[27] Van, A.H., and Roberts, J.W., "Dynamic resource allocation in the Telecom 1 satellite system," *International Conference on Communications*, 1983.

[28] Kaufman, J.S., "Blocking in a shared resource environment," *IEEE Transactions on Communications*, vol. COM-29, pp. 1474–1481, 1981.

[29] Mason, L., Deserres, Y., and Meubus, C., "Circuit-switched multipoint service performance models," *International Teletraffic Congress*, vol. 11, pp. 2.1A-5-1–2.1A-5-6, September 1985.

[30] Gerla, M., "Routing and flow control in ISDN's," in *New Communication Services: A Challenge to Computer Technology*, Kühn, P., ed., pp. 643–647, Elsevier Science Publishers B.V., 1986.

[31] Ash, G.R., "Traffic network routing, control and design for the ISDN era," *ITC Seminar*, 1987.

[32] Ackerly, R.G., Macfayden, N.W., and Songhurst, D.J., "The study of network performance in relation to ISDN," *ITC Seminar*, 1987.

APPENDIX A

Probability Theory

This appendix briefly reviews some elements of probability theory. The review is short because there is an abundance of excellent references on the subject [1]. The topics included here either are not widely known outside the teletraffic community or are of fundamental importance for the traffic analysis presented in this book. This second category is limited to the fundamental results of birth-and-death equations and of the $GI/M/N/\infty/N$ queue. Applications of the general theory to the particular queues of interest in traffic theory are found in the main text.

A.1 Some Functions Used in Probability Models

The following definitions and simple properties are either used extensively in the text or are not well known outside the teletraffic community.

Binomial Coefficients and Related Polynomials

The binomial coefficients are defined for positive integer n as the coefficients of x in the expansion of

$$(1+x)^n = \sum_{k=0}^{n} x^k \binom{n}{k}$$

and are given by

$$\binom{n}{k} = \frac{n!}{(n-k)!k!},$$

which can be written

$$= \frac{n(n-1)\ldots(n-k+1)}{k!}.$$

This definition can be extended to nonnegative real a by defining the binomial coefficient as the coefficient of x in the expansion of

$$(1+x)^a = \sum_{k=0}^{\infty} x^k \frac{a(a-1)\ldots(a-k+1)}{k!},$$

from which we get

$$\binom{a}{k} \triangleq \frac{a(a-1)\dots(a-k+1)}{k!} \quad \forall a \in \Re, \quad \binom{a}{0} \triangleq 1 \tag{A.1}$$

From this, we define the *factorial polynomials*:

$$(a)_k \triangleq a(a-1)\dots(a-k+1). \tag{A.2}$$

The binomial coefficients can be extended to arbitrary real a by looking at the power series of

$$(1+x)^{-a} = \sum_{k=0}^{\infty} x^k (-1)^k \frac{a(a+1)\dots(a+k-1)}{k!}, \quad a \geq 0$$

By defining the extended binomial moment as the coefficient of $-x^k$, we get

$$\binom{-a}{k} = \frac{a(a+1)\dots(a+k-1)}{k!}.$$

A useful relation is obtained if we note that the polynomial $a(a+1)\dots(a+k-1)$ can also be viewed as a factorial polynomial starting at $a+k-1$. Rewriting the identity for the polynomial, we get

$$\binom{-x}{k} = \binom{x+k-1}{k}, \quad x > 0$$

The binomial coefficients also have the following useful inversion property:

$$a_n = \sum_{j=n}^{N} \binom{j}{n} b_j$$

$$b_j = \sum_{n=j}^{N} (-1)^{n-j} \binom{n}{j} a_j$$

The factorial polynomials are related to the $a(a+1)\dots(a+k-1)$ polynomials by the functions

$$(1-x)^{-a} = \sum_{k=0}^{\infty} \frac{a(a+1)\dots(a+k-1)x^k}{k!} \tag{A.3}$$

$$(1+x)^{a} = \sum_{k=0}^{\infty} \frac{(a)_k x^k}{k!} \tag{A.4}$$

Factorial polynomials have a double representation that is related by the so-called *Stirling numbers* [2] $s_{q,i}$ and $S_{q,i}$ of the first and second kind, respectively. We have

$$(x)_q = \sum_{i=0}^{q} s_{q,i} x^i$$

$$x(x+1)\dots(x+q-1) = \sum_{i=0}^{q} S_{q,i}(x)_i$$

with the Stirling numbers defined recursively by

$$\begin{aligned} s_{q+1,i} &= s_{q,i-1} - q s_{q,i} \\ S_{q,i+1} &= S_{q,i-1} + i S_{q,i} \\ s_{0,0} &= S_{0,0} = 1 \\ s_{q,i} &= S_{q,i} = 0 \quad i > q \text{ and } i < -1 \end{aligned}$$

The Kosten and Brockmeyer Polynomials

The Brockmeyer polynomials [3] are defined as

$$S(l,k) = \sum_{i=0}^{l} \frac{A^{l-i}}{(l-i)!}\binom{k-1+i}{i}, \quad S(l,k) = 0 \text{ if } l < 0 \text{ or } m < 0 \quad \text{(A.5)}$$

and the closely related Kosten polynomials [4] by

$$R(k,l) = e^{-A} S(k,l). \quad \text{(A.6)}$$

These functions have some useful properties, such as

$$\begin{aligned} S(k,l) &= S(k-1,l) + S(k,l-1) \\ kS(k,l) &= (A+k-1+l)S(k-1,l) - AS(k-2,l) \\ kS(k,l) &= AS(k-1,l) + lS(k-1,l+1) \\ S(k,l) &= \sum_{j=0}^{i} (-1)^{i-j}\binom{i}{j} S(k-i+j, l+i) \\ S(k,l) &= \sum_{i=0}^{k} S(i,l-1) \end{aligned}$$

These polynomials are frequently useful in analyzing queuing systems with loss.

Moments of a Distribution

The various moments of a distribution ranging from 0 to $N \leq \infty$ are defined as

$$\alpha_q = \sum_{j=0}^{N} j^q p_j \quad \text{(A.7)}$$

for the ordinary moments;

$$\mu_q = \sum_{j=0}^{N} (j - \alpha_1)^q p_j \quad \text{(A.8)}$$

for the central moments;

$$M_q = \sum_{j=q}^{N} (j)_q p_j \tag{A.9}$$

for the factorial moments; and

$$\beta_q = \sum_{j=q}^{N} \binom{j}{q} p_j = \frac{M_q}{q!} \tag{A.10}$$

for the binomial moments.

The various types of moments are interrelated in such a way that, given one set, the other sets can be computed directly. In particular, we have the mean and variance of a given distribution directly from the moments by

$$M \triangleq \alpha_1 = M_1 = \beta_1 \tag{A.11}$$

$$V \triangleq \mu_2 = \alpha_2 - \alpha_1^2 = M_2 - \alpha_1(\alpha_1 - 1) = 2\beta_2 + \beta_1(\beta_1 - 1) \tag{A.12}$$

Generating Functions and Transforms

The generating function, or the z-transform, of a function defined over the nonnegative integers is given by

$$G(z) = \sum_{k=0}^{\infty} p_k z^k, \quad |z| \leq 1$$

This sum converges if p_k does not grow faster than geometrically; the generating function is analytic in the disk $|z| \leq 1$. For instance, the binomial coefficients have the generating function

$$(1+z)^n = \sum_{k=0}^{n} \binom{n}{k} z^k.$$

The generating function contains all the information about a distribution. From this function, we can recover either the probabilities or the moments, as needed. We have

$$G(0) = \sum_{n=0}^{N} p_n$$

$$p_n = \frac{1}{n!} \left(\frac{d^n G(z)}{dz^n} \right)_{z=0}$$

$$\beta_n = \frac{1}{n!} \left(\frac{d^n G(z)}{d^n z} \right)_{z=1}$$

and, in particular, the mean and variance of a distribution:

$$M = G'(1)$$
$$V = G''(1) + G'(1) - (G'(1))^2$$

For example, we have the expression of the distribution in terms of the factorial moments as

$$p_j = \sum_{k=0}^{\infty} (-1)^{k-j} \binom{k}{j} \frac{M_k}{k!}. \tag{A.13}$$

For continuous distributions $F(t)$, we define the Laplace-Stieltjes transform $\Phi(s)$ as follows:

$$\Phi(s) = \int_0^{\infty} e^{-st} dF(t)$$

and, if $F(t)$ is differentiable everywhere, the ordinary Laplace transform

$$\phi(s) = \int_0^{\infty} e^{-st} f(t) dt$$
$$\Phi(s) = s\phi(s)$$

Here too, we can get the moments of the distribution $F(t)$ directly from the Laplace-Stieltjes transform:

$$\int_0^{\infty} t^n dF(t) = (-1)^n \left(\frac{d^n \Phi(s)}{ds^n} \right)_{s=0}$$

Also, if we define the generating function for the binomial moment,

$$F(z) = \sum_{k=0}^{N} \beta_k z^k,$$

it is not hard to show that this function is related to the probability generating function by

$$F(z) = G(z+1).$$

A.2 Birth-and-Death Processes

Birth-and-death (BD) processes are the simplest type of processes that can be used to analyze queuing systems [5]. BD processes are characterized uniquely by their state-dependent arrival and departure rates. For one-dimensional processes, the state is represented by k, the number of customers present in the

system. The arrival rate when the system is in state k is λ_k; the departure rate is given by μ_k. The time-dependent state equation is then given by

$$\frac{dp_k}{dt} = -(\lambda_k + \mu_k)p_k + \lambda_{k-1}p_{k-1} + \mu_{k+1}p_{k+1}, \tag{A.14}$$

and the stationary form

$$0 = -(\lambda_k + \mu_k)p_k + \lambda_{k-1}p_{k-1}p + \mu_{k+1}p_{k+1}. \tag{A.15}$$

The general solution for the stationary state probabilities is given by

$$p_k = p_0 \prod_{j=0}^{k-1} \frac{\lambda_j}{\mu_{j+1}}, \tag{A.16}$$

which satisfies the useful recurrence

$$p_{k+1} = \frac{\lambda_k}{\mu_{k+1}} p_k$$

and the normalization condition

$$\sum_{k=0}^{\infty} p_k = 1.$$

As we saw in Chapter 3, a large number of queuing systems of interest can be modeled by a suitable choice of λ_k and μ_k.

A.3 Renewal Processes and the $GI/M/N/\infty/N$ Queue

A general analysis of overflow processes is based on the $GI/M/N/\infty/N$ queue. In such a queue, the arrival process is of the renewal type, and the service periods have a negative exponential distribution. A complete description of the results available on the $GI/M/N$ system can be found in [6,7]. Here we briefly summarize the derivation of the general equations that can be used to describe this queue.

Distribution of Arrivals in a Fixed Interval

When considering the sequence of requests for service on a particular link, we are naturally led to consider the random variable that represents the length of time that separates two successive arrivals. Following Syski [7], we denote this variable by L_r, calling it the *life* of the interval between the $r-1^{\text{th}}$ and r^{th} calls. The arrival process is a *renewal process* if the L_r variables are independent, nonnegative, identically distributed variables. Although the theory is explained in terms of call arrivals, it applies equally well to the call-departure process, or to any other type of event that is of interest.

We denote the distribution of L_r by $F(t)$, and its density by $f(t)$. For an arbitrary interval $(0, t]$ two quantities are of interest: (1) $N(t)$, the number

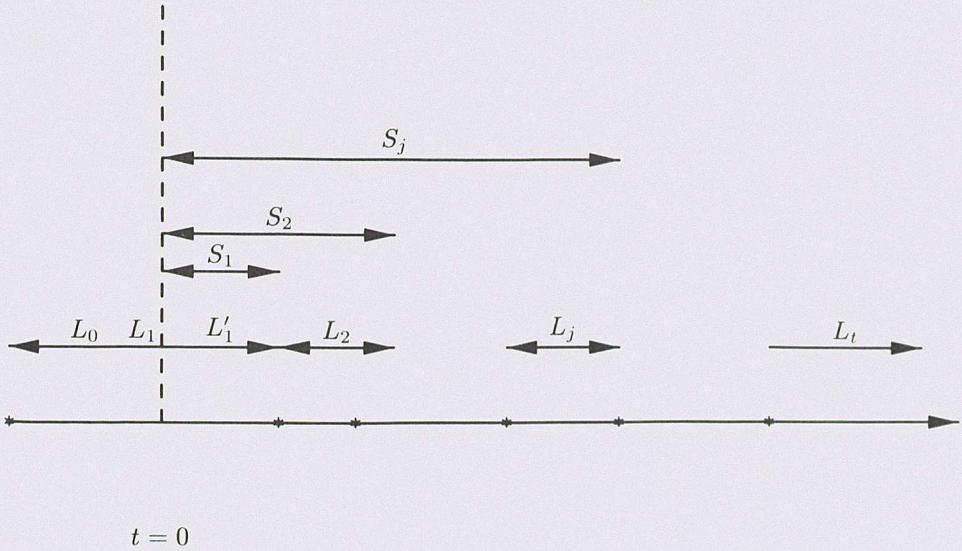

Figure A.1 Definition of Variables Related to a Renewal Process

of arrivals during this interval, and (2) $L(t)$, the time since the last arrival before time t, called the *age* of the interval, with the distribution $Y(s,t) \triangleq Pr\{L(t) \leq s\}$. Note that Y is not simply the value of t because we can select the origin of time at an arbitrary instant, which need not be the time of an arrival.

Calculating the distribution of these random variables require some auxiliary variables. Let S_j be the total lifetime of j calls that arrive in this interval. The distribution functions of the S_js are denoted $F_j(t) = Pr\{S_j \leq t\}$. This is the total duration of the j intervals, and it is obviously given by $S_j = L_1' + L_2 + \ldots + L_j$. Note the special notation for the first interval. Because we wish to be able to select the origin at an arbitrary point, this origin generally does not coincide with an arrival. In this case, the time origin is inside an interarrival period, and the age of the first interval as measured from the origin is different from the ages of the other intervals. This first partial interval is denoted $L_1' = L_1 - L_0$; this term appears in the definition of S_j. In other words, L_1' is the residual life of the first interval after we start looking at the system. Note also that we have

$$L_t = \begin{cases} t - S_{N(t)} & \text{if } N(t) \geq 1 \\ t + L_0 & \text{if } N(t) = 0 \end{cases}$$

The meaning of these variables is shown in Fig A.1.

