CHAPTER 3

EARLY INTERIOR-POINT METHODS

An interior-point algorithm is one that improves a feasible interior solution point of the linear program by steps through the interior, rather than one that improves by steps around the boundary of the feasible region, as the classical Simplex Algorithm does. The earliest interior-point method is due to the famous mathematician John von Neumann. His method for finding a feasible solution to a linear program with a convexity constraint is notable for its simplicity and remarkable convergence properties; see Section 3.1. Since a general linear program combined with its dual can be reformulated into a feasibility problem of this restricted form, *von Neumann's algorithm* may be viewed as a method for solving the general linear program.

Just like there are many variants of the Simplex Method (which we refer to as pivot step algorithms), so there are many variants of interior methods such as *projective* and/or *potential reduction*, *affine*, and *path-following*.

1. *Projective and Potential Reduction Methods.* These methods measure the approach toward an optimal solution by the reduction of the value of a potential function rather than the reduction of the value of the linear objective. For example, Karmarkar's agorithm is typically based on projective geometry but uses a potential function to measure progress of the solution towards optimality. The potential function is typically designed to ensure the following: (a) the objective function decreases at each iteration, (b) the solution point stays in the interior of the feasible space, and (c) the algorithm converges in polynomial time. In practice, these methods have not done well.

2. *Affine Methods.* These methods approximate the feasible region, at each iteration, by an ellipsoid and optimize over the ellipsoid. The implementation of

such methods is easy as we saw in the discussion of one such method in *Linear Programming 1: Introduction*. In this chapter, we discuss Dikin's method, an early affine method. In practice these methods perform quite well but not as well as the path-following methods.

3. *Path-Following Methods*. These methods follow a certain path as the optimal solution is approached. The linear program is first transformed into an unconstrained nonlinear optimization problem, called a *logarithmic barrier function*. The logarithmic barrier function typically consists of the objective function and one or more additional terms, multiplied by a scalar positive parameter, that increase in value as the iterates approach the boundary. In effect, the additional terms throw up a barrier at the boundary. The unconstrained optimization problem is solved and the parameter value reduced for the next iteration. The optimal values of the sequence of unconstrained problems approach the optimal solution of the linear program along a path through the interior of the feasible region.

   Path-following methods have performed the best in theory and practice in recent times. In Chapter 4 we will describe the primal logarithmic barrier method and the primal-dual logarithmic barrier method.

Some other interior-methods inscribe an ellipsoidal ball in the feasible region with its center at the current iterate, or first transform the feasible space and then inscribe a hypersphere with the current iterate at the center. Then an improving direction is found by joining the current iterate to the point on the boundary of the ellipsoid or sphere that maximizes (or minimizes) the linear objective function (obtained by solving a least-squares problem). A point is then selected on the improving direction line as the next current iterate. Sometimes this iterate is found along a line that is a linear combination of the improving direction and some other direction.

In 1967 Dikin proposed an affine method that in its original form is not a finite method but one that converges in the limit. In particular, Dikin's method as described in Section 3.2, has an asymptotic rate of convergence of $1 - 1/\sqrt{m}$. This method has the distinction of having been rediscovered by many; for example, the *primal affine method* is the same as *Dikin's method*.

During the period 1979–2003, there has been intense interest in the development of interior-point methods. These methods are related to classical least-square methods used in numerical analysis for making fits to data or fitting simpler functional forms to more complicated ones. Therefore interior research can tap into the vast literature of approximation theory. A theoretical breakthrough came in 1979: the Russian mathematician L. G. Khachian (based on the work of Shor, 1971–1972) discovered an ellipsoid algorithm whose running time in its worst case was significantly lower than that of the Simplex Algorithm in its worst case. Its iterates are not required to be feasible. Other theoretical results quickly followed, notably that of N. Karmarkar, who discovered an interior-point algorithm whose running-time performance in its worst case was significantly lower than that of Khachian's. This

was followed by the theoretical results of others that improved on the upper-bound estimates of the worst-case performance as the dimensions of the problems and the amount of input data increased indefinitely.

The algorithm best suited for solving a particular problem or a special class of problems may not be the same algorithm best suited for solving any problem from the broad class of problems defined by $Ax \geq b$, $x \geq 0$, $c^T x = \min$. One criterion used for comparing algorithms is upper bounds on worst-case performance times as the dimensions of the problem grow indefinitely in size. This criterion turns out to be totally misleading for deciding which algorithm to use for practical problems because these theoretical upper-bound estimates are many many times greater than any experienced with practical problems.

