URBAN TRANSPORTATION NETWORKS

YOSEF SHEFFI
Massachusetts Institute of Technology

URBAN TRANSPORTATION NETWORKS:

Equilibrium Analysis with Mathematical Programming Methods

PRENTICE-HALL, INC., *Englewood Cliffs, New Jersey 07632*

Library of Congress Cataloging in Publication Data

Sheffi, Yosef. (date)
Urban transportation networks.

Bibliography: p.
Includes index.
1. Urban transportation—Mathematical models.
I. Title.
HE305.S54 1984 388.4'0724 84-9913
ISBN 0-13-939729-9

Editorial/production supervision:
Gretchen K. Chenenko, Ros Herion, and *Nancy DeWolfe*
Interior design: *Gretchen K. Chenenko*
Cover design: *Diane Saxe*
Manufacturing buyer: *Anthony Caruso*
Cover photo: *Ken Karp*

Printed in the United States of America

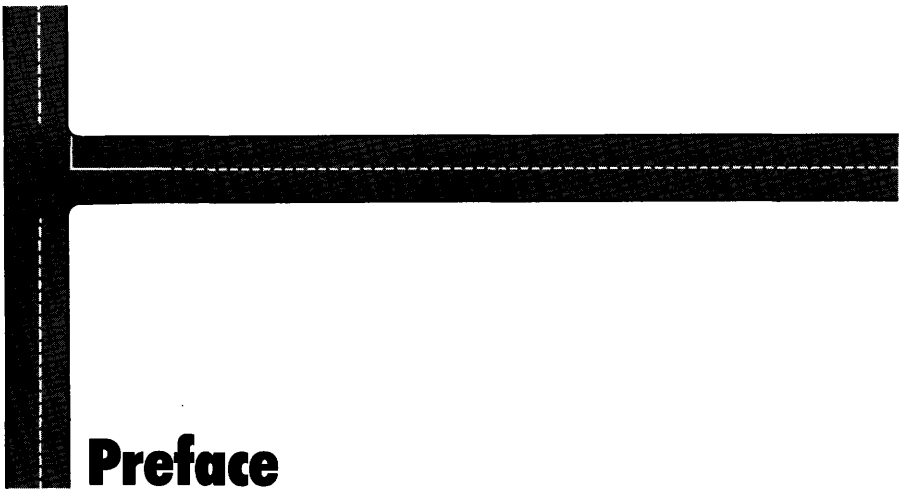
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-939729-9 01

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

To Anat

Contents



Preface

The flow pattern throughout an urban network can be looked upon as the result of two competing mechanisms. On the one hand, users of the system (drivers, passengers, pedestrians) try to travel in a way that minimizes the disutility associated with transportation. For example, motorists driving between a given origin and a given destination are likely to choose the route with the shortest travel time. On the other hand, the disutility associated with travel is not fixed but rather depends in part on the usage of the transportation system. Thus, in the previous example, the travel time on each of the paths connecting the origin and the destination is a function of the total traffic flow due to congestion. It is therefore not clear a priori which path through the network has the shortest travel time. Consequently, it may not be obvious what the flow pattern throughout the network will be under various conditions. This book describes how this flow pattern can be determined for an urban road network by modeling these two mechanisms (travel decisions and congestion).

The analytical approach described in this text draws on analogies between the two mechanisms mentioned here and the interaction of supply and demand in the marketplace. Instead of analyzing the price of a product and the quantity consumed, the analysis here looks at transportation level of service (or its inverse, travel disutility) and flows. The results of the analysis include a set of flow and a set of level-of-service measures that are at equilibrium with each other.

The book looks at many dimensions of travel choice, including the decision to take a trip, the choice of travel mode, the distribution of trips among various possible destinations, and the choice of route between an origin and a

destination. All these decisions, when aggregated and analyzed simultaneously with the congested effects, result in the flow pattern through the network. The analysis of all these travel choices is carried out by using a unified framework that builds on graphical and network representation.

The problem of finding the equilibrium flow pattern over a given urban transportation network is also known as traffic assignment. The basic solution methodology is based on formulating the problem as a nonlinear optimization and solving it as such. This book, however, does not require any prerequisites in mathematical programming or graph theory. All the necessary background is reviewed at an introductory level. The level of mathematics assumed includes college calculus and (in the last parts of the book) introductory probability concepts. The book uses extensively intuitive arguments and network structures, which are utilized to illustrate many situations graphically.

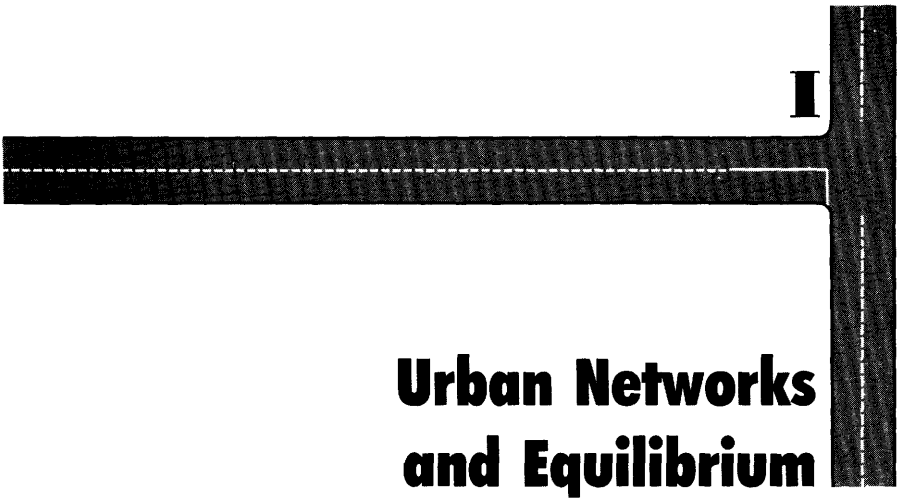
This book grew out of a set of course notes used for two courses at M.I.T., one taken by graduate students and the other by undergraduates. Currently, the courses (and the notes) are combined, drawing both upper-level undergraduate and graduate students. The book was designed as a text for the following classroom environments:

1. A primary text for an intensive one-semester (or two-trimester) course(s).
2. A primary text for a slower one-semester or a one-trimester course (Chapters 1–6, 7, and/or 8, possibly 9, and parts of 13). Such a course would cover only deterministic equilibrium methods.
3. A supporting text in an introductory urban transportation planning course (Chapters 1, 3, 5, 6, and 7 with emphasis on the algorithms).
4. A supporting text in a course on operations research and mathematical programming techniques, demonstrating the applications of the methodology.

The book should also serve practicing transportation engineers and urban planners as a reference guide on issues of traffic assignment and urban transportation networks analysis.

