14.1. LINEAR PROGRAMMING

Many combinatorial parameters are solutions to optimization problems; we seek the largest or smallest value of some function of the input structure. Sometimes there is a natural pairing between a maximization problem and a minimization problem, perhaps even a theorem saying that the solutions always have the same value. In this chapter we explore the theory of such min-max relations.

The general linear programming problem has real variables, linear constraints, and linear objective. Combinatorial problems often restrict variables to integer values. Nevertheless, often we can benefit from phrasing the integer problems as linear programs. We begin with a class of integer programs.

PACKING AND COVERING

Many combinatorial optimization problems (almost all, according to Füredi [1991]) can be phrased as packing or covering in hypergraphs.

14.1.1. DEFINITION. A packing in a hypergraph $H$ is a set of vertices with no two in an edge. A covering of $H$ is a set of edges with union $V(H)$. The packing problem and covering problem for $H$ are the problems of finding a largest packing and a smallest covering.

14.1.2. Example. Dilworth’s Theorem. The chain hypergraph of a poset $P$ is the hypergraph whose vertices are the elements of $P$ and whose edges are the subsets of this set that form chains. The solutions to the packing problem are the maximum antichains in $P$. The solutions to the covering problem are the minimum chain covers. Dilworth’s Theorem states that the optimal solutions to these two problems have the same size.

14.1.3. Example. Hypergraph coloring. Optimal coloring of a hypergraph $H$ is covering its vertex set by a minimum number of sets that contain no edge. Such sets can be viewed as edges in a related hypergraph $H'$, and the coloring on $H$ is the covering problem on $H'$. In Lemma 12.2.13 we expressed poset dimension as a hypergraph coloring problem. When $H$ is a graph, $H'$ becomes the hypergraph of independent sets, and the packing problem in $H'$ is the maximum clique problem in $H$.

In a hypergraph, every covering is as large as every packing, because the elements of a packing must be covered by distinct edges. Called weak duality, this property holds in a more general setting of linear optimization problems with real variables. We begin by encoding the paired packing and covering problems as numerical optimization problems. This generalizes Example 14.1.2, where packing is maximum antichain and covering is minimum chain covering.

Associate a variable $x_j$ with each element $j$ of $V(H)$ and a variable $y_i$ with each element $C_i$ of $E(H)$. The values are restricted to $\{0, 1\}$, corresponding to using or not using elements. The objective is to maximize $\sum x_j$ for the packing problem and to minimize $\sum y_i$ for the covering problem. Additional constraints enforce the chosen elements to yield packings and coverings. For each edge $C_i$, we require $\sum_{j \in C_i} x_j \leq 1$ to avoid capturing more than one element from the edge. To cover the element $j$, we require $\sum_{i: j \in C_i} y_i \geq 1$. The result appears below.

<table>
<thead>
<tr>
<th>Packing problem</th>
<th>Covering problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize $\sum x_j$ subject to $\sum_{j \in C_i} x_j \leq 1$ for all $i$</td>
<td>minimize $\sum y_i$ subject to $\sum_{i: j \in C_i} y_i \geq 1$ for all $j$</td>
</tr>
<tr>
<td>$x_j \in {0, 1}$</td>
<td>$y_i \in {0, 1}$</td>
</tr>
</tbody>
</table>

The maximum chain and minimum antichain covering problems fit this framework using the antichain hypergraph. Often there is a second pair of dual programs arising in this way from a related hypergraph that reverses the role of packings and edges. For example, the packings in the chain hypergraph are antichains, and the packings in the antichain hypergraph are chains. Below we list several examples of this. The lower group more or less reverses the roles of edges and packings from the upper group.

<table>
<thead>
<tr>
<th>Packing problem</th>
<th>Covering problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximize $\sum x_j$ subject to $\sum_{j \in C_i} y_i \geq 1$ for all $i$</td>
<td>minimize $\sum y_i$ subject to $\sum_{i: j \in C_i} x_j \leq 1$ for all $j$</td>
</tr>
<tr>
<td>$x_j \in {0, 1}$</td>
<td>$y_i \in {0, 1}$</td>
</tr>
</tbody>
</table>

Section 14.1: Linear Programming
The constraints of such optimization problems have a compact matrix notation. Let $A$ denote the matrix of the incidence relation between vertices and edges: $a_{i,j} = 1$ if $i \in C_j$, and otherwise $a_{i,j} = 0$. We also use 1 and 0 to denote column vectors of appropriate length whose entries are all 1 or all 0, and when we write an inequality between vectors we mean that the inequality holds in each coordinate. Relaxing the constraints to allow nonnegative integer variables does not change the problem; since $A$ is a 0,1-matrix, no variable will exceed 1 in an optimal solution.

A packing problem can be generalized by weighting the vertices or by loosening the constraints. In the first case, we have a weighted packing problem, in which element $i$ has weight $c_i$, and we want the maximum total weight of a packing. To maintain weak duality, we want each packing to establish a lower bound for the dual problem, so we require element $i$ to be covered $c_i$ times. We substitute $\max c^T x$ for the objective function of the packing problem and $y^T A \geq c^T$ for the constraints in the dual, and the dual becomes the multi-covering problem.

Similarly, if we relax constraint $j$ to allow $b_j$ elements to be used from edge $C_j$, we obtain the multi-packing problem. To maintain weak duality, we count $b_j$ in the cost of a covering when we use $C_j$, since that is the portion of a packing we may cover with $C_j$; the result is the weighted covering problem.

### 14.1.4. Example. Multi-packing and weighted packing problems

The maximum $k$-family problem of Section 11.1 is a multi-packing problem, with variables restricted to $\{0, 1\}$. We allow $k$ elements from a chain, but not $k$ copies of the same element. For the $j$th chain, we set $b_j = \min\{k, |C_j|\}$. The Greene–Kleitman Theorem (proved in Section 11.1 and in Section 14.2) states that the maximum value of a multi-packing equals the minimum value of a weighted covering.

The $b$-matching problem is another multi-packing problem. We specify capacities $b_i$ for the vertices of a graph, and we seek a maximum set of adjacencies having at most $b_i$ incident to vertex $i$, for each $i$.

Given a hypergraph $H$ with vertex weights $\{c_i\}$, the weighted packing problem asks for the maximum total weight of an edge in the antiblocker $H^\ast$. Maximum weighted matching in graphs has this form.
A linear program is **feasible** if some setting for its variables satisfies all constraints. We will see that dual linear programs have the same optimal value when both are feasible. This yields a min-max relation for a class of integer problems if the linear relaxations always have optimal solutions in integers. The discussion is facilitated by a **canonical tableau** format for linear programs.

### CANONICAL TABLEAU FOR DUAL LINEAR PROGRAMS

<table>
<thead>
<tr>
<th></th>
<th>(x^T)</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y)</td>
<td>(A)</td>
<td>(b)</td>
</tr>
<tr>
<td>-1</td>
<td>(c^T)</td>
<td>(d)</td>
</tr>
<tr>
<td></td>
<td>(v^T)</td>
<td>(= f)</td>
</tr>
<tr>
<td></td>
<td>(= g)</td>
<td></td>
</tr>
</tbody>
</table>

We henceforth refer to two dual linear programs as the **max problem** and the **min problem**. The constraints for the max problem appear in the rows: here \(Ax \leq b\) has become \(Ax - b \leq 0\) and then \(Ax - b = -u\), where 

\[
u = \text{a vector of slack variables required to be nonnegative. We treat these as additional variables, even though } x \text{ determines } u.\]

Similarly, the constraints for the dual min problem appear in the columns; we write the constraints as \(y^TA - c^T = v^T\) with nonnegative slack variables \(v\). The box marked “\(d\)” is an additive constant for the objective functions: maximize \(f = c^Tx - d\) and minimize \(g = y^Tb - d\).

#### 14.1.6. DEFINITION.

A choice of values for the variables of a canonical linear program (max or min) is a **solution** if it satisfies the matrix of constraints \((Ax - b = -u\) or \(y^TA - c = v\)). A solution is **feasible** if all variables \((x, u, y, v)\) are nonnegative.

The main result of linear programming is the existence of optimal solutions with equal value to dual pairs of feasible programs. Incorporating infeasibility yields the “Theorem of the Four Alternatives”; each of the dual programs may be feasible or not. When just one is infeasible, the theorem states that the objective function for the other is unbounded.

The case when both programs are feasible is of the most interest (this always holds for packing/covering problems) and is known as the **Duality Theorem of Linear Programming**. Instead of a precise proof of this theorem, here we sketch geometric intuition for why it is true.

Geometric interpretation for systems of linear inequalities often provides insight into algebraic statements. A linear equality constraint on a vector \(x \in \mathbb{R}^n\) confines it to a hyperplane. A linear inequality constraint confines it to a halfspace bounded by that hyperplane. The constraints for the max problem (including \(x \geq 0\)) thus define a feasible region that is an intersection of halfspaces. An intersection of halfspaces is a polyhedron. The intersection of \(k\) hyperplanes in \(\mathbb{R}^n\) has dimension \(n - k\) when nonempty; it is the set of solutions to \(k\) linear equations.

#### 14.1.7. THEOREM. (Duality Theorem; von Neumann [1947], Gale–Kuhn–Tucker [1951])

When dual linear programs both have feasible solutions, they have optimal solutions with equal value.

**Proof:** (sketch of geometric intuition!). Consider a max problem with \(n\) variables and \(m\) constraints, where \(m \geq n\). (When \(m < n\), we can exchange the two programs by negation or exchange max and min in the discussion.)

Let \(P\) be the polyhedron of feasible points: \(P = \{x \in \mathbb{R}^n; Ax \leq b\}\). Maximizing \(c^Tx\) means finding a feasible point farthest in the direction \(c^T\) in \(\mathbb{R}^n\). If the dual problem is feasible, then weak duality implies that the max problem is bounded. Hence there is such a farthest point \(\hat{x}\).

We extend into \(n\) dimensions the intuition that comes from the 2-dimensional picture below. We cannot move \(\hat{x}\) farther in the direction \(c^T\) because we are limited by some of the constraints. If \(c^T\) is not perpendicular to any of the bounding halfplanes, then inability to slide along these bounding halfplanes to increase \(c^Tx\) requires that there are \(n\) bounding halfplanes whose intersection is the single point \(\hat{x}\).

Hence \(\hat{x}\) satisfies the matrix equation formed by the corresponding \(n\) rows of \(Ax = b\); index these as the first \(n\) rows. Let the \(i\)th row be \(a_i\), so

\[
a_1^T\hat{x} = b_1, \quad a_2^T\hat{x} = b_2, \quad \ldots, \quad a_n^T\hat{x} = b_n.
\]
Chapter 14: Hypergraphs and Integer Programming

Section 14.1: Linear Programming

14.1.9. LEMMA. Complementary solutions to dual linear programs in canonical form are optimal and have the same value.

Proof: Complementary slackness forces a 0 into each corresponding pair \( \{u_i, y_i\} \) or \( \{v_j, x_j\} \). When \( u_i = 0 \), we have row \( a_i^T x = b_i \). Thus, in the inequality \( y^T A x \leq y^T b \), the \( i \)th coordinate contributes 0 to each side if \( y_i = 0 \), and it contributes \( y_i b_i \) to each side if \( y_i \neq 0 \). The contribution to the dot product is the same on both sides, and equality holds for the sum. Similarly, \( c^T x = y^T Ax \). Hence we have feasible solutions with the same value \( (c^T x = y^T b) \). By weak duality, they are optimal.

