Final Evaluation Report

Tobenna Peter, Igwe
http://www.github.com/ptigwe/lh-vector

August 20, 2012

1 Introduction

The main aim of this report, is to show the progress made since the midterm evaluations. By the midterm evaluations, the software was able to compute the first set of positively indexed equilibrium gotten from the artificial equilibrium. A complete report of the progress made can be found at `http://www.csc.liv.ac.uk/~cs0tpi/GSOC/mideval.pdf`. In this report, an explanation of the navigation of the bipartite graph of all possible equilibria reachable is provided. Subsequent to these are observations and testing.

2 Implementation

2.1 Restarting from an Equilibrium

Once the first set of positively indexed equilibria has been computed, the next step involves searching for other equilibria from the already computed ones. This involves restarting from one of the computed equilibrium, by choosing a covering vector and pivoting in z_0. When an equilibrium has been computed, the tableau column which represents the covering vector has been changed and a new covering vector would need to be computed which would be able to match the information in the tableau.

The method used to compute the new covering vector is the first method which is explained by Berhnard in his documentation of Lemke's algorithm [2]. This involves the computation of A_B^{-1} for the set of basic variables in B, where A is the tableau which represents the current equilibrium. The column i in A_B^{-1} is given by the column of w_i if w_i is non-basic, or by the column x such that: $x = \begin{pmatrix} 0 & 0 & \cdots & det & \cdots & 0 & 0 \end{pmatrix}^\mathsf{T}$ where det is the determinant of the tableau, and $x[l]$ is equal to det if l is the row of w_i, or 0 otherwise.

The covering vector d which represents a missing label k is given by

$$d = \begin{pmatrix} 1 & 1 & \cdots & 0 & \cdots & 1 & 1 \end{pmatrix}^\mathsf{T}$$

such that $d[i] = 0$ if $i = k$ otherwise $d[i] = 0$. Once the values of d and A_B^{-1} have been computed, the new covering vector d' is calculated as $A_B^{-1}d$. The new covering vector d' can then be substituted in the tableau for the column for z_0.

2.2 Bi-partite Graph

The bi-partite graph containing the equilibria of a game is represented in the program by two linked list, one containing all the negatively indexed equilibria, and the second has as its elements all the positively indexed equilibria. Each node in a linked list stores three values; the equilibrium represented at that node, the list of equilibria the current equilibria is connected with a given missing label, and the next element in the list. By storing each partition of the graph as a list, we can easily get the i-th equilibrium in a list containing n elements, where $0 \leq i < n$, by using the value for next in each node until the i-th node has been found.

Each equilibrium found is stored as the tableau which represents the equilibrium. By doing so the equilibrium for the given node can be copied into the variables used for computation of the new equilibrium.

The list of equilibria which can be gotten from the current equilibria using a set of missing labels is stored as an integer array `link` where the k-th element of the array l, is the location of the equilibrium the current equilibrium is linked to in the opposite list by the missing label $k-1$.

The bi-partite graph is computed using an algorithm similar to the Breadth First Search (BFS) to compute all equilibria reachable from the artificial equilibrium which is listed in Algorithm 2.

Starting from the artificial equilibrium which is stored in `neg[0]`, it first computes the first positively indexed equilibrium by using the missing label 1, stores the resulting equilibrium in `pos[0]` and and links both nodes by setting the first element of both links to 0 (i.e. `neg->link[0] = 0` and `pos->link[0] = 0`). It then makes a call to compute all the equilibrium from the first node of the negatively indexed equilibria (i.e. the artificial equilibrium). Once all the first set of positively linked equilibria have been calculated, it sets `negi` to 1 and `posi` to 0. The value of `negi` is used to store the location of the next negatively indexed equilibrium to restart from, the same goes for `posi` but for positively indexed equilibrium, `isneg` is also set to FALSE (0), to represent that we are going to be restarting from a positively indexed equilibrium.