Our main concern here is with calculating the statistical properties of $N(t)$. Let $p_j(t)$ be the probability of having j arrivals in the interval $(0, t]$. The number of arrivals in the interval is larger than j if and only if the total duration of the first j calls in the interval is not larger than t. This is expressed by

$$\begin{aligned} Pr\{N(t) \geq j\} &= Pr\{S_j \leq t\} \\ &= F_j(t), \end{aligned}$$

and we have, by the definition of F_j,

$$\sum_{k=j}^{\infty} p_k(t) = F_j(t).$$

Taking partial sums from j and $j+1$ and subtracting, we have

$$p_j(t) = F_j(t) - F_{j+1}(t). \tag{A.17}$$

We now need the F_js, the distribution of the cumulative intervals S_j. All these intervals are similar, except for the first one, which depends on the position of the time origin. Let $B(t)$ be the distribution of L'_1. We then have

$$B(t) = \int_0^{\infty} \frac{F(t+s) - F(s)}{1 - F(s)} dY(s,t) \tag{A.18}$$

$$\begin{aligned} F_0(t) &= 1 \\ F_1(t) &= B(t) \\ F_j(t) &= \int_0^{\infty} F(t-\tau) dF_{j-1}(\tau) \qquad \text{(A.19)} \\ F_{\infty}(t) &= 0 \end{aligned}$$

The first equation can be understood as follows. We know that the time origin is inside the first interval. The probability that this interval lasts for another period s after t, given that the time origin is in this interval, is called the *remaining lifetime distribution*; it is given by

$$Pr\{L \leq t + s \mid L \geq t\} = \frac{F(t+s) - F(t)}{1 - F(t)}. \tag{A.20}$$

(Solving this equation is left to the reader.) S_1, then, is the lifetime of the first interval after the origin of the time axis; it is given by the convolution of the distribution of the two events: (1) the residual life must be at least as large as s and (2) there must be no further arrival until t, the density of which is given by dY. The convolution of these two events is the distribution $B(t)$. Similarly, Eq. (A.19) follows from the convolution theorem when we note that $S_j = S_{j-1} + L_j$. Using Eq. (A.19) in Eq. (A.17), we obtain

$$p_j(t) = \int_0^{\infty} [1 - F(t-\tau)] dF_j(\tau). \tag{A.21}$$

The expectation of $N(t)$ is then given by

$$\sum_{j=0}^{\infty} jp_j(t) = \sum_{j=1}^{\infty} F_j(t).$$

From this, we get the fundamental integral equation of renewal theory concerning $G(t)$, the expected number of arrivals in the interval,

$$\begin{aligned} G(t) &\triangleq E\{N(t)\} \\ &= B(t) + \int_0^{\infty} F(t-\tau)dG(\tau), \end{aligned} \tag{A.22}$$

and the corresponding integral equation for the density,

$$g(t) = b(t) + \int_0^t f(t-\tau)g(\tau)d\tau,$$

where $g(t)$ is defined as

$$G(t) = \int_0^t g(\tau)d\tau.$$

The intuitive interpretation of $g(t)$ is the renewal rate of the process — in other words, the probability that there is a renewal in a short interval dt at time t is precisely $g(t)dt$. More important, however, is the stationary behavior of the density. It is known that $\lim_{t\to\infty} g(t) = 1/m$, where $m = E\{L\}$ is the expected lifetime of an interval.

All that remains to be computed is the distribution of the age of the interval $Y(s,t)$ or, equivalently, of its density $y(s,t)$. This is needed because of the arbitrary position of the first interval with respect to the time origin, which appears in Eq. (A.18), the distribution of the residual life of the first interval. Two cases must be considered, depending on whether or not the interval started before the origin of time. If not, the interval started at $t-s$, with probability density $g(t-s)$. It lasts up to time s, with probability $1-F(s)$. Thus the probability of having age s at time t has the density

$$y(s,t) = g(t-s)\,[1-F(s)]\,ds, \quad t > s$$

If, on the other hand, the interval started before the time origin, then it had age $t-s$ at time 0 with probability $y(s-t,0)ds$. Further, the probability that the life of the interval will continue until s, given that it has already lasted $s-t$, is given by $(1-F(s))/(1-F(s-t))$. Hence we have, in this case, from the residual-life formula (A.20),

$$y(s,t) = y(s-t,0)\frac{1-F(s)}{1-F(s-t)}ds, \quad t \le s \tag{A.23}$$

This completes the derivation for the distribution of $N(t)$ for a given interval of size s. In the case of telecommunication networks, we are not particularly

interested in the exact time dependence of Y, but rather in its asymptotic, stationary value. Under suitable conditions, there exists such a stationary distribution of age that is independent of t and of the initial distribution. It is given by

$$Y(s,\infty) = \int_0^s \frac{1-F(\tau)}{m} d\tau \tag{A.24}$$

for finite m. In fact [7], the process $L(t)$ is Markov, has stationary probabilities, and, under suitable conditions, is also ergodic. This expression is the explanation of the so-called *renewal paradox*, a discussion of which is proposed in Problem 3.10.

Distribution of Interarrival Times in a Random Interval

The results obtained so far deal with the number of arrivals during a given interval $(0,t]$. In many cases, we are interested in the number of arrivals during an interval whose length is also a random variable. The application we shall consider is the number of call terminations in the interval between two arrivals in a group of servers. Because of the random nature of the arrivals, the time during which we count the terminations is a random variable, which explains the interest in these processes. These processes are called *compound process*.

Let $N(t)$ be the number of arrivals in a random interval starting at t. It would be useful to relate quantities of interest about $N(t)$ to corresponding quantities about $N(s,t)$, the number of arrivals in an interval of given length s starting at t, since we already have many results in this case. To do this, let the length of time we are considering be a random variable L with distribution $U(t)$. The relation between the distribution of $N(t)$ and $N(s,t)$ is straightforward if we view the distribution of N as conditional on L. We can write

$$Pr\{N=n\} = \int_0^\infty Pr\{N(s,t)=n \mid L=s\} dU(s) \tag{A.25}$$

In many cases, the N and L processes are independent, and $U(s)$ is independent of the starting point of the interval t and is unrelated to the distribution of N. Thus, using Eq. (A.25), we can readily obtain the distribution of N for a random interval if we have this distribution for a fixed interval of size s.

Similar relations exist between other functions of the random variables $N(t)$ and $N(s,t)$. Two important cases are the z transform and the expectations. To get the z transform $G(z,t)$ of $N(t)$, we multiply both sides of Eq. (A.25) by z^n, taking the sum from 0 to ∞. We get the required relation

$$G(z,t) = \int_0^\infty dF(s) G(z,s,t) dz.$$

From this equation, we obtain a similar equation for the expectation,

$$E\{N\} = \int_0^\infty E\{N(s,t)\}dF(s).$$

Eq. (A.25) is of fundamental importance for the discussion in Chapter 3. Most of the results that we will derive for teletraffic are applications of this equation or of one of its variants.

Compound Process in the $GI/M/N/\infty/N$ Queue

Consider now the process by which calls in service in a group terminate. We are interested in $p_{i,j}$, the probability that there are j calls in progress at the end of a single interarrival period, given that there were i calls in progress at the beginning. If we assume that the calls in progress are independent, and have the same holding time distribution $H(s)$ with mean $1/\mu$, then the termination process has all the characteristics of a renewal process, and we can use the results that we previously derived — in particular, Eq. (A.25). First we must compute the probability that j calls are in progress at the end of a given interval of size s, given that i calls were present at the beginning of the interval. Consider first a particular call in progress. The probability that it still exists at time s is just the probability that its age exceeds s. But we know from Eq. (A.24) that this is given by

$$y(s) = \mu \int_s^\infty [1 - H(\tau)]d\tau \qquad (A.24)$$

The probability $p_{i,j}$ that there are j calls in progress at the end of the period, given that there were i calls at the beginning, is a Bernouilli distribution with parameter $y(s)$ and is given by

$$p_{i,j} = \binom{i}{j} y^j (1-y)^{i-j}.$$

Because the length of an interval is the time between two arrivals to the queue, it is a random variable. Thus we must use Eq. (A.25) with $U(t) = F(t)$, which yields

$$p_{i,j} = \int_0^\infty \binom{i}{j} y^j (1-y)^{i-j} dF(s), \quad i \geq j \qquad (A.26)$$

where y stands for $y(s)$ and $F(s)$ is the distribution of the interarrival times.

When the holding time distribution is negative exponential, $y(s) = e^{-\mu s}$. If $F(s)$ is also negative exponential with parameter λ, then we get

$$p_{i,j} = \binom{i}{j} \frac{\lambda}{\mu} B\left(j + \frac{\lambda}{\mu}, i - j + 1\right), \qquad (A.27)$$

where $B(m, n)$ is the Beta function.

References

[1] Feller, W., *An Introduction to Probability Theory and Its Applications*, Wiley, 1957.

[2] Abramowitz, M., and Stegun, I.A., *Handbook of Mathematical Functions*, Dover, 1970.

[3] Brockmeyer, E., "The simple overflow problem in the theory of telephone traffic," (in Danish), *Teleteknik*, vol. 5, pp. 361–374, 1954.

[4] Kosten, L., "On the blocking probability of graded multiples," (in German), *Elektr. Nachr.-Techn.*, vol. 14, pp. 5–12, 1937.

[5] Kleinrock, L., *Queuing Systems*, Wiley, 1975.

[6] Takács, L., *Introduction to the Theory of Queues*, Oxford University Press, 1962.

[7] Syski, R., *Introduction to congestion theory in telephone systems*, North-Holland, 1986.

APPENDIX *B*

Optimization Theory

An enormous literature exists on the subject of optimization, too large to be summarized in a short discussion like this one. Here we can present only a brief review of the more elementary results used in the text. The discussion is on optimization of differentiable functions in subsets of R^n, in the spirit of [1,2]. We leave aside combinatorial optimization and integer programming, which play a relatively minor role here. Good references on these topics are [3,4,5].

After reviewing some definitions related to convexity, we state the necessary conditions for optimality and give a short review of nonlinear duality theory. We then discuss some frequently used methods for computing numerical solutions. We end the appendix by presenting some well-known results of the general theory when it is applied to linear programs, emphasizing network flows.

B.1 Convexity

Convex sets and convex functions play a central role in optimization theory. Let us recall some useful definitions and properties in relation to convexity.

Convex Sets

Let $\mathbf{x}_1, \mathbf{x}_2$ be two points in some set S. We define a convex linear combination (clc) of these points as the vector $\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2$ where $0 < \alpha < 1$. This notion can be generalized to l points by

$$\mathbf{y} \triangleq \sum_{k=1}^{l} \alpha_k \mathbf{x}_k$$

$$\sum_{k=1}^{l} \alpha_k = 1$$

$$\alpha_k \geq 0$$

A set S is said to be *convex* if any clc of any two of its points is also in S. Geometrically, this means that any point on the line segment between $\mathbf{x}_1$ and $\mathbf{x}_2$ is also a point of S. Examples of convex sets are a triangle, a circle, the half-space defined by $<\mathbf{a}, \mathbf{x}> \leq b$, and the positive quadrant in R^2. It is not hard to show that the intersection of any number of convex sets is also convex. From this theorem we get the important class of convex sets called *polytopes*, defined by the set of m linear inequalities $A\mathbf{x} \leq \mathbf{b}$, where A is an $m \times n$ matrix and $\mathbf{b}$ is an m-vector. A bounded polytope is called a *polyhedron*.

Although any clc of points of a convex S is in S, it does not follow that any element of S is a clc of some set of points (think of the summits of the triangle). We call those points that cannot be written as a clc of some other elements *extreme points* of the set. Extreme points of the circle are the points on the perimeter, and the positive quadrant has only one extreme point, that is, the origin.

This last set, which is unbounded, has two other interesting elements: the semi-axes. This notion is generalized to the concept of *rays*. We say that $\mathbf{d}$ is a ray of the convex set S if $\mathbf{x} + \alpha\mathbf{d} \in S$ for any $\mathbf{x} \in S$ and $\forall \alpha \geq 0$. A vector $\mathbf{d}$ is an *extreme ray* if there does not exist any $\mathbf{d}_1$, $\mathbf{d}_2 \in S$ and $\mathbf{d}_2 \neq \mathbf{d}_1$ such that $\mathbf{d} = \alpha\mathbf{d}_1 + (1-\alpha)\mathbf{d}_2$ for some $0 \leq \alpha \leq 1$. Note the analogy with extreme points. We also need the notion of a *cone*. We say that a set S is a cone if, for every $\mathbf{d} \in S$ and every $\alpha \geq 0$, the vector $\mathbf{y} = \alpha\mathbf{d} \in S$. Note that the positive quadrant is a cone.

With these definitions, we are ready to state a fundamental theorem of geometry. Given a convex set S, any point in S can be expressed as a clc of extreme points and extreme rays. From this theorem, we see that the extreme points and rays are in this sense sufficient to characterize the convex set. This property is used extensively in many areas of optimization, especially in decomposition methods for large problems.

Convex Functions

Let f be some function defined on some convex set S. We say that f is a *convex function* if

$$f(\alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1-\alpha) f(\mathbf{x}_2) \qquad \text{(B.1)}$$
$$\forall\ \mathbf{x}_1,\ \mathbf{x}_2 \in S$$
$$0 \leq \alpha \leq 1$$

The function is said to be strictly convex if the inequality is strict in the definition. Examples of convex functions are x^2 and $|x|$. The second function is an example of a convex function that is not differentiable at the origin.

We recall some elementary properties of convex functions defined over a convex set S. If f_1 and f_2 are convex functions, then so are $f_1 + f_2$ and

αf_1 for $\alpha > 0$. From this follows the general property that a positive linear combination of convex functions is also convex. Also note that the set defined as $S = \mathbf{x} \mid f(\mathbf{x}) \leq b$ is a convex set whenever f is a convex function. From this, we see that a set defined by the nonlinear inequalities $g_i(\mathbf{x}) \leq b_i$ is also a convex set whenever all the g functions are convex.

The notion of convexity can be made more explicit if f is differentiable. Recall that the *gradient* of f is the vector ∇f with components $\partial f/\partial x_i$, and that the Hessian matrix is the one with component $H_{i,j} = \partial^2 f/\partial x_i \partial x_j$.

