Discussion - Homework I - OA

Gabriel Bathie

I) Custom rule : Accelerated Bland's Rule

The custom rule I implemented is an improvement of Bland's rule.
The main issue with Bland's rule is that it tends to make a lot of iterations before converging. The counterpart is that it ensures that the simplex terminates.

What we can remark is that cycles can only occur when the value of the objective function increases by 0 between each vertex. Otherwise, we would go through the same vertex with two different value of the objective function, which is impossible.
Therefore, if we can find a strict increase, we are sure that we are not in a cycle. If no strict increase is possible, then we can just apply Bland's rule in to ensure that we will not cycle indefinitely.

My custom rule does this, but instead of searching for any stricly positive increase of the objective function, it will go for the biggest strictly positive increase that is possible at each step. If no positive increase is possible, then we apply Bland's rule.

In order to measure the efficiency of this rule compared to Bland's and the others, I ran experiments. See section **II** for the results.

Even though there are cases where it might lead to more pivots that Bland's rule, we can see that in practice and in average, it is more efficient that Bland's rule. See also section **IV** for a striking example.

Bonus : I also implemented the random pivot rule, (mainly) for the sake of testing.

II) Comparison of behaviors of pivot rules

Performance comparison

I did an experiment to compare the performance of the four rules. I measured performance by counting the average number of pivots in the simplex before reaching the result (optimal point or certificate of unboundedness), on 30 random LPs with an increasing number of variables and constraints. The results are reported in the following table :

For 10 constraints :

N,M	Bland's	Random	Custom	Max Coeff
(10,10)	10	12	10	8
(20,10)	23	25	16	15
(30,10)	23	23	16	14
(40,10)	24	23	16	14
(50,10)	25	22	15	14

For 20 constraints :

N,M	Bland's	Random	Custom	Max Coeff
(10,20)	13	15	14	11
(20,20)	40	41	35	25

N,M	Bland's	Random	Custom	Max Coeff
(30,20)	66	60	53	34
(40,20)	76	68	55	34
(50,20)	75	65	52	32

Other differences

One important aspect of pivoting rules is their interaction with cycles : can they get stuck in a cycle ?

- The maximum coefficient rule can be stuck in a cycle. See section **IV** for an example.
- Bland's rule and the custom rule can't, by design
- The random rule will almost surely get out of any cycle, you just have to wait, potentially a very long time.

III) Exercises from TD1 and random instances

The input and output files for question 1) (exercises from TD sheet 1) are in the `mandatory-exercises/` folder.

The random LPs are in the `random/` folder, and are named the follwing way : `bathie-random-N-M.dat`, where N is the the number of variables, and M the number of constraints. All `.dat` files have their corresponding `.out` file, containing the output of my program when ran on the file, with the `-c` rule. The LPs are generated with the `random_generator.py` program, with rational coefficients between -10 and 10, and with denominator at most 100000.

IV) Interesting instances :

The interesting instances are located in the `interesting/` folder. ### Exponential number of steps for Bland's rule. The first interesting LP that I chose to include is the LP from TD3, Exercise 7 (with epsilon = 0.001), in the file `bathie-exponential.dat`.

It makes Bland's rule make a very high number of steps. I included the output files for `d = 10`, for Bland's rule and my custom rule. Bland's rule makes 188 pivots before reaching the optimal solution (see `bathie-exponential-bland.out`), whereas the custom rule (accelerated bland's rule) takes only 22 pivots before reaching the optimal (see `bathie-exponential-custom.out`).

I also tried to run the program on the case `d = 20` : the custom rule takes 42 pivots to reach the optimal solution (less that 15 seconds), and Bland's rule does not terminate within 15 minutes.

Cycling LP

- LP that cycles for Max

V) Sensitivity of my program

My program is much more sensible to the number of constraints that it is sensible to the number of variables. One reason for that might be the fact that I treat `=` constraints as two separate `<=` constraints : internally, the number of constraints manipulated can be twice the number of constraints given as an input.

Also, in phase I, I add at most 2 variables per *internal* constraint (slack variable + artificial variable). Therefore, I might add up to 4 variable for the = constraints in the input. This slows down a lot the computations.

For example, a computation with 10000 variables and 10 constraints finishes very fast, whereas a computation with 10 variables and 1000 constraints (that is, 10 times less that we had variables) takes a lot of time to complete.

This seems reasonable, as n constraints can make an exponential number of intersections, that is a exponential number of vertices to potentially visit.