The algorithm then loops through the following steps until the condition that `negi` equals the number of negatively indexed equilibria, and `posi` equals the number of positively indexed equilibria, indicating that there are no longer any equilibria which can be found.found.

If `isneg` is FALSE, it loops through values of `posi` until `posi` is equal to the length of `pos`. For each value of `posi` it computes all the negatively indexed equilibria which are reachable from the `posi`-th positively indexed equilibrium, storing any new equilibrium found to `neg` and linking the current equilibrium with the equilibrium found using the label it was found with. After each step, `posi` is incremented by 1.found.

Algorithm 1 Compute Equilibria From Node

procedure COMPUTE EQUILIBRIA FROM NODE($i, isneg$) ▷ Computes all equilibria from the current node i
  if $isneg$ **then**
    $cur \leftarrow neg$ ▷ Set current list as the negatively indexed list
    $res \leftarrow pos$ ▷ Set result list as the positively indexed list
  else
    $cur \leftarrow pos$ ▷ Set current list as the positively indexed list
    $res \leftarrow neg$ ▷ Set result list as the negatively indexed list
  end if

  $cur \leftarrow getNodeAt(cur, i)$ ▷ Gets the ith node in the list
  $maxK \leftarrow nrows + ncols$ ▷ The maximum possible value for the missing label

  for $k2 \leftarrow 1, maxK$ **do**
    if $cur.link[k2-1] \neq -1$ **then** ▷ If the equilibrium it is linked with using label $k2$ is known
      continue
    end if

    $copyEquilibrium(cur.eq)$ ▷ Copy and setup the current equilibrium for computation
    $setupEquilibrium()$
    $runLemke()$ ▷ Run Lemke's algorithm to find the next equilibrium

    $eq \leftarrow createEquilibrium()$ ▷ Create equilibrium using the current information in the tableau
    $j \leftarrow addEquilibrium(res, eq)$ ▷ Add the equilibrium to res and return its location. If eq already exists in res, it returns its location but doesn't add it to res

    $cur.link[k2-1] \leftarrow j$ ▷ Link the two equilibria together using label $k2$
    $p \leftarrow getNodeat(res, j)$
    $p.link[k2-1] \leftarrow i$
  end for
end procedure

Algorithm 2 Compute all Equilibrium Reachable from the Artificial Equilibrium

```
1: procedure COMPUTE ALL EQUILIBRIA ▷
2:     maxK ← nrows + ncols
3:     neg = newnode(maxK)          ▷ Create the two lists with the total number of strategies
4:     pos = newnode(maxK)
5:     isqdok()                     ▷ Check if the LCP is valid and generate the tableau
6:     filltableau()
7:
8:     eq ← createEquilibrium()     ▷ Create the Artificial Equilibrium from the current tableau
9:     neg.eq ← eq                  ▷ Set neg[0] to the Artificial Equilibrium
10:    runLemke()     ▷ Run Lemke's algorithm to find the first positively indexed Equilibrium
11:    eq ← createEquilibrium()
12:
13:    pos.eq ← eq
14:    neg.link[0] ← 0              ▷ Link the two equilibria using label 1
15:    pos.link[0] ← 0
16:
17:    negi ← 1                     ▷ Stores its location in the list of negatively indexed Equilibria
18:    posi ← 1                     ▷ Stores its location in the list of positively indexed Equilibria
19:    isneg ← FALSE                ▷ Stores if it is currently restarting from a negatively indexed
       Equilibrium
20:
21:    while TRUE do
22:        if isneg then
23:            while negi < listlength(neg) do
24:                computeEquilibriaFromNode(negi, isneg)
25:                negi++
26:            end while
27:        else
28:            while posi < listlength(pos) do
29:                computeEquilibriaFromNode(posi, isneg)
30:                posi++
31:            end while
32:        end if
33:        isneg =!isneg
34:
35:        if (negi == listlength(neg)) & (posi == listlength(pos)) then
36:            break
37:        end if
38:    end while
39: end procedure
```

If `isneg` is TRUE, the process is the same as the above paragraph, but with `negi` instead of `posi`, `neg` instead of `pos` and `pos` instead of `neg`.

