Optimization

Fatema Alkhanaizi

February 22, 2019

How to run

There is a python script along with a jupyter notebook included with this report. Python, version 3.6, was used.

Packages and Environemnt

The following packages are required to run both the script and the notebook: **numpy, scipy, pymprog, networkx, matplotlib, sympy**. The first three packages were used for generating and solving the linear programs. **networkx and matplotlib** were used to draw the graphs. **sympy** was used to convert result to rational format. All these packages can be installed to a python environemnt in the university machines using pip install. For the notebook, it will require an anaconda environemnt along with all the packages mentioned which can also be installed with pip.

```
pip install numpy, scipy, pymprog, networkx, matplotlib, sympy
```

Run Instructions

Jupyter notebook

If you have an anaconda distribution, activate the environment that includes all the packages mentioned previously and use python3. Run the command 'jupyter-notebook'; this will print a url that navigates to the notebook UI. Once you navigate to the notebook included for this assignment from the UI, run all the cells (Kernel → Restart and Run All) There are some examples included at the end of the notebook and comments that specify all the included functions. There are two main functions: **lp_clique(G)** and **lp_entropy(G)**. Both functions take a graph as a parameter that is an integer dictionary of sets e.g.

```
G = {1:{2,4,5}, 2:{1,3,4}, 3:{2,5}, 4: {1,2,5}, 5:{1,3,4}}
```

The graph is undirected so an edge uv means that v should be a neighbour of u and vice-versa. Running **lp_clique(G)** in one of the cells will return the optimal value for the fractional clique cover number $\pi^*(G)$ and the vector $\overline{x}$ in rational format. Similarly, running **lp_entropy(G)** will return the optimal value for shannon entropy $\eta(G)$ and the vector $\overline{x}$ in rational format. A mapping between $\overline{x}$ vector and the subsets is included in both outputs for clarity; the columns of the vector represent different subsets depending on the graph G itself. Both functions include an optional paramater for a verbose output; this includes and not limited to all the inequalities used to evaulate the optimal solutions.

Python Script

Make sure you are using an environemnt that includes all the packages and uses python3. Only the following command is required for this script:

```
python3 linear_programs.py
    "{1:{2,4,5}, 2:{1,3,4}, 3:{2,5}, 4: {1,2,5}, 5:{1,3,4}}" > output.txt
```

The script takes the graph in the same format mentioned for jupyter notebook version. This script will print out the output for both linear programs. For a more verbose output, this will require setting the verbose flag at the top of the script file:

```
SET_VERBOSE = True
```

Linear Programs (LP)

Different approaches and packages were tested to create and solve each LP. The LP was modified to be in Standard Equational Form, max $z = c^T x$ s.t. $A\overline{x} = b$.

Fractiona Clique Cover Number $\pi^*(G)$

For implementation using scipy solver, the constraints were modified to be negative, $\sum_{S:v\in S} -x_S \leq -1$, to set an upper boundary for the constraints. By default the scipy solver returns the minimum optimal solution, so no modifications are needed for getting the optimal value of the fractional clique cover number; a clique of a graph $G = (V, E)$ i.e. the subset K satisfy the condition that for any distict $u, v \in K$, $uv \in E$. As part of the constraints for this LP, the variables associated with all subsets that were not cliques were equal to zero. So when creating the matrix A for this LP, all non-clique subsets were ignored. Thus, the solution vector, $\overline{x}$, only included variables of cliques of graph G. An encoding for the vertices was used to represent the position of each vertex. E.g. for a graph with vertices = {1,2,3,4}:

```
1 = [-1, 0, 0, 0];    3 = [0, 0, -1, 0];
2 = [0, -1, 0, 0];    4 = [0, 0, 0, -1];
```

These encodings were used to create the encodings for the cliques which were the sum of the encodings of vertices making each clique. E.g. for clique (1,2):

```
(1,2) = [-1, 0, 0, 0] + [0, -1, 0, 0]
      = [-1, -1, 0, 0]
```

All these encodings represented the columns for matrix A. Each row in matrix A represented the sum constraint for each vertex in G. Matrix b was set to an array of negative 1s. Matrix c was set to an array of positive ones; all variables, $\overline{x}$, are part of the computation for this LP.

Shannon Entropy $\eta(G)$

For shannon entropy LP with scipy and pymprog implementations, all constraints were respresented as upper boundaries, $x \leq y$. Both implementations used similar LP modifications, however scipy package was taking a lot of time to solve the LP, so pymprog was mainly used to solve this LP; code included in the notebook. For modifying the LP, the equality constraints were used to exclude some of the boundary constraints. All unconnected vertices, with $N(v) = 0$ were removed from the graph before starting any computations as they don't contribute to any entropy. A dictionary for

equality constraint, $x_{N(v)} = x_{N(v)\cup\{v\}}$, was created; the smaller set was mapped to the bigger set. In case of a collision, smaller set was mapped to multiple bigger sets, the colluded set was mapped to the smaller set value e.g.

```
(1,2) = (1,2,3);      (1,2) = (1,2,4);
eq_dict = {(1,2):(1,2,3), (1,2,4):(1,2,3)}
```

The constraints for this LP were generated by two nested loops going through all subsets for graph G vertices. In each iteration, the follwoing conditions were checked:

1. If the outer loop subset, i, was of size 1, the constraint $x_{\{v\}} \leq 1$ was included. Otherwise, the inner loop subset, j, was intersected and merged with i resulting in two other subsets, ij and u respectively.

