Optimization Problems

1

1.1 Introduction

Many problems of both practical and theoretical importance concern themselves with the choice of a "best" configuration or set of parameters to achieve some goal. Over the past few decades a hierarchy of such problems has emerged, together with a corresponding collection of techniques for their solution. At one end of this hierarchy is the general nonlinear programming problem: Find \( x \) to
\[
\begin{align*}
\text{minimize } & f(x) \\
\text{subject to } & g_i(x) \geq 0 & i = 1, \ldots, m \\
& h_j(x) = 0 & j = 1, \ldots, p
\end{align*}
\]
where \( f \), \( g_i \), and \( h_j \) are general functions of the parameter \( x \in \mathbb{R}^n \). The techniques for solving such problems are almost always iterative in nature, and their convergence is studied using the mathematics of real analysis.

When \( f \) is convex, \( g \), concave, and \( h \), linear, we have what is called a convex programming problem. This problem has the most convenient property

\( \uparrow \)See the appendix at the end of this chapter for a résumé of terminology and notation.
that local optimality implies global optimality. We also have conditions for optimality that are sufficient, the Kuhn-Tucker conditions.

To take the next big step, when \( f \) and all the \( g_i \) and \( h_i \) are linear, we arrive at the linear programming problem. Several striking changes occur when we restrict attention to this class of problems. First, any problem in this class reduces to the selection of a solution from among a finite set of possible solutions. The problem is what we can call combinatorial. The finite set of candidate solutions is the set of vertices of the convex polytope defined by the linear constraints.

The widely used simplex algorithm of G.B. Dantzig finds an optimal solution to a linear programming problem in a finite number of steps. This algorithm is based on the idea of improving the cost by moving from vertex to vertex of the polytope. Thirty years of refinement has led to forms of the simplex algorithm that are generally regarded as very efficient—problems with hundreds of variables and thousands of constraints are solved routinely. It is also true, however, that there are specially devised problems on which the simplex algorithm takes a disagreeably exponential number of steps.

Soviet mathematicians, in a relatively recent development, have invented an ellipsoid algorithm for linear programming that is guaranteed to find a feasible solution in a number of steps that grows as a polynomial in the \( \text{sizer} \) of the problem—a state of affairs that we shall come to regard as very favorable in the course of this book. At the time of this writing, it is uncertain whether refinements analogous to those of the simplex algorithm will render the ellipsoid algorithm competitive with it. The nesting of the problems mentioned so far—general nonlinear, convex, and linear programs—is indicated in Figure 1-1.

Figure 1-1 The classes of problems considered in this book and the path followed by the chapters.

Certain linear programs, the \( \text{flow and matching problems} \), can be solved much more efficiently than even general linear programs. On the other hand, these problems are also closely related to other problems that are apparently intractable! As an example, the point-to-point shortest-path problem in a graph is in our class of flow and matching problems, and in fact has an \( O(n^2) \) algorithm for its solution, where \( n \) is the number of nodes in the graph. In contrast, the \( \text{traveling salesman problem} \), which asks for the shortest closed path that visits every node exactly once, is in the class of \( NP \)-complete problems, all of which are widely considered unsolvable by polynomial algorithms. This fine line between "easy" and "hard" problems is a recurrent phenomenon in this book and has naturally attracted the attention of algorithm designers.

Another way of looking at the flow and matching problems is as special cases of \( \text{integer linear programs} \). These come about when we consider linear programs and ask for the best-cost solution with the restriction that it have \( \text{integer-valued} \) coordinates. These problems, like linear programs, have a finite algorithm for solution. But there the resemblance stops: The general integer linear programming problem is itself \( NP \)-complete. The complete state of affairs is shown in Figure 1-1, together with an indication of the general path that this book will take through these classes of problems.

We shall begin with the most fundamental and easily accessible facts about convex programs. We then take up linear programming in earnest, studying the simplex algorithm, its geometry, and the algorithmic implications of duality. We shall stress graph-theoretic interpretations of the algorithms discussed, and that will lead naturally to the flow and matching problems. It is an easy step from there to a consideration of complexity issues and the ellipsoid algorithm and to a study of the \( NP \)-complete problems that have become representative of difficult combinatorial optimization problems. The last part of the book is concerned with approaches toward the practical solution of \( NP \)-complete problems of moderate size: approximation, enumerative techniques, and local search.