Upon completion of the loop, the values of `negi` and `posi` would store the number of negatively indexed and positively indexed equilibria found respectively.

3 Testing

Bash scripts have been written to aid with the testing of the program, which can be found in the `test` folder. Some of the games are tested by hand and cross-checked using existing pieces of software, and methods such as best response diagrams. Amongst the games chosen for the test cases are identity games (in `test/identity`), $\Gamma(d \times d)$ and $\Gamma(d \times 2d)$ dual cyclic polytopes, and some other games.

Most of the automated tests for path and path lengths, have two similar file names, `*m1` and `*m2` these files run the specified test using method 1 and method 2 of initialisation for the Lemke's algorithm.

For automated testing of identity matrix games, the size of the game to be tested is required as an argument of the bash script which runs the test. The script then generates the game, and the first set of paths from the artificial equilibrium, and tests to see if the paths generated for the given missing labels, are the same as the paths computed by the program.

The same method for testing applies to $\Gamma(d \times d)$ dual-cyclic polytope games, where the paths are generated and checked to see if it is the same equivalent path which the program computed. The Maple script provided by Rahul Savani used to generate the dual-cyclic polytope games is available at `test/dualcyclic/thesis.mws`. Some sample games which were generated by the Maple script have also been provided, and are used in the tests. In addition to testing the paths, another test case was added to confirm that the equilibrium found by the program was a unique equilibrium.

In testing the $\Gamma(d \times 2d)$ dual-cyclic polytope games, the path length of the first couple of pivots from the artificial equilibrium was tested using already known values for the length of the path [1]. The paths of the (2×4) game was computed using the manipulation of bit strings and is also provided (`test/dualcyclic/dx2d/2x4bit`).

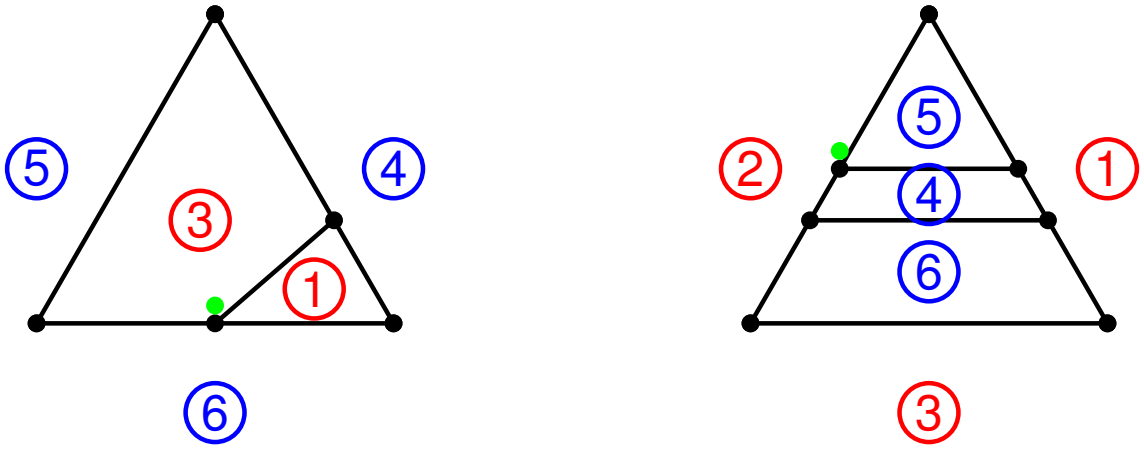
Other tests were carried out using best response diagrams to find the equilibria reachable by Lemke-Howson, and compare with the results gotten from the program. Some of these tests can be found in the figures that are attached below.