We can characterize a convex function that is differentiable once by the property that

$$f(\mathbf{y}) \geq f(\mathbf{x}) + <\nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x}>, \quad \forall\, \mathbf{x}, \mathbf{y} \in S$$

Geometrically, this means that the function f lies above its tangent plane at some point $\mathbf{x}$, and that this holds for all points $\mathbf{x}$.

If the function is twice differentiable, it is not hard to show that f is convex at some point $\mathbf{x}$ inside S if $H(\mathbf{x})$ is positive semidefinite at that point. Also, f is globally convex if it is convex at all points in S. This condition reduces to the familiar condition $d^2 f/dx^2 > 0$ for one-dimensional functions. Note, however, that extending this condition to many variables by the rule $\partial^2 f/\partial x_i^2 > 0$ is *not* sufficient to ensure convexity at some point.

Finally, we say that f is *concave* if $-f$ is convex. Concave functions are often used in cost models where economies of scale are present, and optimization problems with concave functions occur quite frequently.

B.2 Optimality Conditions for Optimization Problems

We are now ready to examine the general optimization problem

$$\begin{aligned} &\min\ f(\mathbf{x}) \\ &\mathbf{x} \in S \subseteq R^n \end{aligned} \tag{B.2}$$

In what follows, we say that a point $\mathbf{x} \in S$ is *feasible*; we denote the solution of this problem by $\mathbf{x}^*$, and the value of f at this point by $f^* \triangleq f(\mathbf{x}^*)$ whenever it is necessary to distinguish these values from more general ones. Otherwise we omit the superscript, simply using $\mathbf{x}$ for the optimal solution and so indicating in the text. We define the local minimum of the function first for unconstrained problems, then for constrained ones. By unconstrained problems, we mean those where either $S = R^n$ or the minimum occurs at an interior point of S, in which case we might as well have extended the domain to R^n in the first place.

A point $\mathbf{x}$ is an *unconstrained local minimum* of f if there is a neighborhood ϵ of $\mathbf{x}$ such that, for any displacement $\mathbf{d}$ from $\mathbf{x}$ in this neighborhood, the function is not increasing. Algebraically, this is written

$$f(\mathbf{x} + \mathbf{d}) \geq f(\mathbf{x}), \qquad |\mathbf{d}| \leq \epsilon \tag{B.3}$$

The minimum is said to be *global* if $\epsilon = \infty$, that is, if the neighborhood in question extends to the whole space.

For constrained problems, where $S \subset R^n$ and the minimum occurs on the boundary, definition (B.3) is no longer valid: The displacement from $\mathbf{x}$ may lead outside S, in which case the solution is no longer feasible. The definition can be amended to take care of this case by adding that condition (B.3) must hold for any *feasible* displacement $\mathbf{d}$. We have

$$f(\mathbf{x}+\mathbf{d}) \geq f(\mathbf{x}), \qquad |\mathbf{d}| \leq \epsilon, \ \mathbf{x}+\mathbf{d} \in S \tag{B.4}$$

First-Order Conditions

The definition of a local minimum is usually not easy to check because it involves conditions on *all* feasible displacements. When the problem functions are differentiable, it is possible to derive more easily verifiable conditions from Eq. (B.3). Here we state the first-order necessary conditions, expressed in terms of the first derivatives only. From this point on, we assume that the set S is given by the inequalities $g_i(\mathbf{x}) \leq 0$, $i = 1, \ldots m$. Theoretically, the case of equality constraints $h_i(\mathbf{x}) = 0$ can be included by replacing h_i by two inequality constraints of the form $g_i(\mathbf{x}) \leq 0$ and $-g_i(\mathbf{x}) \leq 0$. Also, we say that a constraint i is *active*, or *tight*, at point $\mathbf{x}$ if $g_i(\mathbf{x}) = 0$. Otherwise, the constraint is said to be inactive, or slack.

For the unconstrained case, the necessary condition reduces to the familiar case $\nabla f = 0$, which can easily be derived by expanding f to first order in the neighborhood of the optimal point $\mathbf{x}$.

For the constrained case, this condition is no longer necessary. The necessary conditions for optimality have been stated by Kuhn and Tucker as follows. Assume that $\mathbf{x}$ is a minimum. Then there exist a set of quantities called the Kuhn-Tucker multipliers u_i such that

$$\nabla f(\mathbf{x}) = -\sum_{i=1}^{m} u_i \nabla g_i(\mathbf{x}) \tag{B.5}$$

$$g_i(\mathbf{x}) \leq 0 \quad i = 1 \ldots m \tag{B.6}$$

$$u_i \geq 0 \quad i = 1 \ldots m \tag{B.7}$$

$$u_i g_i(\mathbf{x}) = \quad i = 1 \ldots m \tag{B.8}$$

The first condition states that, at an optimal solution $\mathbf{x}$, the direction opposite to that of the gradient must lie in the cone of the gradients of active constraints at that point. The second set, called the *primal feasibility* condition, states that the point $\mathbf{x}$ must be feasible for the original problem. The third set, called the *dual feasibility* conditions, defines the domain of the dual problem (see Section B.3). The last set, called the *complementary slackness* condition,

states that, if a constraint is slack, then the corresponding multiplier must vanish. Note, however, that the inverse statement is not necessarily true, and that there can be tight constraints with zero multipliers.

These conditions hold if the functions are differentiable, and also if certain regularity conditions hold on f and g. Although these regularity conditions are theoretically important, they have no effect in practice, and can be ignored for the purpose of this discussion. In the case of equality constraints, the equations are the same, except that condition (B.7) is no longer present. This is easy to check by replacing the equality constraints by a pair of inequality constraints of opposite sign, using conditions (B.5–B.8).

The first condition can be expressed more concisely by defining the *Lagrange function*, or Lagrangian, as a function of both the $\mathbf{x}$ and the $\mathbf{u}$ variables

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) \triangleq f(\mathbf{x}) + \sum_{i=1}^{m} u_i g_i(\mathbf{x}) \tag{B.9}$$

This function is built by incorporating the constraints linearly into the objective function, using the multipliers as coefficients. The first optimality condition then reduces to

$$\nabla_x \mathcal{L}(\mathbf{x}, \mathbf{u}) = 0, \tag{B.10}$$

where ∇_x denotes the gradient with respect to the $\mathbf{x}$ variables.

Second-Order Conditions

The Kuhn-Tucker conditions are necessary but not sufficient for an optimal solution. A sufficient but not necessary condition for a minimum is that f and g_i all be convex functions at that point. For unconstrained problems, this is equivalent to saying that the Hessian is positive semidefinite at the solution. In the constrained case, the condition is somewhat more complex; the interested reader should consult the appropriate references [1].

Global Optimality

The discussion so far has covered conditions for a local optimum. In general, we are often interested in having a global optimum, that is, a solution with the lowest value over all possible solutions in the set, not only in the neighborhood of some point.

Although there exist methods for finding a global solution of an arbitrary optimization problem, they are extremely time consuming and are limited to very small problems with only a few constraints and variables. The only useful result on the question of global optimality is in the case of convex problems, where both the objective function and the domain are convex. In this case,

one can show that any local minimum of the problem is also a global one. In practice, this condition is often not easy to verify, and we must be content to find a local solution.

When it is not know whether the problem is convex, some measure of reassurance about the global nature of the solution can be obtained by starting the solution algorithm from widely different initial solutions. If at least two of the final solutions have different values, then we have demonstrated that the problem is not convex. If all the solutions are identical, we have an indication that the minimum *may* be global. The degree of confidence in this conclusion, however, depends largely on the nature of the domain and the function.

Finally, we recall an important theorem concerning the minimization of concave function over convex sets. The theorem states that the global minimum can be found on at least one of the extreme points of the set. In cases when this set is finite, this reduces the optimization to a search over a finite set that may be easier to perform than the search in R^n.

B.3 Duality Theory

To verify that a solution $\mathbf{x}$ satisfies the necessary conditions for optimality, the Kuhn-Tucker multipliers must be known. Of course, these multipliers usually are not known, and calculating them leads us into the subject of duality.

We start by remarking that the Lagrange function can be viewed as a function of two vectors, that is, $\mathbf{x}$ and $\mathbf{u}$. Since we are interested in $\mathbf{u}$, we may think of eliminating the other vector. This defines the *dual function*

$$w(\mathbf{u}) \triangleq \min_{\mathbf{x} \in S} \mathcal{L}(\mathbf{x}, \mathbf{u}) \tag{B.11}$$

which is a function of $\mathbf{u}$ only. This function has a number of interesting properties:

1. It is continuous over its domain.

2. It is concave everywhere over its domain.

3. It is not necessarily differentiable everywhere.

4. Let $\mathbf{x}^*(\mathbf{u})$ be the solution of the minimization (B.11) for some value of $\mathbf{u}$. Suppose also that w is differentiable everywhere. The gradient of w at this point is given by $\nabla w = \mathbf{g}(\mathbf{x}^*)$. When w is not differentiable, the constraint vector is a subgradient of w.

5 The function w satisfies the weak duality theorem. For any pair of feasible solutions $\mathbf{x} \in S$ and $\mathbf{u} \geq 0$, we have $w(\mathbf{u}) \leq f^* \leq f(\mathbf{x})$.

These properties hold under very general assumptions about S, which need not be convex, or for that matter may be finite or denumerably infinite. This is why the theory of duality plays such an important role in combinatorial optimization and also in optimization methods for large systems.

As we said at the beginning of this section, we are interested in calculating the multipliers. This leads us into the definition of the *dual problem*

$$\max w(\mathbf{u}) \qquad \text{(B.12)}$$
$$\mathbf{u} \geq 0$$

where problem (B.2) is now called the *primal problem*. Let $\mathbf{u}^*$ be the optimal solution of the dual. The difference $f(\mathbf{x}^*) - w(\mathbf{u}^*)$ is called the *duality gap*. One question of interest is the size of this gap. It turns out that in general the duality gap does not vanish. If, on the other hand, the primal is a convex problem, at least in a small neighborhood of the solution, then we can show that the duality gap does vanish, and that the optimal value of the dual is precisely equal to the optimal value of the primal. Furthermore, the optimal solution $\mathbf{u}^*$ is precisely the set of multipliers required by the Kuhn-Tucker equations for the primal at that point.

Even when this is not the case, duality has great practical importance in relation to property 5, called the *weak duality* theorem, which states that the value of *any* feasible dual solution is a lower bound on the optimal value. Because many problems are intrinsically difficult to solve, sometimes the calculation of a solution must be terminated before the optimal solution is reached. In such cases, knowing the value of a dual solution can be a useful bound on the error made because of this premature termination.

B.4 Numerical Methods for Unconstrained Problems

Virtually all methods for obtaining a numerical solution to an optimization problem are iterative. Because unconstrained optimization methods are conceptually simpler, and also because they are used as subalgorithms in constrained problems, we first explain several such methods. Most are of the descent type and operate as follows. A starting point $\mathbf{x}^0$ is chosen by some unspecified technique. Let $\mathbf{x}^k$ be the current point at iteration k:

1. Check some optimality condition. In general, we use $|\nabla f(\mathbf{x}^k)| \leq \epsilon$. If the condition is met, stop. If not, proceed to the next iteration.

2. Compute a new value $\mathbf{x}^{k+1}$:

    a. Compute a direction of displacement $\mathbf{d}^k$. This a vector such that for sufficiently small $\epsilon > 0$, we have $f(\mathbf{x} + \epsilon\mathbf{d}) < f(\mathbf{x})$.

b. Compute α^k, the amount of displacement in this direction. Called the *line search*, this consists of finding a minimum of f on the half-line $\mathbf{x}^k + \alpha \mathbf{d}^k$, where $\alpha \geq 0$. The value of α^* at this minimum, called the *step size*, becomes α^k.

c. Compute a new position $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{d}^k$.

We now examine some algorithms frequently used to perform the line search and to calculate directions.

Line Search

The line search is a one-dimensional optimization problem of the form

$$\begin{aligned} \min \ & g(\alpha) = f(\mathbf{x} + \alpha \mathbf{d}) \\ & \alpha \geq 0 \end{aligned} \tag{B.13}$$

where we have dropped the iteration index k for simplicity of notation. Because of this simplified structure, efficient techniques can be used to solve (B.13). The methods are classified according to the type of information used in the minimization, as measured by the order of the derivatives of g that are used.

Zeroth-order methods, also called *interval methods*, do not use any derivative information. The basic assumption is that an initial interval has been determined in which the function is unimodal, that is, has a single minimum. This happens when three increasing values α_0, α_1, and α_2 have been found such that $g(\alpha_1) < g(\alpha_2)$ and $g(\alpha_1) < g(\alpha_0)$. The uniqueness of the minimum is always assumed; if the algorithm fails, this is taken as a sign that the assumption was unjustified. The interval is then reduced at each iteration, while ensuring that it always contains the minimum. We now reintroduce an iteration index l, but with a different meaning. Here l denotes the number of the current iteration when a minimum in the direction $\mathbf{d}$ is being searched for; it must not be confused with the iteration index k, used to denote the iteration number in the search for a minimum in R^n. The particular method used for this interval reduction defines the technique. Simple bisection of the current interval into two equal parts can be used, as can the so-called *golden section method* and the Fibonnaci search, where the interval is divided into unequal parts. This last technique produces the largest relative decrease in the interval for a given number of function evaluations. The golden section method, while not quite optimal, will achieve as good a reduction as that of the Fibonnaci search method, with one additional function evaluation at most.

Other methods construct a quadratic interpolation through the three points, selecting the minimum of the interpolation as the third point required to construct a smaller interval. The convergence rate of these methods is generally linear, often with a rather slow rate near the solution.

First-order methods use the information contained in the first derivative of g. A frequently used technique is the *secant method*, where an iteration is calculated by

$$\alpha^{l+1} = \alpha^l - g'(\alpha^l)\frac{\alpha^l - \alpha^{l-1}}{g'(\alpha^l) - g'(\alpha^{l-1})}. \tag{B.14}$$

Second-order methods use the information contained in the second derivative of g. The most widely known, Newton's method, uses the iteration

$$\alpha^{l+1} = \alpha^l + \frac{g'(\alpha^l)}{g''(\alpha^l)} \tag{B.15}$$

Provided the initial value is close enough to the solution, the convergence is quite rapid when expressed as the number of iterations required to meet some stopping rule. Because each iteration requires the calculation of second derivatives, however, the total time to reach the solution may still be quite large. The method is also subject to oscillation whenever the initial value is not close enough to the solution, and there is no easily computable rule to determine when this is the case.