Many people have contributed to the development of this book. Warren Powell of Princeton University and Carlos Daganzo of the University of California at Berkeley joined me in most of my research in this area. Joel Horowitz of the University of Iowa and Hani Mahmassani of the University of Texas at Austin helped in the preparation of the manuscript by commenting extensively on its early versions. Useful comments were also provided by Brian Brademeyer, Stella Dafermos, Mark Daskin, Patrick Harker, Haris Koutsopoulos, Larry LeBlanc, Joffre Swait, and all the students who took my networks course at M.I.T. during the years in which the manuscript was in preparation. The contributions of Karen and Jonathan are also greatly appreciated.

YOSEF ("YOSSI") SHEFFI



I

Urban Networks and Equilibrium

1

Urban Transportation Networks Analysis

The amount of travel taking place at a given moment on any street, intersection, or transit line in an urban area is the result of many individuals' decisions. Travelers choose if and when to take a trip for some purpose, which mode of transportation to use (e.g., whether to drive or take public transit), where to go, and which way to get there. These decisions depend, in part, on how congested the transportation system is and where the congested points are. Congestion at any point of the transportation system, however, depends on the amount of travel through that point. This book describes how these interactions between congestion and travel decisions can be modeled and solved simultaneously to obtain the flow pattern throughout the urban transportation network.

One of the major problems facing transportation engineers and urban planners is that of predicting the impact of given transportation scenarios. The analytical part of this problem can be dealt with in two main stages. First, the scenario is specified mathematically as a set of inputs that are used to predict the flow pattern resulting in from such a scenario. Second, the flow pattern is used to calculate an array of measures that characterize the scenario under study. The inputs to the analysis typically include a description of the following (either existing or projected):

1. The transportation infrastructure and services, including streets, intersections, and transit lines.
2. The transportation system operating and control policies
3. The demand for travel, including the activity and land-use patterns

The first stage of the analysis uses these inputs to calculate the flow through each component of the urban network. Flow is measured in terms of the number of travel units (vehicles, passengers, pedestrians) that traverse a given transportation facility in a unit time. This part of the analysis builds on functional relationships between flows and congestion as well as relationships between congestion and travel decisions.

The second stage of the analysis involves the calculation of an array of measures of interest, given the flow pattern. These may include the following:

1. Level of service measures, such as travel time and travel costs. These affect the users of the transportation system directly.
2. Operating characteristics, such as revenues and profits. These are of primary concern to the operators of the system.
3. Flow by-products, such as pollution and changes in land values. These affect the environment in which a transportation system is operated.
4. Welfare measures, such as accessibility and equity. These may indirectly affect various population groups.

The focus of this book is on the first stage of the analysis, that of determining the flow pattern, given the inputs. Many of the aforementioned measures can be calculated from these flows in a straightforward manner. Others (such as pollution level or equity measures) may require sophisticated methodologies and complex models which are not within the scope of this book. These analyses, however, use the flow pattern as a major input. In addition, a complete analysis of a given scenario may involve important considerations which are not based on the flow, such as construction costs and political and institutional factors.

The perspective of the modeling approach presented in this text is *descriptive* rather than *normative*. In other words, it describes how individuals travel through an urban transportation system given the components of that system (which are specified as inputs). It does not attempt to determine the optimal system configuration (in terms, say, of which project should be built or what kind of control policies should be utilized). In many cases, however, the analysis is motivated by the need to decide on a given transportation investment, regulation, or policy from among a set of alternative courses of action. In this case, each of the alternative scenarios should be specified mathematically as a set of inputs to the analysis. The models described in this book are then formulated and solved for each alternative scenario in order to predict the flow pattern that is used, in turn, to develop many of the measures of interest. The full set of measures associated with each alternative is then used to compare their respective impacts. These measures are, typically, also compared to the set of measures characterizing a base case (which can be the current system or a projection of how the current system will operate under the "do nothing" alternative). These comparisons serve as a basis for recommendation regarding the preferred course of action.

This book presents the various assumptions, models, and algorithms used to calculate the flow pattern from the inputs. The thrust of the book is the interaction of congestion and travel decisions that result in this flow. Since congestion increases with flow and trips are discouraged by congestion, this interaction can be modeled as a process of reaching an *equilibrium* between congestion and travel decisions.

The remainder of this chapter introduces and explains the notion of equilibrium in the context of urban transportation analysis. Section 1.1 presents this equilibrium in general terms, and contrasts it with the more familiar economic equilibrium framework. Section 1.2 describes the mathematical representation of the urban transportation system as a network. The network structure is a fundamental characteristic of the equilibrium discussed in this text. Section 1.3 gives the actual definition of several equilibria between congestion and travel decisions. These equilibria are defined specifically for urban transportation networks. The section ends with an outline of the text. Finally, Section 1.4 summarizes the first chapter and Section 1.5 suggests some further readings.

1.1 EQUILIBRIUM ANALYSIS OF TRANSPORTATION SYSTEMS

The concept of equilibrium analysis is tied to the systems-oriented view of urban transportation taken in this text. This section discusses three related issues that explain this view: (1) the need for the systems-based approach to the analysis of urban transportation, (2) the general notion of equilibrium and the various types of equilibria, and (3) the applications of these notions to transportation systems analysis.

Systems Approach to Urban Transportation

The traditional approach taken in many engineering analyses is to isolate a component of a system and analyze this component individually. This approach is also used in analyzing the effect of small changes in the urban transportation system. For example, traffic lights are typically timed by considering only the traffic using the intersection under consideration. The effects of the change on adjacent intersections are usually assumed to be negligible. Similarly, parking regulations, intersection designs, and other changes in the urban transportation system are typically analyzed by considering only the immediate environment.

If the impact of a particular policy or design is likely to be small, this type of analysis may be sufficient. When the change being analyzed is more substantial, however, it will affect not only the component that is being changed but other parts of the system as well. As an illustration of this type of ripple effect, consider a congested segment of an urban arterial. In order to

reduce congestion, the local transportation authority considers the costs and benefits of widening this segment of road. Given the existing traffic flow, the addition of one lane is judged to be sufficient, since the added lane will mean that the existing traffic will be able to flow smoothly and with only minor delays. The project benefits calculated in this manner may, however, be misleading since travel decisions were not taken into account. For example, it is likely that motorists who did not use the arterial segment before, will subsequently make use of the improved facility. Thus the traffic conditions on the widened facility will not be as good as anticipated due to the increased flow, which will cause increased congestion. In turn, some of the roads leading into the improved facility may get congested as motorists try to access and exit from the widened segment. In addition, the flow conditions on roads parallel to the widened facility may improve as a result of these shifts since the traffic flow on them may decrease. Drivers on other parts of the system may then realize a change in their flow conditions and alter their route accordingly. Each route change will lead to further changes in congestion levels and consequently more route changes. Ultimately, however, these ripple effects will lessen and, after a short while (several days or a few weeks), the system will stabilize at a new *equilibrium* point with no more significant changes occurring.