1.2 Optimization Problems

Optimization problems seem to divide naturally into two categories: those with \( \text{continuous} \) variables, and those with \( \text{discrete} \) variables, which we call combinatorial. In the continuous problems, we are generally looking for a set of real numbers or even a function; in the combinatorial problems, we are looking for an object from a finite, or possibly countably infinite, set—typically an integer, set, permutation, or graph. These two kinds of problems generally have quite different flavors, and the methods for solving them have become quite divergent. In our study of combinatorial optimization we start—in some sense—at its boundary with continuous optimization.

Linear programming plays a unique role in optimization theory; it is in
Chapter 1  OPTIMIZATION PROBLEMS

one sense a continuous optimization problem, but, as we mentioned above, it can also be considered combinatorial in nature and in fact is fundamental to the study of many strictly combinatorial problems. We shall therefore give a definition of an optimization problem general enough to include linear programming (and almost any other optimization problem).

Definition 1.1

An instance of an optimization problem is a pair \((F, c)\), where \(F\) is any set, the domain of feasible points; \(c\) is the cost function, a mapping \(c : F \rightarrow \mathbb{R}^1\)

The problem is to find an \(f \in F\) for which

\[c(f) \leq c(y)\]

for all \(y \in F\).

Such a point \(f\) is called a globally optimal solution to the given instance, or simply an optimal solution.

In many examples the cost function will take on only nonnegative integer values.

Definition 1.2

An optimization problem is a set \(I\) of instances of an optimization problem.

We have been careful to distinguish between a problem and an instance of a problem. Informally, in an instance we are given the "input data" and have enough information to obtain a solution; a problem is a collection of instances, usually all generated in a similar way. Thus, in the following example, an instance of the traveling salesman problem has a given distance matrix; but we speak in general of the traveling salesman problem as the collection of all instances associated with all distance matrices.

Example 1.1  (Traveling Salesman Problem (TSP))

In an instance of the TSP we are given an integer \(n > 0\) and the distance between every pair of \(n\) cities in the form of an \(n \times n\) matrix \([d_{ij}]\), where \(d_{ij} \in \mathbb{Z}^+\).

A tour is a closed path that visits every city exactly once. The problem is to find a tour with minimal total length. We can take

\[F = \{\text{all cyclic permutations } \pi \text{ on } n \text{ objects}\}\]

A cyclic permutation \(\pi\) represents a tour if we interpret \(\pi(j)\) to be the city visited after city \(j, j = 1, \ldots, n\). Then the cost \(c\) maps \(\pi\) to

\[\sum_{j=1}^{n} d_{\pi(j)}\]

Example 1.2  (Minimal Spanning Tree (MST))

As above, we are given an integer \(n > 0\) and an \(n \times n\) symmetric distance matrix \([d_{ij}]\), \(d_{ij} \in \mathbb{Z}^+\). The problem is to find a spanning tree on \(n\) vertices that has minimal total length of its edges. In our definition of an instance of an optimization problem, we choose

\[F = \{\text{all spanning trees } (V, E) \text{ with } V = \{1, 2, \ldots, n\}\}\]

\[c : (V, E) \rightarrow \sum_{(i,j) \in E} d_{ij}\]

(By a spanning tree we mean an undirected graph \((V, E)\) that is connected and acyclic. See the appendix at the end of this chapter.)

Example 1.3  (Linear Programming (LP))

Let \(m, n\) be positive integers, \(b \in \mathbb{Z}^m, c \in \mathbb{Z}^n\), and \(A\) an \(m \times n\) matrix with elements \(a_{ij} \in \mathbb{Z}\). An instance of LP is defined by

\[F = \{x : x \in \mathbb{R}^n, Ax = b, \ x \geq 0\}\]

\[c : x \rightarrow c'x\]

Stated as such, linear programming is a continuous optimization problem, with, in fact, an uncountable number of feasible points \(x \in F\). To see how it can be considered combinatorial in nature, consider the simple instance defined by \(m = 1, n = 3\) and

\[A = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}\]

\[b = (2)\]

Figure 1–2 shows the feasible set \(F\) in this instance, the intersection of a plane with the first octant in \(\mathbb{R}^3\).