In a canonical tableau for dual linear programs, we can set \( x = 0 \) and \( y = 0 \) to achieve complementary slackness. We then obtain \( u = b \) and \( v = -c \). This is fine if \( b \geq 0 \) and \( c \leq 0 \). If some coordinate of \( b \) is negative, then this solution to the max problem is infeasible. If some coordinate of \( c \) is positive, then this solution to the min problem is infeasible.

14.1.10. REMARK. The aim of the Simplex Algorithm. In the canonical tableau for dual packing/covering problems, setting \( x = 0 \) for the packing problem is feasible, but \( y = 0 \) for the covering problem is not. Instead of \( y = 0 \), we must “use” some edges to cover elements. To allow this, we will rewrite the constraint equations so that when some other variables are set to 0 we obtain nonzero values for some of \( y \).

Meanwhile, in the packing problem we are starting at \( x = 0 \), which is the worst feasible point when \( c \) is all-1. To move to a better vertex of the feasible polyhedron, we want to introduce some other constraint to the set determining the feasible point (and eliminate one to maintain a set of size \( n \)), doing so to move in the direction of \( c \).

Say we make constraint \( r \) tight and drop the tightness of \( x_r = 0 \). The geometric discussion suggests that \( y_r \) can now be nonzero, along with \( x_r \), while the variables \( u_r \) and \( v_r \) will become 0. To determine what values \( x_r \) and \( y_r \) will have in the resulting solution vectors, in the max problem we want to rewrite the equations to solve for \( \{u_{1}, \ldots, u_{r-1}, x_r, v_{r+1}, \ldots, v_n\} \) in terms of \( \{x_{1}, \ldots, x_{r-1}, u_{r}, x_{r+1}, \ldots, x_m\} \). In the min problem, we want to solve for \( \{v_{1}, \ldots, v_{r-1}, y_r, v_{r+1}, \ldots, v_m\} \) in terms of \( \{y_{1}, \ldots, y_{r-1}, v_r, y_{r+1}, \ldots, y_n\} \).

Magically, both happen together, by the same computation, producing a new tableau, and the pairs of variables for complementary slackness remain the same. The operation of exchanging \( y_r \) with \( v_r \) and \( x_r \) with \( u_r \) in the set of variables to be “solved for” is called a pivot on position \((r, s)\).

14.1.11. DEFINITION. In a canonical tableau expressing the equations \( Ax - b = -u \) and \( y^T A - c = v \), the vectors of variables \( u \) and \( v \) are the basic variables; the equations are solved for these variables in terms of \( x \) and \( y \).
of the others. Setting the nonbasic variables \( x \) and \( y \) to all 0 yields the basic solution for this tableau.

14.1.12. REMARK. Searching for complementary solutions. When the equations are rewritten by pivoting to obtain \( A'x' - b' = -u' \) and \( y'A' - c' = v' \), the variables that have moved into the positions of \( u' \) and \( v' \) are now the basic variables for the new tableau. The new equations still impose the same constraints; we are merely rewriting the equations. The aim in doing so is to reach a tableau \( T' \) where both basic solutions are feasible (that is, \( b' \geq 0 \) and \( c' \leq 0 \)). Since \( T' \) merely rewrites the equations of the original tableau \( T \), the basic feasible solutions for \( T' \) are feasible solutions also for \( T \). Since basic solutions for a tableau satisfy complementary slackness, Lemma 14.1.9 implies that these solutions are optimal for the programs in \( T' \), which are the same problems expressed in \( T \).

It is time to state the computation that a pivot performs to rewrite the equations and exchange the variables.

14.1.13. LEMMA. Consider dual linear programs \( Ax - b = -u \) and \( y^T A - c = v \) in canonical form. A pivot step on position \((r,s)\) can be performed when \( a_{r,s} \neq 0 \). It exchanges the positions of \( x_r \) and \( u_r \) to make the new tableau and simultaneously exchanges \( s_r \) and \( y_r \) to form a new tableau with variables \((x',u',y',v')\) where the pairs of complementary variables remain the same. If \( A \) has \( m \) rows and \( n \) columns and we write \( a_{m+1,j} = c_j \) and \( d_{i,n+1} = b_i \), then the new entries \( A', c', b' \) after the pivot are

\[
\begin{align*}
\quad a'_{r,s} &= \frac{1}{a_{r,s}} \\
\quad a'_{r,j} &= \frac{a_{r,j}}{a_{r,s}} \\
\quad a'_{i,s} &= \frac{-a_{i,s}}{a_{r,s}} \quad \text{for } i \neq r, \\
\quad a'_{i,j} &= a_{i,j} - \frac{a_{i,s} a_{r,j}}{a_{r,s}} \quad \text{when } i \neq r \quad \text{and } j \neq s.
\end{align*}
\]

Proof: The old equations are \( Ax - b = -u \) and \( y^T A - c = v \). To put \( x_r \) into the basis, the \( r \)th row equation must be solved for \( x_r \). This yields the new row \( r \) as described; note that \( u_r \) becomes an input in the new expression, moving it into \( x' \). The other new row equations are obtained by substituting the resulting expression for \( x_r \) into those equations, thereby expressing the other basic variables in terms of \( x' \).

\[
\begin{pmatrix}
  a_{r,s} & a_{r,j} \\
  a_{i,s} & a_{i,j}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  a'_{r,s} & a'_{r,j} \\
  a'_{i,s} & a'_{i,j}
\end{pmatrix} = \begin{pmatrix}
  \frac{1}{a_{r,s}} & \frac{a_{r,j}}{a_{r,s}} \\
  \frac{-a_{i,s}}{a_{r,s}} & \frac{-a_{i,s} a_{r,j}}{a_{r,s}}
\end{pmatrix}
\]

Alternatively, solving the \( s \)th column equation for \( y_s \), and substituting this expression for \( y_s \) into the other column equations yields exactly the same formulas for the new tableau. Hence the pivot step performs both exchanges as claimed.

The Simplex Algorithm uses pivoting to move through successive tableaus (or successive bases, or successive sets of \( n \) hyperplanes) in search of complementary solutions. There may be exponentially many bases; we don't want to examine them all. We seek pivots likely to move us toward a good basis. In the packing/covering case, where we start with a feasible solution for the max problem, we choose pivots that maintain its feasibility \((b' \geq 0)\) while increasing the value of the objective function.

14.1.14. COROLLARY. Consider a canonical tableau with \( b \geq 0 \). If \( b_r > 0 \), \( c_r > 0 \), and \( b_r/a_{r,s} = \min(b_r/a_{r,s}; a_{r,s} > 0) \), then pivoting at position \((r,s)\) maintains basic feasibility of the max problem and increases the value of its basic feasible solution.

Proof: By Lemma 14.1.13, the amount subtracted from \( b_r \) is \( b_r a_{r,s}/a_{r,s} \) for \( i \neq r \), which is at most \( b_r \) by the choice of \( r \). Hence the new tableau is basic feasible for the max program.

The value of the basic solution to the max problem of the current tableau is \(-d\). Since \(-d\) is decreased by the positive number \( b_r c_r/a_{r,s} \), performing the pivot increases the value of the basic solution.

14.1.15. REMARK. Further ideas in the Simplex Algorithm. Using pivots as in Corollary 14.1.14 to seek an optimum by working through successive feasible solutions for the max problem is called the primal simplex algorithm. Since we are increasing the value of the basic feasible solution for the max problem (we are moving to points where some of the \( x \) variables are positive), we are moving more in the direction of \( c^T \). We can continue improving as long as we find pivots satisfying the hypotheses.

Meanwhile, what happens in the dual min problem? We seek a final tableau with \( b' \geq 0 \) and \( c' \leq 0 \), to have complementary solutions. The conditions \( b_r > 0 \), \( c_r > 0 \), and \( b_r/a_{r,s} > 0 \) imply that \( c'_r < 0 \), by Lemma 14.11.13, so we seem to move toward basic feasibility of the min problem. Unfortunately, since the amount subtracted from \( c_j \) is \( c_j a_{r,j}/a_{r,s} \) and \( a_{r,j} \) may be negative, it is possible that other entries of the bottom row may turn positive.

We have hope. If the dual min problem is actually feasible, then weak duality implies that we cannot make the objective value of a feasible solution to the max problem arbitrarily large.

What happens if the max problem is feasible, the min is not, and no pivot satisfies the conditions of Corollary 14.1.14? The tableau would then be as on the left below, indicating immediately that the min problem is infeasible and the max problem is unbounded.
The dual case of min infeasibility is on the right; this never occurs when starting with \( b \geq 0 \). These are two of the possibilities in the Theorem of the Four Alternatives; a tableau can also indicate infeasibility of both problems. (Ability to pivoting at position \((r,s)\) requires only \(a_{r,s} \neq 0\), not feasibility of either problem.)

Although pivoting cannot grow the value arbitrarily high in the Primal Simplex Algorithm when the dual is feasible, perhaps it could increase infinitely many times, gaining less each time but never terminating. This cannot happen because there are only finitely many bases.

Even though there are only finitely many bases, there is a difficulty. Pivoting may produce a 0 in the right column. A later pivot in a row \( r \) with \( b_r = 0 \) would not increase the value of the max program. This raises the danger of cycling through basis choices. The algorithm needs a rule for choosing pivots in this degenerate situation that allows it to proceed and avoid cycling.

14.1.16. Example. **Degeneracy - Cycles under simple pivot choice rules.** Natural choices for the pivot column would be to select the first (or the largest) positive entry in \( c \), and then pick the first (or largest) positive entry in the column as the pivot entry. Kuhn and Tucker provided the following example where these rules produce a pivot cycle of length 11, with all pivot values during the cycle equaling 1.

<table>
<thead>
<tr>
<th>( z_1 )</th>
<th>( z_2 )</th>
<th>( z_3 )</th>
<th>( z_4 )</th>
<th>−1</th>
</tr>
</thead>
<tbody>
<tr>
<td>−12.5</td>
<td>−2</td>
<td>12.5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1*</td>
<td>0.24</td>
<td>−2</td>
<td>−0.24</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1.92</td>
<td>(−1)*</td>
<td>−0.96</td>
<td>0</td>
</tr>
</tbody>
</table>

A lexicographic rule was proved to avoid cycling in Dantzig-Orden-Wolfe [1955]. Bland [1977] found another such rule, simpler. Furthermore, if the initial tableau is not basic feasible for the max problem, another use of Bland’s Rule produces pivots that lead to a tableau with \( b \geq 0 \) (or exhibits infeasibility of the max problem by reaching a tableau like that on the right above).

