Combining Incremental Precision Boosting and Iterative Refinement for Exact Linear Programming

Jules Nicolas-Thouvenin
Supervisors at Zuse Institute Berlin: Ambros Gleixner, Leon Eifler
Supervisors at University of Nantes (master ORO): Anthony Przybylski, Xavier Gandibleux

Linear programming solvers use floating-point numbers for computational speed, as well as tolerances to deal with roundoff-errors introduced during the solving process. As a result, the optimal solutions returned by these solvers have no guarantee of being exactly optimal (i.e. primal and dual violations are exactly zero). In fact, violations below the tolerances cannot be detected by the solver. On the other hand, exact linear programming solvers guarantee that the returned solution is indeed exactly optimal. They often mix rational arithmetic with the floating-point computations to ensure the exactitude of the results.

There are two methods for solving LPs exactly: iterative refinement and precision boosting. The LP solver SoPlex, if used as an exact LP solver, implements the iterative refinement method. Although this method works well on a majority of instances, there are numerically hard LPs that the solver cannot yet solve.

The goal of our work is to implement an algorithm combining the iterative refinement method of SoPlex with the precision boosting method and study whether our algorithm outperforms both methods separately.

Precision Boosting

The precision boosting method, originally presented in 2006 by Espinoza [1], aims to solve numerically challenging LPs by dynamically increasing the floating-point precision. An iteration of the precision boosting method consists of three main steps. The first step solves the LP using the revised simplex method. The second step checks exactly (i.e. using rational arithmetic) whether the returned basis is indeed optimal. If not, the third step (which we can call the "precision boosting step") increases the floating-point numerical precision and decreases the tolerances of the simplex solver. The hope behind this precision boosting step is that the additional precision will help to successfully solve the LP next iteration.

An important detail on the precision boosting method is the use of a warm start strategy: when possible, the LP solver is warm started with the basis returned at the previous iteration.

[1] Daniel G Espinoza. *On linear programming, integer programming and cutting planes. Georgia Institute of Technology, 2006.*

Iterative Refinement

The iterative refinement method, presented in 2016 by Gleixner et al [2], aims to solve numerically challenging LPs by ”zooming” the LP around increasingly accurate solutions. An iteration of the iterative refinement method consists of three main steps. As with precision boosting, the first two steps involve solving the LP with the revised simplex method and checking exactly whether the final basis is exactly optimal. If the exact check proves that the basis is not exactly optimal, the third step *shifts* and *scales* the original LP to create a *refined* LP. The refinement of the LP can be thought of as ”zooming” the LP around the vertex associated with the final basis. The hope is that this ”zooming” will reveal new simplex pivots available for the next iteration.

Just like in precision boosting, the floating-point solve is warm started at each iteration.

Our algorithm and its configuration

Our algorithm is similar to the precision boosting method, except for the part that solves the LP. While precision boosting uses a standard floating-point solver as an oracle, we use the iterative refinement method to gain numerical accuracy. In this way, if iterative refinement fails to solve the LP, a precision boosting step increases the floating-point precision in the hope that the iterative refinement will successfully solve the LP at the next iteration. In this thesis, we present several flowcharts to better understand how our algorithm works.

There are two main settings for our algorithm. The first is the use of a wam start strategy. Without a warm start strategy, the solving step starts solving the LP again from scratch each time the floating-point precision is increased, which can take a long time. The second setting is the decrease of tolerances during the precision boosting step. If this parameter is set to true, the tolerances are decreased linearly with the precision increase. Otherwise, the tolerances remain the same during the whole run.

Computational Study and Results

The computational study has two goals: identify the best configuration for our algorithm and investigate whether our algorithm outperforms iterative refinement and precision boosting separately. The experiments are performed on a test set of 269 numerically challenging LPs.

We found that the best configuration for our algorithm was the one using warm start and not using the decrease of tolerances. We also found that our algorithm outperforms both iterative refinement and precision boosting. Our algorithm solves 44.59% more numerically difficult instances than iterative refinement and 9.66% more than precision boosting. Furthermore, on instances successfully solved by both our algorithm and precision boosting, our algorithm is 9.41% faster than precision boosting.

[2] Ambros Gleixner, Daniel Steffy, and Kati Wolter. *”Iterative Refinement for Linear Programming”*. In: INFORMS Journal on Computing 28.3 (2016), pp. 449–464. doi: 10.1287/ijoc.2016.0692.