The problem is to minimize the value of the linear function \(c'x = c_1x_1 + c_2x_2 + c_3x_3\)
\[ c_2 x_2 + c_3 x_3 \] on this triangle. It is not hard to see intuitively that a minimum
will always occur at one of the corners \( v_1, v_2, \) or \( v_3 \) indicated in the figure. If
we grant this, we can solve this instance by finding all the vertices and evaluating
\( c'x \) at each one. This may be a formidable task in a larger instance, but the point
is that it is a \textit{finite} one. It is in this sense that LP is combinatorial.

In many cases it is possible to do the opposite: express a purely combinatorial
problem as an LP.

Example 1.3 (Continued)

Consider an instance of the MST problem with \( n = 3 \) points. There are
three spanning trees of these points, shown in Figure 1–3. They can also be
thought of as points in 3-dimensional space if \( x_i = 1 \) whenever edge \( e_i \) is in
the tree considered, and zero otherwise, \( i = 1, 2, 3 \). These three spanning trees
then coincide with the vertices \( v_1, v_2, \) and \( v_3 \) of the feasible set \( F \) in Figure
1–4 defined by the constraints

\[
x_1 + x_2 + x_3 = 2 \\
x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0 \\
x_1 \leq 1, \quad x_2 \leq 1, \quad x_3 \leq 1
\]

(We shall allow inequalities as well as equalities in LP.) Finding the minimal
spanning tree with distance matrix \( d_{12} = c_2, d_{23} = c_3, \) and \( d_{13} = c_1 \) is exactly
the same as solving the LP with the feasible set in Figure 1–4.

Thus this purely combinatorial problem can, in principle, be solved by
LP. This point of view will be very useful later for developing algorithms for
certain combinatorial problems.

1.3 Neighborhoods

Given a feasible point \( f \in F \) in a particular problem, it is useful in many
situations to define a set \( N(f) \) of points that are "close" in some sense to the
point \( f \).

Definition 1.3

Given an optimization problem with instances \((F, c)\), a \textit{neighborhood} is a
mapping

\[ N : F \rightarrow 2^F \]

defined for each instance.

If \( F = \mathbb{R}^n \), the set of points within a fixed Euclidean distance provides a
natural neighborhood. In many combinatorial problems, the choice of \( N \) may
depend critically on the structure of \( F \).

Example 1.4 \([\text{Lin1}]\)

In the TSP we may define a neighborhood called \textit{2-change} by

\[ N_2(f) = \{ g : g \in F \quad \text{and} \quad g \text{ can be obtained from } f \text{ as follows:} \]

\[ \text{remove two edges from the tour; then} \]

\[ \text{replace them with two edges} \]

Figure 1–5 shows an example of a tour \( f \) and another tour \( g \in N_2(f) \) for an
instance of the TSP with seven cities and a distance matrix determined by
Euclidean distance between points in the plane.

This neighborhood can be generalized in the obvious way to \( N_k \), called...
**Example 1.5**

In the MST, an important neighborhood is defined by

\[ N(f) = \{ g : g \in F \text{ and } g \text{ can be obtained from } f \text{ as follows: add an edge } e \text{ to the tree } f, \text{ producing a cycle; then delete any edge on the cycle} \} \]

**Example 1.6**

In LP, we can define a neighborhood by

\[ N_s(x) = \{ y : A y = b, \ y \geq 0, \text{ and } \| y - x \| \leq \epsilon \} \]

This is simply the set of all feasible points within Euclidean distance \( \epsilon \) of \( x \) for some \( \epsilon > 0 \).

**1.4 Local and Global Optima**

Finding a globally optimal solution to an instance of some problems can be prohibitively difficult, but it is often possible to find a solution \( f \) which is best in the sense that there is nothing better in its neighborhood \( N(f) \).

**Definition 1.4**

Given an instance \((F, c)\) of an optimization problem and a neighborhood \( N \), a feasible solution \( f \in F \) is called *locally optimal with respect to \( N \) (or simply locally optimal whenever \( N \) is understood by context)* if

\[ c(f) \leq c(g) \text{ for all } g \in N(f) \]

**Example 1.7**

Consider the instance of an optimization problem \((F, c)\) defined by

\[ F = \{0, 1\} \subseteq R^1 \]

and the cost function \( c \) sketched in Fig. 1–6.

**Figure 1–6** A 1-dimensional Euclidean optimization problem.

Further, let the neighborhood be defined simply by closeness in Euclidean distance for some \( \epsilon > 0 \).

\[ N_s(f) = \{ x : x \in F \text{ and } \| x - f \| \leq \epsilon \} \]

Then if \( \epsilon \) is suitably small, the points \( A, B, \) and \( C \) are all locally optimal, but only \( B \) is globally optimal.