For all these reasons, in practice the derivative methods and interval-reduction methods are often combined. The latter are used to safeguard against a lack of convergence in the early iterations, while the faster secant or Newton methods are used in the final iterations to obtain an accurate solution faster than can the interval techniques. Also, the line search is generally not performed very accurately, at least in the first stages of the calculation: There is not much point in computing a very accurate solution to the line search since the current value of $\mathbf{x}$ may be far from the optimal solution in any case. Theoretical results state the minimum decrease in f that must be achieved in each line search to guarantee that the overall algorithm will still converge.

Direction Calculation

The second element of descent methods is the computation of a search direction. A number of techniques are available, depending on the order of the derivatives used. All these techniques meet the obvious requirement that the chosen direction must allow a decrease of the objective function in the neighborhood of the current solution. In the following, we denote $\nabla f(\mathbf{x}^k) = \mathbf{g}^k$.

First-order methods use only the gradients, either those at the current iteration or some of the values of preceding iterations. The simplest direction, using $\mathbf{d}^k = -\mathbf{g}^k$, is called the *steepest-descent method*. The reason for this name is that, at the point $\mathbf{x}^k$, the direction $-\mathbf{g}^k$ gives the direction of largest decrease of f, *at least in the neighborhood of* $\mathbf{x}$. Although this method appears simple, it can exhibit an extremely slow convergence near the solution, and thus is almost never used in practice.

The most frequently used first-order method, called the *conjugate-direction method*, uses values of the gradient at the present iteration, modified with the directions used at the previous iteration. The direction is given by

$$\begin{aligned} \mathbf{d}^0 &= -\mathbf{g}^0 \\ \mathbf{d}^k &= -\mathbf{g}^k + \beta_k \mathbf{d}^{k-1} \end{aligned} \tag{B.16}$$

where the choice of β_k determines the particular type of conjugate-direction method used. The most frequent choices are

$$\beta_k = \begin{cases} \dfrac{<\mathbf{g}^k, \mathbf{g}^k>}{<\mathbf{g}^{k-1}, \mathbf{g}^{k-1}>} & \text{for the Fletcher-Reeves method} \\ \dfrac{<\mathbf{g}^k - \mathbf{g}^{k-1}, \mathbf{g}^k>}{<\mathbf{g}^{k-1}, \mathbf{g}^{k-1}>} & \text{for the Polak-Ribière method} \end{cases}$$

The conjugate-direction methods have a number of interesting properties. First, given exact arithmetic, they find the minimum of a quadratic function of n variables in precisely n iterations. This feature is not present in the steepest-descent method; for some quadratic functions, the algorithm converges extremely slowly near the solution. Although this result does not apply to general functions, convergence of the conjugate-direction method is often quite good near a minimum since, at this point, the function is well approximated by its second-order term, and looks very much like a quadratic function. Also, the storage requirements are relatively modest, and, because only vector multiplications are involved, the calculation can be vectorized quite easily, also lending itself to a sparse representation when the problem data is sparse. For these reasons, conjugate-direction methods are currently the only realistic methods for truly large-scale optimization problems that do not have some special structure to be exploited.

Second-order methods achieve faster convergence rates than the conjugate-direction techniques by using information contained in the Hessian matrix of f. The prototype for second-order methods is Newton's method, which can be stated as

$$\mathbf{x}^{k+1} = \mathbf{x}^k - H^{-1}(\mathbf{x}^k)\mathbf{g}^k. \tag{B.17}$$

Note that the method has no line search, but computes a new solution directly from the Hessian matrix and the current point. It is possible to show that the algorithm converges quadratically to the solution, provided it is started close enough to the solution, but there is no computational test to determine this radius of convergence. Even if such a test were possible, the amount of work required per iteration is quite large, since it involves computing the Hessian matrix and inverting it. Unless the function has a simple analytic form, the computation must be done by finite differences, with a corresponding loss of accuracy, and the inversion can become time consuming as the problem size

increases. Although seldom used, Newton's method has been the source of more practical algorithms that can achieve rapid convergence with moderate computation times.

All quasi-Newton methods are based on the fact that, when far from the solution, the exact computation of the inverse of the Hessian matrix is really unnecessary, and that an approximate value for this inverse would probably be quite sufficient. Such methods use the general iteration scheme

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha^k S^k \mathbf{g}^k, \tag{B.18}$$

where S^k is some symmetric, positive-definite matrix that may depend on the iteration. If $S^k = I$, we have the steepest-descent method; if $S^k = H^{-1}$, Newton's method. The quasi-Newton methods lie between these two extremes since they construct successive approximations F^k of the inverse Hessian matrix in such a way that the amount of computation at each iteration is relatively small and that the exact Hessian inverse is obtained after n iterations for a quadratic function of n variables. These methods differ in the way the inverse is updated at each iteration; the two techniques are called *rank-one* and *rank-two* updates — so named because the approximation is updated at each iteration by matrices of rank one or two, respectively. Some frequently used update formulas are as follows:

$$F^{k+1} = F^k + \frac{(\mathbf{p}^k - F^k\mathbf{q}^k)(\mathbf{p}^k - F^k\mathbf{q}^k)^T}{\mathbf{q}^{k\,T}(\mathbf{p}^k - F^k\mathbf{q}^k)} \tag{B.19}$$

or

$$F^{k+1} = F^k + \frac{\mathbf{p}^k\mathbf{p}^{k\,T}}{\mathbf{p}^{k\,T}\mathbf{q}^k} - \frac{F^k\mathbf{q}^k\mathbf{q}^{k\,T}F^k}{\mathbf{q}^{k\,T}F^k\mathbf{q}^k}, \tag{B.20}$$

where

$$\mathbf{p}^k = \mathbf{x}^k - \mathbf{x}^{k-1}$$
$$\mathbf{q}^k = \mathbf{g}^k - \mathbf{g}^{k-1}$$

Eq. (B.19) is called the Davidon-Fletcher-Powell rank-one update formula, while Eq. (B.20) is known as the Broyden-Fletcher-Goldfarb-Shanno rank-two update. In practice, second-order methods can solve problems with a few hundred variables and perhaps tens of nonlinear constraints; the major limitation is the storage requirements of the partial inverse Hessian matrix. It is quite obvious that second-order techniques are inadequate for network problems, where the number of variables can range in the thousands and where the number of constraints is of the same order of magnitude unless some particular simplification related to the problem structure can be used.

B.5 Numerical Methods for Constrained Problems

We now consider numerical algorithms for computing the solution of problems of the type

$$\min f(\mathbf{x})$$
$$g_i(\mathbf{x}) \leq 0$$

We concentrate on the more difficult case, that with inequality constraints; most methods can readily be adapted to the case of equality constraints. Numerical methods for constrained problems fall into two classes: (1) primal methods and (2) relaxation methods. There is a wide variety of each type, and since our purpose is not to give an exhaustive survey, we concentrate on a few successful techniques, those with the best computation times for small and medium problems.

Primal Methods

Primal methods assume an initial feasible solution and maintain feasibility throughout the calculation by ensuring that all the constraints are met at all stages. The obvious advantage is that, if the algorithm must be terminated prematurely, the current solution, being feasible, can be used. The disadvantage is that an initial feasible solution must be found; maintaining feasibility at all times means that the algorithm must follow the boundary of the domain — quite difficult if the constraints are nonlinear. These methods operate as unconstrained algorithms as long as none of the constraints is active — that is, as long as the current solution is inside the domain. The presence of active constraints affects both elements of the descent algorithms: the line search and the direction calculation.

In this section, we assume for ease of exposition that the constraints are linear, of the form

$$<\mathbf{a}^i, \mathbf{x}> \leq b_i,$$

or, in matrix form,

$$A\mathbf{x} \leq \mathbf{b},$$

where A is an $m \times n$ matrix with $m > n$. We assume that A is of rank n, again for simplicity. The theory can be extended to the nonlinear case by considering the tangent plane at the current point, but in practice the numerical solutions are generally much more difficult. Because our interest lies in the conceptual nature of the method, the limitation to linear constraints is not really important; using it clarifies the exposition.

The basic notion underlying primal techniques is that of the *active set*, which is, as indicated by its name, the set of constraints active at the current

point. For a given active set defined by l equality constraints, define A_l as the $l \times n$ matrix of the rows $\mathbf{a}^i$, $i = 1, \ldots l$ of the active constraints. Assume also that the A_l matrix is of rank l. Each active constraint $<\mathbf{a}^i, \mathbf{x}> = b_i$ defines a hyperplane in R^n. The vector $\mathbf{a}^i$ is orthogonal to this plane, and the set of vectors $\mathbf{a}_i = (\mathbf{a}^i)^T$, $i = 1, \ldots l$ defines a subspace $R^l \subset R^n$. The set of hyperplanes can be characterized as the set of vectors $\mathbf{d}$ orthogonal to all of the $\mathbf{a}^i$; it also forms a subspace R^{n-l} of R^n and is the orthogonal complement of R^l. This is expressed by the set of linear equations $<\mathbf{d}, \mathbf{a}^i> = 0$, or, in matrix form, $A_l\mathbf{d} = 0$.

All active-set methods proceed in the same manner:

1. Determine, at the current point, a suitable active set that can be a subset of the active constraints.

2. Minimize f on this active set.

3. At the minimum of f, test some stopping rule. One of three cases can occur:

   a. A previously inactive constraint has become active. In this case, add it to the active set and start another iteration.

   b. It is possible to decrease f by moving away from the surface defined by the current active set. In this case, drop one or more constraints from the set, and start another iteration.

   c. Neither of these cases is true. We have found a point that satisfies the first-order necessary conditions.

Primal methods differ in the way that the first two steps are actually carried out. Two of the most widely used methods are gradient projection and reduced gradient.

The Gradient-Projection Method. In the gradient-projection technique, the gradient of the objective function at the current solution is projected onto the surface defined by the active set, becoming the search direction in this subspace. The direction $\mathbf{d}$ is a vector that lies in R^{n-l}, the subspace orthogonal to the $\mathbf{a}^i$ vectors that define the active set. Recall that any vector of R^n can be expressed as a linear combination of vectors in R^l and its orthogonal complement. This means that we have

$$\mathbf{g} = \mathbf{d} + A^T\mathbf{u} \tag{B.21}$$

for some vector $\mathbf{u} \in R^n$. Because $\mathbf{d}$ is on the surface, we must have $A\mathbf{d} = 0$; replacing in Eq. (B.21), we can solve for $\mathbf{u}$,

$$\mathbf{u} = -(A_l A_l^T)^{-1} A_l \mathbf{g},$$

and replacing in Eq. (B.21), we get for the projection of the gradient on the plane

$$\mathbf{d} = -\left[I - A_l^T(A_l A_l^T)^{-1} A_l\right] \mathbf{g},$$

where the projection operator is defined

$$P_l \triangleq I - A_l^T(A_l A_l^T)^{-1} A_l.$$

This direction is then used to search for a minimum *on the surface*. The line-search algorithm must be modified to recognize two kinds of outcomes:

1. A displacement in the direction would lead outside the domain; in other words, a new constraint has become active.

2. A minimum has been reached on the surface.

In the first case, a new active set is constructed, and the procedure is restarted. The situation is somewhat more complex in the second case, where a minimum has been found on the surface. This happens whenever $\mathbf{d} = 0$. In this case, from Eq. (B.21), we have the conditions

$$\mathbf{g} + \mathbf{u}^T A_l = 0.$$

If $\mathbf{u} \geq 0$, then the Kuhn-Tucker conditions are satisfied, and a point has been found that satisfies the first-order necessary conditions. If, on the other hand, one or more components of $\mathbf{u}$ is negative, then it is possible to find a direction of motion where the function decreases and remains feasible. In this case, constraints corresponding to negative u_is are dropped from the active set, and the procedure is restarted.

Reduced-Gradient Method. The reduced-gradient method is currently the most efficient primal method for linearly and nonlinearly constrained problems. It is best explained in the context of a problem of the form

$$\begin{aligned} \min\ & f(\mathbf{x}) \\ A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq 0 \end{aligned} \tag{B.22}$$

where A is an $m \times n$ matrix with $m < n$. Here too, we assume for simplicity that A is of full rank. Being a primal method, the reduced-gradient method ensures that the optimization proceeds on the constraint surfaces. This is equivalent to saying that, of the n components of $\mathbf{x}$, only $n - m$ are truly independent variables, and that the other m are determined by the requirement that the point $\mathbf{x}$ must satisfy the constraints.

Given a feasible point, the vector $\mathbf{x}$ is partitioned into two parts $\mathbf{y}$ and $\mathbf{z}$, corresponding, respectively, to the dependent and independent variables. This

partition is reflected by a similar partition of the constraint matrix into two parts B and D, the first matrix being $m \times m$ and regular, and the second $m \times n - m$. The B matrix is called the *basis*; the $\mathbf{y}$ variables are thus called the *basic variables*.

The original problem can be rewritten

$$\begin{aligned} &\min f(\mathbf{y}, \mathbf{z}) \\ &B\mathbf{y} + D\mathbf{z} = \mathbf{b} \\ &\mathbf{y}, \mathbf{z} \geq 0 \end{aligned}$$

For a given basis, however, the problem can also be viewed as a minimization on the $\mathbf{z}$ variables only, and can be written

$$\begin{aligned} &\min F(\mathbf{y}(\mathbf{z}), \mathbf{z}) \\ &\mathbf{y} = B^{-1}(\mathbf{b} - D\mathbf{z}) \\ &\mathbf{y}, \mathbf{z} \geq 0 \end{aligned} \tag{B.23}$$

F is minimized over $\mathbf{z}$ by a gradient-search method; the search direction requires the calculation of $\mathbf{G}$, the gradient of F, which is called the *reduced gradient* of the problem. It is not hard to see that

$$\mathbf{G} = \nabla_z f(\mathbf{y}, \mathbf{z}) - \nabla_y f(\mathbf{y}, \mathbf{z}) B^{-1} D \tag{B.24}$$

and that a point satisfies the first-order optimality conditions if

$$G_i \begin{cases} = 0 & \text{if } z_i > 0 \\ \geq 0 & \text{if } z_i = 0 \end{cases}$$

The search proceeds in the space of independent variables as in the gradient-projection method.