14.1.17. **DEFINITION.** Bland’s rule for pivot choices. Index the variables \( x_1, \ldots, x_n, u_1, \ldots, u_m \) for the max program and \( v_1, \ldots, v_n, y_1, \ldots, y_m \) for the min program as \( z_1, \ldots, z_{m+n} \) and \( w_1, \ldots, w_{m+n} \), respectively. Among \( \{j: c_j > 0\} \), choose the pivot column \( s \) whose original index is smallest. Among \( \{i: a_{i,s} > 0 \text{ and } b_i/a_{i,s} \text{ attains its minimum}\} \), choose the pivot row \( r \) whose original index is smallest.

For both column and row, we are selecting the variable to enter or leave the basis that has the smallest original index among the eligible variables, where eligible means that the exchange preserves feasibility of the basic solution to the max program.

14.1.18. **THEOREM.** (Bland [1976]) Beginning with a tableau in which the basic solution to the max program is feasible, the primal simplex algorithm with Bland’s rule for selecting pivots concludes after finitely many pivot steps with complementary basic feasible solutions or with a tableau guaranteeing that the max program is unbounded and the min program is infeasible.

**Proof:** We have observed that there are finitely many basis choices, and the pivots performed never decrease the value of the basic solution. We can always choose a pivot, except in the stated stopping conditions. Termination can fail only by having a cycle among pivots where \( b_i = 0 \) and the value of the solution \( −d \) remains unchanged. Let \( q \) be the maximum index of the variables that enter and leave the basis during this cycle of pivots. To obtain a contradiction, we examine a pair of solutions obtained from the tableaus when \( z_q \) enters and leaves the basis.

Because \( Ax−b = −u \) and \( y^TA−c = v \), solutions \( z = x : u \) and \( w = v : y \) satisfy \( w^Tz = y^TAx − c^Tx − y^TAx + y^Tb = g − f \), where \( f = c^Tx − d \) and \( g = y^Tb − d \). This holds whether \( w, z \) are feasible or not and is A.W. Tucker’s **Duality Equation** of Linear Programming.

We may restrict our attention to the subproblem \( A^* \) consisting only of the columns and rows corresponding to variables that enter and leave the basis during the cycle, since the others won’t affect the entries in \( A^* \). Each row of \( A^* \) must have a pivot during the cycle, so \( b \) is 0 in these rows and \( d \) is constant throughout the cycle. Suppose that \( z_q \) leaves the basis when \( z_p \) is introduced, pivoting on position \( i, j \) with \( c_{ij} > 0 \). Let \( \hat{z} \) be the solution obtained from the tableau before the pivot nonbasic variables to 0, except
308 Chapter 14: Hypergraphs and Integer Programming

Section 14.1: Linear Programming

We now have a method for choosing pivots that solves every linear program with finitely many steps. It is the **Anticycling Simplex Method**. It produces complementary solutions if both programs are feasible, thereby proving the duality equation.

The canonical tableau is ideal for discussing duality. An alternative “standard” tableau is better for computation, because one can avoid recomputing the entire tableau at each pivot (some authors switch the two names).

In practice or in expectation (Borgwardt [1982]) the Simplex Algorithm runs quickly, but it is not a polynomial-time algorithm. There are instances with $n$ variables and $2n$ constraints where (with Bland’s Rule) it makes nearly $2^n$ pivots (Klee–Minty [1972]; Avis–Chvátal [1978]; see Papadimitriou–Steiglitz [1982, p.??], Chvátal [1983, p.253ff]). The “ellipsoid method” (Khachiyan [1980]) runs in polynomial time but is inefficient in practice. It does not require an explicit statement of the LP (which may have exponentially many constraints), just a quick test for feasibility of solution vectors. This leads to proofs of polynomial complexity for many combinatorial optimization problems (see Korte–Viggen [2002]).

“Interior point methods”, starting with Karmarkar [1984], search for improvement via the interior of the feasible polytope. They are useful in practice. Ellipsoid and interior point methods also extend to more general convex (“semidefinite”) programming problems; this is beyond our scope.

Our brief sketch here does not fairly state the importance of computational methods for solving linear programs. In 2006, a search for “Simplex Algorithm” on MathSciNet returned 95 books. In the January/February issue of *Computing in Science and Engineering*, an article by Jack Dongarra and Francis Sullivan named the Simplex Algorithm one of the “Top Ten Algorithms of the Century”. Meanwhile, when the algorithms by Khachian and Karmarkar were announced, they were front-page news in both *The New York Times* and *The Wall Street Journal*.

**Farkas’ Lemma and Second Neighborhoods**

Although our motivation in introducing linear programs was the study of integer programs, we postpone to Section 14.3 the discussion of conditions on linear programs that guarantee integer solutions. In the remainder of this section we present applications of linear programming duality.

The Duality Theorem of Linear Programming (Theorem 14.1.7) is equivalent to various statements about systems of linear equations. One is Farkas’ Lemma, which itself has various forms (Exercises 5–8). We state

$$z_p = -1.$$ Because $b = 0$, the resulting values for the basic variables are the entries in column $j$. Because Bland’s rule chooses the smallest eligible index for the pivot row, we have $\hat{z}_t \leq 0$ for all $t < q$, whether $z_t$ is in the basis or not. The resulting value of $f$ is $-c_j - d$.

Now consider the tableau just before $z_q$ returns to the basis, when $\hat{w}_q$ and $w_q$ are in column $l$ and there is about to be a pivot in position $k,l$. Let $\hat{w}_i$ be the basic solution for the min program in this tableau; the nonbasic variables have values $-c_r$. By Bland’s choice of the pivot column, $\hat{w}_i \geq 0$ for all $t < q$. The resulting value of $g$ is $-d$, as always for a basic solution.

By the duality equation, $\hat{w}^T \hat{z} = g - f = -d - (-c_j - d) = c_j > 0$. Recall that $q$ was defined to be the maximum index of variable involved in the cycle, so $q$ is the maximum index of variables in $A^*$. Hence the properties of $\hat{w}$ and $\hat{z}$ we have derived imply

$$\hat{w}^T \hat{z} = \sum_{k < q} \hat{w}_k \hat{z}_k + \hat{w}_q \hat{z}_q \leq \hat{w}_q \hat{z}_q = a_i,j(-c_j) < 0.$$ 

This contradiction forbids cycles.

It may be that the basic solution for the initial tableau is not feasible for the max program; that is, some $b_i$ is negative. Another application of Bland’s rule (“make the eligible choice whose index has the least value”) allows us to extend the algorithm by making pivots to reach a tableau where the max program is basic feasible ($b \geq 0$), if possible.

**14.1.19. THEOREM.** Pivot choices by Bland’s rule lead to a tableau with a basic feasible solution for the max program or a tableau proving infeasibility for the max program. When $b \geq 0$ does not hold, choose the row $r$ with $b_r < 0$ for which the original index of the corresponding variables is minimized. Among the positions in this row such that $a_{r,i} < 0$, choose the pivot column $s$ for which the original index of the variables is minimized.

**Proof:** In studying the feasibility of the max program, the answer and the sequence of pivots will be the same if we set $c = 0$. Now interchange the roles of the max program and min program (and variable names). The new max program is max $-b^T y + d$ such that $-A^T y \leq 0$. Here the max program is feasible, and the sequence of pivots performed by applying the previous algorithm to it is the “transpose” ( interchange rows and columns) of the sequence of pivots performed by the algorithm here on the original tableau. In particular, if at some step there is no $i$ with $a_{s,i} < 0$, then row $r$ has $A_r \geq 0$ and $b_r < 0$, which indicates that the max program is infeasible and the min program is unbounded; at the corresponding moment the transposed algorithm finds an unbounded max program.
the common form used in our first application of linear programming duality. It is another Theorem of Alternatives. First we extend the canonical expression of linear programming problems to incorporate equality constraints and unrestricted variables in a cleaner way.

14.1.20. Lemma. A linear programming problem may involve any of the following: inequality constraints, equality constraints, nonnegative variables, and unrestricted variables. To obtain the dual program with the transposed matrix of constraints, the corresponding dual variables and constraints are restricted in the following way:

<table>
<thead>
<tr>
<th>primal</th>
<th>dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>inequality constraint</td>
<td>nonnegative variable</td>
</tr>
<tr>
<td>nonnegative variable</td>
<td>inequality constraint</td>
</tr>
<tr>
<td>equality constraint</td>
<td>unrestricted variable</td>
</tr>
<tr>
<td>unrestricted variable</td>
<td>equality constraint</td>
</tr>
</tbody>
</table>

Proof: Recall that we model equality constraints in a max problem by two inequalities: \( a^T x = b \), becomes \( a^T x \leq b \), and \( -a^T x \leq -b \). In the canonical form, these inequalities yield two nonnegative dual variables, \( y_i^+ \) and \( y_i^- \). The \( j \)th dual constraint contains the expression \( y_j^+ a_{i,j} - y_j^- a_{i,j} \), which equals \( (y_j^+ - y_j^-) a_{i,j} \). Although \( y_j^+ \) and \( y_j^- \) are restricted to be nonnegative, the difference \( y_j^+ - y_j^- \) may have any value. Hence we obtain an equivalent statement of the dual by having an unrestricted variable (having any sign) as the dual variable corresponding to an equality constraint. The geometric explanation in Theorem 14.1.7 also extends to this setting.

Similarly, we model an unrestricted variable \( x_j \) as the difference of two nonnegative variables. This yields two inequality constraints in the dual that can be written as a single equality constraint \( \sum y_i a_{i,j} = c_j \).

Analogous arguments apply for the dual of a min problem.

14.1.21. Theorem. (Farkas’ Lemma; Farkas [1894], Minkowski [1896])

For an \( m \)-by-\( n \) matrix \( A \) and vector \( b \in \mathbb{R}^m \), either \( Ax = b \) has a nonnegative solution, or \( y^T A \geq 0^T \) has a solution with \( y^T b < 0 \); not both.

Proof: Consider the max problem with constraints \( Ax = b \), nonnegative \( x \in \mathbb{R}^n \), and objective function \( c^T x \), where \( c^T \) is the vector 0 in \( \mathbb{R}^n \). The statement of the dual is minimization \( y^T b \) such that \( y^T A \geq 0^T \), where \( y^T \in \mathbb{R}^m \) is a vector of unrestricted variables.

If the max problem is feasible, then its value is 0, since \( c^T = 0 \). In that case, weak duality yields \( y^T b \geq 0^T \) whenever \( y^T A \geq 0^T \). If the max problem is infeasible, then by the Duality Theorem the min problem is unbounded, which yields a solution to \( y^T b \geq 0^T \) with \( y^T b < 0 \).

Farkas’ Lemma can be proved directly using “Fourier–Motzkin elimination”, developed by Fourier in the 1820s for solution of linear inequalities. Kuhn [1956] discussed these connections; see also Chvátal [1983], Schrijver [1986, pp.155-157], and Ziegler [1995].

We apply Farkas’ Lemma to a special case of a conjecture on digraphs.

14.1.22. Conjecture. (Second Neighborhood Conjecture; Seymour) If \( G \) is a simple digraph with no loop or 2-cycle, then \( G \) has a vertex \( v \) such that \( |N^2(v)| \geq |N^+(v)| \), where \( N^+(v) \) is the set of (immediate) successors of \( v \), and \( N^2(v) \) denotes the set of vertices whose shortest path from \( v \) takes two steps. Call such a vertex expansive.■

Various partial results for general graphs appear in Kaneko–Locke [2001], Chen–Shen–Yuster [2003], and Godbole–Cohn–Wright [2007]. The restriction to tournaments (orientations of complete graphs) was known as Dean’s Conjecture (see Dean–Latka [1995]). Dean’s Conjecture was proved by Fisher [1996] using Farkas’ Lemma and more combinatorially by Havet–Thomasse [2000] (using median orders).