**Example 1.8**

In the TSP, solutions locally optimal with respect to the \( k \)-change neighborhood \( N_k \) are called *\( k \)-opt* [Linl]. To find a \( k \)-opt tour in an instance of the TSP, define the function \( \text{improve}(t) \), where \( t \in F \), as follows:

\[ \text{improve}(t) = \begin{cases} \text{any } s \in N_k(t) \text{ such that } c(s) < c(t) \text{, if } \text{such an } s \text{ exists} \\ \text{no} \text{ otherwise} \end{cases} \]

That is, \( \text{improve}(t) \) searches \( N_k(t) \) for a better tour \( s \). If one is found, it returns the improved tour; otherwise it returns the value 'no.' An algorithm for finding a \( k \)-opt tour is then

\[
\text{procedure } k\text{-opt} \\
\text{begin} \\
t := \text{some initial tour}; \\
\text{while } \text{improve}(t) \neq \text{no} \text{ do} \\
\quad t := \text{improve}(t); \\
\text{return } t \\
\text{end}
\]

†Algorithms are written in an informal notation called *pidgin algol*. See the appendix at the end of this chapter.
Because we are generally interested in finding a global optimum and because many algorithms can do no more than search for local optima, it is important to know whether a local optimum is or is not global. This depends, of course, on the neighborhood $N$. The following terminology describes the happy situation when every local optimum is also a global optimum.

**Definition 1.5**

Given an optimization problem with feasible set $F$ and a neighborhood $N$, if whenever $f \in F$ is locally optimal with respect to $N$ it is also globally optimal, we say the neighborhood $N$ is **exact**. □

**Example 1.9**

In the instance sketched in Fig. 1–6, the neighborhood $N_{\epsilon}$ is exact when $\epsilon \geq 1$ but not exact for sufficiently small $\epsilon > 0$. □

**Example 1.10**

In the TSP, $N_2$ is not exact; but $N_n$, where $n$ is the number of cities, is exact. (See Problem 2.) □

**Example 1.11**

In the MST, the neighborhood described in Example 1.5 is exact. (See Problem 3.) □

### 1.5 Convex Sets and Functions

We now turn our attention to the class of problems where $F \subseteq \mathbb{R}^n$. In particular, we should like very much to find classes of problems where $N_{\epsilon}$ is exact for every $\epsilon > 0$, for in such problems we can be assured that any local optimum found is a global one. Such a property is enjoyed by the class of convex programming problems, of which linear programming is a special case. We start with some important definitions.

**Definition 1.6**

Given two points $x, y \in \mathbb{R}^n$, a convex combination of them is any point of the form

$$ z = \lambda x + (1 - \lambda)y, \quad \lambda \in \mathbb{R}^1 \text{ and } 0 \leq \lambda \leq 1 $$

If $\lambda \neq 0, 1$, we say $z$ is a strict convex combination of $x$ and $y$. □

**Definition 1.7**

A set $S \subseteq \mathbb{R}^n$ is convex if it contains all convex combinations of pairs of points $x, y \in S$. □

**Example 1.12**

The entire set $\mathbb{R}^n$ is convex, as is the empty set $\emptyset$ and any singleton set. □

**Example 1.13**

In $\mathbb{R}^1$, any interval is convex and any convex set is an interval. □

**Example 1.14**

In $\mathbb{R}^2$, convex sets, loosely speaking, are those without indentations. Thus, in Fig. 1–7, set $A$ is convex but $B$ is not. □

![Figure 1–7 A convex set $A$ and a nonconvex set $B$.](image)

An important property of convex sets is expressed in the following lemma.

**Lemma 1.1** The intersection of any number of convex sets $S_i$ is convex.

**Proof** If $x$ and $y$ are two points in $\cap S_i$, they are in every $S_i$. Any convex combination of them is then in every $S_i$ and therefore in $\cap S_i$. □

We now introduce the idea of a convex function defined on a convex set.

**Definition 1.8**

Let $S \subseteq \mathbb{R}^n$ be a convex set. The function

$$ e : S \longrightarrow \mathbb{R}^1 $$
is convex in $S$ if for any two points $x, y \in S$
\[ c(\lambda x + (1 - \lambda)y) \leq \lambda c(x) + (1 - \lambda)c(y), \quad \lambda \in \mathbb{R}^1 \text{ and } 0 \leq \lambda \leq 1 \]
If $S = \mathbb{R}^n$, we say simply that $c$ is convex.