The method can be extended to nonlinear equality constraints of the form $\mathbf{h}(\mathbf{x}) = 0$ by assuming that the Jacobian matrix of the transformation $\nabla_y \mathbf{h}(\mathbf{y}, \mathbf{z})$ is regular at the current point. In this case, the reduced gradient has the form

$$\mathbf{G} = \nabla_z f(\mathbf{y}, \mathbf{z}) + \mathbf{u}^T \nabla_z \mathbf{h}(\mathbf{y}, \mathbf{z}), \tag{B.25}$$

where $\mathbf{u}$ satisfies

$$\nabla_y f(\mathbf{y}, \mathbf{z}) + \mathbf{u}^T \nabla_y \mathbf{h}(\mathbf{y}, \mathbf{z}) = 0.$$

In practice, the procedure is similar to the linear case, but computationally is much more difficult. The $\mathbf{z}$ variables change in the direction of the reduced gradient, but the $\mathbf{y}$ variables must move *nonlinearly* to ensure that the constraints $\mathbf{h}(\mathbf{y}, \mathbf{z}) = 0$ are continually met while $\mathbf{z}$ changes. Procedures exist for doing this efficiently, either by moving temporarily away from the surface and then projecting back onto it, or by iteratively resolving the constraint equations by an approximate Newton method at each step during the line search. This last procedure, implemented in one of the most efficient existing codes for nonlinear programming, allows the solution of problems with a few hundred variables and tens of nonlinear constraints.

Relaxation Methods

Relaxation methods attempt to solve a constrained problem by using the well-established algorithms for unconstrained minimization. This is done by constructing a sequence of unconstrained problems whose sequence of solution converges to the solution of the original constrained problem. Relaxation methods differ depending on the way the unconstrained, or relaxed, problems are constructed. Most frequently used are the penalty and barrier methods, the dual methods, and the multiplier method.

Penalty and Barrier Methods. Penalty and barrier techniques are the oldest used to solve constrained problems. They were developed in the 1960s, when the theory of constrained optimization was relatively undeveloped and almost no practical algorithmic implementations existed. The main difference between penalty techniques and barrier techniques is that the former may leave the domain, while the latter are strictly feasible throughout the calculation.

In the case of the penalty method, each unconstrained problem is constructed by removing the constraints and adding a penalty term $P(\mathbf{x})$ to the objective function that somehow represents the "cost" of violating the constraints. Intuitively, it is clear that, if this penalty becomes large enough, the solution of the unconstrained problem should be feasible; if the penalty term was chosen correctly, the solution should be close to that of the original problem. At iteration k, the unconstrained problem is of the form

$$\min f(\mathbf{x}) + C^k P^k(\mathbf{x}),$$

where the penalty function must satisfy the following requirements:

1. P is continuous.
2. $P(\mathbf{x}) \geq 0$ for all $\mathbf{x}$ outside the domain.
3. $P(\mathbf{x}) = 0$ if and only if $\mathbf{x}$ is inside the domain.

A typical penalty function could be of the form $P(\mathbf{x}) = \sum_i \max\,[0, g_i(\mathbf{x}]^2/2$. Given a set of weights $C^{k+1} > C^k$, it is possible to show that the limit of the sequence of solutions to the unconstrained problems converges to the solution of the original problem.

Barrier methods operate on the same principle but are limited to domains with an interior. The subproblems are of the form

$$\min f(\mathbf{x}) + \frac{1}{C^k} B(\mathbf{x}),$$

where $C^k \to \infty$ and the barrier term $B(\mathbf{x})$ is now defined in such a way that

1. $B(\mathbf{x})$ is continuous.

2. $B(\mathbf{x}) \geq 0$.

3. $B(\mathbf{x}) \to \infty$ as $\mathbf{x}$ approaches the boundary of the domain.

For a domain defined by the set of inequalities $g_i(\mathbf{x}) \leq 0$, a typical barrier function could be $B(\mathbf{x}) = -\sum_i 1/g_i(\mathbf{x})$. Clearly, if the initial solution to the first subproblem is feasible, then all solutions will remain feasible. Here again, the appropriate choice of the barrier term guarantees that the limit point of the unconstrained solutions converges to the optimal solution of the original problem.

Penalty and barrier methods have the advantage that they are easy to program, given the existence of an unconstrained minimization algorithm. Because the weight of the penalty or barrier term must increase with the number of iterations, individual problems become increasingly ill conditioned and difficult to solve numerically. As a consequence, the number of iterations is generally small, and the accuracy of the final solution may not be very high.

Dual Methods and the Subgradient Algorithm. The process by which the constraints are incorporated in the objective function to construct the subproblems is strongly reminiscent of the way the Lagrange function is built. In fact, the only difference is that the coefficients of the penalty or barrier terms are not the multipliers, but some increasing value more or less arbitrarily chosen.

The theory of duality can be used to compute the solution of a convex constrained problem by solving the dual, taking the solution obtained in the minimization of the Lagrangian as the solution of the primal. More precisely, we solve

$$\max_{\mathbf{u} \geq 0} w(\mathbf{u})$$

Calculating the value of the dual function for some value of the multipliers $\mathbf{u}$ by

$$w(\mathbf{u}) = \min_{\mathbf{x} \in S} \mathcal{L}(\mathbf{x}, \mathbf{u})$$

we let $\mathbf{x}^*(\mathbf{u})$ be the optimal solution of the minimization for the current value of $\mathbf{u}$. It is possible to show that if $\mathbf{u}^*$ is the optimal solution of the dual, then $\mathbf{x}^*(\mathbf{u}^*)$ is the optimal solution of the primal.

This method is not generally useful for an arbitrary problem because evaluating the dual function can be quite difficult, and also because $w(\mathbf{u})$ may not be differentiable everywhere. In this case, the solution of the dual can be computed only by methods of nondifferentiable optimization, which require long computation times. The approach has some advantages when a special structure can be exploited.

One such case is the so-called *separable* problem. Typically, a separable problem is of the form

$$\min \sum_{j=1}^{l} f_j(\mathbf{x}_j)$$

$$\sum_{j=1}^{l} \mathbf{g}_j(\mathbf{x}_j) \leq 0$$

$$\mathbf{h}_j \leq 0$$

The structure is such that the vector of variables $\mathbf{x}$ can be partitioned in l sets of vectors $\mathbf{x}_j$; similarly, each constraint function is the sum of constraint functions defined over these sets. In this case, the calculation of the dual function becomes

$$w(\mathbf{u}) = \sum_{j=1}^{l} \min \; f_j(\mathbf{x}_j) + <\mathbf{u}_j, \mathbf{g}_j(\mathbf{x}_j)>$$

$$\mathbf{h}_j \leq 0$$

In other words, the evaluation of the dual function separates into l independent minimization problems. In many cases, this reduction in complexity may justify recourse to the dual approach, even though the dual problem itself may be harder to solve.

Another frequent situation in this context occurs when a very precise solution is not required — for instance, when the actual data for the problem is not known with high accuracy. It thus does not make much sense to compute a solution to many significant digits; an approximate solution is often sufficient. In this case, the dual problem is not solved very accurately; rather, a crude search technique is used. In this method, called the *subgradient* method, a direction of ascent is given by $\mathbf{g}(\mathbf{x})$ at each value of $\mathbf{u}$, where $\mathbf{x}$ is the solution obtained in calculating the dual function. Instead of trying to find a maximum of w in this direction, a single step is taken, given by

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \alpha_k \mathbf{g}^k$$

$$\alpha_k = \frac{\lambda_k[w^* - w(\mathbf{u})]}{||\mathbf{g}(\mathbf{x})||^2}$$

$$0 < \epsilon_1 < \alpha_k \leq 2 - \epsilon_2$$

$$\epsilon_2 > 0$$

Although no attempt is made to find a maximum in the direction, it is still possible to prove convergence of the method under the conditions stated. In practice, w^* is not known, and an approximation is used. The method is particularly useful when the dual calculation is separable, as explained earlier,

and when there exists a fast algorithm to compute a feasible solution to the primal $\mathbf{x}'$. In this case, we have, by the weak theorem of duality, $w(\mathbf{u}) \leq f^* \leq f'$. We can use f' to approximate w^* in the step-size calculation, and use the gap $f' - w$ as a stopping rule. The value of this gap is an upper bound on the error made by stopping the algorithm before optimality is reached; if small enough, it indicates that the suboptimal solution thus computed may be sufficient for practical purposes. This method has found widespread application in the areas of integer programming and of large-scale nonlinear problems with structure, for which it is probably the most efficient technique.

Multiplier Method. The preceding ideas have been extended to the multiplier method, a synthesis of the penalty and barrier methods on the one hand, and of Lagrangian techniques on the other, without the drawbacks of either approach. The multiplier method does not suffer from the stability problems of the standard penalty method; all the minimization subproblems are differentiable and can be solved by standard unconstrained techniques.

In the multiplier method, a penalty is added to the Lagrangian instead of to the function itself. The general formulation for problems with equality constraints

$$\begin{aligned} &\min\ f(\mathbf{x}) \\ &h_i(\mathbf{x}) = 0 \end{aligned}$$

is to solve a sequence of subproblems of the form

$$\min\ L(C^k, \mathbf{x}, \mathbf{u}^k) = f(\mathbf{x}) + \sum_i u_i^k h_i(\mathbf{x}) + \frac{1}{2} C^k \sum_i [h_i(\mathbf{x})]^2. \tag{B.26}$$

The advantage is that one can prove convergence with a *finite* weight of the penalty term, making the subproblems much easier to solve. The case with inequality constraints can be reduced to formulation (B.26) through the use of slack variables.

Multiplier methods, along with the reduced-gradient technique, have the best numerical performance for small- to medium-size problems.

B.6 Linear Programming

The theory and practice just described take a particularly simple form when the objective and constraint functions are linear. Linear programming dates back to the beginning of operations research during the 1940s; there exists a very large literature on the subject. We thus do not pursue the subject in any depth, but instead point out certain features that are used in this text. The problem is written

$$\begin{aligned} \min\ z &= <\mathbf{c}, \mathbf{x}> \\ A\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq 0 \end{aligned} \tag{B.27}$$

which is known as the standard form of linear programming. Here, A is an $m \times n$ matrix with $m < n$ and of rank m.

The domain of the linear program is convex, since it is the intersection of m half-planes and the positive orthant. In the discussion that follows, we assume that the set of constraints is feasible, that the minimum is finite, and that the constraints are not redundant. In practice this is unnecessary, but it simplifies the exposition of the basic principles required here.

The objective function, being linear, is both convex and concave. For this reason, we know that any local minimum of the problem is also a global minimum and that this minimum occurs on at least one of the extreme points of the domain. At such a point, the m inequalities are tight, which means that m of the variables are determined by this system of equations and that $n - m$ are undetermined. The partition of the vector $\mathbf{x}$ in these two sets determines what is called a *basis*, where $\mathbf{x}$ is the concatenation of two vectors: (1) a vector of *basic variables* $\mathbf{x}_B$ of m components and (2) a vector of *nonbasic variables* $\mathbf{x}_D$ of $n - m$ components. The coefficient matrix A is partitioned in the same way by regrouping the columns corresponding to $\mathbf{x}_B$ into an $m \times m$ regular submatrix B, and the other columns in an $n - m \times n$ matrix D. A similar partition can be made for the cost vector, which is written $\mathbf{c}_B$, $\mathbf{c}_D$. The system of equations corresponding to the constraints can then be written

$$B\mathbf{x}_B + D\mathbf{x}_D = \mathbf{b}. \tag{B.28}$$

A *basic solution* is defined as a solution of the linear system where $\mathbf{x}_D = 0$. In this case, the basic variables are simply given by

$$\mathbf{x}_B = B^{-1}\mathbf{b}, \tag{B.29}$$

which is always possible since B is regular by definition. Note, however, that the solution may have negative components $x_i < 0$. A basic solution is said to be *feasible* if it does meet the inequality constraints $\mathbf{x} \geq 0$.

This concept of a basic feasible solution is central to the theory of linear programming. The fundamental theorem states that to each extreme point of the constraint polytope there corresponds a basic feasible solution, and that the converse is also true. Two extreme points are said to be *adjacent* if they differ by a single component of their basic vector. Because of the concave nature of the objective function, we know that it is possible to find an optimal solution by examining only the extreme points; this theorem gives a simple algebraic characterization of them. The solution of a linear program is therefore reduced to the search over the basic feasible solutions — a considerably easier task than a full search over the domain.

A simple enumeration algorithm would not be practical since the number of bases is of the order of $\binom{n}{m}$, which grows very rapidly with the number of variables and constraints. The most widely used algorithm for finding an

optimal solution, called the *simplex algorithm*, can be viewed as the application of the steepest-descent method to the linear program (B.27). The simplex method explores the set of extreme points of the polytope, but proceeds in directions of steepest descent at each iteration, remaining feasible at all times once a feasible solution is found. Given a basic feasible solution $\mathbf{x}_B$, $\mathbf{x}_D = 0$, the algorithm computes another basic feasible solution with a lower value of the objective function. It does this by replacing one of the current basic variables x_i with one of the nonbasic variables x_j.

The first step consists of choosing the nonbasic variable x_j, called the *candidate* variable. It is chosen such that the marginal decrease of the objective function is the largest possible at the current point. In this sense, the simplex algorithm behaves as the steepest-descent method for general nonlinear programming. Recall that the current solution is a basic feasible one. Hence $x_j = 0$ at the beginning, and making it basic means that its value *increases*. Because the solutions must remain feasible, the linear system (B.28) must always be verified. It can be written as

$$\mathbf{x}_B = B^{-1}\mathbf{b} + \sum_j x_j B^{-1}\mathbf{a}_j.$$

Since only one nonbasic variable is being changed, it becomes

$$\mathbf{x}_B = B^{-1}\mathbf{b} - x_j B^{-1}\mathbf{a}_j, \tag{B.30}$$

where $\mathbf{a}_j$ is the column of A corresponding to x_j. By replacing the basic variables, the objective function can also be written as a function of x_j, yielding

$$z(x_j) = <c_B, B^{-1}\mathbf{b}> + \left[c_j - <\mathbf{c}_B, B^{-1}\mathbf{a}_j>\right] x_j. \tag{B.31}$$

The term $c_j - <\mathbf{c}_B, B^{-1}\mathbf{a}_j>$, called the *reduced cost* for the nonbasic variable x_j, is denoted r_j. This reduced cost can also be written

$$\begin{aligned} r_j &= c_j - <\mathbf{c}_B, B^{-1}\mathbf{a}_j> \\ &= c_j - <\mathbf{c}_B B^{-1}, \mathbf{a}_j> \\ &= c_j - <\mathbf{u}, \mathbf{a}_j>. \end{aligned} \tag{B.32}$$

The vector $\mathbf{u} \triangleq \mathbf{c}_B B^{-1}$, called the *shadow-price* vector for the current basis, is intimately related to the dual of the linear program. The objective function changes linearly with the nonbasic variable at the rate r_j. Now, since we want to minimize z, we should choose the nonbasic variable that has the smallest such reduced cost. If the smallest $r_j \geq 0$, then all nonbasic variables have a nonnegative rate, and the current solution is a local — and by the convexity of the problem, a global — minimum. If some $r_j < 0$, then the j for which the minimum occurs is chosen as the variable that will become basic, and is called the *entering* variable.