The preceding example illustrated the flow changes resulting from a change to the transportation infrastructure. Similar changes may occur if transportation control policies are changed or if new transporation services are introduced. Furthermore, new flow patterns may result from changes not directly tied to the transportation system but rather changes in the general activities in the urban area. These activities create the need and the demand for transportation. For example, consider the opening of a new shopping center. The center attracts shopping trips which are partially new (generated by people who now shop more frequently) and partially diverted from other destinations. The streets leading to the new center will get congested, causing some travelers who are only passing through the area to change route, mode of travel, or the timing of their trips. These diversions will change the flow and congestion throughout the system, causing further changes in travel decisions. Other changes will occur around other shopping centers, which may experience some decline in the level of flow. Again, after a while the flows will reach a new equilibrium point. At this point, the frequency of trips, the trip destinations, the modes of travel, and the chosen routes are stable throughout the transportation network.

The notion of equilibrium in this context parallels the physical notion of stable equilibrium, which is the state in which there are no (net) forces that try to push the system to some other state. Furthermore, when the system is in disequilibrium, there are forces that tend to direct the system toward the equilibrium state. In the case under consideration, the flows are being "pushed" toward equilibrium by the route-switching mechanism. At equilibrium, the flows will be such that there is no incentive for route switching. It is this state of equilibrium that is the focus of this text.

As the examples above indicate, the equilibrium flow pattern associated with a given scenario may involve unanticipated flow levels and congestion in various parts of the urban network. This bears directly on the question of who is affected under this scenario. Whereas some travelers will be better off, others may be worse off. A consideration of the entire system is therefore neccessary in order to account for the wide spectrum of effects associated with a particular scenario. In other words, the equilibrium flow pattern that would prevail under each scenario can be found only by analyzing simultaneously all elements of the urban transportation network.

The next section explains the equilibrium concept in several contexts, leading to the role of such analysis in the study of urban transportation systems.

Equilibrium in Markets

The classical view of a (perfectly competitive) economic market for a certain good includes two interacting groups: the producers and the consumers. The behavior of the producers is characterized by a supply function and the behavior of the consumers is characterized by a demand function. The supply function expresses the amount of goods that the suppliers produce as a function of the price of the product. As the price increases, it becomes profitable to produce more and the quantity supplied increases. The demand function describes the aggregate behavior of consumers by relating the amount of the product consumed to its price. As the price increases, the amount consumed decreases.

Figure 1.1 depicts simple demand and supply functions for a certain product. The point where the two curves intersect is characterized by the "market clearing" price, P^*, and quantity produced, Q^*. This is the point where the price of the product is just right, so that the entire quantity produced is consumed. If the price is higher than P^*, production will be higher than consumption, as shown in Figure 1.1b. Such an imbalance cannot be sustained, since not all of the product sells and inventories will grow indefinitely. Prices will eventually fall and consumption will increase accordingly. If the price is lower than P^*, the quantity demanded is higher than the production. Such a situation is again unstable since producers will try to increase prices in order to capture the consumers' willingness to pay more. Such price increases will lead to higher production and lower demand. Thus, if the prices are either lower or higher than P^*, market forces will tend to push the price toward its "market clearing" level. At this point the price will be stable and thus the point (P^*, Q^*) is known as the *equilibrium* point.

The monetary price of a product is not always the only determinant of the quantity consumed. Products can be characterized by many attributes that influence consumption. Furthermore, some of these characteristics are not constant but rather a function of the quantity consumed.

Consider, for example, the demand for gasoline at a certain gas station.

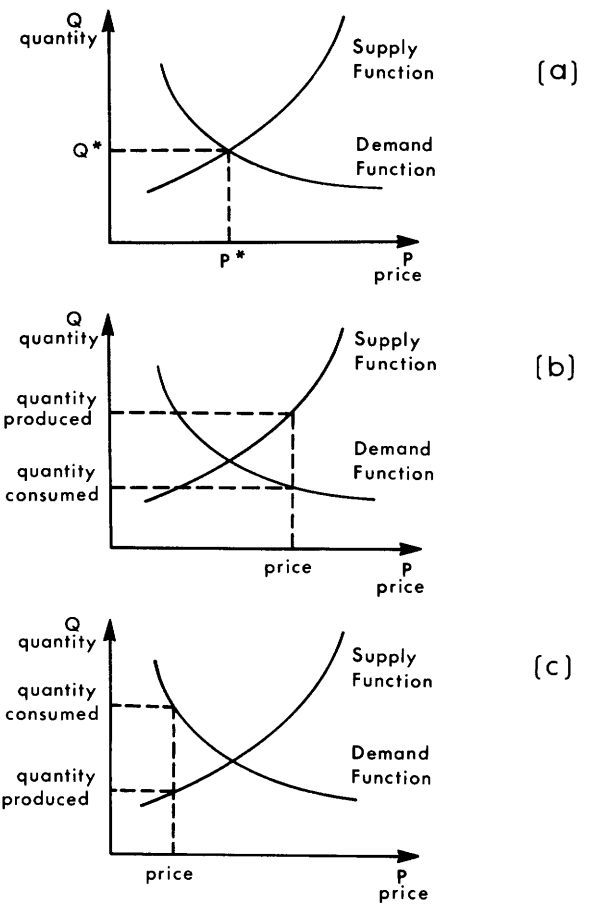


Figure 1.1 Demand/supply equilibrium: (a) market-clearing quantity and price; (b) price too high; (c) price too low.

The lower the price of gasoline at this station, the larger the number of motorists who would want to purchase gasoline there. As the number of customers grows, a queue appears and customers have to wait in line. At this point, the number of additional customers is influenced by two elements: price and waiting time. The station operator can set only the price. For a given price, the volume of sales will be determined by two factors: the waiting time, which is influenced by the number of customers in the queue, and the willingness of customers to wait, which determines the demand for gasoline at the station.

For a given gasoline price,† then, the situation can be characterized by two functions. The first is a demand function that relates the customers' arrival

†Assuming also that the price and waiting time in all other stations in the area is fixed and known.

rate to the delay (in other words, it gives the number of customers entering the station per unit time as a function of the waiting time). The second function relates the waiting time to the arrival rate of customers. This second function, unlike a supply function, does not capture the behavior of any economic agent. Instead, it depicts merely a physical phenomenon, which in this case is the delay associated with queueing. This function is termed a *performance function.* Unlike the supply and demand functions, the performance function does not have the price or any other service characteristic as its argument, and it does not give the quantity consumed or produced. Instead, its argument is the quantity consumed (or the arrival rate in the case of the example considered here) and it gives the service characteristic (which is the delay in this case).

