Practical guidelines for solving difficult linear programs

Klots and Newman

Linear Programming Fundamentals

$x \in \mathbb{R}^{n \times 1}$ is continous-valued, non-negative decision variables

$A \in \mathbb{R}^{m \times n}$ is the left hand side constraint coefficients

$c \in \mathbb{R}^{n \times 1}$ is the objective function coefficients

$b \in \mathbb{R}^{m \times 1}$ is the right hand side data values

$$(P_{LP}) : \min c^T x$$
$$\text{subject to } Ax = b$$
$$x \geq 0$$

Without loss of generality, any linear program can be written as above.
The related dual problem is:

$$(D_{LP}) : \max y^T b$$
$$\text{subject to } y^T A \leq c^T$$

Linear Programming Fundamentals

The most commonly used LP algorithms solve linear systems of equations dimensioned by the number of constraints.

- The number of constraints is a more significant measure of solution time than the number of variables.

A basis for the primal system consists of m variables whose associated matrix columns are linearly independent.

- Basic variables are obtained by solving $Ax=b$.
- Each vertex of polyhedron formed by $Ax=b$ is a basic solution.
- If the solution also satisfies non-negativity constraints it is a basic feasible solution.
- The basic feasible solution is a candidate for the optimal solution

Primal Simplex Algorithm

Obtain basis with corresponding feasible solution

1. Backsolve - Obtain dual variables by solving $y^T*A_B=c^T_B$
2. Pricing - Calculate the reduced cost
3. Pick the entering variable x_t
4. Forward solve - Calculate the corresponding column
5. Ratio test - Determine θ, how much the entering variable can increase
6. Basis update by calculating A_B
7. Recalculate value of basic variables - Either update **or** refactorize
    a. Update rows by: $x_t=\theta$; $x_i \leftarrow x_i-\theta*w_i$; for $i \in\{B\}-\{t\}$
    b. Refactorize - LU factorization: solve $A_B*x_B=b$
8. Return to step 1

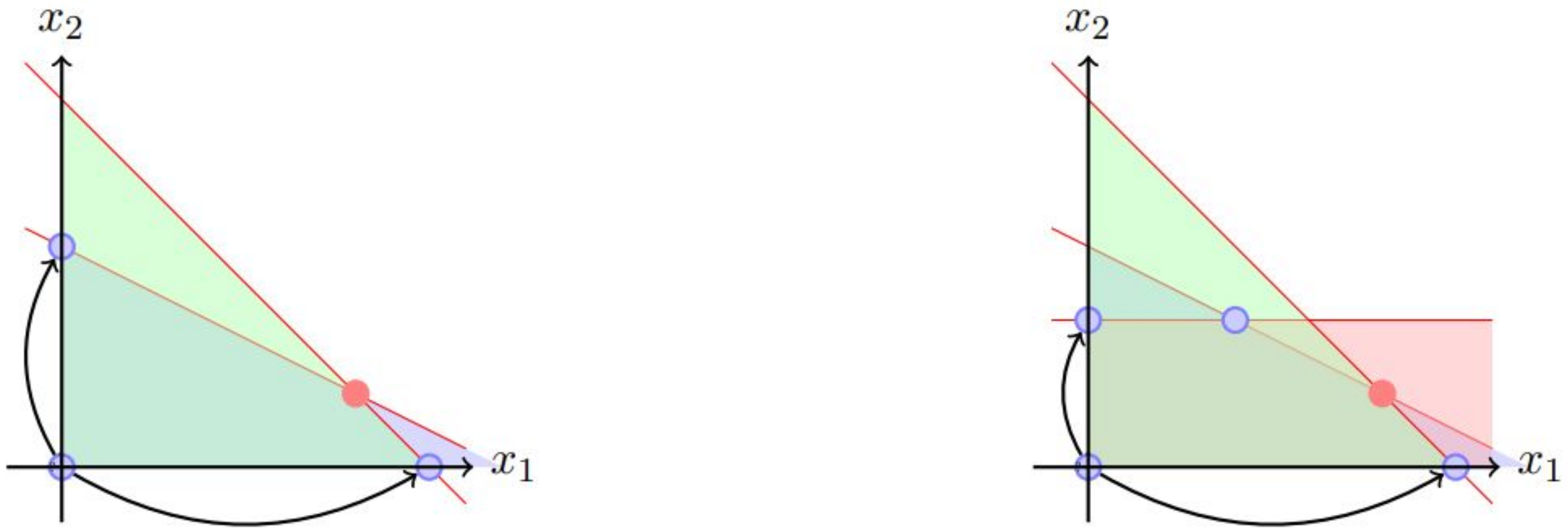

Figure 2.6: The search process of the simplex.