Having decided which nonbasic variable will become basic, we must now decide what value it should take. This is determined again by the requirement that the system (B.28) must remain feasible as x_j increases. Rewriting the system (B.30) for the ith component, we have

$$x_i = (B^{-1}\mathbf{b})_i - x_j \left(B^{-1}\mathbf{a}_j\right)_i, \tag{B.33}$$

where the subscript i indicates that we are interested in the ith component of the vector in question. Here again, we see that each basic variable varies linearly with the nonbasic variable, with coefficient $z_i = -\left(B^{-1}\mathbf{a}_j\right)_i$. Clearly, if $z_i \geq 0$ for all values of i, all basic variables will increase as x_j is increased, and x_j can be increased without limit. Since the effect of increasing x_j is to decrease z, however, the value of the linear program is unbounded. If, on the other hand, some of the z_is are positive, then some of the basic variables will decrease when x_j increases. Because we want to maintain feasibility of the solution, the basic variables can only decrease to zero. In this case, the value of x_j is limited by Eq. (B.33), which must be maintained for all i. Setting $x_i = 0$ in this equation, we get the largest value for x_j permitted by this basic variable, which is

$$x_j(i) = \frac{\left(B^{-1}\mathbf{b}\right)_i}{\left(B^{-1}\mathbf{a}_j\right)_i}.$$

The value of x_j will be given by $\min_i \; x_j(i)$. At this value, $x_i = 0$ and becomes the nonbasic variable, and $x_j > 0$ becomes the new basic variable.

The third and final step is to recompute the inverse basis B^{-1}, which is needed for the next iteration. In theory, this poses no difficulty since the columns of the basis are known from the set of basic variables. In practice, very efficient numerical methods can be used to compute the new value of B^{-1} since this is the inverse of a matrix that differs from the basis of the previous iteration by the interchange of a single column. Given the inverse of the basis at the previous iteration, it is possible to recompute a new inverse much more efficiently than by standard Gaussian elimination, with corresponding efficiency in the numerical solution of linear programs.

Linear Programming Duality

The general theory of duality takes a particularly simple form in the context of linear programming. Given the primal problem (B.27), we can construct the Lagrangian by incorporating the linear constraints into the objective function

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = <\mathbf{c}, \mathbf{x}> + <\mathbf{u}, \mathbf{b} - A\mathbf{x}>. \tag{B.34}$$

The dual function is then given by

$$w(\mathbf{u}) = <\mathbf{u}, \mathbf{b}> + \min_{x \geq 0} \; <\mathbf{c} - \mathbf{u}A, \mathbf{x}>.$$

The solution of this minimization is

$$x_i = \begin{cases} 0 & \text{if } (\mathbf{c} - \mathbf{u}A)_i > 0 \\ \infty & \text{if } (\mathbf{c} - \mathbf{u}A)_i < 0 \\ \text{arbitrary} & \text{if } (\mathbf{c} - \mathbf{u}A)_i = 0 \end{cases}$$

The second case is interpreted by saying that the domain of the dual problem should be restricted to the set $\mathbf{u}A \leq \mathbf{c}$ since the value of the dual function is $w(\mathbf{u}) = -\infty$ outside this set. Thus the dual problem becomes

$$\max\ w(\mathbf{u}) = <\mathbf{u}, \mathbf{b}>$$
$$\mathbf{u}A \leq \mathbf{c}$$

The dual problem has a number of symmetries with the primal. Like the primal problem, it is a linear program, and can be solved by the simplex algorithm. In fact, it can be shown that the shadow prices $\mathbf{u}$ at the optimal solution are identical to the optimal values of the variables of the dual problem. The objective is maximization, and the cost and constraint vectors are inverted from one problem to the other. Note also that the number of rows of the dual is the number of variables of the primal, and vice versa. Given that the computation time of the simplex algorithm depends more strongly on the number of constraints than on the number of variables, it may be preferable to solve the dual rather than the primal problem whenever the latter has a large number of constraints.

Network Flows

One class of linear problems has a special structure that can be exploited in numerical algorithms. These problems are related to the transportation of material in a network from a given source node to another destination, or *sink node*. In practice, the same structure may appear in contexts totally unrelated to transportation. Whenever possible, it is advantageous to transform a particular linear problem to exhibit the presence of network flows.

In this case, the variables, denoted f_{ij}, represent the amount of material carried on the arc (i, j) in the network. The flow domain $\mathcal{F}$ is defined by the constraints

$$\sum_j f_{ij} - \sum_j f_{ji} = \begin{cases} v & \text{if } i \text{ is the source} \\ -v & \text{if } i \text{ is the sink} \\ 0 & \text{otherwise} \end{cases}$$
$$f_{ij} \geq 0 \tag{B.35}$$

The conservation equations can be put in matrix form $\mathcal{I}\mathbf{f} = \mathbf{v}$ by using the node-arc incidence matrix $\mathcal{I}$, defined as

$$\mathcal{I}_{i,j} = \begin{cases} +1 & \text{if arc } (i, j) \text{ goes out of node } i \\ -1 & \text{if arc } (i, j) \text{ goes into node } i \\ 0 & \text{otherwise} \end{cases}$$

In this expression, the sum over j is taken over all arcs having their origin at i and their end node at j. The first set of equations is called the *conservation constraints*.

In practice, some physical limitation is often imposed on the amount of material that can go through an arc. This is modeled by the set of capacitated flows $\mathcal{F}_c$, which is given by

$$\begin{aligned} \mathbf{f} &\in \mathcal{F} \\ f_{ij} &\leq C_{ij} \end{aligned}$$

Typical problems that arise in this context are the maximum-flow problem,

$$\max_{\mathbf{f} \in \mathcal{F}_c} v(\mathbf{f})$$

and the minimum-cost flow problem

$$\min_{\mathbf{f} \in \mathcal{F} \quad \text{or} \quad \mathbf{f} \in \mathcal{F}_c} \sum_{ij} a_{ij} f_{ij}$$

where $a_{i,j}$ is the transportation cost per unit flow on arc (i, j). The obvious solution to the uncapacitated minimum-cost flow problem is to have all the flow on the path with minimum cost from source to sink. This can be computed easily by using any one of the many shortest-path algorithms, with the unit costs a_{ij} as the length of the arc. Since the problem is a linear problem, its solutions lie on extreme points of the uncapacitated-flow domain $\mathcal{F}$. For this reason, we see that the extreme points of $\mathcal{F}$ are defined by the shortest paths from source to sink.

Although the constrained problems are more difficult to solve, there are specialized versions of the simplex algorithm [6] that allow solutions to be calculated for networks with tens of thousand of nodes and arcs — substantially larger than can be solved by the general-purpose simplex algorithm.

The single-commodity flow problems also have an important property that is not present in linear programs in general. Assume that all the data (capacities, costs, and requirement) is integer. For a general linear program, this is no guarantee that the solution of the simplex algorithm will have only integer values; in general, it will not. In the case of network flow, however, integer solutions appear naturally from the simplex method, without the necessity of using special integer programming techniques. This is very important in the many cases where the flow variables represent technical systems that must be represented by whole numbers.

Flow models are useful to represent the transportation of a single item in a network, but are generally insufficient in many applications. This happens whenever more than one item, or commodity, must use the network. In this

case, a set of flows must be defined, one for each commodity k. The equations that define the domain of multicommodity flows $\mathcal{F}_m$ are quite similar:

$$\sum_j f_{ij}^k - \sum_j f_{ji}^k = \begin{cases} v^k & \text{if } i \text{ is the origin of commodity } k \\ -v^k & i \text{ is the sink of commodity } k \\ 0 & \text{otherwise} \end{cases}$$

$$f_{ij}^k \geq 0$$

The capacitated version $\mathcal{F}_{mc}$ has the additional constraints that link the commodities together:

$$\sum_k f_{ij}^k \leq C_{ij}.$$

The most frequent optimization problem is that of minimum cost,

$$\min \sum_{ij} \sum_k a_{ij}^k f_{ij}^k$$

$$\mathbf{f} \in \mathcal{F}_m \quad \text{or} \quad \mathbf{f} \in \mathcal{F}_{mc}$$

where the unit transportation cost $a_{i,j}^k$ is different for each link and each commodity. The solution of the uncapacitated version is once again to route each commodity on the least-cost path from its source to its sink; this can be calculated by a shortest-path algorithm. In the capacitated version, however, the solution is much more difficult. In fact, there is currently no specialized algorithm that will perform substantially better than the simplex method when it is tailored to sparse problems (which is the case for the minimum-cost multicommodity flow). Also, the integrality of solutions no longer holds, and special integer programming methods must be used whenever these integer requirements are important.

References

[1] Luenberger, D.G., *Linear and Nonlinear Programming*, Addison-Wesley, 1984.

[2] Shapiro, J. F., *Mathematical Programming: Structures and Algorithms*, Wiley, 1979.

[3] Lawler, E., *Combinatorial Optimization, Networks and Matroids*, Holt, Rinehart & Winston, 1976.

[4] Papadimitriou, C.H., and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, 1982.

[5] Salkin, H.M., *Integer Programming*, Addison-Wesley, 1975.

[6] Grigoriadis, M.D., and Hsu, T., "RNET: Rutgers minimal cost network flow subroutines — user documentation," Report No. V 3.61, Department of Computer Science, Rutgers University, October 1979.

APPENDIX C

Notation and List of Symbols

In this appendix we collect the main conventions for notation used in this book. We also include an alphabetic list of symbols used most frequently.

C.1 Notation

Vectors are denoted in boldface $\mathbf{x}$, and matrices in uppercase italic A. Unless absolutely necessary, we do not explicitly distinguish between row and column vectors, and do not use the transpose notation to simplify notation. Thus the expression $A\mathbf{x}$ denotes the product of the vector $\mathbf{x}$ by the matrix A; we assume that the matrix and vectors have the adequate dimensions. In this case, the assumption is that $\mathbf{x}$ is a column vector.

It is sometimes necessary to use row vectors, especially for matrix multiplications. Most such cases are recognizable because the vector appears on the left of the matrix, as in $\mathbf{x}A$, which represents the left multiplication of the row vector $\mathbf{x}$ by the matrix A. This is equivalent to $A^T\mathbf{x}$, where now $\mathbf{x}$ is a column vector, but avoids the need for yet another superscript in situations where the vector may already have many others.

The ith component of $\mathbf{x}$ is denoted by x_i, the jth column of matrix A by $\mathbf{a}_j$, and the ith row by $\mathbf{a}^i$. Recall that the product $A\mathbf{x}$ can be read $\sum_i x_i\mathbf{a}_i$ — that is, as the linear combination of the columns of A, with the components of $\mathbf{x}$ as the coefficients. When it is necessary to identify a particular component of the vector $A\mathbf{x}$, we denote this by $(A\mathbf{x})_i$. The inner product of two vectors is denoted $<\mathbf{x},\mathbf{y}>$. Vector inequalities or equalities hold component-wise.

Many variables appear in a network context, where it is often necessary to indicate to which arc the variable belongs, which commodity it represents, or both. This is simplified as much as possible by the following convention for indices. Many variables in networks are double-index variables, referring either to arcs in a graph or to some matrix element. The double-index notation is sometimes essential, and sometimes too cumbersome. Something described by two indices may be described equally well with a single index if we assume that there is a mapping from one representation to the other. For instance, we may

denote the flow on the arc (i, j) of a graph by $f_{i,j}$, or, if we know that arc (i, j) is the kth in some given list of arcs, by f_k. In the text, we assume that such a unique transformation always exists, and we use the single and double indices freely according to the needs of the exposition.

In general, superscripts refer to a commodity, and subscripts to a link. Thus $f_{i,j}^{k,l}$ denotes the flow of commodity having its source at k and its sink at l on the link (i, j). This same quantity could as well have been denoted f_s^m, or $f_s^{k,l}$ if it is important for the discussion to know the origin and destination of the flow, but not the origin and extremity of the arc in question.

A similar simplification is made for denoting paths in networks. A path from i to j can be specified by giving the list of nodes encountered on the path, as in $(i, k_1, k_2, \ldots k_l, j)$, or the list of arcs $((i, k_1), (k_1, k_2), \ldots (k_l, j))$. Using the simplified notation for arcs, we could denote it as well by $(s_1, s_2, \ldots s_l)$. In many cases, it is not useful to know the precise composition of the path. In this case, we assume that all paths have been placed in a list, referring to the kth path between i and j as (i, j, k). Even more generally, if the particular origin and destination nodes are not needed, we may assume that there is a global list of all paths in the network, (or of all paths of interest in the particular situation) and refer to the kth path.

There is another convention regarding traffic and the relation between offered, carried, and overflow traffic. If x is a symbol representing traffic, then the variable x by itself denotes offered traffic, $\overline{x}$ denotes carried traffic, and $\hat{x}$ denotes blocked, or overflow, traffic. This convention applies to link, path, or route traffic, with the appropriate notation in each case.

C.2 List of Symbols

Here we list the symbols that occur regularly throughout the text. A particular symbol may be used with a different meaning in some short section, in which case this is indicated in the text. The reason is the convenience of conforming with specialized notations that are customary in certain areas.

$A^{i,j}$ = The value of the external traffic from i to j. This is the intensity of the Poisson process that describes the arrival of new calls into the network.

$A_k^{i,j}$ = The value of the external traffic from i to j that is offered to the kth path available to this stream.

