Improving Feasibility of the Starting Basis for Simplex

Ivet Galabova

2017

1 Introduction

Linear programming problems are commonly solved with the simplex algorithm. The Idiot crash is a heuristic algorithm which can improve the performance of simplex-type solvers significantly for certain classes of problems. The crash is a part of CLP, a solver for linear problems written by John Forrest. The algorithm is a way of constructing a point which may be feasible and may be close to the optimal. Using this point to get a better starting basis for simplex could be beneficial. Other than the code itself, the only documentation available for the crash is the following quote by the author:

"John Forrest (2014): "Idiot" is not the barrier method but a cheap imitation - so bad that I called it the Idiot algorithm and gave a bad talk on it years ago - any bad heuristic followed by an algorithm is an algorithm. The simplified basic idea is that you minimize mu*objective + sum of squared primal infeasibilities. This is done on a very local basis i.e. column by column where you just solve a quadratic to get new value. Periodically you reduce mu. For many problems you finish with a small sum of infeasibilities and an objective a bit higher than the optimal one. You then finish with primal."

2 Background

A linear programming problem in standard form is formulated as

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \tag{1}$$

where $\boldsymbol{x}, \boldsymbol{c}, \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. One of the most widely applied ways to solve large linear programming problems (LPs) is to use the simplex algorithm. Simplex performs well from a suitable starting point, however, if the starting point is very infeasible this could hinder performance significantly.

3 The crash

The Idiot crash is a procedure applied to an LP problem prior to the simplex algorithm. A crash is a procedure aiming to find a better starting point for the simplex algorithm. Generally, the solution returned from Idiot is not a basic solution, as required by simplex. After the Idiot crash an additional procedure called a crossover is applied, so a basic solution is found and simplex can start.

The Idiot crash works as follows. Firstly, an initial solution is chosen so it satisfies the bounds on $\boldsymbol{x}$. Then instead of solving one large minimization problem, many small ones are solved. A series of minimizations is performed, column by column. For each $i \in \{1..m\}$, let $e_i(\boldsymbol{x})$ be the infeasibility of row i as a function of $\boldsymbol{x}$. Suppose, that all of the components of $\boldsymbol{x}$ are fixed, except for x_j. The problem being solved for each $j \in \{1..n\}$ is

$$\min_{x_j} \quad \mu\tilde{c}_j x_j + \sum_{i\in 1..m} e_i(\boldsymbol{x})^2, \tag{2}$$

where $\tilde{c}_j$ is a cost described in more detail in the subsections below. There is an outer loop of "major iterations" and an inner loop of "minor iterations". During each minor iteration the minimizations are performed over all the columns. The number of major iterations can be specified as a parameter, otherwise CLP determines it heuristically, usually between 30 and 200. During each major iteration there is between a couple and just over a hundred minor iterations.

3.1 Adjustments

During the execution of the crash, different parameters are adjusted depending on the improvements made over the first major iterations. Below are given more details about the logic behind the analysis and adjustments. In this section, the terms "column value" and "column solution" will refer to the value of variable x_j, the coefficients of which correspond to column j of the matrix A. The "row value" of row i denotes the value of the linear combination of variables defined by the i^{th} constraint of the LP.

3.2 Parameters

Below are given details about the more significant of the various parameters determining the execution.

- The scalar μ controls the weight ratio cost to infeasibility of the function that is minimized for each column:
$$\min_{x_j} \quad \mu\tilde{c}_j x_j + \sum_{i\in 1..m} e_i(\boldsymbol{x})^2.$$
It will be referred to as the weight parameter. The value of μ is set prior to each major iteration and remains unchanged during its execution. After every few major iterations μ is reduced.
- The parameter $\boldsymbol{\lambda} \in \mathbb{R}^m$ determines the cost $\tilde{c}_j$ of the variable x_j in the function we minimize:
$$\tilde{c}_j = c_j + \sum_{i\in 1..m} a_{ij}\lambda_i,$$
where c_j is the original objective cost of variable x_j.
- There is a parameter `maxIts` defining how many times the minimization over all columns should be performed for each major iteration. Recall, that a single run over all the columns is a minor iteration.
- The value of `maxBigIts` determines the maximum number of major iterations which can be performed with the same value of the weight parameter μ.
- The parameter `djExit` determines how much progress is considered enough to continue the major iterations. Details on how progress is measured are given below.

All of the parameters above are set for each major iteration and not modified during its execution.