**Example 1.15**

Any linear function is convex in any convex set $S$. 

**Example 1.16**

Intuitively, a convex function is one that "bends up." Figure 1–8 shows a sketch of a convex function in $[0, 1] \subseteq \mathbb{R}^1$.
\[ c: [0, 1] \rightarrow \mathbb{R}^1 \]

The convexity condition implies that chords always lie above the function. 

![Figure 1–8 A function $c$ convex in $[0, 1]$.](image)

Finally, functions that in some sense behave oppositely to convex functions are called concave functions.

**Definition 1.9**

A function $c$ defined in a convex set $S \subseteq \mathbb{R}^n$ is called concave if $-c$ is convex in $S$.

**Example 1.18**

Every linear function is concave as well as convex. Loosely speaking, a linear function bends neither down nor up, and so walks the line between convexity and concavity.

### 1.6 Convex Programming Problems

An important class of optimization problems concerns the minimization of a convex function on a convex set. These problems have the convenient property (mentioned above) that local optima are global. More precisely, we establish the following fact.
Theorem 1.1 Consider an instance of an optimization problem \((F, c)\), where 
\(F \subseteq \mathbb{R}^n\) is a convex set and \(c\) is a convex function. Then the neighborhood defined by Euclidean distance 
\[ N_\epsilon(x) = \{ y : y \in F \text{ and } ||x - y|| \leq \epsilon \} \]
is exact for every \(\epsilon > 0\).

Proof We refer to Figure 1–10.

![Figure 1–10](image)

Figure 1–10 The points in the proof of Theorem 1.1.

Let \(x\) be a local optimum with respect to \(N_\epsilon\) for any fixed \(\epsilon > 0\) and let \(y \in F\) be any other feasible point, not necessarily in \(N_\epsilon(x)\). We can always choose \(\lambda\) sufficiently close to 1 that the strict convex combination 
\[ z = \lambda x + (1 - \lambda)y, \quad 0 < \lambda < 1 \]
lies within the neighborhood \(N_\epsilon(x)\). Evaluating the cost function \(c\) at this point, we get, by the convexity of \(c\), 
\[ c(z) = c(\lambda x + (1 - \lambda)y) \leq \lambda c(x) + (1 - \lambda)c(y) \]
Rearranging, we find that 
\[ c(y) \geq \frac{c(z) - \lambda c(x)}{1 - \lambda} \]
But since \(z \in N_\epsilon(x)\) 
\[ c(z) \geq c(x) \]
so 
\[ c(y) \geq \frac{c(x) - \lambda c(x)}{1 - \lambda} = c(x) \]
Note that we have made no extra assumptions about the function \(c\); it need not be differentiable, for example.

For our purposes, the convex feasible region will always be defined by a set of inequalities involving concave functions. Such problems are conventionally known as convex programming problems.

Definition 1.10

An instance of an optimization problem \((F, c)\) is a convex programming problem if \(c\) is convex and \(F \subseteq \mathbb{R}^n\) is defined by 
\[ g_i(x) \geq 0, \quad i = 1, \ldots, m \]
where 
\[ g_i : \mathbb{R}^n \rightarrow \mathbb{R}^1 \]
are concave functions.

It is not hard to see that the set \(F\) defined in this way is in fact convex.

Lemma 1.3 The feasible set \(F\) in a convex programming problem is convex.

Proof The functions \(-g_i\) are convex, so by Lemma 1.2, the sets 
\[ F_i = \{ x : g_i(x) \geq 0 \} \]
are convex. Hence, by Lemma 1.1, 
\[ F = \bigcap_{i=1}^m F_i \]
is also convex.

With this, we have shown the following theorem.

Theorem 1.2 In a convex programming problem, every point locally optimal with respect to the Euclidean distance neighborhood \(N_\epsilon\), is also globally optimal.

Example 1.19

A convex function \(c(x)\) defined on \([0, 1] \subseteq \mathbb{R}^1\) can have many local optima but all must be global, as illustrated in Fig. 1–11.

![Figure 1–11](image)

Figure 1–11 A convex programming problem with many local optima, all of which are global.
Example 1.20

Every instance of LP is a convex programming problem, because linearity functions are both convex and concave. Thus a local optimum of an instance of LP must also be global. □

PROBLEMS

1. Formulate the following as optimization problem instances, giving in each case the domain of feasible solutions $F$ and the cost function $c$.