Fisher approached Dean’s Conjecture by generalizing the notion of a “loser”; in any digraph, a vertex of outdegree 0 is expansive (see Exercise 10 for other small degrees). A density \( f \) on a digraph \( D \) is a nonnegative vertex weighting with sum 1. For \( S \subseteq V(D) \), let \( f(S) = \sum_{v \in S} f(v) \). A losing density is a density \( f \) such that \( f(N^+(v)) \geq f(N^-(v)) \) for every vertex \( v \). Intuitively, a random vertex \( u \) picked using \( f \) is at least as likely to lose to any given vertex \( v \) (via \( v \rightarrow u \)) as to beat it (via \( u \rightarrow v \)). Note that weight 1 on a vertex of outdegree 0 is a losing density. Once we interpret the definition as a system of inequalities, Farkas’ Lemma provides a losing density.

14.1.23. Theorem. (Fisher [1996]) Every digraph \( D \) has a losing density. Also, if \( f(v) > 0 \) for a losing density \( f \), then \( f(N^+(v)) = f(N^-(v)) \).

Proof: Let \( n = |V(D)| \). Let \( A \) be the adjacency matrix of \( D \). That is, \( a_{i,j} = 1 \) if \( i \neq j \) and \( a_{i,i} = 0 \) if \( 1 \leq i \leq n \); otherwise the entries are 0. Let \( I_0 \) denote the all-1 and all-0 vectors, respectively, and let \( f \) be the \( n \)-by-\( n \) identity matrix.

The statement that \( D \) has a losing density \( f \) (writing \( f \) as a vector) is the statement that \( -A f + \theta \) has a nonnegative solution. The bottom row states that the weights sum to 1, and the row for vertex \( v \) states that \( -f(N^+(v)) + f(N^-(v)) + z(v) = 0 \), which becomes \( f(N^+(v)) \geq f(N^-(v)) \) when \( z \) is nonnegative. Furthermore, if \( f(N^+(v)) \geq f(N^-(v)) \), then \( z(v) \) can be chosen to complete the solution.

If this matrix equation has no nonnegative solution, then by Farkas’ Lemma there is a solution to \( (g^T w) \geq (0^T 0^T) \) with \( (g^T w)(1) < 0 \).
The second condition states that \( w < 0 \). The latter columns of the matrix inequality state that \( g^T \geq 0^T \), and the first column prevents \( g^T = 0^T \) since \( w < 0 \). Hence we can normalize so that \( \sum g(v) = 1 \) to treat \( g \) as a density. Now the column for \( v \) states that \( -g(N^-(v)) + g(N^+(v)) + w = 0 \), which becomes \( g(N^+(v)) > g(N^-(v)) \) since \( w < 0 \). Hence \( g \) is a losing density.

Now let \( f \) be a losing density, so \( -Af \geq 0 \). Thus \( f(v)(Af)(u) \leq 0 \) for each \( v \). On the other hand, since \( -A = A^T \), we have \( f^T(-A)f = f^T Af \), and hence \( f^T Af = 0 \). Since \( f^T Af \) is the sum of the nonpositive terms \( f(v)(Af)(v) \), we conclude that \( f(v) = 0 \) or \( (Af)(v) = 0 \) for every \( v \). In particular, \( f(v) > 0 \) implies \( f(N^+(v)) = f(N^-(v)) \).

The problem now is to use a losing density to find an expansive vertex. Fisher did this for tournaments by studying the expected outdegree. The remainder of the proof does not involve Farkas’ Lemma but is a nice application of expectations over vertex weightings. Meanwhile, the moral of our story is that when a property of a structure can be expressed by linear equations (or inequalities; see Exercises 5–9), Farkas’ Lemma or its variants can yield a characterization theorem for when the property holds.

14.1.24. LEMMA. The expected outdegree of a vertex chosen from a density \( f \) on a digraph \( D \) is \( \sum_{v \in V(D)} f(N^-(v)) \).

**Proof:** The number of times the expectation counts \( f(v) \) is \( d^+(v) \), once for each edge leaving \( v \). That is, for every edge in \( D \), we count the weight of the source vertex. This is what the sum counts.

14.1.25. LEMMA. If \( f \) is a losing density on a tournament \( T \), then \( f(N^-_T(v)) \geq 2f(N^+_T(v)) \) for all \( v \in V(T) \).

**Proof:** Here \( T^2 \) denotes the square of \( T \) in the usual digraph sense; \( uv \in E(T^2) \) if \( d_T(u, v) \leq 2 \). Thus \( N^-_T(v) \subseteq N^-_{T^2}(v) \). Let \( S = N^-_{T^2}(v) \).

Since \( f \) is a losing density, \( f(N^-_T(v)) \leq \frac{1}{2} \). Hence we may assume that \( f(S) < 1 \). Let \( Q \) be the subtournament obtained by deleting \( S \) from \( T \); note that \( v \in V(Q) \) and that \( f(V(Q)) > 0 \). Since \( f(N^-_T(v)) = \sum_{x \in V(Q), f(x) = 0} f(x) f(N^-_Q(x)) = \sum_{x \in V(Q), f(z) = 0} f(x) f(N^-_Q(x)) \), we can compare the terms for which \( f(x) > 0 \) in the two outer sums to obtain a vertex \( u \in V(Q) \) such that \( f(u) > 0 \) and \( f(N^-_Q(u)) \geq f(N^+_Q(u)) \).

Since \( u \notin S \) and \( T \) is a tournament, \( N^-_T(v) \subseteq N^-_T(u) \) and \( N^+_T(v) \subseteq N^+_T(u) \). Thus \( N^-_T(v) \subseteq S \cap N^-_T(u) \) and \( N^+_T(v) \cap S \subseteq S - N^-_T(v) \). Under \( f \), we obtain

\[
\begin{align*}
    f(N^-_T(u)) &= f(N^-_Q(u)) + f(N^-_T(u) \cap S) \geq f(N^-_Q(u)) + f(N^-_T(v)), \\
    f(N^+_T(u)) &= f(N^+_Q(u)) + f(N^+_T(u) \cap S) \leq f(N^+_Q(u)) + f(S - N^-_T(v)).
\end{align*}
\]

Since \( f(u) > 0 \), by Theorem 14.1.23 the terms on the left are equal. Thus \( f(N^-_Q(u)) + f(S - N^-_T(v)) \geq f(N^-_Q(u)) + f(N^-_T(v)) \). By the choice of \( u \), we have \( f(S - N^-_T(v)) \geq f(N^-_T(v)) \), and finally we apply \( N^-_T(v) \subseteq S \).

14.1.26. THEOREM. Every tournament \( T \) has a losing density \( f \). Under \( f \), the expected outdegree is at least twice as big in \( T^2 \) as in \( T \). In particular, the Second Neighborhood Conjecture is true for \( T \).

**Proof:** Summing the result of Lemma 14.1.25 over all \( v \) and applying Lemma 14.1.24 to both sides yields the claim about the expected outdegree. By the Pigeonhole Property of the expectation, there is then a vertex \( v \) in \( T \) with \( d^+_{T^2}(v) \geq 2d^+_T(v) \), and hence \( |N^2(v)| \geq |N^+(v)| \).

**SHANNON CAPACITY AND STRONG PRODUCTS**

Our next application of linear programming duality consider the problem of sending messages using an alphabet in which some pairs of letters are confusable. Only nonconfusable messages are allowed; we want to find a large set of these. We create a graph \( G \) whose vertices are the letters of the alphabet, with edges between confusable pairs of letters. The largest set of nonconfusable letters is the largest stable set in this graph.

Now consider messages of length \( k \). One message is confusable with another if for all \( i \) the \( i \)-th letters in the messages are confusable or identical. Since we can treat the letters of each position independently, we can obtain a nonconfusable set of size at least \( \alpha(G)^k \). However, we may be able to do better, as shown below. In each case, we create a graph of confusable words and look for the largest stable set. What makes the problem interesting is that the graph for longer messages can be obtained from the original graph via a natural product operation.

14.1.27. Example. Confusion graphs. In \( G \) below, two-letter words do better than the square of one-letter words, using the nonconfusable set \{aa, ec, cd, be, db\}. In the graph of \( H^2 \), the broken edges “wrap around”.
14.1.28. DEFINITION. The strong product $G \cdot H$ of two graphs $G$ and $H$ is the graph with vertex set $V(G) \times V(H)$ defined by $(x, y) \leftrightarrow (x', y')$ in $G \cdot H$ if and only if $(x = x'$ or $xx' \in E(G))$ and $(y = y'$ or $yy' \in E(H))$.

The largest set of nonconfusable words of length $k$ is the largest independent set in the strong product $G^k$ of $k$ copies of $G$. We seek some measure of the “capacity” of an alphabet, which we think of as the “effective number of nonconfusable letters” in the alphabet. The Shannon capacity $\Theta(G)$ of a graph $G$ (the confusion graph) is $\sup(\alpha(G^k))^{1/k}$. The values increase with $k$ but are bounded by the number of vertices in $G$. Thus we seek the limit.

To improve the upper bound on $\Theta(G)$, consider the weighted independent set problem. An independent set uses at most one vertex from each clique. This constraint can be relaxed by using a value between 0 and 1 for every vertex and maximizing the overall total value such that the total value on each clique is at most 1. Letting $w$ be the vector of variables, this yields a linear program

$$\alpha^*(G) = \max \ 1^T w \text{ such that } Aw \leq 1,$$

where $A$ is the clique-vertex incidence matrix. Since each independent set yields a feasible solution to this problem, we have $\alpha(G) \leq \alpha^*(G)$. The optimal value $\alpha^*(G)$ is the fractional independence number of $G$.

For the independence number, the natural dual minimization problem is the clique covering problem. The clique cover number $\theta(G)$ is the minimum number of cliques in $G$ that cover $V(G)$. For the linear relaxation of the clique covering problem, we obtain

$$\theta^*(G) = \min q^T 1 \text{ such that } q^T A \geq 1_n.$$

Again the relaxation argument yields $\theta(G) \geq \theta^*(G)$, and linear programming duality yields $\alpha^*(G) = \theta^*(G)$. We now have all the relations below except one, which says that we can bound $\Theta(G)$ by solving the linear program for $\alpha^*(G)$. The proof follows from other results later.

14.1.29. LEMMA. (Shannon [1956]) $\alpha(G) \leq \Theta(G) \leq \alpha^*(G) = \theta^*(G) \leq \theta(G)$. 

14.1.30. Example. For $C_5$, we can set $w = 1/2$ for each vertex and $q = 1/2$ for each clique. Then each clique contributes total weight 1 in the max program and each vertex receives total cover 1 in the min program, so both are feasible and we have $\Theta(C_5) \leq \alpha^*(C_5) = 5/2$.