Attempts to characterize in a simple way the class (or classes) of practical problems from which one might be able to derive a theoretical explanation of the excellent performance times of some of the algorithms used in practice have, in general, failed. In special cases, such as the shortest-path problem, the performance of shortest-path algorithms for the entire class of shortest-path problems is comparable to that observed on actual problems. There has been progress proving that *average performance* on classes of *randomly generated* problems using a parametric variant of the Simplex Method resembles that obtained on practical problems, but no one claims these randomly generated problems are representative of the class of practical linear programs.

Because the theoretical results can be totally misleading as to what algorithm to choose to solve a practical problem, an emperical approach is used. The linear programming profession has accumulated a large collection of test problems drawn from practical sources. These are used to compare the running times of various proposed algorithms. The general goal of these efforts is to find *the algorithm* that surpasses the performance of all other algorithms in the collection.

For example, Karmarkar claimed (when he developed his method) that on very large problems his technique would be significantly faster. As of this writing, as far as the authors can ascertain, there appears to be no one algorithm that is a clear winner, i.e., that solves *all (or almost all)* of the test problems faster than all the other proposed methods. On problems with many bounding hyperplanes in the neighborhood of the optimum point, an interior method will probably do better than an exterior method. On problems with relatively few boundary planes (which is often the case in practice) an exterior method will be hard to beat. For this reason, it is likely that the commercial software of the future will be some sort of a hybrid because one does not know which kind of problem is being solved or because one wishes to obtain an extreme-point solution. Many specialized efficient codes have been proposed for solving structured linear programs such as network problems, staircase problems, block-angular systems, and multi-stage stochastic systems.

> *Definition (Polynomial Time)*: Let the problem data size (length of the input data stream) be $\mathcal{L}$, the total number of bits required to store the data of a linear program in m equations and n variables in the computer. An algorithm is said to have a *polynomial worst-case* running time if the algorithm's execu-

tion time (in, say, seconds) to find an optimal solution on the computer is less than some polynomial expression in $\mathcal{L}$, m, and n. Otherwise the algorithm is said to be NP (nonpolynomial).

Worst-Case Measures Can be Misleading. For example, given a linear program in m equations and n variables, it may be stated that a method requires less than $O(n^p m^q)$ iterations, where $O(n^p m^q)$ means some fixed constant times $n^p m^q$. If the constant for the worst-case bound were huge, say 10^{100} (which may be larger than the number of all the electrons in the universe), then such a bound would be ridiculous. Implicit in such statements about a worst-case bound is the assumption that the fixed constant is small, say 10 or 100. Usually this assumption is valid, and it has become common practice to compare worst-case bounds of algorithms *as if* the fixed constants for each of the algorithms are the same.

In general, given a linear program in m equations and n variables, projective methods require less than $O(n)$ iterations. Path-following methods require less than $O(\sqrt{n})$ iterations. Each of these also require $O(n^3)$ arithmetic operations per iteration to solve a linear least-squares subproblem in order to find a steepest descent direction. However, with refinements (such as rank-one updates) it is possible to solve each least-squares problem in $O(n^{5/2})$ arithmetic operations instead of $O(n^3)$. When the number of operations is multiplied by the bound on the number of iterations, we find that Karmarkar's projective method is bounded by $O(n^{7/2})$ arithmetic operations while path-following methods are bounded by $O(n^3)$ arithmetic operations to obtain an optimal solution within a tolerance $\epsilon > 0$. The time required to carry out the arithmetic operations depend on $\mathcal{L}$, the digits of input data. Thus the bound on the *time* to execute Karmarkar's algorithm is $O(n^{7/2}\mathcal{L})$.

Because the number of arithmetic operations (and iterations) can depend critically on $\epsilon > 0$, the bound on the accuracy of the computed optimal solution, we will use the following definition of a polynomial-time algorithm.

> *Definition (ϵ-Optimal Polynomial Time)*: An algorithm to solve a linear program in m rows and n columns is said to have a *polynomial worst-case* running time (measured in seconds, say) if the time to execute it is less than some polynomial expression in $\mathcal{L}$, m, n, $q = -\log_{10}\epsilon$, where $\mathcal{L}$ is the total number of bits required to store the problem's data in the computer, q is the number of significant decimal digits of accuracy of the optimal solution, and the tolerance ϵ is some measure of how much the calculated solution differs from the true optimal objective value, or an approximate feasible solution to a feasibility problem differs from the right-hand side of a true feasible solution.

3.1 VON NEUMANN'S METHOD

Von Neumann, in a discussion with George Dantzig in 1948, proposed the first interior algorithm for finding a feasible solution to a linear program with a convexity

http://www.springer.com/978-0-387-98613-5

Linear Programming 2
Theory and Extensions
Dantzig, G.B.; Thapa, M.N.
2003, XXVI, 448 p., Hardcover
ISBN: 978-0-387-98613-5