Figure 1.2a depicts a hypothetical performance curve showing how the

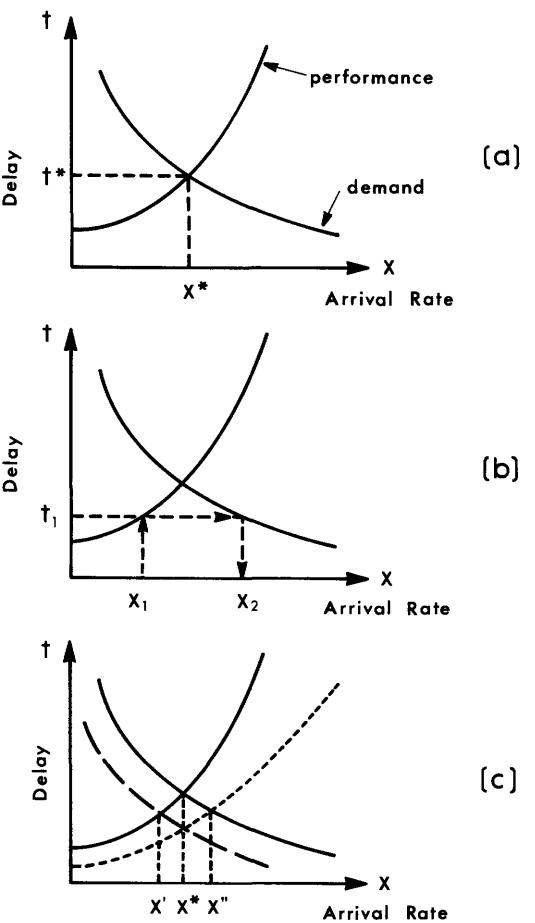


Figure 1.2 Demand/performance equilibrium for the gas station example: (a) equilibrium arrival rate and waiting time; (b) a waiting time that is too low; (c) a shifted demand curve (higher price) and a shifted performance curve (higher service rate).

delay, t, increases with the arrival rate, x. The figure also depicts a demand curve which shows the arrival rate associated with any given delay (this curve should be read by entering the vertical axis). When the arrival rate is low, there will be little queueing and the delay will be low, attracting more customers. When the arrival rate is very high, the delays are very high and no customers will join the queue. The only stable situation will be when the arrival rate is such that it creates a delay that corresponds to the same arrival rate. This equilibrium point is denoted by (x^*, t^*) in Figure 1.2a. Figure 1.2b shows a situation in which the arrival rate, x_1, is too low (i.e., $x_1 < x^*$). The delay associated with this arrival rate is t_1, in accordance with the performance function. This delay, however, generates an arrival rate of x_2 (where $x_2 > x^*$), in accordance with the demand function. As the arrival rate increases, the delay increases toward the equilibrium point where the delay matches the arrival rate.

Note that the demand function given in Figure 1.2 holds only when everything else (price, number of pumps, delay and price at other stations, etc.) is fixed. Since the demand is a function of both price and delay, a change in the price would show as a different demand function in the arrival rate/delay space. The demand curve associated with a higher price is depicted by the dashed line in Figure 1.2c. The equilibrium quantity here, x', will be lower than x^*, as expected. The dotted line in Figure 1.2c depicts a performance function associated with more pumps operating. In this case the delay associated with each arrival rate is smaller and the equilibrium quantity, x'', will be higher than x^*, again as expected.

The economics of the gas station operation can still be analyzed in the framework of supply/demand equilibrium. In this case the demand function gives the quantity consumed (or the customers' arrival rate) as a function of both price and delay. A supply function will give the price charged by the operator as a function of the quantity consumed. This supply/demand equilibrium, however, cannot be solved in isolation since the delay depends on the arrival rate. The latter relationships are given by a performance function. A complete economic analysis utilizes all three functions to solve, simultaneously, for the arrival rate, the delay, and the price. The demand/performance analysis given in the preceding paragraphs assumed that the price is fixed and known.

This example should clarify the differences between a supply/demand and performance/demand equilibrium. In a sense, the performance/demand analysis is a special case of the supply/demand equilibrium. It is obtained by fixing some of the variables in the supply/demand equilibrium problem.

The Transportation Arena

Transportation is a service that the transportation industry (including road builders, vehicle manufacturers, transit operators, traffic managers, etc.) offers travelers. As in many other service industries, the transportation prod-

ucts can be characterized by the level of service offered in addition to the price charged. Transportation level of service can be measured in terms of travel time, schedule convenience, reliability, safety, comfort, spatial coverage, accessibility to the service, and many other factors.

Many of the components of transportation level of service are not fixed but rather are flow dependent, in that their value depends on the usage of the transportation system. Consider, for example, a transit line connecting two points in an urban area. If many individuals choose to use this line, the buses will get congested, the waiting time will increase, the probability of obtaining a seat will be lower, and the driver is less likely to be friendly. The level of service, then, will be lower with increased patronage even if the fare remains fixed. In response, some patrons may drive instead of taking transit, others may change the timing of the trips, and yet others may forgo the trip altogether. This situation can be analyzed by using two functions: (1) a performance function that describes how the level of service deteriorates with increasing volume; and (2) a demand function that describes how the volume of passengers increases with improved level of service. These functions are defined for a given situation in terms of fare structure, schedule, equipment, and so on. As was the case with the aforementioned gas station example, the framework for a performance/demand equilibrium analysis is set by fixing those service characteristics that are under the operator's control. In the case of the gas station, this includes only the price. In the case of a transit line, these characteristics include a range of variables.

The dependence of the level of service on the flow is a fundamental characteristic of the transportation market. In the context of urban transportation, performance functions are rooted in the congestion phenomenon, which causes increased delays and costs with increased flow. The focus of this book is on the calculation of the demand/performance equilibrium in the urban passenger transportation market. The notion of equilibrium in the remainder of this book refers solely to this type of equilibrium.

The equilibrium in the urban transportation market is necessarily reached over the urban network of streets and transit lines. To define the notion of equilibrium over a transportation network, the network concept has to be presented and discussed. That is the topic of the next section.

1.2 NETWORK REPRESENTATION

The term "network" is commonly used to describe a structure that can be either physical (e.g., streets and intersections or telephone lines and exchanges, etc.) or conceptual (e.g., information lines and people, affiliation relationships and television stations, etc.). Each of these networks includes two types of elements: a set of points and a set of line segments connecting these points. This observation leads to the mathematical definition of a network as a set of *nodes* (or *vertices* or *points*) and a set of *links* (or *arcs* or *edges*) connecting

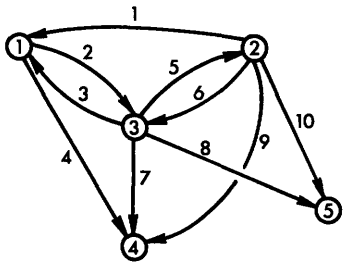


Figure 1.3 Directed network of links and nodes.

these nodes.† Figure 1.3 depicts a network including five nodes connected by 10 links. Each link in this network is associated with a direction of flow. For example, link 5 represents flow from node 3 to node 2, while link 6 represents the reverse flow, 2→ 3. This text deals exclusively with *directed* networks (in which all links are directed).

Each network link is typically associated with some impedance that affects the flow using it.‡ The units of measurement of this impedance depend on the nature of the network and the link flows. Impedance can represent electrical resistance, time, costs, utility, or any other relevant measure. When the flow involves people, the term "level of service" is usually used instead of "impedance." (The magnitudes of these terms are the opposite of each other; that is, a high level of service means low impedance.) Only links can be associated with impedance. Nodes represent merely the intersection of links and are not associated with any impedance to flow.