The best bounds known for $\Theta(C_5)$ remained $\sqrt{5}$ and $5/2$ for 23 years, until Lovász proved that the lower bound is the correct value. Lovász obtained a better general upper bound on $\Theta(G)$ using linear algebra. For $C_5$, the improved bound equals $\sqrt{5}$.

The linear algebra involves embedding the graph in Euclidean space in a clever way. The representation is fairly natural, but the immediacy of the bound on the capacity is quite astonishing. For the remainder of this discussion, we denote vertices by integers and use $u, v, x, y$ to denote vectors.

14.1.31. DEFINITION. An orthonormal representation of $G$ is an assignment of unit vectors $\{u_1, u_2, \ldots, u_n\}$ to the vertices of $G$ such that $i \leftrightarrow j$ implies that $u_i \cdot u_j = 0$.

The definition does not mention the dimension of the space containing the representation. Certainly we need never more than $n$ dimensions, because we could assign each vertex an element of an orthonormal basis.

To build representations for $G \cdot H$, we use the tensor product. For vectors $x, y$ of dimensions $r, s$, the tensor product $x \circ y$ is defined by $x \circ y = (x_1y_1, \ldots, x_1y_s, x_2y_1, \ldots, x_2y_s)$. In other words, we record the rows in order from the matrix product $xy^T$. When the numbers of coordinates agree, dot products of tensor products obey a simple formula.

14.1.32. LEMMA. $(x \circ y)^T (u \circ v) = (x^T u)(y^T v)$.

Proof: $(x \circ y)^T (u \circ v) = \sum_i \sum_j (x_i y_j)(u_i v_j) = \sum_i x_i u_i \sum_j y_j v_j = (x^T u)(y^T v)$.

We have no further need to mention individual coordinates of vectors, so we henceforth use subscripts to indicate members of a set of vectors.

14.1.33. LEMMA. If $G$ and $H$ have orthonormal representations $\{u_i\}$ and $\{v_j\}$, then $G \cdot H$ has an orthonormal representation $\{u_i \circ v_j\}$.

Proof: Suppose $(i, j) \leftrightarrow (k, l)$ in $G \cdot H$. By Lemma 14.1.32, $(u_i \circ v_j)^T (u_k \circ v_l) = (u_i^T u_k)(v_j^T v_l)$. But $(i, j) \leftrightarrow (k, l)$ means $i \leftrightarrow j$ or $k \leftrightarrow l$, so at least one of these factors is 0.
14.1.34. DEFINITION. Given an orthonormal representation \( \{u_i\} \), the cost of a unit vector \( c \) is \( \max \frac{1}{(c \cdot u_i)^2} \). A vector of minimum cost is a handle of the representation; this is a unit vector that is as far as possible from being orthogonal to any vector in the representation. The cost of a handle is the value of the representation. A representation with minimum value is an optimal representation, and its value \( \vartheta(G) \) is the Lovász theta function of \( G \).

Lovász proved that \( \vartheta(G) \) is an upper bound on \( \Theta(G) \). For the proof, we need to know how \( \vartheta(G) \) behaves with respect to independent sets and strong products.

14.1.35. LEMMA. \( \vartheta(G) \geq \alpha(G) \).

Proof: Let \( \{u_i\} \) be an optimal representation with handle \( c \) and value \( \vartheta(G) \). Let \( u_1, \ldots, u_k \) be the vectors corresponding to a maximal stable set; these vectors are orthonormal. Project \( c \) on the space spanned by \( u_1, \ldots, u_k \); this yields \( \sum_{i=1}^k (c \cdot u_i)u_i \). The length of this vector cannot exceed the length of \( c \) itself, so we have
\[
1 = c \cdot c \geq \sum_{i=1}^k (c \cdot u_i)u_i \sum_{i=1}^k (c \cdot u_i)u_i = \sum_{i=1}^k (c \cdot u_i)^2.
\]
Applying this to \( \vartheta(G) \) yields
\[
\vartheta(G) \geq \vartheta(G) \sum_{i=1}^k (c \cdot u_i)^2 = \frac{\sum_{i=1}^k (c \cdot u_i)^2}{\min_{i \in [n]} (c \cdot u_i)^2} \geq k = \alpha(G).
\]
The behavior of \( \vartheta \) under strong products is opposite to that of \( \alpha(G) \).

14.1.36. LEMMA. \( \vartheta(G \cdot H) \leq \vartheta(G)\vartheta(H) \).

Proof: Let \( \{u_i\} \) and \( \{v_j\} \) be optimal representations for \( G \) and \( H \) with handles \( c \) and \( d \). The tensor product of two unit vectors is a unit vector, so we can use \( c \circ d \) and the product representation \( \{u_i \circ v_j\} \) (Lemma 14.1.33) to obtain an upper bound on \( \vartheta(G) \). Combining this with Lemma 14.1.32 and the definitions, we have
\[
\vartheta(G \cdot H) \leq \max_{i,j} \frac{1}{[(c \circ d) \cdot (u_i \circ v_j)]^2} = \max_{i,j} \frac{1}{[(c^T u_i)(d^T v_j)]^2} \leq \max_i \frac{1}{(c \cdot u_i)^2} \max_j \frac{1}{(d \cdot v_j)^2} = \vartheta(G)\vartheta(H).
\]

14.1.37. THEOREM. (Lovász [1979]) \( \Theta(G) \leq \vartheta(G) \).

Proof: By Lemmas 14.1.35–14.1.36, \( \alpha(G^k) \leq \vartheta(G^k) \leq [\vartheta(G)]^k \). Hence \( [\alpha(G^k)]^{1/k} \leq \vartheta(G) \) for every \( k \). Therefore \( \vartheta(G) \geq \sup(\alpha(G^k))^{1/k} = \Theta(G) \).

14.1.38. COROLLARY. \( \Theta(C_5) = \sqrt{5} \).

Proof: It suffices to show \( \vartheta(C_5) \leq \sqrt{5} \). Consider a set of choices for \( \{u_i\} \) in \( R_3 \) that behaves like an umbrella. The handle \( c \) of the umbrella is \((0,0,-1)\). The ribs \( \{u_i\} \) are equally spaced. When the umbrella is closed, all \( u_i \) is in the \( x, y \)-plane, the angle between them is \( 72^\circ \), and the angle between vectors representing nonadjacent vertices is \( 104^\circ \). Somewhere before the umbrella opens completely, there is an orthogonal representation. Its handle is \((0,0,-1)\). At this point, the Spherical Cosine Theorem yields \( c \cdot u_i = 5^{1/4} \). Therefore, \( \vartheta(C_5) = \max_i (c \cdot u_i)^2 = 5^{1/2} \).

14.1.39. REMARK. We have not yet shown that \( \Theta(G) \leq \alpha^*(G) \). Proving that \( \vartheta(G) \leq \alpha^*(G) \) shows this and also proves that \( \vartheta(G) \) is a better upper bound than \( \alpha^*(G) \) on \( \Theta(G) \).

The proof relies on eigenvalues of matrices. Lovász showed that \( \vartheta(G) = \min_A \lambda_1(A) \), where \( A \) ranges over all symmetric matrices with \( a_{i,j} = 1 \) if \( i = j \) or \( i \leftrightarrow j \), and \( \lambda_1(A) \) is the largest eigenvalue of \( A \). If we set \( a_{i,j} = 0 \) when \( i \leftrightarrow j \), then we obtain the adjacency matrix \( A(G) \), where \( A(H) \) is the adjacency matrix of \( H \), and hence \( \vartheta(G) \leq 1 + \lambda_1(A(G)) \). From the result of Wilf [1967] that \( \chi(H) \leq 1 + \lambda_1(A(H)) \) (see Chapter 10), this is also an upper bound on \( \vartheta(G) \), but that does not prove the desired inequality.

14.1.40. REMARK. We close the discussion of \( \vartheta \) by stating several other known results.

1. If \( G \) is vertex transitive, then \( \vartheta(G) \vartheta(G) = n \).
2. The minimum dimension in which \( G \) has an orthonormal representation is at least \( \vartheta(G) \).
3. Finally, the most important: \( \vartheta(G) = \max_{B \subseteq B} \sum_i \sum_j b_{i,j} \), where \( B \) ranges over the convex set of all matrices such that \( \text{Trace}(B) = 1 \) and \( b_{i,j} = 0 \) if \( i \leftrightarrow j \). The importance of this is in expressing \( \vartheta(G) \) as the solution to a convex programming problem. This, along with the ellipsoid algorithm for convex programming, yields the existence of a polynomial-time algorithm for computing the independence number in perfect graphs (Grotschel–Lovász–Schrijver [1981]), although the problem is NP-complete for general graphs. The proof uses that \( \alpha(G) = \alpha^*(G) \) when \( G \) is a perfect graph. Computing \( \alpha^*(G) \) is not enough to yield a polynomial algorithm, since this linear program may have an exponential number of constraints. However, since \( \alpha(G) = \alpha^*(G) \) for a perfect graph, and \( \alpha(G) \leq \vartheta(G) \leq \alpha^*(G) \) for any graph, the desired polynomial-time algorithm is obtained.
\[ \alpha^*(G) \] always, computing \( \vartheta(G) \) yields \( \alpha(G) \) when \( G \) is perfect. The convex programming problem given here for \( \vartheta(G) \) is small enough to yield a polynomial time algorithm.

**Matrix Games**

On each play of a two-person zero-sum game, each player chooses some option, and the chosen options determine a payoff from one player to the other. Neither player knows what the other is choosing, but both know all the strategies and payoffs. Our players are \( P \) and \( Q \).

**14.1.41. Example. The Odd/Even Finger Game.** On each play of the game, each player shows 1 or 2 fingers. One player pays the other \( k \) dollars, where \( k \) is the total number of fingers showing. If \( k \) is even, then \( P \) pays \( Q \). If \( k \) is odd, then \( Q \) pays \( P \). The payoffs from \( Q \) to \( P \) appear in the matrix below for the four possible outcomes of each play of the game.

\[
\begin{array}{ccc}
\text{P shows 1} & \text{Q shows 2} \\
\text{P shows 1} & -2 & +3 \\
\text{P shows 2} & +3 & -4 \\
\end{array}
\]

Although this seems like a fair game, in fact it favors \( P \). If \( P \) always shows 1 or always shows 2, then \( Q \) can use this information to win. Hence \( P \) should show 1 sometimes and 2 sometimes, perhaps showing 1 with probability \( x \) and 2 with probability \( 1-x \). Player \( Q \) may know the strategy \( x \), but \( Q \) does not know the random number determining a particular play.

Knowing \( x \), player \( Q \) can compute the expectations of the columns and play only the one with the smaller expectation. Hence the expected amount that \( x \) guarantees for \( P \) is the minimum of the two column expectations. These are \(-2x + 3(1-x)\) and \(3x - 4(1-x)\), which simplify to \(3 - 5x\) and \(7x - 4\); \( P \) wants to maximize the minimum of these. Since one decreases with \( x \) and the other increases, the minimum is maximized when they are equal; \(3 - 5x = 7x - 4\) yields \( x = 7/12 \). By choosing \( x = 7/12 \), \( P \) guarantees an average payoff per game of at least \( 1/12 \).