$a_{s,t}$ = The total traffic offered to link s, t. This is generally made up of two components, direct trafic $A^{s,t}$ and internal traffic, that is, calls that either originate at a node other than s or terminate at a node other than t.

$a_{s,t}^{i,j}$ = The traffic offered to link (s,t) that belongs to the stream (i,j). This is often called a traffic *parcel*.

B_s = The blocking probability of link s. This is the probability that all circuits on link s are occupied.

B'_s = The probability of reaching or exceeding the protection threshold on link s. This is the probability that m or more circuits on link s are occupied.

C = A cost (most often). It can be a function of the group size, or a constant when the cost model is linear. It is also used occasionally in the discussion of flow models to denote the arc capacity.

$d(k)$ = The destination (sink) node of commodity k.

$E(A,N)$= The Erlang B function.

f = A flow variable or a generic function.

$F(t)$ = A distribution of interarrival times. Its density is represented by $f(t)$.

H = The marginal occupancy on a link. This is the incremental amount of traffic that can be carried on a link for a small increment of capacity, given a constant offered traffic.

L^k = The end-to-end blocking probability for stream k. This is the probability that a k-call cannot be connected.

M = The mean of a traffic. In some cases, this can also be represented by the lowercase m.

m = The protection level of a link. This is the number of busy circuits at which some calls begin to be blocked.

N_s= The number of circuits on link s.

$o(k)$ = The origin (source) of commodity k.

$P_k^{i,j}$ = The probability of overflow to the kth path in the route tree of stream (i,j).

Q = The connection probability; $1-P$.

p_k = The probability of being in state k for a stochastic process.

r_s = The reservation level for link s. We always have $r = N - m$.

$r_{i,j}$ = The reward for the transition from state i to state j in a Markov model.

R_s = The residual capacity of link s. This can also be used for paths, with the appropriate change in indices.

V = The variance of a traffic. In some cases, this is represented by the lowercase variable v.

$\mathbf{x}$ = A path-flow vector. This is the flow carried on all paths in a network.

$\mathbf{u}, \mathbf{v}, \mathbf{w}$ = Vectors of Kuhn-Tucker multipliers.

$\mathcal{I}$ = The incidence matrix of a network.

$\mathcal{L}$ = The Lagrange function, also called the Lagrangian.

$\mathcal{P}_k^{i,j}$ = The probability that the kth path available to stream (i, j) is blocked.

$\alpha_k^{i,j}$ = The load-sharing coefficient for the kth path available to stream (i, j). This is also the tandem selection probability in residual capacity adaptive routing.

β = The marginal capacity of a link. This is the increment of traffic that can be offered to a link for a small increment in the link size such that the blocking probability remains constant.

γ = The marginal overflow of a link. This is the increment of overflow traffic corresponding to a small increase in the offered traffic, for a given constant link size.

$\delta_{i,j}$ = The Kroenecker symbol. $\delta_{i,j} = 1$ if $i = j$, and 0 otherwise.

λ_k = The arrival rate when the system is in state k for an arrival process.

μ_k = The average service rate when the system is in state k. In other words, $1/\mu$ is the average service time.

$\Phi(s)$ = The Laplace-Stieltjes transform of the distribution $F(t)$. The corresponding transform for the density is denoted by $\phi(s)$.

APPENDIX *D*

Numerical Data

In Figs. D.1–D.7, this appendix provides the information necessary to do those exercises that require numerical calculations. The information is by no means intended for real design problems. The reason is that the data is applicable only to the case of Poisson traffic — not a realistic approximation for all hierarchical networks and some nonhierarchical networks. Also, as cheap microcomputers are being introduced, manual network design is rapidly becoming obsolete; such design should be done only for educational purpose, as in the present case.