(a) Find the shortest path between two nodes in a graph with edge weights representing distance.

(b) Solve the Tower of Hanoi problem, which is defined as follows: We have $n$ diamond needles and $64$ gold disks of increasing diameter. The disks have holes at their centers, so that they fit on the needles, and initially the disks are all on the first needle, with the smallest on top, the next smallest below that, and so on, with the largest at the bottom. A legal move is the transfer of the top disk from any needle to any other, with the condition that no disk is ever placed on top of one smaller than itself. The problem is to transfer, by a sequence of legal moves, all the disks from the first to the second needle. (There is a story that the world will end when this task is completed [k] which may be an optimistic expectation.) Generalize to $n$ gold disks.

(c) Win a game of chess. How many instances of this problem are there?

(d) Find a cylinder with a given surface area $A$ that has the largest volume $V$.

(e) Find a closed plane curve of given perimeter that encloses the largest area.

*2. Show by example that 2-change does not define an exact neighborhood for the TSP. Repeat for 3-change, and for $(n - 3)$-change, where $n$ is the number of cities.

*3. Show that the neighborhood defined in Example 1.5 for the MST is exact.

4. The moment problem is that of finding a permutation $\pi$ of $n$ weights $w_i$ so that the moment

$$\sum_{i=1}^{n} w_{\pi(i)} = \min$$

Show that the neighborhood determined by all possible interchanges of two adjacent weights is exact.

5. In the $n$-city TSP, what is the cardinality of $N_2(i)$, the neighborhood of tour $i$ determined by 2-change? What is the cardinality of $N_1(i)$?

6. Suppose we are given a set $S$ containing $2n$ integers, and we wish to partition it into two sets $S_1$ and $S_2$ so that $|S_1| = |S_2| = n$ and so that the sum of the numbers in $S_1$ is as close as possible to the sum of those in $S_2$. Let the neighborhood $N$ be determined by all possible interchanges of two integers between $S_1$ and $S_2$. Is $N$ exact?

7. Is the product of two convex functions convex? If yes, prove it; if not, give a counterexample.

8. Let $f(x)$ be convex in $R^i$. Is $f(x + b)$, where $b$ is a constant, convex in $R^i$?

9. Let $f(x)$ be convex in $R^i$. Fix $x_2, \ldots, x_n$ and consider the function $g(x_1) = f(x_1, \ldots, x_n)$. Is $g$ convex in $R^i$?

10. Let $f(x)$ be a convex function of the single variable $x$. Then $g(x) = f(x)$ can also be considered as a function of $x \in R^i$. Is $g(x)$ convex in $R^i$?

11. Show that the sum of two convex functions is convex.

12. Justify the inclusion of integer linear programming within the class of nonlinear programs in Figure 1–1.

13. The following is a very useful criterion for determining if a function is convex [SW, vol. 1, p. 152]:

Let $C$ be an open convex set in $R^n$ and let $f$ have continuous second partial derivatives in $C$. Then $f$ is convex in $C$ if and only if the matrix of second partial derivatives

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_i \partial x_j} \end{bmatrix}$$

is positive semidefinite for all $x \in C$.

Determine whether or not the following functions are convex in the indicated domains.

(a) $f = x_1 x_2$, $C = R^2$

(b) $f = e^{x_1 + x_2}$, $C = R^2$

(c) $f = x_1^2 + x_2^2 - x_1 x_2$, $C = R^3$

(d) $f = x_1 + x_2^2$, $C = \{x \in R^2 : x > 0\}$

(e) $f = \tan x_1$, $C = \{x_1 : 0 < x_1 < 1\}$

14. Consider the nonlinear program

$$\min f = x_1 x_2$$

such that $g = (x_1 - 1)^2 + (x_2 - 1)^2 = 1$

Find all the global and local minima.

*15. Formulate the minimal spanning tree problem for an $n$-node graph as a linear program with $\binom{n}{2}$ variables, one for each edge, thus generalizing Example 1.3. (Hint: this may require many constraints.)