Player \( Q \) can keep \( P \) from winning more than \( 1/12 \) on average. Player \( Q \) plays column 1 with probability \( y \), guaranteeing that the expected payoff to \( P \) is at most \( \min(2y + 3(1-y), 3y - 4(1-y)) \). The minimum occurs when the two formulas are equal, at \( y = 7/12 \).

**14.1.42. Definition.** Every real \( m \)-by-\( n \) matrix \( A \) defines a matrix game in which the rows index the options for player \( P \), the columns index the options for player \( Q \), and the payoff from \( Q \) to \( P \) if \( P \) chooses option \( i \) and \( Q \) chooses option \( j \) is \( a_{ij} \). The matrix \( A \) is the payoff matrix; the available choices are pure strategies. A mixed strategy for a player is a probability distribution over the pure strategies.

The Minimax Theorem of matrix games states that the maximum gain that \( P \) can guarantee equals the minimum loss that \( Q \) can guarantee; these two values are the left and right sides of the equality stated below. This theorem was proved much earlier than the development of linear programming, but linear programming duality provides a short proof.

\[
\max_p \min_q p^T A q = v = \min_q \max_p p^T A q.
\]

**14.1.43. Theorem.** (Minimax Theorem of Matrix Games; von Neumann [1928]) Given a matrix game with payoff matrix \( A \), there exist optimal strategies \( p^* \) and \( q^* \) and a constant \( v \) (the value of the game) such that \((p^*)^T A \geq v 1_n\) and \(A q^* \leq v 1^T_m\).

**Proof:** When \( P \) chooses a strategy \( p \), the expected outcome for \( Q \) will be a convex combination of the column expectations, which lie in the vector \( p^T A \). That is, \( P \) can guarantee gaining \( v_1 \) using strategy \( p \) if and only if \( p^T A \geq v_1 1 \). The problem for \( P \) is thus to maximize \( v_1 \) such that a strategy \( p \) can be chosen to achieve these inequalities.

Similarly, the problem for \( Q \) is to minimize \( v_2 \) such that a strategy \( q \) can be chosen to achieve \( A q \leq v_2 \). We express these as linear programs. Showing that they are dual programs proves the desired equality.

To put the programs in canonical form, treat \( p, q, v_1, v_2 \) as variables. \( P \) wants to minimize \(-v_1 \) subject to \( p^T A - v_1 1 \geq 0 \) and \( \sum p_i = 1 \). Similarly, \( Q \) wants to maximize \(-v_2 \) subject to \( A q - v_2 1 \leq 0 \) and \( \sum q_j = 1 \). These programs exhibit duality with the constraint matrix \((A^T 1)\). The nonnegative variables \( p \) correspond to the inequality constraints for the rows of \( A \), and the variable \(-v_1 \) is unrestricted, since it corresponds to the equality constraint \( \sum q_j = 1 \).

Both programs are feasible, using any probability vector, and hence there is an optimal solution, which implies the desired equality.
We next define a matrix game based on a graph. Let $G$ be a connected weighted multigraph in which edge $e$ has positive weight $w(e)$. The edge player $P$ picks an edge $e$, and the tree player $Q$ picks a spanning tree $T$. The payoff $c(T, e)$ to the edge player is 0 if $e \in T$ and is $c(T, e)/w(e)$ if $e \notin T$, where $c(T, e)$ is the total weight of the cycle formed by adding $e$ to $T$. Let $Val(G, w)$ be the value of this tree-edge game to the edge player $P$. We write this as $Val(G)$ in the unweighted case (every edge has weight 1).

**14.1.44. PROPOSITION.** If $G$ is an $n$-vertex graph, then $Val(G, w) \leq n$.

**Proof:** We claim that $n$ is an upper bound on the payoffs in the column corresponding to a minimum spanning tree $T$. The weights of edges in $T$ that lie in the path in $T$ joining the endpoints of $e$ are at most $w(e)$, else we can substitute $e$ for one of these to obtain a cheaper spanning tree. Because the length of the cycle is at most $n$, we obtain $c(T, e) \leq nw(e)$. Hence the tree player can use a minimum spanning tree $T$ as a pure strategy to establish this bound. 

To obtain a lower bound on the value of the game in the worst case, we consider 4-regular graphs with large girth. There exists a sequence of 4-regular graphs whose girth grows at least logarithmically with $n$. Even in the unweighted case, such graphs have $n + 1$ edges with cost at least $c \log n$, and by playing all edges with equal probability the edge player can guarantee $Val(G) > c/2 \log n$.

**14.1.45. Example.** For the unweighted complete graph, $Val(K_n) = 3 - 6/n$. If the edge player chooses uniformly among the edges, then every tree has probability $2/n$ of payoff 0 and probability $(n - 2)/n$ of a nonzero payoff. The minimum payoff is the length of the shortest cycle, which is at least 3. Therefore the expected payoff is at least $3(n - 2)/n = 3 - 6/n$. Equality holds only for the $n$-vertex stars, because other trees have longer fundamental cycles. If the tree player chooses uniformly among the $n$ stars, then every edge has probability $2/n$ of payoff 0 and probability $(n - 2)/n$ of payoff 3, with expected payoff $3 - 6/n$. Hence uniform edge selection and uniform star selection are optimal strategies, and $Val(K_n) = 3 - 6/n$.

**14.1.46. Example.** For every weight function $w$, $Val(C_n, w) = 1$. Let $T_i$ be the tree that omits edge $i$, and let $w_i$ be the weight on edge $i$, with $W = \sum w_i$. If the tree player assigns probability $p_i = w_i/W$ to $T_i$, then every edge has expected payoff 1. If the edge player assigns probability $w_i/W$ to edge $i$, then every tree has expected payoff 1. Hence $Val(G, w) = 1$.

We apply the tree-edge game to a problem in computer science.

**14.1.47. Example.** The $k$-Server Problem. Consider a metric space $M$ with $k$ servers to handle service requests. A request is processed by moving a server to the location where the request occurs. The cost of serving a sequence of requests is the total distance the servers move. Given initial positions $\pi$ and request sequence $\rho$, let $OPT(\pi, \rho)$ denote the optimal offline cost; this is the cheapest way to process the requests, given that the entire sequence of requests is known in advance. Given an on-line algorithm $A$, let $A(\pi, \rho)$ denote its cost on the input $(\pi, \rho)$. An on-line algorithm is $c$-competitive if $A(\pi, \rho) \leq c \cdot OPT(\pi, \rho) + a$ for all inputs $(\pi, \rho)$.

If $|M| \geq k$ and $c < k$, then there is no $c$-competitive deterministic on-line algorithm (Manasse–McGeoch–Slater [1990]). Bounded competitiveness is always achievable, but not with the greedy algorithm. If we alternate requests at two points close to one of the initial positions, then the optimal off-line cost is just the cost of moving two servers to those points, but the greedy algorithm continues to move one server back and forth, with unbounded cost.

When $|M| = k + 1$, there is a simple $k$-competitive algorithm: do nothing if the requested point is occupied. Otherwise, move the server such that adding this move to its previous moves gives the least total cost (Manasse–McGeoch–Slater [1990]). This algorithm is not $k$-competitive if $|M| > k + 1$. In general, it is open whether a $k$-competitive algorithm always exists. A general algorithm presented in Chrobak–Larmore [1992] was shown to be $(2k - 1)$-competitive in Koutsoupias–Papadimitriou [1994].

We consider the $k$-server problem for a metric space on a road network. Points occur all along the roads, and $d(x, y)$ is the minimum road distance from $x$ to $y$. Chrobak and Larmore [1991] developed a $k$-competitive deterministic algorithm for the $k$-server problem on a road network whose graph is a tree (edge weights encode length). To obtain a competitive algorithm for a general network, we allow the additional flexibility of randomization.

**14.1.48. DEFINITION.** A randomized algorithm uses random bits to guide its choices, making $A(\pi, \rho)$ a random variable. An oblivious adversary specifies the entire input (request sequence $\rho$) in advance, not knowing the results of any choices by the algorithm. An adaptive adversary may specify the next request based on the server’s choices. A randomized on-line algorithm $A$ is $c$-competitive (against an oblivious adversary) if $E(A(\pi, \rho)) \leq c \cdot OPT(\pi, \rho) + a$ for each input $(\pi, \rho)$.

For deterministic algorithms, there is no difference between the two adversaries, because the adversary knows in advance what the algorithm will do against each initial sequence.
14.1.49. THEOREM. When a road network $M$ is modeled by $(G, w)$, there is a $k(1 + \text{Val}(G, w))$-competitive randomized on-line algorithm for the $k$-server problem on $M$ against an oblivious adversary.

Proof: We define such an algorithm $A$. First use the optimal strategy for the tree player in the tree-edge game on $(G, w)$ to select a spanning tree $T$. For each edge $e \notin T$, choose a random point $x_e$ as a cut. This produces a tree-like network $G'$. Process $(\pi, \rho)$ using the $k$-competitive Chrobak–Larmore algorithm on $G'$.

If $OPT'(\pi, \rho)$ is the optimal off-line cost on $G'$ for this input, then $A(\pi, \rho) \leq kOPT'(\pi, \rho)$ for each $(\pi, \rho)$ and each experimental outcome $G'$. Taking the expectation on each side over the optimal strategy for the game yields $E(A(\pi, \rho)) \leq kE(OPT'(\pi, \rho))$. We show that $E(OPT'(\pi, \rho)) \leq (1 + \text{Val}(G, w))OPT(\pi, \rho)$; this part involves no on-line algorithms.

To prove the inequality, we use $G'$ to simulate the moves for the optimal off-line scheme achieving $OPT(\pi, \rho)$ on $G$. When the scheme on $G$ would cross the roadblock at $x_e$ on edge $e$, we traverse the cycle in $T + e$ to reach the other side. The cost of this detour is $c(T, e)$. Given $T$ chosen with probability $q(T)$, let $d(e)$ denote the total distance traveled on edge $e$ by the optimal scheme for $(\pi, \rho)$; note that $OPT(\pi, \rho) = \sum_e d(e)$.

If $e \notin T$, then the expected number of times the scheme crosses the random point $x_e$ on $e$ is $d(e)/w(e)$. Hence the expected total cost of detours caused by edge $e$ is $d(e)c(T, e)/w(e)$ if $e \notin T$, otherwise 0. The expected cost of all detours, over all experiments, is

$$\sum_T \sum_e q(T) d(e) c(T, e) = OPT(\pi, \rho) \sum_T \sum_e q(T) \frac{d(e)}{OPT} c(T, e) = OPT(\pi, \rho) \text{Val}(G, w).$$

Here the inequality holds because $\sum_e d(e) = OPT$ implies that $\frac{d(e)}{OPT}$ is a possible distribution for the edge player. Since $q$ is an optimal strategy for the tree player, the expected loss against this edge strategy is at most the loss limit $\text{Val}(G, w)$.

To bound the total expected cost for this simulation, we add this bound on the detours to $OPT(\pi, \rho)$. Since $E(OPT'(\pi, \rho))$ is at most the expected cost using this particular simulation, we obtain $E(OPT'(\pi, \rho)) \leq (1 + \text{Val}(G, W))OPT(\pi, \rho)$, as desired.