The game in Figure 1 contains exactly one equilibrium, which is reached from the artificial equilibrium using all the strategies 1 through 6. The game shown in Figure 2 contains three different Nash Equilibrium, but only one equilibrium the pure strategy $\big((0 \quad 0 \quad 1) \quad (0 \quad 0 \quad 1)\big)$ ● is reachable with the Lemke-Howson algorithm. Finally, the game represented in Figure 3, has 5 Nash Equilibria all of which can be found with the Lemke-Howson algorithm. Using the missing labels 1 through 6 from the artificial equilibrium, we get the following equilibria.

- Labels 1 and 4 leads to equilibrium
- Labels 2 and 5 leads to equilibrium
- Labels 3 and 6 leads to equilibrium
- Restarting from equilibrium with labels 2, 3, 5, 6 leads to
- Restarting from equilibrium with labels 1, 4 leads to
- Restarting from equilibrium with labels 3, 6 leads to
- Restarting from equilibrium with labels 1, 2, 4, 5 leads to

Figure 1: 3×3 Game

$$A = \begin{bmatrix} 5 & 8 & 4 \\ 4 & 5 & 8 \\ 8 & 5 & 10 \end{bmatrix} \qquad B = \begin{bmatrix} -5 & -8 & -4 \\ -5 & -8 & -4 \\ -8 & -5 & -10 \end{bmatrix}$$



References

[1] Rahul Savani. *Finding Nash Equilibria of Bimatrix Games*. PhD thesis, London School of Economics, 2006.

[2] Berhnard von Stengel. Implementation of lemke's algorithm `www.csc.liv.ac.uk/~cs0tpi/GSOC/doculemke.pdf`, 2012.

Figure 2: 3×3 Game

$$A = \begin{bmatrix} 3 & 3 & 0 \\ 4 & 0 & 1 \\ 0 & 4 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 4 & 0 \\ 3 & 0 & 4 \\ 0 & 1 & 5 \end{bmatrix}$$

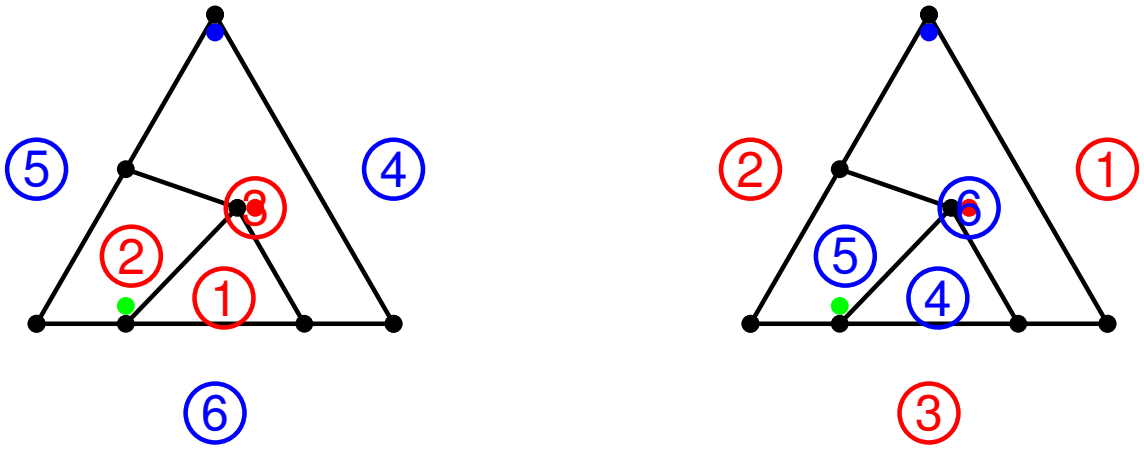


Figure 3: 3×3 Game

$$A = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 2 & 1 \\ -1 & 0 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 & -1 \\ 1 & 2 & 0 \\ -1 & 1 & 2 \end{bmatrix}$$