The networks discussed in this text are typically "connected." In other words, it is possible to get from any node to any other node by following a *path* (or a *route*) through the network. A path is a sequence of directed links leading from one node to another. The impedance along a path is the sum of the impedances along the links comprising that path. Note that a pair of nodes is usually connected by more than one path. For example, to get from node 2 in Figure 1.3 to node 4, the following paths (designated by the link numbers along the path) can be used: 9; 6, 7; 6, 3, 4; 1, 2, 7; 1, 4. In large networks, the number of paths connecting each pair of nodes can be extremely large.

Representation of the Urban Road Network

The focus of this text is on urban transportation. The roadway network in an urban area includes intersections and streets through which traffic moves. These elements can be translated naturally into a structure of nodes and links (including impedance measures), respectively. For a measure to

†The term "network" is used, then, to represent both a physical structure and its mathematical representation (known also as a graph).

‡Formally, the difference between a network and a graph is that network links are associated with impedance, whereas the links of a graph are not (they only represent connection and/or direction).

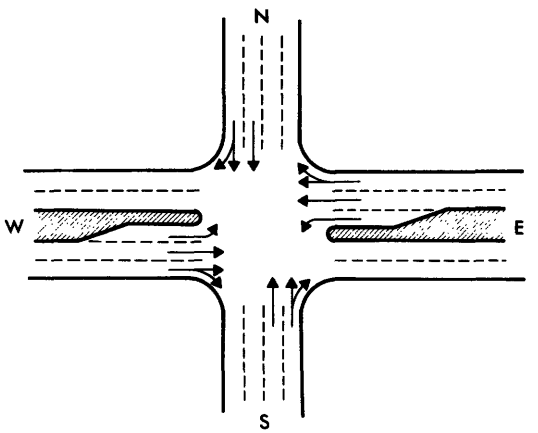


Figure 1.4 Four-approaches intersection layout.

properly represent link impedance it has to be a deterrent to flow. A particularly relevant measure for links representing urban streets is the time required to traverse the street. This impedance measure is discussed in detail in the last part of this section.

The graph representation of a physical network is not unique. In other words, there are many networks that can be used to represent the same physical structure. Consider, for example, the intersection shown in Figure 1.4. This intersection can be represented simply as a node with the streets leading to and from it represented by links connected to that node, as shown in Figure 1.5a. Note that a two-way street is represented by two opposite-direction links. The impedance on the links entering the intersection (the approaches) represents the intersection delay as well as the travel time along the approaching street. The impedance on the outbound links represents the travel time on the streets in that direction and the delay at downstream intersections.

The drawbacks of this representation are twofold. First, it cannot be

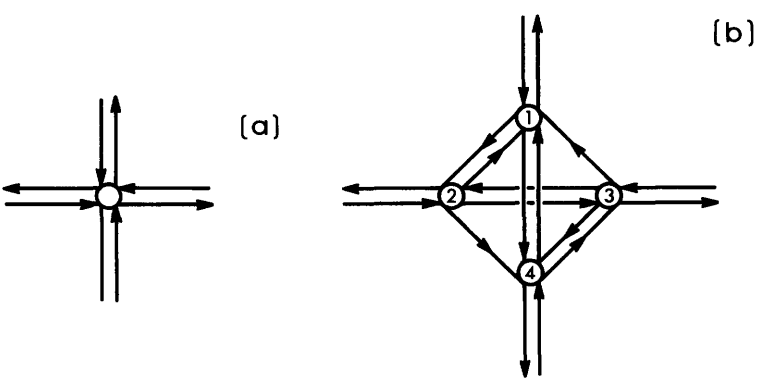


Figure 1.5 Network representation of the intersection in Figure 1.4: (a) representing the intersection as a node; (b) a detailed intersection representation.

used to represent turning restrictions. Second, it assumes that all flow entering the intersection from a particular direction will experience the same travel time, regardless of where it is destined. Clearly, right turns are easier to execute than straight-through movements, which, in turn, are easier to execute than left turns. These different delays can be accounted for by a more elaborate representation of the same intersection, shown in Figure 1.5b. In this figure the single node representation of the intersection is replaced by a network of four nodes and 10 links. Each permissible movement through the intersection is represented by a separate link. (Note that the graph is not completely symmetric due to the lack of left-turning movements from the northern and southern intersection approaches.) In this representation, each intersection movement can be associated with the appropriate delay. Link $3 \rightarrow 4$, for example, will be associated with left-turning delay that would be experienced by the traffic turning from the eastern approach into the southern one. Link $3 \rightarrow 1$, on the other hand, will represent right-turning maneuvers for flow from the east to the north, with the appropriate delay. In such a representation the links leading into the intersection will be associated only with the travel time on the incoming approaches, not with the intersection delay itself. The intersection delay is captured by the links connecting the four nodes in the figure.

Representation of the Transit Network

The movement of vehicular traffic through streets and intersections is not the only flow in the urban area; this text looks at the flow of transit passengers as well. A transit link can be represented by a simple linear network in which the transit stations (or bus stops) are represented by nodes and the line-haul portion by links. Such a graph is shown in Figure 1.6a. The impedance on each of the links in this network includes "in-vehicle" travel disutility elements such as travel time. Other measures that are not associated with the travel in the transit vehicle, however, serve also as travel impedance. These include, for example, the waiting time at the station and the fare charged. Furthermore, the transit station may be located away from the actual destination, requiring a walking trip from the point of alighting to the destination node. Figure 1.6b depicts the transit network associated with the flow between a particular origin node to a particular destination node. The loading link, in this figure, corresponds to the waiting time and fare, while the unloading link corresponds to the walking time. Note that in order for the various types of links to be compatible with each other, the impedance on all of the network links has to be expressed in the same units. If this were not the case, the impedance of a path could not be calculated.

A representation of a complete transit network would include not only loading and unloading links but also transfer links. The impedance on such links would be associated with line transfer charges (which may be lower than the fare at the start of the trip) and the waiting time at the transfer point. Figure 1.6c depicts a network representation of a transit station serving two

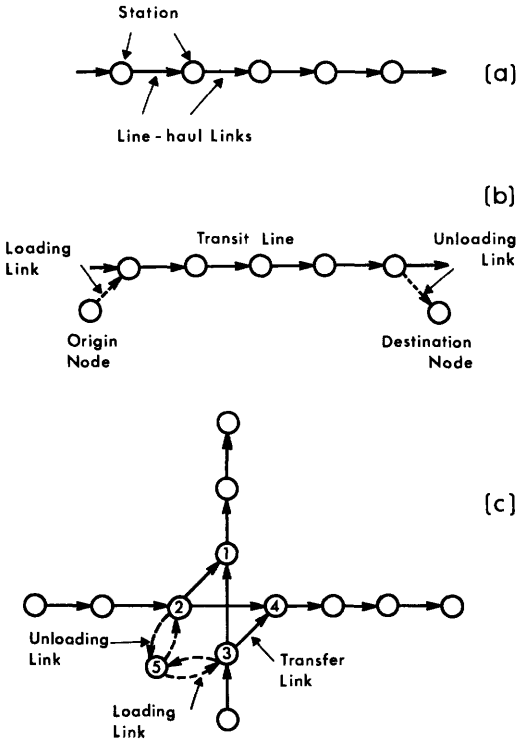


Figure 1.6 Transit-line representation: (a) a representation including only the "in-vehicle" movement; (b) a representation including boarding and alighting movements; (c) a detailed representation of a transit station, including transfer, boarding, and alighting links.

