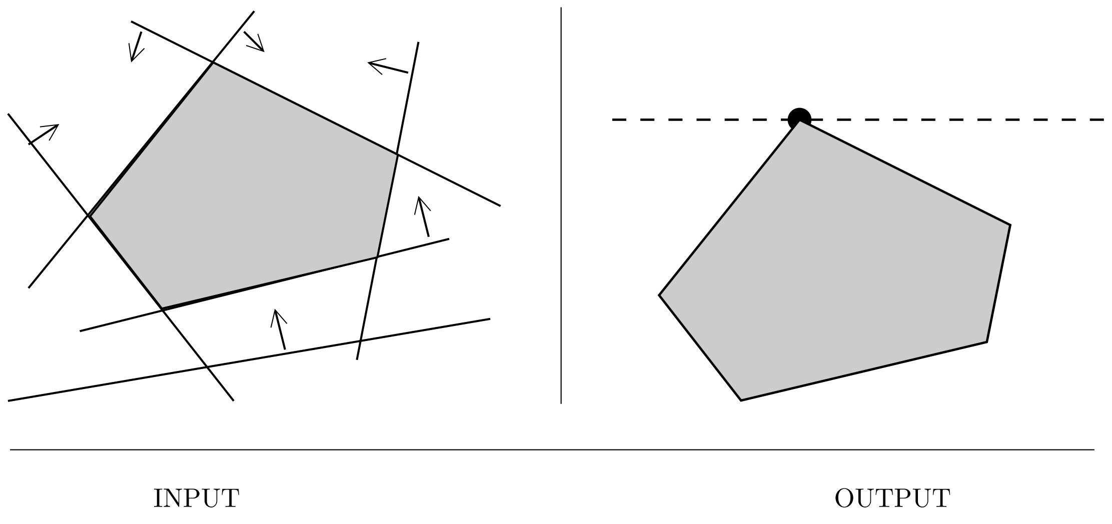

13.6 Linear Programming

Input description: A set S of n linear inequalities on m variables

$$S_i := \sum_{j=1}^{m} c_{ij} \cdot x_j \geq b_i, \ 1 \leq i \leq n$$

and a linear optimization function $f(X) = \sum_{j=1}^{m} c_j \cdot x_j$.

Problem description: Which variable assignment X' maximizes the objective function f while satisfying all inequalities S?

Discussion: Linear programming is the most important problem in mathematical optimization and operations research. Applications include:

- *Resource allocation* – We seek to invest a given amount of money to maximize our return. Often our possible options, payoffs, and expenses can be expressed as a system of linear inequalities such that we seek to maximize our possible profit given the constraints. Very large linear programming problems are routinely solved by airlines and other corporations.

- *Approximating the solution of inconsistent equations* – A set of m linear equations on n variables x_i, $1 \leq i \leq n$ is overdetermined if $m > n$. Such overdetermined systems are often *inconsistent*, meaning that there does not exist an assignment to the variables that simultaneously solves all the equations. To find the assignment that best fits the equations, we can replace each variable x_i by $x'_i + \epsilon_i$ and solve the new system as a linear program, minimizing the sum of the error terms.

- *Graph algorithms* – Many of the graph problems described in this book, including shortest path, bipartite matching, and network flow, can be solved as special cases of linear programming. Most of the rest, including traveling salesman, set cover, and knapsack, can be solved using *integer linear programming.*

The *simplex method* is the standard algorithm for linear programming. Each constraint in a linear programming problem acts like a knife that carves away a region from the space of possible solutions. We seek the point within the remaining region that maximizes (or minimizes) $f(X)$. By appropriately rotating the solution space, the optimal point can always be made to be the highest point in the region. The region (simplex) formed by the intersection of a set of linear constraints is convex, so unless we are at the top there is always a higher vertex neighboring any starting point. When we cannot find a higher neighbor to walk to, we have reached the optimal solution.

While the simplex algorithm is not too complex, there is considerable art to producing an efficient implementation capable of solving large linear programs. Large programs tend to be sparse (meaning that most inequalities use few variables), so sophisticated data structures must be used. There are issues of numerical stability and robustness, as well as choosing which neighbor we should walk to next (so-called *pivoting rules*). There also exist sophisticated *interior-point* methods, which cut through the interior of the simplex instead of walking along the outside, and beat simplex in many applications.

The bottom line on linear programming is this: you are much better off using an existing LP code than writing your own. Further, you are probably better off paying money than surfing the Web. Linear programming is an algorithmic problem of such economic importance that commercial implementations are far superior to free versions.

Issues that arise in linear programming include:

- *Do any variables have integrality constraints?* – It is impossible to send 6.54 airplanes from New York to Washington each business day, even if that value maximizes profit according to your model. Such variables often have natural integrality constraints. A linear program is called an *integer program* if all its variables have integrality constraints, or a *mixed integer program* if some of them do.

  Unfortunately, it is NP-complete to solve integer or mixed programs to optimality. However, there are integer programming techniques that work reasonably well in practice. *Cutting plane techniques* solve the problem first as a linear program, and then add extra constraints to enforce integrality around the optimal solution point before solving it again. After sufficiently many iterations, the optimum point of the resulting linear program matches that of the original integer program. As with most exponential-time algorithms, run times for integer programming depend upon the difficulty of the problem instance and are unpredictable.

