matrix is held, and "output-file.lp" is the user-supplied name of the file that will hold the concrete ILP formulation based on the input genotype matrix.

After solving a concrete ILP formulation and saving the solution to a file (assuming there is a solution), you can construct the $2n$ solution haplotypes from the ILP solution. The values for the haplotypes are given by the values of the Y variables. For example, the entry "Y7,4 1" means that the 7'th haplotype has a 1 at position 4. It would be easy for an experienced programmer to write a script to create the haplotypes from the solution in the .sol file.

Example Data: *mpph.txt*