1 The Running Time of the Simplex Method

1.1 Worst-Case Running Time

The worst-case running time of the Simplex method is exponential. This is known to be true for all proposed pivoting rules. It is not known whether there can be no pivoting rule which leads to polynomial running time.

The worst-case examples are variants of the Klee-Minty cube. The basic idea is simple. Consider the unit-cube C in R^n defined by the inequalities $0 \leq x_i \leq 1$ for all i, $1 \leq i \leq n$; the cube has 2^n vertices corresponding to the binary strings of length n. There is a path along edges of C from vertex 0^n to vertex 10^{n-1} visiting every vertex exactly once.

This is easy to see by induction. It is certainly true for $n = 1$. For $n > 1$ we move as follows:

$$000^{n-2} \xrightarrow{*} 010^{n-2} \rightarrow 110^{n-2} \xrightarrow{*} 100^{n-2}$$

The path p from 00^{n-2} to 010^{n-2} exists by induction hypothesis. For all nodes on this path we have $x_1 = 0$. From 010^{n-2} to 110^{n-2} we follow an edge. For the third part we use a path p' which is simply the path p in reverse order but now with all nodes having $x_1 = 1$.

Assume next that our goal is to maximize x_1 and we start in vertex 0^n. On the unit cube the simplex method will find an optimal vertex in a single iteration.

In the path constructed above the objective functions never decreases. It is now possible to distort the unit cube such that every edge of the path is improving (and not only non-decreasing) and such that the simplex algorithm with the particular pivoting rule follows the path. The details can be found in [Sch86, Section 11.4].

1.2 Average Case Running Time

The average case running time of the Simplex method is much better. In the early 80s it was shown by Borgwardt, Smale and later Haimowich that the number of iterations is polynomially bounded. In fact, even linearly bounded. This assumes a particular pivoting rule (the so-called Schatteneckenrule) and a particular probability distribution on LP-problems. It is not clear (at least, to me) what the connection is between probabilistic model and reality. This is true for probabilistic analysis in general. Fact is that the simplex method performs very well in practice.

The Schatteneckenrule has a nice geometric interpretation. Assume that we are trying to maximize $c^T x$ subject to $Ax \leq b$ and assume also that we know an optimal vertex x_0^* with respect to a cost function d. We then look at the plane (= two-dimensional space) spanned by c and d and (conceptually) project the polytope P_0 defined by $Ax \leq b$ into this plane. The projection is a polygon P (= the shadow of P_0 under parallel light orthogonal to the plane spanned by c and d.) in two-dimensional space, the projection of x_0^* is optimal in direction d, and the projection of the optimal solution x^* is optimal in direction c. We can walk from one projection to the other along the boundary of P. This corresponds to a walk where we slowly transform the direction d into the direction c. The Schattenecken rule performs this walk on P_0, i.e., it walks exactly through those vertices of P_0 which project onto the part of the boundary of P between the two optimal vertices.

Very recently, Spielman and Teng [ST01] strengthened the probabilistic analysis of the simplex method. They consider LPs of the form

$$\max c^T x \quad \text{subject to} \quad (A + \sigma G)x \leq b$$

where c, A, and b are arbitrary (of course of the right dimension), G is a matrix of independently chosen Gaussian random variables with mean 0 and variance 1 and $\sigma \geq 0$ is an arbitrary real.

The density function of the Gaussian distribution is

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} .$$

Recall that $\int_{-\infty}^{+\infty} f(x)dx = 1$. The Gaussian distribution has the well known bell shape.

For $\sigma = 0$, we are in the worst case scenario. For very large σ the random part dominates and the result is basically Borgwardts. The spectacular case is $\sigma > 0$ and small. Spielman and Teng prove that the running time of the simplex method with the Schatteneckenrule is polynomial for any fixed $\sigma > 0$, i.e., averaging over the random perturbations of any fixed problem $Ax \leq b$ suffices to get polynomiality.

The paper is available at `http://www-math.mit.edu/~spielman/simplex/index.html`. Rene Baier will discuss the paper in more detail in the seminar of the Graduiertenkolleg.

1.3 Randomized Simplex Algorithms

Randomized simplex algorithms select the variable entering or leaving the basis at random among the set of candidate values. The analysis of randomized pivot rules has received a lot of attention in recent years and it has been shown that some of them guarantee subexponential (although not polynomial) running time. We refer the reader to [MSW96] for one such result.

A particularly simple randomized simplex algorithm is due to Raimund Seidel [Sei91]. It is suitable for problems with a fixed number d of variables and an arbitrary number m of constraints. The running time is linear in m, but exponential in d.