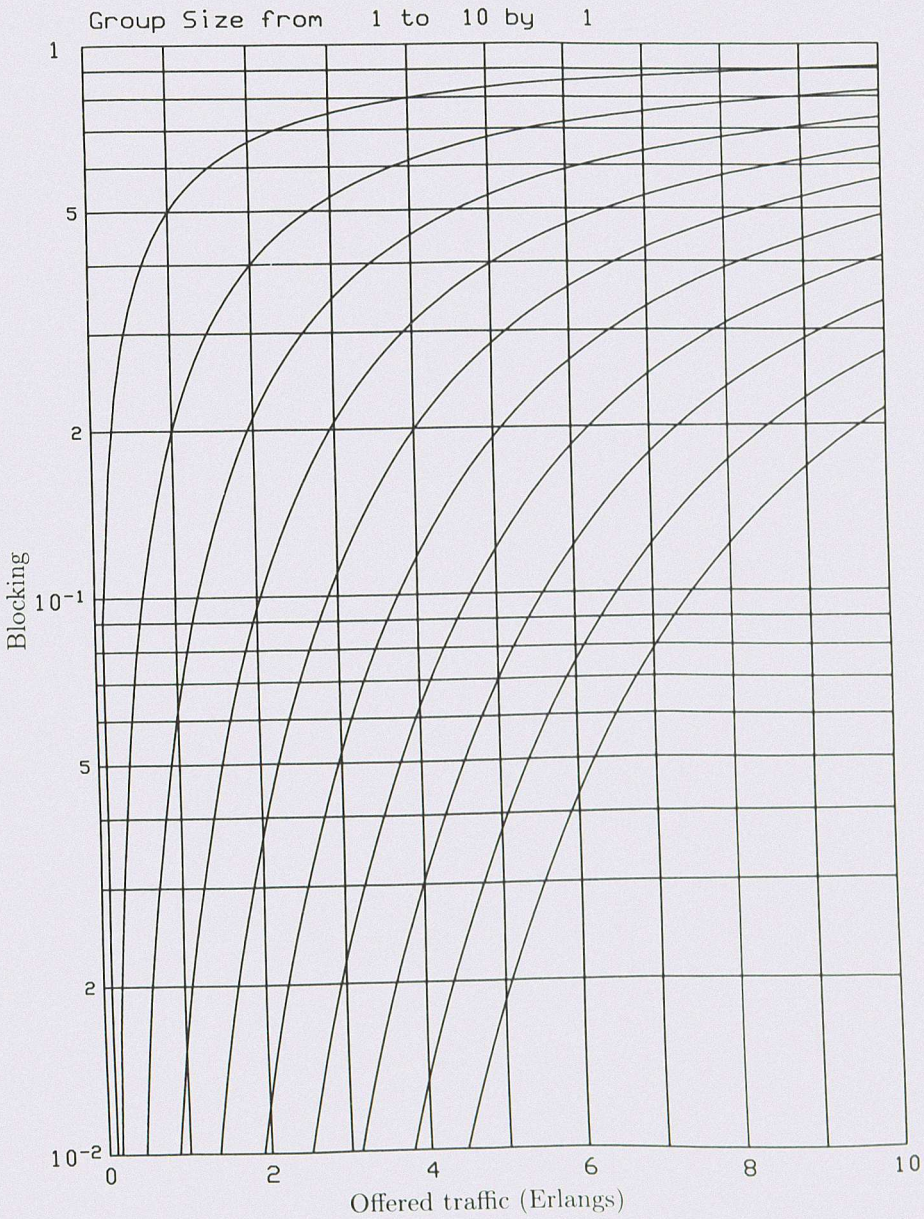

Figure D.1 Erlang B Function

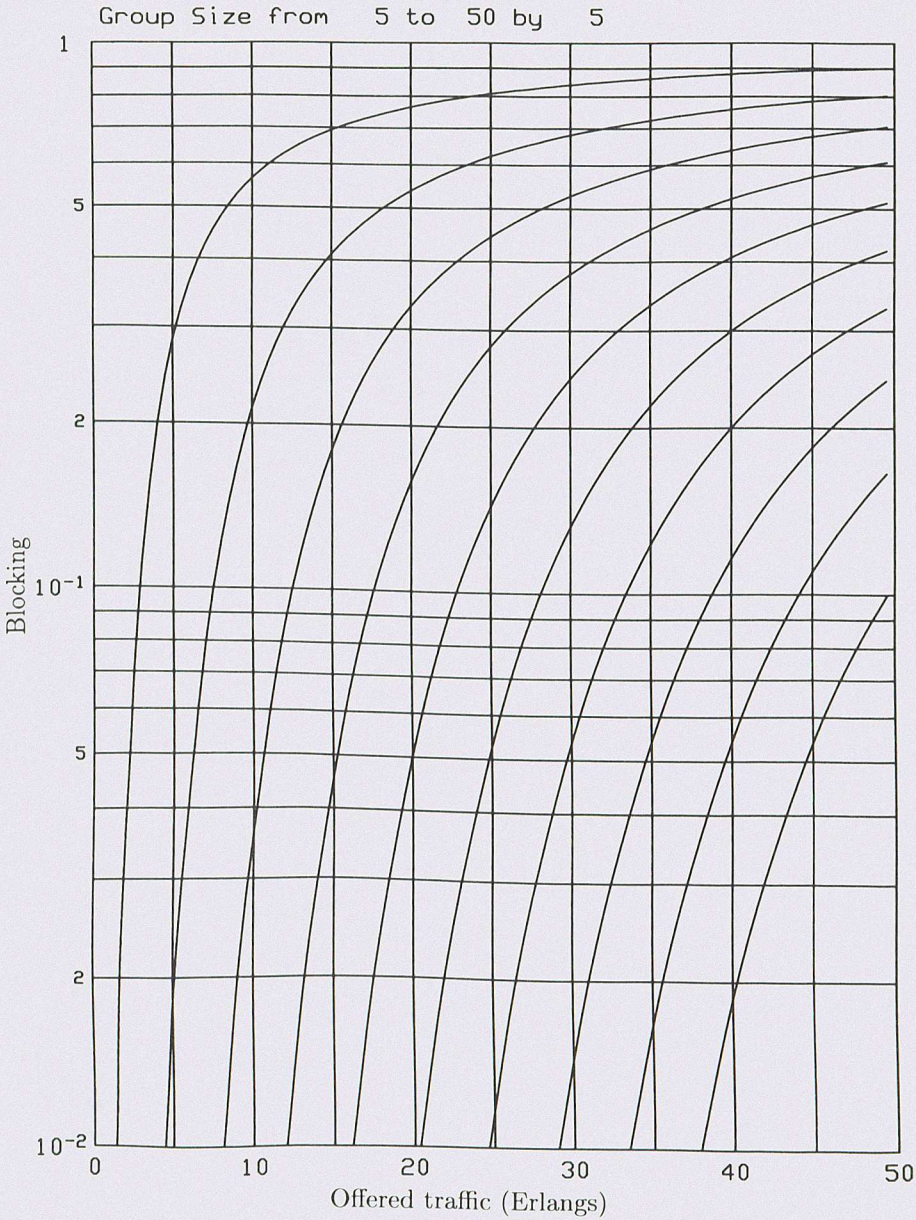

Figure D.2 Erlang B Function

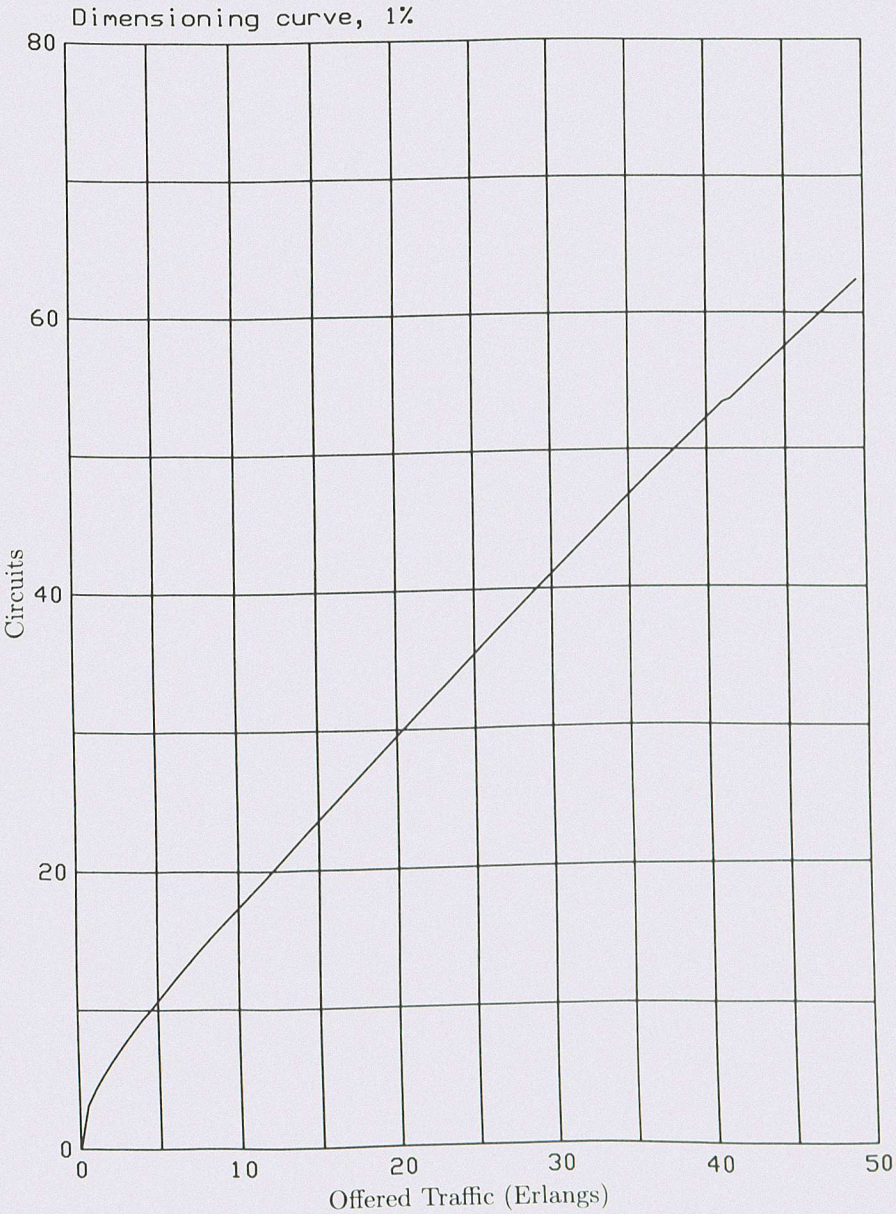

Figure D.3 Dimensioning Curve

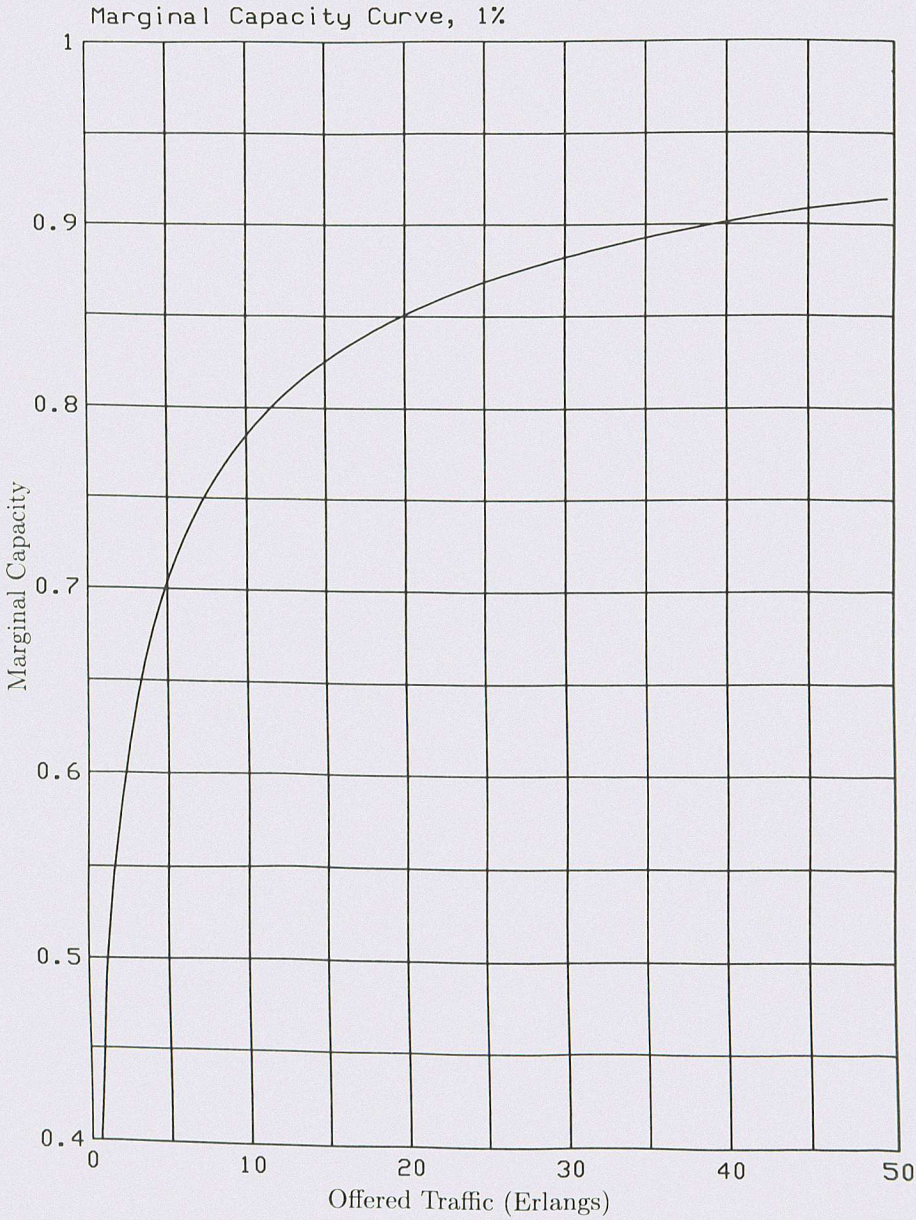

Figure D.4 Marginal Capacity Curve

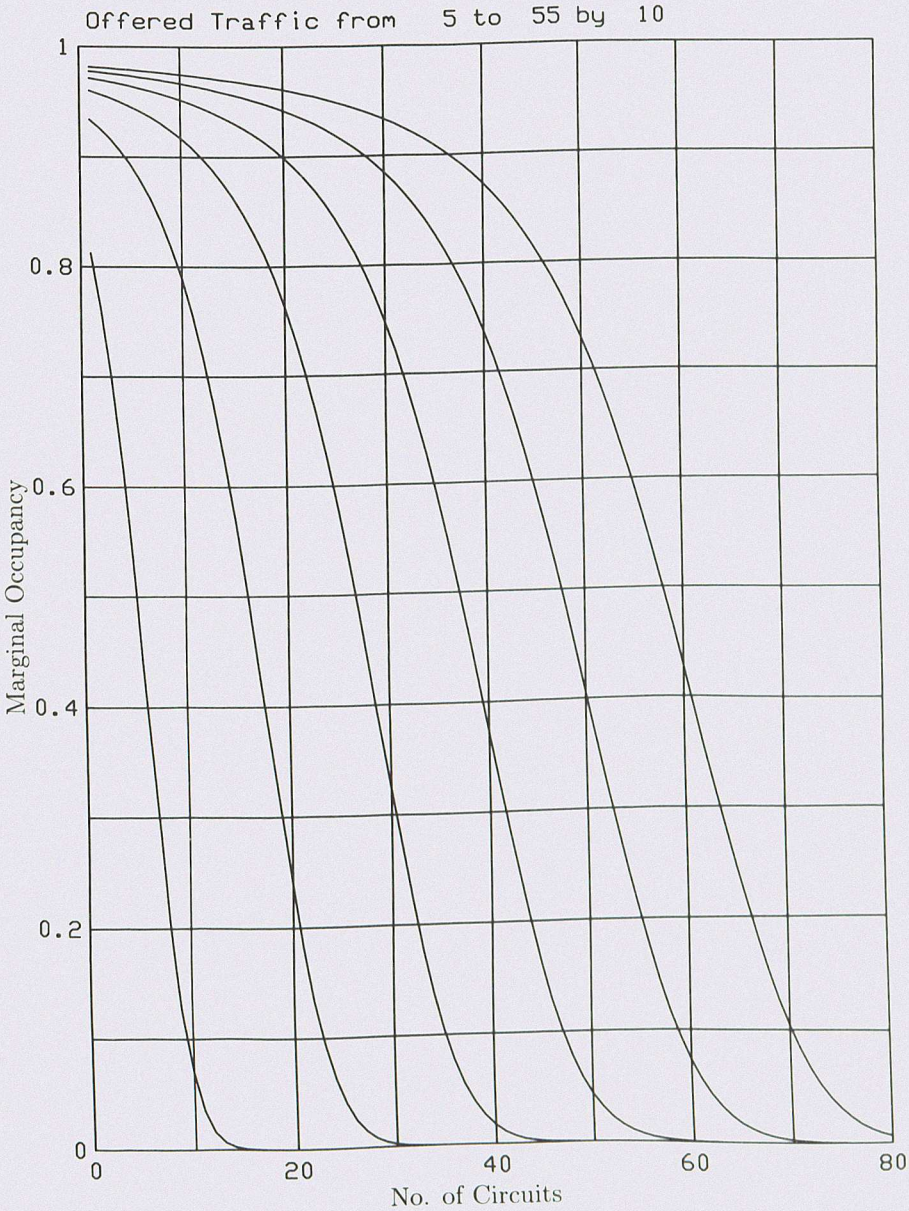

Figure D.5 Marginal Occupancy

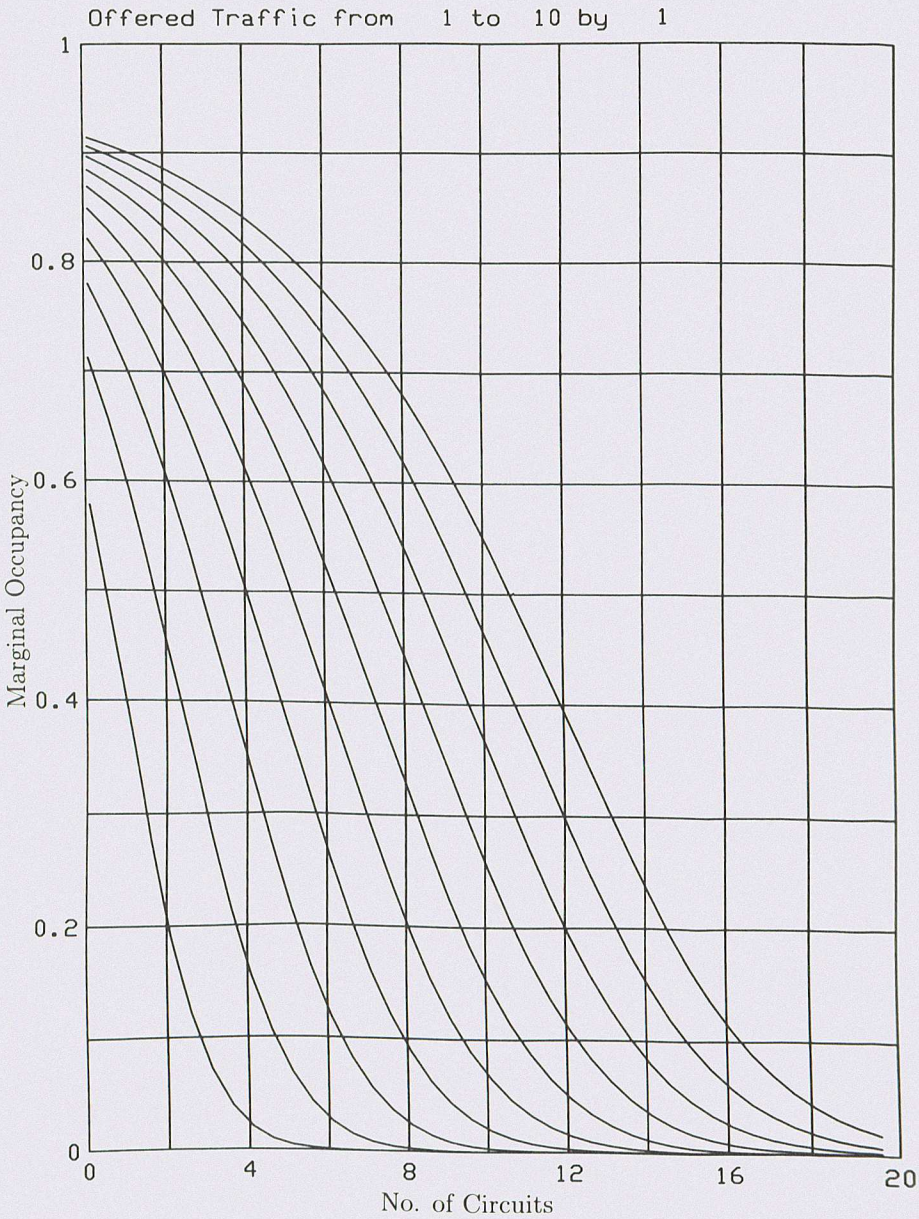

Figure D.6 Marginal Occupancy

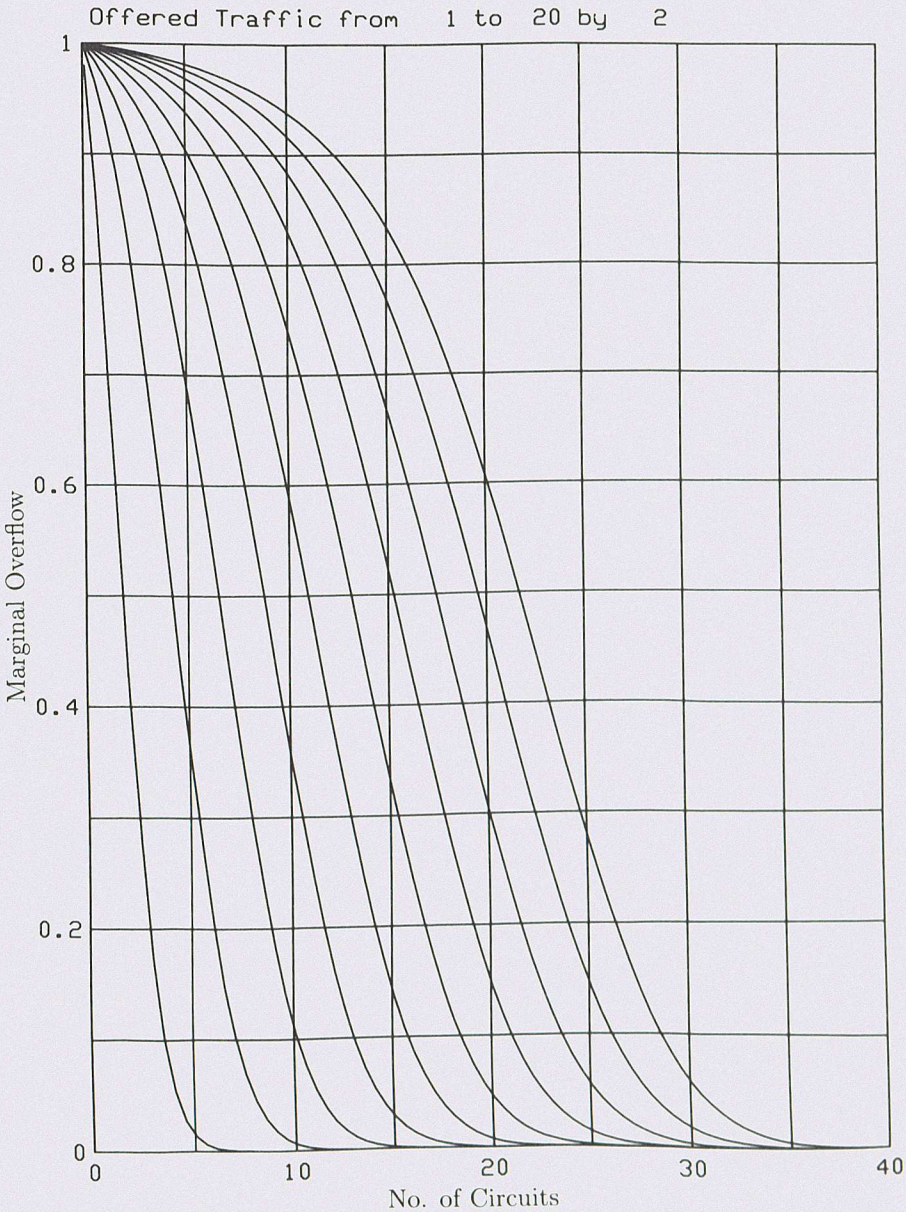

Figure D.7 Marginal Overflow