The asymptotic worst-case growth of $\text{Val}(G)$ for unweighted $n$-vertex multigraphs remains unknown. We mentioned the lower bound of $\Omega(\log n)$, and there is an upper bound of $e' \sqrt{\log e \log \log n}$ (Alon–Karp–Peleg–West [1995]). More recently, an upper bound of $O((\log n \log \log n)^2)$ was obtained in Elkin–Spielman–Teng [2005].

Section 14.1: Linear Programming

**DIRECT COMBINATORIAL APPLICATIONS**

Combinatorial applications of linear programming duality have the following flavor. We seek to maximize or minimize some parameter, such as the "size" of a combinatorial structure or the number of operations needed to accomplish a task. Meanwhile, some constraints can be expressed as linear inequalities. The solution of a linear program can then bound the size. If that bound can be achieved, then the linear program shows that the structural achieving it is optimal.

Packing and covering problems are well-behaved examples of this in two respects. First, the structures are completely described by the linear inequalities (modulo integrality constraints). Second, the dual problem is also a natural combinatorial optimization problem. In many applications of linear programming duality, these properties do not hold. In particular, numerical feasible solutions for the variables in the linear program (even integral feasible solutions) need not be realizable as instances of the combinatorial structure. Nevertheless, a numerical bound on the optimal value of the structures is provided by any feasible solution to the dual problem.

We provide two examples of this technique. The first is from Gates–Papadimitriou [1979], the only mathematical paper by Bill Gates.

14.1.50. Example. The Pancake Problem. Writing as “Harry Dweighter”, Jacob Goodman [1975] described a harried waiter with a stack of pancakes. To avoid disaster, he wants to sort them in order by size. He can lift a top portion of the stack, invert it, and replace it. The problem is to sort by such **prefix reversals**, also called “flips”.

Let $f(n)$ be the number of flips needed to sort a stack of $n$ pancakes, in the worst case. By successively flipping the next largest pancake to the top of the stack and then down on top of the larger ones already sorted, sorting can be done in at most $2n - 3$ flips, so $f(n) \leq 2n - 3$ for $n \geq 2$. Gates–Papadimitriou [1975] proved that $f(n) \leq (5n + 5)/3$, and this upper bound has not been improved in general. They also showed that $f(n) \geq 17n/16$, which was later improved to $15n/14$ (Heydari–Sudborough [1997]).

The upper bound is proved by providing an algorithm and using a linear program to prove that it never takes more than $(5n + 5)/3$ steps. We use two measures of the current permutation. An adjacency consists of two neighboring elements with consecutive sizes. A **block** is a maximal nonempty consecutive string of adjacencies. An adjacency can be gained by flipping the top element to have a desirable neighbor, unless doing so breaks an adjacency. In particular, the bottom element of a block is **buried**; if its desired neighbor is the top of the stack, we cannot make them adjacent in one flip without breaking an adjacency.
There are many variations on the pancake problem; most are unsolved. More flips may be needed in the “burnt pancake problem”, where each pancake must have the burnt side down in the final stack. This is the problem of sorting “signed permutations” by prefix reversal (Cohen–Blum [1995]; see also Heydari–Sudborough [1997], Hannenhalli–Pevzner [1995], and Kamplan–Shamir–Tarjan [1997]). Problems using fewer moves result from more general cutting and pasting operations; see Aigner–West [1987], Eriksson et al. [2001], and Cranston–Sudborough–West [2007].

14.1.51. ALGORITHM. (Sorting by prefix reversal). We define several operations that consist of several flips and increase the number of adjacencies. Given an element \( i \), let \( \bar{i} \) denote \( i + 1 \) or \( i - 1 \), as appropriate, with addition modulo \( n \), and let \( i \) denote the value next to \( i \) in the other direction from \( \bar{i} \). We use the language of permutations (horizontal) instead of pancake stacks (vertical).

Let \( i \) be the front element of the current permutation. If \( i \) is not in a block, then we gain an adjacency in one flip if \( i \) (or \( \bar{i} \)) is unburied (see \( A \) or \( A' \) below). If \( i \) and \( \bar{i} \) are both buried, then we gain two adjacencies in four flips (\( B \)).

\[
A: \begin{array}{ccccccc}
  \bar{i} & \cdots & i & \cdots & 1 & \cdots & i \bar{i} & \cdots \\
A': \begin{array}{ccccccc}
  i & \cdots & \bar{i} & \cdots & 1 & \cdots & i \bar{i} & \cdots \\
B: \begin{array}{ccccccc}
  i & \cdots & \bar{i} & \cdots & \bar{i} & \cdots & i \bar{i} & \cdots \\
C: \begin{array}{ccccccc}
  i & \cdots & \bar{i} & \cdots & 1 & \cdots & i \bar{i} & \cdots \\
C': \begin{array}{ccccccc}
  i & \cdots & \bar{i} & \cdots & 1 & \cdots & i \bar{i} & \cdots \\
D: \begin{array}{ccccccc}
  j & \cdots & \bar{j} & \cdots & 2 & \cdots & j \bar{j} & \cdots \\
E: \begin{array}{ccccccc}
  \bar{i} & \cdots & \bar{j} & \cdots & 4 & \cdots & \bar{i} j & \cdots \\
E': \begin{array}{ccccccc}
  \bar{i} & \cdots & \bar{j} & \cdots & \bar{j} & \cdots & \bar{i} j & \cdots
\end{array}
\end{array}
\end{array}
\end{array}
\end{array}
\]

If \( i \) is in a block and \( \bar{i} \) is unburied, then we gain an adjacency in one flip (\( C \) or \( C' \)). Assume that \( \bar{i} \) is buried, and let \( j \) be the other end of the block containing \( i \). If \( j \) is in a block then we make \( j \) and \( \bar{j} \) adjacent in two flips (\( D \)). If \( j \) is singleton, then we gain two adjacencies in four flips (\( E \)).

With these moves, we can successively increase the number of adjacencies until we have \( n - 1 \). Among these we may have \( n \) and 1 adjacent. In at most four additional flips, we produce the full sorted permutation in increasing order (this is like sorting two burnt pancakes).

14.1.52. THEOREM. Algorithm 14.1.51 sorts every permutation of \([n]\) in at most \((5n + 5)/3\) steps.

**Proof:** Let \( x_\alpha \) be the number of moves of type \( \alpha \) performed by the algorithm, where \( \alpha \in \{A, A', B, C, C', D, E\} \). To obtain an upper bound on the number of moves used, we want to see numerically how large \( \sum_\alpha f_\alpha x_\alpha \) can be, where \( f_\alpha \) is the number of flips in a move of type \( \alpha \); every instance of the algorithm will take at most this many steps.

There are two constraints. Each case in the algorithm has an effect on the number of adjacencies (gaining 1 or 2) and on the number of blocks (gain 1, 0, or -1). The number of adjacencies starts with a nonnegative value \( a \) and ends this phase at \( n - 1 \). The number of blocks starts with a nonnegative value \( b \) and winds up at 1. Note that \( a \geq b \). Hence the net gain in adjacencies is at most \( n - 1 - b \), and the net loss in blocks is at most \( b - 1 \). This yields a linear program with the tableau below.

\[
\begin{array}{cccccccc}
  y_a & x_A & x_{A'} & x_B & x_C & x_{C'} & x_D & x_E \\
  1 & 1 & 2 & 1 & 1 & 2 & 2 & n - 1 - b \\
  -1 & 0 & 1 & 0 & 1 & 1 & 1 & b - 1 \\
  1 & 1 & 4 & 1 & 1 & 2 & 4 & \end{array}
\]

Since the algorithm may make four flips in the last phase, to show that in total it never uses more than \((5n + 5)/3\) flips, it suffices to obtain an upper bound of \( (5n - 7)/3 \) on the value of the maximization problem. To prove such a bound, it suffices to exhibit a feasible solution to the dual minimization problem that has this value.

Set \( y_a = 5/3 \) and \( y_b = 2/3 \). The linear combination in each column is at least the value in the bottom row; hence this is a feasible solution to the dual. Its value is \( (5/3)(n - 1 - b) + (2/3)(b - 1) \), which equals \( (5n - 7)/3 - b \).

14.1.53. REMARK. When \( b = 0 \), these linear programs have value \((5n - 7)/3\). This does not imply that there is a permutation that needs \((5n + 5)/3\) flips. It says only that there is no permutation on which this algorithm uses more than \((5n + 5)/3\) flips, and even that may not be sharp.

In proving Theorem 14.1.52, giving a dual solution sufficed, but how was it found? The simplex algorithm was not needed. In studying the maximization problem, one may observe that some variables are no help. For example, any amount given to \( x_D \) would be better given to \( x_E \), and any amount given to \( x_A \) or \( x_{C'} \) of \( x_E \) would be better given to \( x_A \). Hence the nonzero variables in an optimal solution of the maximization problem may be assumed to be only \( x_A \) and \( x_E \). By complementary slackness, the corresponding dual constraints will hold with equality in an optimal solution. This yields the two linear equations \( y_a - y_b = 1 \) and \( 2y_a + y_b = 4 \).
Our second example is from extremal set theory. The intuition is that if an antichain is big, then the average size of its members is big. More precisely, Kleitman–Milner [1973] showed that if the size of an antichain $F$ in $2^{|n|}$ is at least $\binom{2n}{n}$, where $k \leq n/2$, then the average size of the members of $F$ is at least $k$. The bound is sharp, of course, since it holds with equality for $F = \binom{2n}{n}$. The argument applies to all LYM posets (Section 11.2).

**14.1.54. THEOREM.** (Greene–Kleitman [1978]) Let $P$ be an LYM order with log-concave rank sizes $N_0, \ldots, N_m$ (that is, $N_i^2 \geq N_{i-1}N_{i+1}$ for all $i$) such that $N_m = \max_i N_i$. If $k \leq m$ and $A$ is an antichain in $P$ with $|A| \geq N_k$, then the average rank of elements of $A$ is at least $k$.

**Proof:** We seek the inequality $\sum_{x \in A} r(x)/|A| \geq k$, where $r$ is the rank (the size) of $x$. Let $y_i = |A \cap P_i|/N_i$; this is the fraction of elements with rank $i$ that are in $A$. The LYM Inequality states that $\sum_i y_i \leq 1$. The actual number of elements having rank $i$ is $y_i N_i$, so the hypothesis of the theorem gives another linear constraint: $\sum_i y_i N_i \geq N_k$.

We rewrite the desired inequality in this notation by using $\sum_{x \in A} r(x)/|A| = \sum_{i=0}^m y_i N_i = \sum_{i=0}^m y_i N_i$ and $\sum_{x \in A} r(x)/|A| = \sum_{i=0}^m y_i N_i = k$. Here $y_i$ is a variable, depending on the choice of $A$, and $N_i$ is a constant coefficient. Given the constraints above, minimizing $\sum_i y_i N_i$ is a linear program. If the dual problem tells us, numerically, that there is no feasible choice of the variables with objective less than $k|A|$, then the theorem is proved.