So assume we are given m constraints $a_i^T x \leq b_i$ in d variables x_1 to x_d. The goal is to maximize $c^T x = \sum c_j x_j$. Call the system LP. We assume that the problem is feasible and that the cost vector has at least one non-zero entry. Otherwise, the problem is trivial. For technical reasons we also add special constraints $|x_i| \leq M$ for some large M. This is to guarantee boundedness. We make the technical assumption that the optimum to our LP is unique and that the LP is non-degenerate, i.e., there is no point satisfying $d + 1$ of the inequalities with equality.

If $m = 0$, an optimal solution is trivial to find. Set $x_i^* = M$ if $c_i > 0$ and set $x_i^* = -M$ if $c_i \leq 0$.

If $d = 1$, the problem is also trivial. We only have a single variable x_1 and hence our constraints limit x_1 to an interval. The optimal solution is one of the endpoint of the interval.

So assume $m > 0$. We chose one of the constraint at random (by selecting a random integer i between 1 and m), drop it, and solve the resulting system LP' of $m-1$ constraints. Let x^* be the optimal solution. If x^* is also feasible for LP we are done (since the objective value for LP cannot be larger than the objective value for LP'). So assume that x^* is not feasible. Then any optimal solution of LP must satisfy the i-th constraint with equality. We take the i-constraint and solve it for any of the variables, say x_j, which has a non-zero coefficient in the i-th constraint ($a_{ij} \neq 0$). We obtain an expression E for x_j in the remaining variables. We plug E into the other constraints (giving us $m-1$ constraints in $d-1$ variables) and also into the special constraints for x_j (giving us the two additional constraints $-M \leq E \leq M$). We then solve the resulting system of $m+1$ constraints in $d-1$ variables. This gives us values for all variables but x_j. We compute the value of x_j by substituting the values of the other variables into E.

```
opt_solution LP_solve(linear_program LP)
{ /* returns an optimal solution to LP. LP has d >= 1 variables and
     m>=0 normal constraints;
     in addition there are 2d special constraints. */
  if ( d == 1 ) compute and return optimal solution;
  if ( m == 0 ) compute and return optimal solution
  i = a random integer in [1 .. m];
  let LP' be obtained from LP by deleting the i-th constraint;
  x* = LP_solve(LP')
  if ( x* satisfies the i-th constraint ) return x*;
  solve i-th constraint for one of its variables, say xj;
  substitute expression E for xj into the other constraints,
  remove the special constraints -M <= xj <= M,
  and add the normal constraints -M <= E <= M;
  call the resulting linear program LP''
  x' = LP_solve(LP'');
  obtain x* from x' by setting xj = E(x');
  return x*;
}
```

We want to analyze the expected running time of this randomized procedure. We count the number of calls to *LP_solve*. Let $T(LP)$ be the expected number of calls (top-level plus recursive calls) to *LP_solve* in a call *LP_solve*(LP) and let

$$T(d, m) = \max \{T(LP) \; ; \; LP \text{ is a linear program in } d \text{ variables with } m \text{ constraints}\} \ .$$

We derive a recurrence relation for $T(d,m)$. We have

$$\begin{aligned} T(d,0) &= 1 \quad \text{for all } d \\ T(1.m) &= 1 \quad \text{for all } m \\ T(d,m) &\leq 1 + T(d,m-1) + \frac{d}{m} T(d-1,m+1) \quad \text{for } d > 1 \text{ and } m > 0 \end{aligned}$$

The last inequality requires proof. Let x^* be the optimal solution of LP; x^* satisfies d inequalities with equality. Among these inequalities there are d' normal inequalities for some $d' \leq d$. If the d inequalities defining x^* are also inequalities in LP', x^* is also the optimal solution for LP'. The probability for this to happen is $\frac{m-d'}{m}$. Thus the probability that we also have the recursive call $LP\_solve(LP'')$ is $\frac{d'}{m} \leq \frac{d}{m}$. LP'' has $m+1$ constraints and $d-1$ variables.

We prove $T(d,m) \leq 1 + c_d m$ for all d and m where $c_1 = 0$ and $c_d = (d+1) + 2dc_{d-1}$ for $d \geq 1$. In particular, $c_1 = 3$, $c_3 = 4 + 2 \cdot 3 \cdot 3 = 22$, ... Can you find an explicit expression for c_d?

The claim is certainly true for $d = 1$ and $c_1 = 0$. So assume that $d > 1$. For $m = 0$, the claim is true (for any definition of c_d). For $m > 0$ we have

$$\begin{aligned} T(d,m) &\leq 1 + T(d,m-1) + \frac{d}{m} T(d-1,m+1) \\ &\leq 1 + 1 + c_d(m-1) + \frac{d}{m}(1 + c_{d-1}(m+1)) \\ &= 1 + c_d m + (-c_d + 1 + \frac{d}{m} + dc_{d-1}\frac{m+1}{m}) \\ &\leq 1 + c_d m + (-c_d + 1 + d + 2dc_{d-1}) \\ &= 1 + c_d m \end{aligned}$$

where the last equality follows from the definition of c_d.