lines (west → east and south → north). The station itself is represented by the four nodes 1, 2, 3, and 4. Links 3 → 1 and 2 → 4 represent the continuation of a trip on the same line and are therefore associated with zero impedance. (Note that all four nodes represent the same physical facility.) Links 3 → 4 and 2 → 1 represent the transfer movements between the two lines. The impedance on these links include the transfer costs and delays expressed in appropriate units. Node 5 represents both a source for trips (i.e., it is an origin node) and a terminus for trips (i.e., it is also a destination node). This node is connected to both transit lines by loading and unloading links (shown as dashed lines in Figure 1.6c). It represents a set of trip origins and trip destinations which are located in the vicinity of the transfer station.

Centroids and Connectors

The transportation planning process for urban areas is typically based on a partition of the area into traffic zones. The size of each traffic zone can vary from a city block to a whole neighborhood or a town within the urban area. The number of traffic zones can vary from several dozens to several thousands. Each traffic zone is represented by a node known as *centroid* (the name stems from the practice of placing the node at the geometrical center of

gravity of the traffic zone). The network representation of the urban area will include many other nodes, representing intersections, bus stops, and other transportation facilities. The centroids, however, are those "source" and "sink" nodes where traffic originates and to which traffic is destined. (Node 5 in Figure 1.6c is a centroid.) Once the set of centroids is defined, the desired movements over an urban network can be expressed in terms of an *origin–destination matrix*. This matrix specifies the flow between every origin centroid and every destination centroid in the network.

Figure 1.7 shows a traffic zone that is surrounded by four two-way streets. The node in the middle of the zone is the centroid. It is connected to the roadway network by special *centroid connector* links (known also as "connectors" or "dummy links"). The centroid shown in the figure is a trip origin and therefore all the connectors are directed away from the centroid and into the nodes of the road network. Connectors directed toward the centroid would be added if trips can also terminate there.

Each centroid represents an aggregation of all the actual origins and all the actual destinations in its traffic zone. The centroid connectors represent the ubiquitous street network within a traffic zone. If the actual origin and destination points are uniformly distributed in the traffic zone, the connectors can be associated with the travel time between the geometrical centroid of each zone and the appropriate nodes. If the distribution of origin and destination points is not uniform, the connector travel times should be weighed accordingly.

The centroids can be connected directly to nodes that are part of the transit network. In this case, the connectors represent loading and unloading links, as shown in Figure 1.6c. The impedance on the loading centroid connector represents the walking time to the station, waiting at the station, and the transit fare. The impedance on the unloading connectors would represent the walking time from the transit station to the centroids.

The level of detail at which an urban area is represented is determined in large part by the size of the traffic zones. The analysis of the flows in the urban area does not focus on the flows within each traffic zone; the centroid and centroid connectors representation has to ensure only that the flows *between* the traffic zones are correct. If the flows within a certain zone are of interest to the transportation analyst, the zone under consideration should be divided

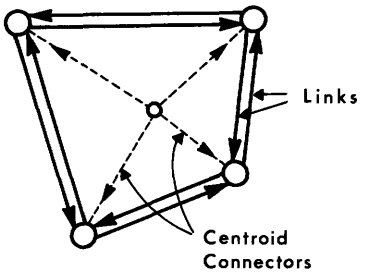


Figure 1.7 Network representation of a traffic zone, including a centroid node and centroid connector links.

into smaller zones and the road network between these zones modeled explicitly. Such detailed analysis is likely to be more accurate. Costs, however (including both direct analysis costs and data collection costs), escalate dramatically as the number of traffic zones and the size of the network increase. The size of the network to be analyzed is typically determined by this trade-off between the required accuracy and the available budget.

Similar considerations apply to the level of detail in which the road network itself is represented. Although it is important to show all the major streets and arterials, smaller streets may not have to be represented at all. The issue of network representation is as much an art as it is a science; practice and experience are required to carry it out successfully.

Link Performance Functions

The travel impedance, or level of service, associated with the links representing an urban network can include many components, reflecting travel time, safety, cost of travel, stability of flow, and others. The primary component of this impedance, however, is travel time, which is often used as the sole measure of link impedance. The reasons for using travel time in this context are threefold. First, empirical studies seem to indicate that it is a primary deterrent for flow. Second, almost all other possible measures of travel impedance are highly correlated with travel time and thus exhibit the same trends. Third, it is easier to measure than many of the other possible impedance components. Generalized impedance, which combines several measures, can, however, be used and in the remainder of this book, the term "travel time" can be understood as such a combined impedance. Furthermore, impedance measures other than travel times can be used explicitly in some parts of an urban network. For example, the appropriate impedance measure for links in the transit network are transit (in vehicle) time, waiting time, fare, and so on. To be compatible with the impedance over the vehicular traffic-carrying links of an urban network, these impedance measures should all be expressed in travel-time units.†

As mentioned in Section 1.1, the level of service (or impedance) offered by many transportation systems is a function of the usage of these systems. Because of congestion, travel time on urban streets and intersections is an increasing function of flow. Consequently, a *performance function* rather than a constant travel-time measure should be associated with each of the links representing the urban network. The performance function relates the travel time (which, as mentioned above, may stand for generalized impedance) on each link to the flow traversing this link.

A performance function for a typical approach to a signalized intersection is shown in Figure 1.8. This function captures both the time spent in

†The conversion coefficient should be rooted in travelers' values. The estimation of such coefficients is discussed in Chapter 13.

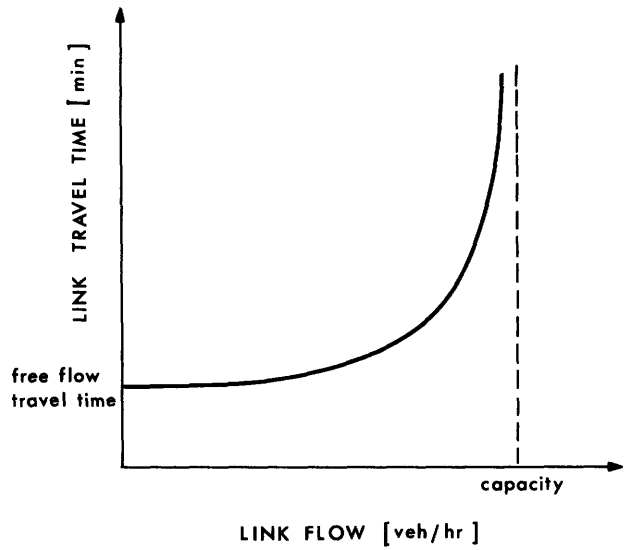


Figure 1.8 Typical link performance function for an approach to a signalized intersection.

traveling along the approach under consideration and the delay at the downstream intersection. The shape of the curve shown in the figure is typical of performance functions for links representing urban network components.