- *Do I have more variables or constraints?* – Any linear program with m variables and n inequalities can be written as an equivalent *dual* linear program with n variables and m inequalities. This is important to know, because the running time of a solver might be quite different on the two formulations. In general, linear programs (LPs) with much more variables than constraints should be solved directly. If there are many more constraints than variables, it is usually better to solve the dual LP or (equivalently) apply the dual simplex method to the primal LP.

- *What if my optimization function or constraints are not linear?* – In least-squares curve fitting, we seek the line that best approximates a set of points by minimizing the sum of squares of the distance between each point and the line. In formulating this as a mathematical program, the natural objective function is no longer linear, but quadratic. Although fast algorithms exist for least squares fitting, general *quadratic programming* is NP-complete.

  There are three possible courses of action when you must solve a nonlinear program. The best is if you can model it in some other way, as is the case with least-squares fitting. The second is to try to track down special codes for quadratic programming. Finally, you can model your problem as a constrained or unconstrained optimization problem and try to solve it with the codes discussed in Section 13.5 (page 407).

- *What if my model does not match the input format of my LP solver?* – Many linear programming implementations accept models only in so-called *standard form*, where all variables are constrained to be nonnegative, the object function must be minimized, and all constraints must be equalities (instead of inequalities).

  Do not fear. There exist standard transformations to map arbitrary LP models into standard form. To convert a maximization problem to a minimization one, simply multiply each coefficient of the objective function by -1. The remaining problems can be solved by adding *slack variables* to the model. See any textbook on linear programming for details. Modeling languages such as AMPC can provide a nice interface to your solver and deal with these issues for you.

Implementations: The USENET Frequently Asked Question (FAQ) list is a very useful resource on solving linear programs. In particular, it provides a list of available codes with descriptions of experiences. Check it out at *http://www-unix.mcs.anl.gov/otc/Guide/faq/*.

There are at least three reasonable choices in free LP-solvers. `Lp_solve`, written in ANSI C by Michel Berkelaar, can also handle integer and mixed-integer problems. It is available at *http://lpsolve.sourceforge.net/5.5/*, and a substantial user community exists. The simplex solver `CLP` produced under the Computational Infrastructure for Operations Research is available (with other optimization software) at *http://www.coin-or.org/*. Finally, the GNU Linear Programming Kit (GLPK) is intended for solving large-scale linear programming, mixed integer programming (MIP), and other related problems. It is available at *http://www.gnu.org/software/glpk/*. In recent benchmarks among free codes (see *http://plato.asu.edu/bench.html*), `CLP` appeared to be fastest on on linear programming problems and `lp_solve` on mixed integer problems.

NEOS (Network-Enabled Optimization System) provides an opportunity to solve your problem on computers and software at Argonne National Laboratory. Linear programming and unconstrained optimization are both supported. This is worth checking out at *http://www.mcs.anl.gov/home/otc/Server/* if you need an answer instead of a program.

Algorithm 551 [Abd80] and Algorithm 552 [BR80] of the *Collected Algorithms of the ACM* are simplex-based codes for solving overdetermined systems of linear equations in Fortran. See Section 19.1.5 (page 659) for details.

Notes: The need for optimization via linear programming arose in logistics problems in World War II. The simplex algorithm was invented by George Danzig in 1947 [Dan63]. Klee and Minty [KM72] proved that the simplex algorithm is exponential in worst case, but it is very efficient in practice.

Smoothed analysis measures the complexity of algorithms assuming that their inputs are subject to small amounts of random noise. Carefully constructed worst-case instances for many problems break down under such perturbations. Spielman and Teng [ST04] used smoothed analysis to explain the efficiency of simplex in practice. Recently, Kelner and Spielman developed a randomized simplex algorithm running in polynomial time [KS05b].

Khachian's ellipsoid algorithm [Kha79] first proved that linear programming was polynomial in 1979. Karmarkar's algorithm [Kar84] is an interior-point method that has proven to be both a theoretical and practical improvement of the ellipsoid algorithm, as well as a challenge for the simplex method. Good expositions on the simplex and ellipsoid algorithms for linear programming include [Chv83, Gas03, MG06].

Semidefinite programming deals with optimization problems over symmetric positive semidefinite matrix variables, with linear cost function and linear constraints. Important special cases include linear programming and convex quadratic programming with convex quadratic constraints. Semidefinite programming and its applications to combinatorial optimization problems are surveyed in [Goe97, VB96].

Linear programming is P-complete under log-space reductions [DLR79]. This makes it unlikely to have an NC parallel algorithm, where a problem is in NC iff it can be solved on a PRAM in polylogarithmic time using a polynomial number of processors. Any problem that is P-complete under log-space reduction cannot be in NC unless P=NC. See [GHR95] for a thorough exposition of the theory of P-completeness, including an extensive list of P-complete problems.

Related Problems: Constrained and unconstrained optimization (see page 407), network flow (see page 509).