It would seem that $|A|$ is also a variable, but in this case we can simplify matters by discarding the elements of $|A|$ with highest ranks to obtain $|A| = N_k$, since that will not increase the average rank. We may also assume that $A$ has no elements above rank $m$, since otherwise the normalized matching property would allow us to replace the elements at the highest rank with the same number of elements in $P_m$, thereby reducing the average rank of the elements.

In canonical form, we now have the problem in nonnegative variables of minimizing $\sum_i y_i N_i$ subject to $\sum_{i=0}^m (-y_i) \geq -1$ and $\sum_{i=0}^m N_i y_i \geq N_k$.

<table>
<thead>
<tr>
<th>$y_0$</th>
<th>$y_1$</th>
<th>$\ldots$</th>
<th>$y_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-1$</td>
<td>$N_0$</td>
<td>$\ldots$</td>
<td>$N_m$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$N_k$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From these two inequalities, the dual maximization problem has just two (nonnegative) variables, say $\alpha$ and $\beta$. We obtain a constraint in the dual for each variable $y_i$ in the minimization problem. These constraints are $-\alpha + N_i \beta \leq i N_i$ for $0 \leq i \leq m$, and the objective is to maximize $-\alpha + N_k \beta$.

We know that the minimization problem has a feasible solution with $y_k = 1$, all other $y_i = 0$, and value $k N_k$; this corresponds to the antichain $P_k$. The linear programming approach will prove that this is optimal if there is also a solution to the dual with value $k N_k$. Hence we set $-\alpha + N_k \beta = k N_k$, or $\alpha = (\beta - k) N_k$.

To complete such a solution, we need a value of $\beta$ such that $\beta \geq k$ (for $\alpha \geq 0$) and $(k - \beta) N_k + N_k \beta \leq i N_i$ for $0 \leq i \leq m$. We rewrite this as $\beta (N_i - N_k) \leq i N_i - k N_k$. The inequality always holds when $i = k$. Since $N_0, \ldots, N_m$ is nondecreasing (due to log-concavity and the maximization at $m$), we need $\beta \geq k N_i - i N_i$ for $i < k$ and $\beta \leq j N_j$ for $j > k$.

Such a value of $\beta$ exists if and only if $\frac{k N_i - i N_i}{N_i - N_k} \leq \frac{j N_j - N_j}{N_j - N_k}$ whenever $i < k < j \leq m$. Algebraic manipulation transforms this to $\frac{1}{N_i - N_k} \leq \frac{1}{N_j - N_k} + (1 - \lambda) \frac{1}{N_j}$, where $\lambda = \frac{j - 1}{j - k}$ and $1 - \lambda = \frac{k - 1}{j - k}$. With this definition of $\lambda$, we also have $k = \lambda i + (1 - \lambda) j$.

We have therefore reduced our problem to the following lemma: If $a_0, \ldots, a_m$ is logarithmically concave as a function of the index, then $a_1^{-1}, \ldots, a_m^{-1}$ is convex as a function of the index. For this it suffices to prove that $\frac{1}{x} \leq \frac{\lambda}{x}$ implies $\frac{1}{x} - \frac{1}{y} \geq \frac{1}{x} - \frac{1}{y}$. We compute

$$\frac{1}{z} \geq \frac{1}{y^2} \iff \frac{x - y}{y^2} \geq \frac{x - y}{yx} = \frac{1}{x} - \frac{1}{y}.$$

In Theorem 14.1.54, there was quite a bit of work to do after setting up the linear program, because we proved a fairly general theorem in which we did not know the entries in the constraint matrix; we knew only the log-concavity relationship among them. Often we have an explicit matrix, and we can obtain a bound merely by exhibiting a feasible solution to the dual. Our search for a good solution to the dual can be guided by the complementary slackness conditions that we know must hold for optimal solutions to both programs, since we may have a candidate optimal solution arising from the construction for the original problem.

**EXERCISES**

14.1.1. (−) Construct a pair of dual linear programs that are both infeasible.

14.1.2. (−) The Finger Game (Example 14.1.1).

a) Which values of $x$ in the interval $[0, 1]$ guarantee a positive expectation for
14.1.13. Use weak duality of linear programming to prove the weak duality property for matroid intersection: \(|I| \leq r_1(X) + r_2(\overline{X})\) for any \(I \subseteq I_1 \cap I_2\) and \(X \subseteq E\). (Hint: Set up an appropriate tableau.)

14.1.14. Prove that if \(\hat{x}\) is an optimal solution to \(\max[c^T x : Ax \leq b, x \geq 0]\) and \(\hat{y}\) is an optimal solution to \(\min[y^T c : y^T A \geq c, y \geq 0]\), then complementary slackness holds. That is, a variable in one of these solutions is nonzero only when the corresponding constraint is tight in the other problem.

14.1.15. (\() Prove that Farkas’ Lemma (Theorem 14.1.21) is equivalent to the following: Either the system \(Ax \leq b\) has a nonnegative solution or \(y^T A \geq 0\) has a nonnegative solution such that \(yb < 0\). (Hint: Consider the matrix \([I A]\).)

14.1.16. (\() Prove that Farkas’ Lemma (Theorem 14.1.21) is equivalent to the following: Either the system \(Ax \leq b\) has a solution (unrestricted variables) or \(y^T A = 0^T\) has a nonnegative solution such that \(yb < 0\). (Hint: Consider the matrix \([I A -A]\).)

14.1.17. (\() Use the variant of Farkas’ Lemma in Exercise 14.1.6 to prove the Duality Theorem of Linear Programming (Theorem 14.1.7). (Hint: Use the system below to produce complementary solutions.)

\[
\begin{bmatrix}
A & 0 \\
-c^T & b^T \\
0 & A^T \\
0 & -A^T
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\leq
\begin{bmatrix}
b \\
0 \\
c
\end{bmatrix}
\]

14.1.18. Use Farkas’ Lemma (Theorem 14.1.21) and the Duality Theorem (Theorem 14.1.7) to prove an “affine form” of Farkas’ Lemma: If \(Ax \leq b\) is feasible, and \(c^T x \leq \delta\) for each \(x\) such that \(Ax \leq b\), then for some \(\delta’ \leq \delta\) the inequality \(c^T x \delta’\) is a nonnegative linear combination of the inequalities in \(Ax \leq b\). (Haar [1918])

14.1.19. (+) Use the variant of Farkas’ Lemma in Exercise 14.1.6 to prove the Scott–Suppes Theorem (Theorem 12.1.18): a poset is a semiorder if and only if it neither \(2 + 2\) nor \(3 + 1\) is a subposet. (Isaak)

14.1.10. Show that every vertex of degree at most 2 in an oriented graph is expansive (see Conjecture 14.1.22). In general, having minimum degree does not guarantee being expansive. Construct an oriented graph with eight vertices in which every vertex has outdegree 3 but not every vertex is expansive.

14.1.11. Prove that every tournament \(T\) has a unique regular subtournament \(Q\) such that every vertex outside \(Q\) has at least as many successors in \(Q\) as predecessors in \(Q\). Use this and Theorem 14.1.23 to prove that every tournament has exactly one losing density. (Fisher–Ryan [1992], Laslier–Laffond–Le Breton [1993])

14.1.12. Give a necessary and sufficient condition on a payoff matrix so that the uniform strategies are optimal strategies.

14.1.13. Let \(G\) be a multigraph whose underlying graph is the cycle \(C_n\), with \(G\) having \(n_i\) copies of edge \(i\). Let \(w\) be a weight function on \(G\) in which each copy of edge \(i\) has weight \(w_i\). Prove that \(\text{Val}(G, w) < 3\) and that \(\text{Val}(G, w)\) can be arbitrarily close to 3 when \(n\) and \(n_i\) are appropriately chosen.


a) Let \(G\) be an unweighted edge-transitive graph (\(w = 1\) for every edge). Prove that \(\text{Val}(G)\) equals the minimum, over all spanning trees, of the average cost of an edge against that tree. (The average cost of a spanning tree \(T\) is obtained by summing the payoffs for each edge against \(T\) and dividing by \(|E(G)|\).)

b) Use (c) to show that the value of the game on the unweighted \(d\)-dimensional hypercube is at most \((d + 3)/2\) (conjectured to be asymptotically optimal).

c) Consider a matrix game with payoff matrix \((c_{ij})\) (to the row player).

a) Determine the value of the game for all choices of these numbers.

b) Determine when the players each have an optimal strategy consisting of playing their two options with equal probability.

14.1.16. Fix a computational problem. Let \(G\) be the set of possible inputs, and let \(A\) be the set of deterministic algorithms. Every algorithm \(A \in A\) incurs some cost \(c(A, G)\) on input \(G \in G\). Randomized algorithms can choose among alternatives in \(A\) by a probability distribution. Against a fixed input \(G \in G\), a particular randomized algorithm has an expected cost. The maximum of this over all inputs is the worst-case expected cost.

Let \(\alpha\) be the least worst-case expected cost of a randomized algorithm. Given a distribution \(q\) on \(G\), let \(\beta\) be the minimum expected cost of a deterministic algorithm against \(q\). Prove that \(\alpha > \beta\). Thus a lower bound on the worst-case expected performance of a randomized algorithm follows by giving an input distribution against which every deterministic algorithm has high expected cost. (Yao [1977])

14.1.17. Prove Sperner’s Theorem (Theorem 11.2.5) and the Erdős–Ko–Rado Theorem (Theorem 13.1.8) using weak duality of linear programming.

14.1.18. A -generator of a simple graph \(G\) is a list of subsets of \(V(G)\) (not necessarily distinct) such that \(u \leftrightarrow v\) if and only if \(u\) and \(v\) appear together in at least \(p\) sets in the list. Let \(T\) be a -generator of \(K_{n,n}\).

a) Let \(\gamma_{k}\) be the number of sets of size \(k\) in \(T\). Use counting arguments to prove that \(\sum \gamma_{k}a_{k} \leq 2(p - 1)\binom{n}{k}\) and \(\sum \gamma_{k} \beta_{k} \geq pn^{k}\), where \(a_{k} = \binom{k}{2} + \binom{k}{2}\) and \(\beta_{k} = \left\lfloor \frac{k^{2}}{4}\right\rfloor\).

b) By exhibiting a solution to the dual of the problem of minimizing \(\sum \gamma_{k}\) subject to the constraints in part (a), prove that \(|T| \geq (n^2 + 2p - 1)n/p\). (Comment: Eaton–Gould–Rödl [1996] and Füredi [1997] independently proved that this lower bound is asymptotically best possible.) (Chung–West [1994])

14.1.19. (+) Choose \(m, n \in \mathbb{N}\) with \(1 \leq m < 2^{n+1}\). Let \(k\) be the largest integer such that \(\sum_{i=1}^{k} (n+1)^{i-1} \leq m\). Let \(F\) be an intersecting family in \(2^{[n]}\) consisting of \(m\) sets. Show that \(I_{\supseteq F} \{x\}\) is minimized by using all sets of size at most \(k\) that contain the element 1 together with enough sets of size \(k + 1\) containing 1 to bring the total to \(m\). (Hint: Introduce variables \(x_0, \ldots, x_n\) to record the number of sets in \(F\) of various
sizes, and use the Erdős–Ko–Rado Theorem (Theorem 13.1.8) to establish a linear programming relaxation of the problem.) (Griggs [2002])