The travel time at zero flow is known as the free-flow travel time. At this point, a traveling car would not be delayed because of interaction with any other car moving along the link. The only source of delay at this point is the time associated with traversing the link and the expected delay associated with the probability of being stopped by a red signal indication. As the flow increases, the travel time monotonically increases since both the travel time along the approach increases (because of vehicle interactions at higher traffic densities) and the intersection delay increases (because of queueing phenomena) with the flow. Characteristically, the performance function is asymptotic to a certain level of flow known as the *capacity* of the transportation facility under consideration. The capacity is the maximum flow that can go through any transportation facility. The performance function is undefined for higher values of flow, since such flows cannot be observed. When the flow approaches capacity, the queues at the intersection will start growing, clogging upstream intersections and finally causing traffic to come to a halt.

The general shape of the performance functions is similar for links representing most types of urban streets. The physical characteristics of each street (e.g., length, width, parking restrictions, turning pockets, or signal green time) determine the exact parameters of the function for each street.

The centroid connectors are typically modeled as fixed travel-time links. In other words, the travel time on the connectors does not vary with the flow since these links represent a ubiquitous local street network and not a particu-

lar facility. Similarly, loading and unloading links represent time and cost components that do not vary with the flow.

It is important to remember that the physical network and its graph representation do not have to be similar. As shown in later chapters, links can be used to represent many types of flow, some of which do not correspond to physical movements. For example, in Chapter 6 the individuals who choose not to travel at all are represented as a flow on an imaginary link added to the network. It is always true, however, that each link represents a distinct flow.

The next section discusses equilibrium over urban transportation networks, building on the system analytic approach and the equilibrium framework introduced in Section 1.1.

1.3 EQUILIBRIUM OVER URBAN TRANSPORTATION NETWORKS

The notion of equilibrium in the analysis of urban transportation networks stems from the dependence of the link travel times on the link flows. Assume that the number of motorists who wish to travel between a given origin point and a given destination point is known. Furthermore, assume that these points are connected by several possible paths. The question of interest here is how these motorists will be distributed among the possible paths. If all of them were to take the same path (which may initially be the shortest one in terms of travel time), congestion would develop on it. As a result, the travel time on this path might increase to a point where it is no longer the minimum travel-time path. Some of these motorists would then use an alternative path. The alternative path can, however, also be congested, and so on.

The determination of the flows on each of these paths involves a solution of a demand/performance equilibrium problem. The flow on each link is the sum of the flows on many paths between many origins and destinations. A performance function is defined independently for each link, relating its travel time to this flow. The demand for travel, however, is rooted in motorists' behavior and is not defined for each link separately. Instead, it specifies how motorists choose among the alternative paths (routes) connecting each origin–destination (O–D) pair. This dichotomy in the definition of the performance and the demand functions gives this performance/demand equilibrium analysis its "systems" nature. In other words, this is why no link, path, or origin–destination pair can be analyzed in isolation.

This section discusses the route choice rule and the associated definitions of equilibrium used in this text.

Definitions of Equilibria

This text analyzes a variety of urban network equilibrium problems, including both the transit and automobile mode and encompassing several possible travel decisions. The basic problem, however, is presented here for a

simplified case, including a network representing only the automobile flows. The only travel choice examined here is the motorists' choice of routes between their origin and their destination. This problem can be put as follows:

Given:

1. A graph representation of the urban transportation network
2. The associated link performance functions
3. An origin–destination matrix

Find the flow (and travel time) on each of the network links.

This problem is known as that of *traffic assignment* since the issue is how to assign the O–D matrix onto the network. The resulting link flows are then used to calculate an array of measures, which can be used, in turn, to evaluate the network. Particular designs of transportation infrastructure or transportation control policies usually enter the analysis through the specification of the network itself and the performance functions.

To solve the traffic assignment problem, it is required that the rule by which motorists choose a route be specified. As explained above, this rule can be viewed as the function or the procedure that specifies the demand for travel over paths. The interaction between the routes chosen between all O–D pairs, on the one hand, and the performance functions on all the network links, on the other, determines the equilibrium flows and corresponding travel times throughout the network.

It is reasonable to assume that every motorist will try to minimize his or her own travel time when traveling from origin to destination. As explained above, this does not mean that all travelers between each origin and destination pair should be assigned to a single path. The travel time on each link changes with the flow and therefore, the travel time on several of the network paths changes as the link flows change. A stable condition is reached only when *no traveler can improve his travel time by unilaterally changing routes.* This is the characterization of the *user-equilibrium* (UE) condition.

Since individual motorists can be expected to behave independently, the UE situation ensures that at this point there is no force that tends to move the flows out of the equilibrium situation. Consequently, this point will be stable and, in fact, a true equilibrium.

The user-equilibrium condition, however, is not the only possible definition of equilibrium. The assumption that each motorist chooses the minimum-travel-time route may be reasonable in some cases, but it includes several presumptions that cannot always be assumed to hold even approximately. For example, the UE definition implies that motorists have full information (i.e., they know the travel time on every possible route) and that they consistently make the correct decisions regarding route choice. Furthermore, it assumes that all individuals are identical in their behavior. These presumptions can be

relaxed by making a distinction between the travel time that individuals *perceive* and the actual travel time. The perceived travel time can then be looked upon as a random variable distributed across the population of drivers. In other words, each motorist may perceive a different travel time, over the same link. Equilibrium will be reached when *no traveler* believes *that his travel time can be improved by unilaterally changing routes.* This definition characterizes the *stochastic-user-equilibrium* (SUE) condition.

The stochastic user equilibrium is a generalization of the user-equilibrium definition. If the perceived travel times are assumed to be entirely accurate, all motorists would perceive the same travel time and the stochastic user equilibrium will be identical to the (deterministic) user equilibrium. In other words, the flow patterns resulting from both models would be identical.

The definitions of equilibria given above are not easy to use in an operational manner to solve for equilibrium flow patterns. To be useful, the equilibrium definitions have to be characterized and formulated mathematically. The following section discusses the user equilibrium, while the more general stochastic user equilibrium is treated only in Part IV.

An important point, however, should be mentioned here with regard to both types of equilibria. The demand for urban travel is derived from the pattern of activity in the urban area. The timing and locations of these activities mean that travel demand is not uniform throughout the day. Steady-state equilibrium analyses of the type discussed here, however, are applicable only if the flows can be considered stable over the period of analysis. Consequently, transportation planners analyze urban transportation systems for certain time periods such as the "morning peak," "evening peak," or "midday," depending on the purpose of the analysis. The origin–destination flows within each of these respective periods is considered constant in order for steady-state analyses to apply. The larger the period of analysis, the less accurate this assumption is. The period of analysis cannot be very small, however, since each such period has to be appreciably longer than the typical duration of trips at this time.