NOTES AND REFERENCES

Further discussion of nonlinear programming problems and optimality conditions can be found in


The idea of a $k$-change neighborhood for the TSP is due to


The origin of the Tower of Hanoi problem (Problem 1) is described in


APPENDIX

TERMINOLOGY AND NOTATION

A.1 Linear Algebra

The real number line is denoted by $R$ (or sometimes $R^1$) and the $n$-dimensional real vector space, the set of ordered $n$-tuples of real numbers, by $R^n$. Other fixed sets are the set of nonnegative reals, $R^+$; the set of integers, $Z$; the set of nonnegative integers, $Z^+$; and the set of ordered $n$-tuples of integers, $Z^n$.

A set of elements $s_1, s_2, s_3, \ldots$ is written explicitly as

$$S = \{s_1, s_2, s_3, \ldots\}$$

and a set defined to contain all elements $x$ for which a condition $P$ is true is defined by writing

$$S = \{x : P(x)\}$$

For example, $Z^+$ can be defined by

$$Z^+ = \{i : i \in Z \text{ and } i \geq 0\}$$

The size of a finite set $S$ is denoted by $|S|$. A mapping $\mu$ from set $S$ to set $T$ is written

$$\mu : S \rightarrow T$$

and $2^S$ stands for the set of all subsets of $S$.

An $m \times n$ matrix with element $a_{ij}$ in Row $i$ and Column $j$ is written

$$A = [a_{ij}]$$

The $n$-vector consisting of the $i$th row of $A$ is denoted by $a_i$ and the $m$-vector which is the $j$th column of $A$ by $A_j$. All vectors $x$ (no prime) are column vectors; vectors $x'$ (prime) are row vectors. Thus the matrix equation

$$Ax = b$$

is equivalent to the set of scalar equations

$$a_i x = b_i \quad i = 1, \ldots, m$$

where $b$ is an $m$-vector and $b_i$, its $i$th component, is a scalar. The transpose of a matrix $A$ is written $A^T$ and the determinant of a square matrix is denoted by $\det (A)$.

The unit square matrix is denoted by $I$, where its dimension is usually understood by context. It is defined by

$$I_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Similarly, $0$ denotes either the zero scalar, vector, or matrix, depending on the context.

To construct an $n$-vector $x$ with $i$th component $x_i$, we write

$$x = \col (x_1, \ldots, x_n)$$
or, when there is no danger of confusion,
\[ x = (x_1, \ldots, x_n) \]

To construct an \((n+m)\)-vector \(z\) if its first \(n\) components are those of \(x\) and next \(m\) components those of \(y\), we write
\[ z = (x | y) \]

### A.2 Graph Theory

A graph \(G\) is a pair \(G = (V, E)\), where \(V\) is a finite set of nodes or vertices and \(E\) has as elements subsets of \(V\) of cardinality two called edges. The vertices of \(V\) are usually called \(v_1, v_2, \ldots\). For example, the graph
\[ G = \{[v_1, v_2], [v_2, v_3], [v_3, v_4], [v_4, v_5], [v_5, v_6]\} \]
is shown in Fig A-1. (Notice that we denote edges using brackets.) It is occasionally useful to consider multigraphs, that is, graphs with repeated edges (see Figure A-2).

![Figure A-1 A graph.](image)

![Figure A-2 A multigraph.](image)

A directed graph, or digraph, is a graph with directions assigned to its edges. Formally, a digraph \(D\) is a pair \(D = (V, A)\) where \(V\) is again a set of vertices and \(A\) is a set of ordered pairs of vertices called arcs; that is, \(A \subseteq V \times V\).

In Figure A-3, we have drawn the digraph \(D = \{(v_1, v_2), (v_2, v_3), (v_4, v_5), (v_5, v_1), (v_1, v_3)\}\).

![Figure A-3 A digraph.](image)

If \(G = (V, E)\) is a graph and \(e = [v_1, v_2] \in E\), then we say that \(v_1\) is adjacent to \(v_2\) (and vice-versa) and that \(e\) is incident upon \(v_1\) (and \(v_2\)). The degree of a vertex \(v\) of \(G\) is the number of edges incident upon \(v\). So, for the graph of Figure A-1, the degree of \(v_1\) is 3. A walk in \(G\) is a sequence of nodes \(w = [v_1, v_2, v_3, \ldots, v_k]\), \(k \geq 1\), such that \([v_j, v_{j+1}] \in E\) for \(j = 1, \ldots, k - 1\). The walk is closed if \(k > 1\) and \(v_k = v_1\). A walk without any repeated nodes in it is called a path; a closed walk with no repeated nodes other than its first and last one is called a circuit or cycle. For example, in Figure A-1, \([v_1, v_2, v_3, v_4, v_5, v_1]\) and \([v_2, v_3, v_4]\) are all walks; the second and third are closed, the first and fourth are arc paths, and the third is a cycle. The length of the path \([v_1, \ldots, v_k]\) is \(k - 1\); the length of the cycle \([v_1, \ldots, v_k = v_1]\) is \(k - 1\).