3.3 Determining the initial weight

The value of μ is initialized at the start, before any major iterations have been performed. It is reduced every few major iterations, thus allowing for a larger weight of the squared sum of infeasibilities as Idiot progresses. It is initialized to a power of 10 between 10^3 and 10^{-3}. In the special case when the average nonzero cost is very large we allow for larger initial values of μ:
$$\mu := \max\left(1000, \frac{\sum_{j:c_j\neq 0} c_j}{\sum_{j:c_j\neq 0} 1}\right)$$

If the first major iteration has not reduced the initial primal infeasibility by more than 10%, μ is reduced by a factor of 10 and one more major iteration is performed. If the 10% reduction is still not present this is repeated until either the reduction is achieved or μ drops below 10^{-30}. If μ drops below 10^{-30} the crash is considered not likely to help so simplex is started from an all logical basis. Alternatively, if the initial infeasibility is reduced the crash continues execution with re-adjusted parameters as described in the section below.

3.4 Number of iterations

As mentioned above, there are normally between 30 and 200 major iterations but this number could be set to any value as a command line parameter by the user. The number `maxBigIts` of major iterations performed with the same value of μ is equal to either 3 or 6, depending on the matrix coefficients and the number of major iterations.

The number `maxIts` of minor iterations is set to two initially. It does not change until a 10% reduction of the initial primal infeasibility is achieved. Once this happens, `maxIts` is set to 105. Note that the value of this parameter does not mean that the full 105 runs over all the columns will be performed. It is possible to exit a major iteration sooner if progress is not sufficient.

Early exit

After 52 minor iterations a check is performed as follows:

A moving average of expected progress is calculated during each 10^{th} minor iteration. For each column j is noted the change in the objective of the function we are minimizing per unit increase in x_j. Then the maximum absolute value of all such changes over the columns is determined. After 52 iterations five such values will have been saved (at minor iterations $10, 20, ..., 50$). If the average of these is below `djExit` the execution of the minor iterations is stopped and $\boldsymbol{\lambda}$ and μ are readjusted for the next major iteration. Otherwise, the minor iterations continue for another 10. The value calculated at iteration 60 replaces the one from iteration 10. The average is checked again: if lower than `djExit` the minor iterations are stopped and $\boldsymbol{\lambda}$ and μ readjusted, otherwise another 10 minor iterations are performed and a check follows and so on.

The outer loop is terminated if the problem becomes feasible. A check is made at every iteration except for the first 50. During the execution of the outer loop, if feasibility is not being improved, then the major iterations can be terminated early too. However, if the number of iterations is specified from the command line by the user, then early exit is not allowed unless the problem becomes feasible.

3.5 The cost term in the quadratic

Recall, that the function minimized for each column $j \in \{1..n\}$ is

$$\min_{x_j} \quad \mu\left(c_j + \sum_{i\in 1..m} a_{ij}\lambda_i\right)x_j + \sum_{i\in 1..m} e_i(\boldsymbol{x})^2.$$

Each time μ is updated $\boldsymbol{\lambda}$ is set to zero. This means that the cost $\tilde{c}_j$ of each column in the function we minimize is the same as the column cost c_j in the original problem. Then, in the next few major iterations μ remains unchanged but the $\boldsymbol{\lambda}$s are modified:

$$\lambda_i = \frac{1}{\mu}e_i(\boldsymbol{x}),$$

where $e_i(\boldsymbol{x})$ is the infeasibility of row i at the end of the previous major iteration.

4 Performance

The Idiot crash works very well on some problems and can be disastrous on others. Care must be taken with the decision whether to apply it or not. A particular class of problems for which Idiot is very beneficial is the quadratic assignment problems.

4.1 Quadratic assignment problems

The quadratic assignment problem (QAP) is a combinatorial optimization problem - a special case of the facility location problem. There are a set of facilities and a set of locations. For each pair of locations there is a distance and for each pair of facilities there is a weight or flow specified, for instance the amount of items transported between the two facilities. The problem is to assign all facilities to different locations so that the sum of the distances multiplied by the corresponding flows is minimized. Often, instead of the quadratic a linearisation is solved. QAPs are well known for being very difficult to solve, even for small instances (they are NP-hard and the travelling salesman problem can be seen as a special case of QAP).

The Idiot crash contributes for a significant improvement of the solution time of QAPs. An example is the qap12 instance from the Netlib test set. The solution time of CLP with no Idiot is about 20 seconds whilst Idiot brings that down to 6 seconds.