2 Implementations of the Simplex Method

The simplex method is trivial to implement, the simplex method is very difficult to implement. Both statements are true.

In what sense is it trivial to implement? Our formulation of the simplex method (primal or dual) is quite close to real code. It is not difficult (does not require more than a few days of work) to turn our formulation into executable code in any programming language which offers arrays and vectors. In this sense it is trivial to implement the simplex method.

In what sense is it difficult to implement. There are at least two issues which we did not discuss: exploitation of sparsity and numerical stability. Most linear programs arising in practice are sparse in the sense that most entries of the constraint matrix are zero.

For example, in the LP formulation of the minimum cost flow problem we had $n + m$ constraints, where n is the number of nodes and m is the number of edges of the graph, and m variables. The variables correspond to the edges of the graph. For each variable x_e, $e = (u, v)$, we had three non-zero entries in the constraint matrix, one is the balance constraint for u, one in the balance constraint for v, and one in the capacity constraint for e. Thus we have only $3m$ non-zero entries in a matrix of size $(n + m)m$. Any efficient implementation of the simplex method must exploit sparsity. Techniques for exploiting sparsity are described in [Chv93, Section 24] and in [Kwa98].

The second issue is numerical stability. Implementations of the simplex method use floating point arithmetic and floating point arithmetic incurs round-off errors. I strongly recommend the article by Goldberg [Gol90]. Section 9.6 of the LEDAbook [MN99] shows some surprising examples of what can go wrong when floating point arithmetic is used naively. The fact that floating point arithmetic incurs round-off error implies that quantities are only known approximately and hence decisions cannot be made with certainty. Numerical analysis has developed techniques for coping with round-off error [DH91]. A good implementation of the simplex method must use these techniques.

Many implementations of the simplex method are available, commercial ones and non-commercial ones. I have used CPLEX [CPL] and SOPLEX [SoP]. SOPLEX is freely available. Michael Trick[1] (`http://mat.gsia.cmu.edu/`) maintains a web page on combinatorial optimization. The page contains links to implementations and many other things.

How good are these implementations? How often do they find the optimum? Karsten Kwappik [Kwa98] addressed this issue in his Diplomarbeit and Dennis Weber is now continuing and extending the work (FOPRA und Diplomarbeit). We pursue the following approach. LP-solvers returns a numerical object (= the coordinates of a solution, the objective value) and a combinatorial object (= a basis). Primal and dual feasibility of a basis can be checked by solving *one* linear system of equations *exactly*. Exactly means that the system is solved with exact arithmetic (arbitrary precision integers, arbitrary precision rationals). This is costly. On the other hand, only a single system has to be solved.

Kwappik was able to check about 80 to 90% of the examples in the netlib-collection of LPs. The running time for checking was comparable to what it took CPLEX or SOPLEX to find the solution. In 10 to 20% of the cases, the checker was too slow and could not run to completion. Dennis Weber is exploring alternative approaches for solving large sparse linear systems and we hope to make a considerable step forward through his work. In the cases which Kwappik was able to check, CPLEX and SOPLEX had actually computed the optimal solution in about 90% of the cases. Thus state-of-the-art LP-solvers are very good and find the correct solution in most cases. But they can also err.

As part of Dennis Weber's work we will set up a server for checking LP solutions. One can send pairs consisting of an LP and a basis to the server and the server computes and returns whether the basis is primal and/or dual feasible. The server will use the exact

[1] Michael computes the tournament schedules of some of the baseball and basketball conferences in the US.

algorithms developed by Kwappik and Weber.

References

[Chv93] V. Chvatal. *Linear Programming.* Freeman, 93.

[CPL] CPLEX. `www.cplex.com`.

[DH91] P. Deuflhard and A. Hohmann. *Numerische Mathematik: Eine algorithmisch orientierte Einführung.* Walter de Gruyter, 1991.

[Gol90] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1990.

[Kwa98] C. Kwappik. Exact linear programming. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 1998.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing.* Cambridge University Press, 1999. 1018 pages.

[MSW96] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, October/November 1996.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[Sei91] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete and Computational Geometry*, 6:423–434, 1991.

[SoP] SoPlex. `www.zib.de/Optimization/Software/Soplex`.

[ST01] D. Spielman and S.-H. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *STOC*, 2001.