# Simplex - How To

Vladan Jovičić

October 18, 2016

## Contents

## 1 Introduction

This package contains the following:

1. Implementation of the simplex algorithm
2. Folder `tests` consisting of various LPs used to test the algorithm
3. Folder `vertex_cover` consisting of generator of LPs for vertex cover problem on randomly generated graph (if you want to use it, run `python3 vc_gen.py n` where `n` denotes the number of vertices. The set of edges is generated randomly with probability $\frac{1}{2}$ for each edge.) It also contains files `graph.out` (mentioned below) and `vc_graph.dat` which is LP formulation of the vertex cover problem on the graph described in `graph.out`.
4. Folder `KleeMinty` consisting of generator for the Klee-Minty cubes (i.e. it prints LP formulation of this problems) and LP formulation of Klee-Minty cubes (files `KleeMinty3.dat` ... `KleeMinty10.dat`). If you want to use this generator, run the following `python3 generator.py n` where $n$ denotes the size.

Description of the implemntation is given below.

## 2 Requirements

In order to be able to run the algorithm, the following packages are needed:

- *Python 3*
- *fractions* (a built-in package)
- *argparse* - a package for parsing arguments easily. You can install it by running `pip3 install argparse`

## 3 Usage

To read the rules for usage, you can type `python3 main.py -h`. Detailed description is given below. You can specify the following options for the algorithm:

1. the input file (positional argument)
2. verbose mode: True or False (optional argument). The default value is False
3. pivot rule: bland, max_coef or my_rule (optional argument). If you do not specify this argument, the algorithm will use Bland's rule.
4. maximum number of iterations (optional argument)

### 3.1 Examples

To run the algorithm for a LP given in the file `input.dat` do the following: `python3 main.py -f input.dat`.

If you want verbose mode then do the following: `python3 -f input.dat -v True`. The value True is mandatory, otherwise you will get an error.

If you want to use my_rule then do the following: `python3 -f input.dat -p my_rule`

If you want to set the number of maximum iterations, you can specify it as follows: `python3 main.py -f input.dat -iter-num 100`

Any combination of this will work correctly.

## 4 Limitations

There is no limitations on the number of variables and constraints. The runing time of one iteration of the algorithm is $O((n+m)*m)$. But you must take into account the fact that the algorithms are implemented in python. For example, for the below mentioned vertex cover instance, the number of variables is 30 and the number of constraints is 250. The execution time (on average machine) is less than 3 minutes.

## 5 Solutions of the problems Thief, Power Plants and Fractions

The LP formulation of these problems is given in the files `test_thief.dat`, `test_powerplants.dat`, `test_fractions.dat`. You can find them in `tests` folder. Here are the obtained solutions:

1. Thief problem: the optimal value is $\frac{1567}{21}$. One optimal solution is $(1, 0, 1, 1, \frac{19}{42}, 1, 0)$.

2. Power Plants: the optimal value returned by the algorithm is -25000. Since this is a minimization problem, the objective function is multiplied by $-1$. Thus, the optimal value is 25000. One optimal solution is: $(300, 400, 0, 400, 500, 0)$ meaning that the first power plant will power city1 with 300 megawatts and city2 with 400 megawatts, the second power plant will not power city1 and it will power city2 with 400 megawatts, the third power plant will just power city1 with 500 megawatts.

3. Fraction: the optimal solution is 3. One optimal solution is $(1, 0, 0, 0)$.

## 5.1 Other interesting problems

1. *Klee-Minty cubes* The optimal solution for $n = 3 \ldots 10$ is obtained in one step using my_rule. Using Bland's rule, the number of iterations is: 5, 9, 15, 25, 41, 67, 109, 177 for $n = 3 \ldots 10$ respectively.

2. *Vertex Cover problem* The vertex cover problem is interesting since LP relaxation gives a 2-factor approximation. I have tested the algorithm for various instances of the vertex cover problem. For the graph described in the file `graph.out` (randomly generated graph with probability for the edge eqaul to $\frac{1}{2}$) with 30 vertices and 220 edges, the optimal solution using my_rule is obtained after 160 iterations while the number of iterations needed to solve the same instance using Bland's rule is 261.

3. *Max-flow problem* Another interesting problem is the Max-flow problem. If all capacities of the given graph $G$ are integers then the value and assigned capacites in the optimal solution are also integers. In practice, various problems can be modeled using max-flow (including perfect matching in a bipartite graph) and thus we can solve it using simplex algorithm. I have tested the algorithm on the graph given in Figure 1.

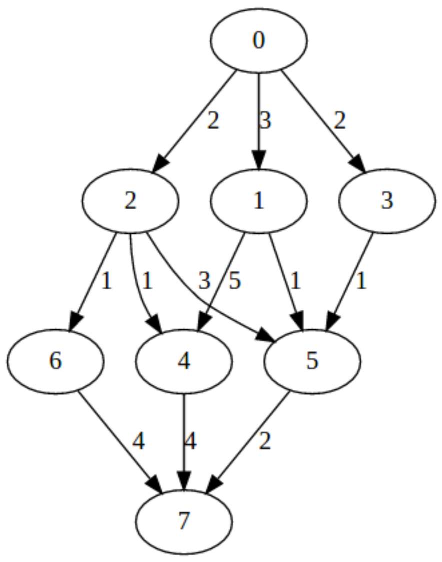

Figure 1: An instance of max-flow problem. The goal is to find max-flow from 0 to 7. The capacities are defined on the edges. The maximum value obtained by the algorithm is 6.

## 6 Description of my_rule

The following rules are used for choosing pivot:

1. Among all variables which are candidates for entering the basis, choose one such that the objective function is maximally increased. This is done by precomputing the value of objective function for each candidate, i.e., for each variable compute coresponding leaving variable and compute the value which will be added to the current value of the objective function and then choose the maximum such value. To break a ties, the maximum subscript rule is used.

2. the leaving variable is chosen by "the most restrictive" principle.

Although the number of iterations using my_rule is less than the number of iterations using Bland's rule in many cases, it can be arbitrary poor compared to Bland's rule. One situation is presented in Figure 2. The algorithm will iterate over vertices $Q, H, I, J, K, P$ using my_rule while it will need only two iterations using Bland's rule to achieve the maximum value.

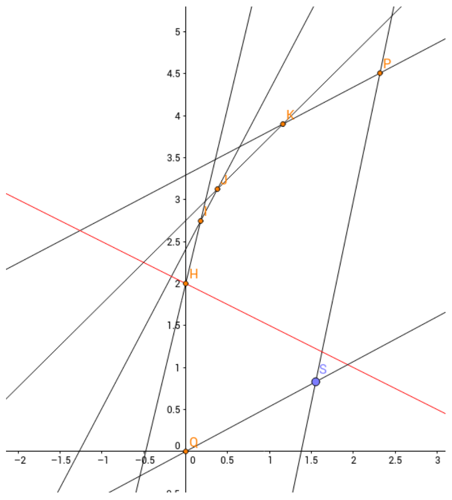

Figure 2: An example of polytope when the number of iteration of the algorithm using my_rule is greater than using Bland's rule. The horizontal axis is $x\_1$ and the vertical is $x\_2$. The orange points are processed by the my_rule. The objective function is represented by red line.

From the other side, we can also construct an instance of LP such that the number of iterations using Bland's rule is arbitrary poor compared to my_rule