2. If i, j and u sets were part of the equality dictionary, they were mapped to their perspective equivalent sets; the positions in $\overline{x}$ were used to represent the mapping of these sets.

3. If the set ij was the same as either i or j, and i and j didn't have the same position in $\overline{x}$ then either i or j could be a subset of the other, setting the constraint $x_S \leq x_T, S \subseteq T$. Only subsets i and j with difference of 1 in length were considered. This was due to the cascading of constraints e.g. $x_1 \leq x_{1,2} \leq x_{1,2,3}$. There was no need to include that $x_1 \leq x_{1,2,3}$ and $x_1 \leq x_{1,2}$ as it was implied by cascading constraints, $x_1 \leq x_{1,2}$ and $x_{1,2} \leq x_{1,2,3}$.

4. Otherwise, if i and u had the same postion in $\overline{x}$ but j didn't and vice-versa, continue to the next iteration of the inner loop. If i and u were mapped to the same position, this would imply that $x_{ij} \leq x_j$ by the constraint $x_{S\cup T} + x_{S\cap T} \leq x_S + x_T$ and $x_{ij} \leq x_j$ would have been covered by the previous condition so it is safe to skip it.

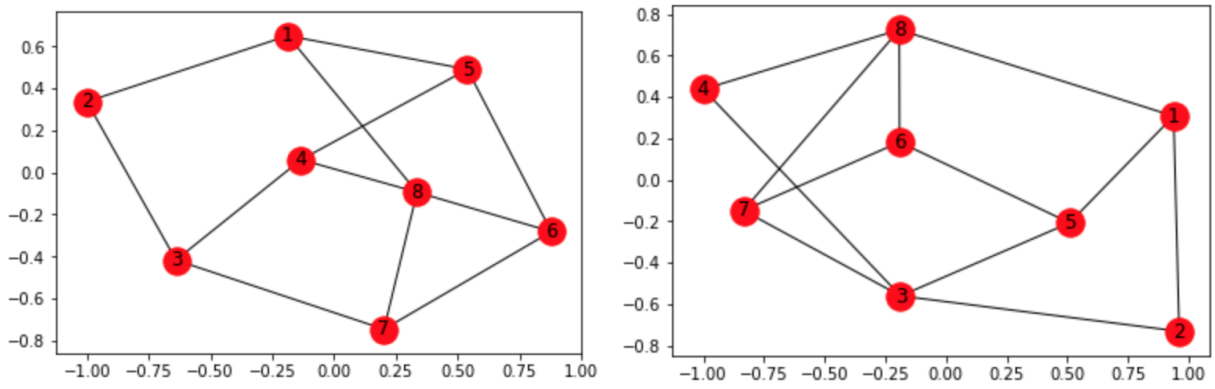
5. If ij was the empty set, if i, j and u were mapped to the same position then continue to the next iteration of the inner loop. Otherwise set the constraint $x_{S\cup T} \leq x_S + x_T$. If i, j and u were the same then this constraint will be reduced to be $0 \leq x_i$ which was already included as all $\overline{x}$ values were non-negative.

6. If ij was not the empty set, and if ij was not a key in equality dictionary, then $x_u + x_ij \leq x_i + x_j$ would be included. If ij was a key and if i and j had the same position then $x_u + x_ij \leq x_i + x_j$ would be included. On the other hand if i and j didn't have the same position but ij had the same position as i then this implies that $x_u \leq x_j$ but $x_u \geq x_j$ by the third condition so $x_u = x_j$. This equality would be added to the equality dictionary following the same proceduce for mapping any collisions. If ij had the same position as j but not i, the same proceduce would be applied. Otherwise, $x_u + x_ij \leq x_i + x_j$ would be included.

There were $(2^n-1)\times(2^{n-1}-1)$ possible pairings (constraints) for the subsets; empty set was exluded as it was evaluated to 0. Those conditions reduced the number of constraints to be evaulated by at least 20%.

Example

The following is the outputs for a graph G with 8 vertices:



(a) G = {1:{2,5,8}, 2:{1,3}, 3:{2,4,7}, 4: {3,5,8}, 5:{1,4,6}, 6:{5,7,8}, 7:{3,6,8}, 8:{1,4,6,7}}

(b) G = {1:{2,5,8}, 2:{1,3}, 3:{2,4,5,7}, 4: {3,8}, 5:{1,3,6}, 6:{5,7,8}, 7:{3,6,8}, 8:{1,4,6,7}}

Figure 1: Graphs with 8 vertices

For the graph in Figure 1a, $\pi^*(G) = \frac{7}{2}$, $\eta(G) = \frac{9}{2}$. For the graph in Figure 1b, $\pi^*(G) = 4$, $\eta(G) = 4$. Both graphs complete outputs are available in the textfile, g8-output-verbose.txt, included with this report.