Algorithm performance contrast (2.3)

Section describes when we should use Primal/Dual - Simplex/Interior Point

IPA less likely to outperform Simplex if (or)

- Many more non-zeros in lower triangle of AA^T than number of non-zeros in A
- Number of non-zeros in Cholesky factor grows dramatically relative to number of non-zeros in AA^T

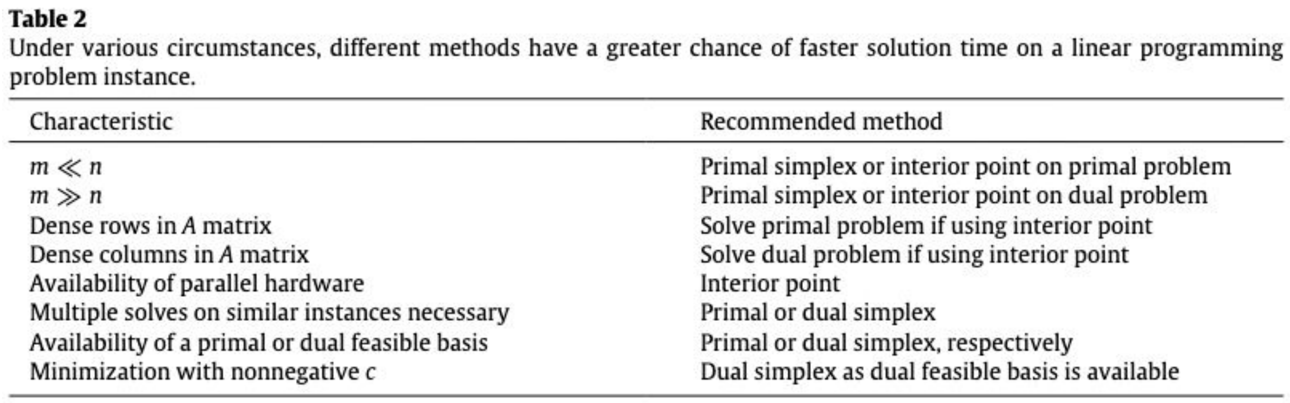

Table 2
Under various circumstances, different methods have a greater chance of faster solution time on a linear programming problem instance.

Characteristic	Recommended method
$m \ll n$	Primal simplex or interior point on primal problem
$m \gg n$	Primal simplex or interior point on dual problem
Dense rows in *A* matrix	Solve primal problem if using interior point
Dense columns in *A* matrix	Solve dual problem if using interior point
Availability of parallel hardware	Interior point
Multiple solves on similar instances necessary	Primal or dual simplex
Availability of a primal or dual feasible basis	Primal or dual simplex, respectively
Minimization with nonnegative *c*	Dual simplex as dual feasible basis is available

~ n/m > 2
~ n/m < 0,5

(Characteristics are not mutually exclusive)

Numerical stability and ill conditioning (3.1)

- We say a matrix A is ill conditioned when small changes to input lead to large changes in output.
  Therefore numerical error are amplified when solving ill conditioned systems on the form: Ax = b

- Condition number

$$\kappa = \|A_B\| \|A_B^{-1}\|$$

  Large condition number implies a more ill conditioned system.

Numerical stability and ill conditioning (3.1)

- Computational errors can come from many sources, here are some of the main ones listed in the paper:
  - Floating point errors due to finite precision of computers.
  - Poorly chosen parameters of the computation (tolerance, etc.)
  - Rounding errors in input data (e.g. single precision programs -> double precision programs).
  - Small numerical values relative to the tolerance of the optimizer.

- The interior point method is more susceptible to numerical stability issues, as it tries to maintain an interior solution, compared to the Simplex method.

Degenerancy (3.2)

Evidence when objective function value does not change throughout iterations

Good idea to try all other LP algorithms. Primal degenerancy does not imply the same in the dual and interior point algs do not have degenerancy.

Bland rule although theoretically avoids cycling seldom used in practice.

Perturbations introduced on input data (b values if primal simplex, c values if dual simplex) either by the practitioner or by the solver.

Excessive simplex algorithm iteration counts (3.3)

- Excessive iterations because of inferior choice of basic variables
- Use Steepest edge and Devex pricing for selection rules
  - “Steepest edge computes L2 norm of each non-basic matrix column relative to the current basis”
  - These extra calculations almost require the same amount of work for the primal simplex iteration but it can be advantageous for the dual as it only requires as 20% iteration reduction as opposed to 50%
- Important to choose optimizer tolerances to distinguish legitimate values from values from round-off error
- Reducing the maximum allowable violation in the Harris ratio test can improve performance

Excessive barrier algorithm iteration counts (3.4)

- The barrier algorithm relies on convergence criteria.
    - It may struggle to converge, even though the total number of iterations may be modest.
    - Use adjustable convergence tolerance.
- A larger barrier convergence tolerance may save significant time when the barrier method run time dominates the crossover run time.
- However, Increasing the barrier convergence tolerance avoids barrier iterations of little, if any, benefit to the crossover procedure.
- Reducing convergence tolerance may provide better interior points.
    - May yield better starting points for crossover, improving performance.
    - Possible when the crossover time comprises a significant part of optimization time with a modest barrier iteration count.

Excessive time per iteration (3.5)

- After many iterations of the simplex the time per iteration can increase dramatically
  - Memory consumption increases with denser matrices
  - Dense bases and factorization
- Reduce cholesky factor by column splitting
  - This introduces *more* constraints, but there are fewer non-zeros
- If neither barrier or simplex algorithms work
  - Use sifting
    - If constraint matrix has many more constraints, then use sifting on the dual LP
  - Dantzig Wolfe or Bender's Decomposition