In a digraph \(D = (V, A)\) the indegree of a node \(v\) is the number of arcs of the form \((u, v)\) that are in \(A\); similarly, the outdegree of \(v\) is the number of arcs of \(A\) that have the form \((v, u)\). We can readily extend the above definitions to digraphs. A directed walk \(w = (v_1, v_2, \ldots, v_k)\) of \(G\) is a sequence of nodes in \(V\) such that \((v_{j-1}, v_j) \in A\) for \(j = 1, \ldots, k - 1\). Furthermore, if \(k > 1\) and \(v_k = v_1\), then \(w\) is closed. A directed path in \(G\) is a walk without repetitions. A directed circuit or cycle is a closed directed path. The length of a directed path and cycle are defined by analogy to the undirected case. (Notice that we always use brackets for the edges of a graph and parentheses for the arcs of a digraph.)

Suppose that \(B = (W, E)\) is a graph that has the following property. The set of vertices \(W\) can be partitioned into two sets, \(V\) and \(U\), and each edge in \(E\) has one vertex in \(V\) and one vertex in \(U\) (Figure A-4). Then \(B\) is called a bipartite graph and is usually denoted by \(B = (V, U, E)\). Not all graphs have such a partition. The precise conditions under which they do are the following:

**Proposition 1** A graph is bipartite iff it has no circuit of odd length.
Another interesting class of graphs is the class of trees. A graph is connected if there is a path between any two nodes in it. A tree $G = (V, T)$ is a connected graph without cycles. For example, Fig. A-5 shows a tree. A forest $F = (V, E)$ is a set of node-disjoint trees $F = \{(V_1, T_1), \ldots, (V_s, T_s)\}$ (Fig. A-6). We usually say that a tree $(V, T)$ spans its set of nodes $V$ or is a spanning tree of $V$. Similarly, a forest $F = \{(V_1, T_1), \ldots, (V_s, T_s)\}$ spans $V_1 \cup V_2 \cup \cdots \cup V_s$.

**Proposition 2** Let $G = (V, E)$ be a graph. Then the following are equivalent:

1. $G$ is a tree.
2. $G$ is connected and has $|V| - 1$ edges.
3. $G$ has no cycles, but if an edge is added to $G$, a unique cycle results.

We shall frequently consider weighted graphs, that is, graphs $G = (V, E)$ together with a function $w$ from $E$ to $\mathbb{Z}$ (usually just $\mathbb{Z}^+$); it can also be $\mathbb{R}$, when, for example, the weights are Euclidean distances). In certain cases we shall use more mnemonic names for weights, such as $c$ (for costs) or $d$ (for distances). We denote the weight of the edge $[u, v]$ by $w[u, v]$, or $w_{uv}$. Notice that a symmetric $n \times n$ distance matrix $[d_{ij}]$ (recall Examples 1.1 and 1.2) can also be thought of as a weighted complete graph $G = ([v_1, \ldots, v_n], K_n)$, where $K_n = \{[u_i, v_j]: 1 \leq i < j \leq n\}$.

A network $N = (s, t, V, A, b)$ is a digraph $(V, A)$ together with a source $s \in V$ with 0 indegree, a terminal $t \in V$ with 0 outdegree, and with a bound (or capacity) $b(u, v) \in \mathbb{Z}^+$ for each $(u, v) \in A$ (Fig. A-7). A flow $f$ in $N$ is a vector in $\mathbb{R}^{|A|}$ (one component $f(u, v)$ for each arc $(u, v) \in A$) such that:

1. $0 \leq f(u, v) \leq b(u, v)$ for all $(u, v) \in A$.
2. $\sum_{v \in \delta^+(u)} f(u, v) = \sum_{v \in \delta^-(u)} f(v, u)$ for all $v \in V - \{s, t\}$.

The value of $f$, sometimes denoted by $|f|$, is the following quantity: $|f| = \sum_{(u, v) \in A} f(u, v)$. For example, in Figure A-8, we show a legitimate flow for $N$; its value is $|f| = 5$. 