A Simple User-Equilibrium Example

Consider the two-link network shown in Figure 1.9a. This network represents one origin–destination pair connected by two alternative routes. Let t_1 and t_2 represent the travel time on links 1 and 2, respectively, and let x_1 and x_2 represent the traffic flow on these links. The total origin–destination flow is designated by q, where

$$q = x_1 + x_2 \qquad [1.1]$$

The performance functions on these links $t_1(x_1)$ and $t_2(x_2)$ are shown in Figure 1.9b. For each link, the performance function gives the travel time on that link as a function of the flow on the link.

Assume now that the trip rate between O and D is very small. In other

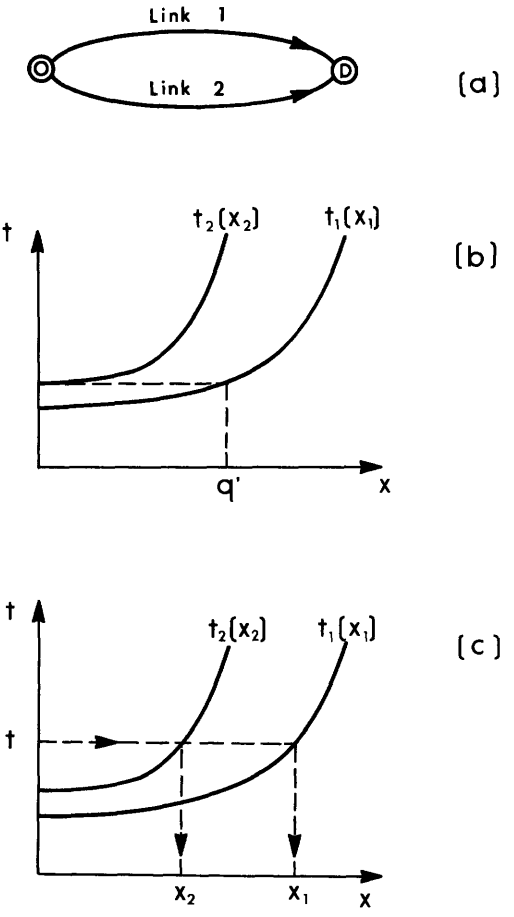


Figure 1.9 Equilibrium in a simple network: (a) a two-link network; (b) the two performance functions—if the flow is less than $q' = x_1 + x_2$, it is assigned to link 1; (c) the travel time on both links is equal if the flow is higher than q'.

words, q is very small. If all motorists are trying to minimize their own travel time, each of the q motorists should choose to travel over link 1. As shown in the figure, this link is associated with a lower free-flow travel time than link 2. If q is small, the increased delay due to the traffic on link 1 is not sufficient to increase the travel time on this link even to the point where it is equal to the free-flow travel time on link 2. Thus, all q motorists will use link 1 and no one will use link 2. This is an equilibrium situation since none of the motorists using link 1 has an incentive to switch routes to the longer link. Such an equilibrium will hold as long as $q < q'$, where q' is the flow that causes the travel time on link 1 to equal the free-flow travel time on link 2. At this point, an additional motorist may choose either link. If the additional motorist chooses link 2, the travel time on it will increase and the next motorist will choose link 1. If on the other hand, the first motorist (above q') chooses link 1, the next one will choose link 2.

Looking at the flow of traffic as a continuous flow, it is clear that beyond

the point $q = q'$, equilibrium can be maintained only if the travel time on both links is equal. Beyond this point both links are used, and if the travel times are not equal, some motorists can change route and lower their own travel time. The route-switching process will not occur only if the travel time on both routes is equal, giving motorists no incentive to switch.

The two characterizations of equilibrium that can occur in this two-link example (the first for $q < q'$ and the second for $q > q'$) give rise to an operational definition of user equilibrium over transportation networks:

UE Definition. For each O–D pair, at user equilibrium, the travel time on all used paths is equal, and (also) less than or equal to the travel time that would be experienced by a single vehicle on any unused path.

This definition means that at equilibrium, the paths connecting each O–D pair can be divided into two groups. The first group includes paths that carry flow. The travel time on all these paths will be the same. The other group includes paths that do not carry any flow. The travel time on each of these paths will be at least as large as the travel time on the paths in the first group.

Using this definition, the two-link example in Figure 1.9 can now be solved for any value of q. The assignment of any amount of flow that is less than q' is obvious—it should all be assigned to link 1. The only problem is to ensure that the traffic assignment is such that beyond the point of $q = q'$, both links are assigned at a rate that keeps the travel time on these links equal. If the equilibrium travel time, t, between the origin and the destination is known (for the case in which $q > q'$), the equilibrium definition above means that $t = t_1 = t_2$. Consequently, the appropriate link flows can be determined by the inverse link performance functions [i.e., $x_1 = t_1^{-1}(t)$ and $x_2 = t_2^{-1}(t)$]. Graphically, this determination can be accomplished by entering the link performance curves horizontally, given t, as shown in Figure 1.9c. This method is applicable at any value of t, even those associated with $q < q'$. The problem then is to determine t.

The equilibrium travel time can be determined by creating a new "performance curve" which depicts the O–D travel time as a function of the O–D flow. This curve can be constructed by summing up the link performance function horizontally, as shown in Figure 1.10. The curve $t(q)$ in this figure is an O–D performance function giving the equilibrium O–D travel time as a function of the total O–D flow. The construction of the O–D performance function in this fashion ensures that for each value of the travel time, the O–D flow is the sum of the flows on the individual links. Once this curve is constructed, the equilibrium travel time can be determined by simply entering this function with the O–D flow. Given the equilibrium travel times, the flows on the individual links can be determined graphically, as shown in the figure. Note that the O–D performance function coincides with link 1 performance function for $q \leq q'$. This means that for $q \leq q'$ the equilibrium travel time will

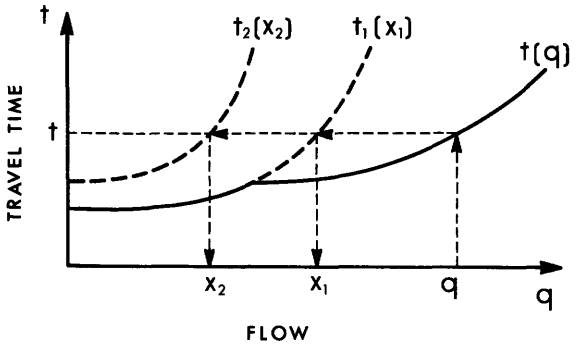


Figure 1.10 The O–D performance function, $t(q)$, is used to find the O–D travel time, t and equilibrium flows, x_1 and x_2.

be given by the travel time on link 1. The travel time on link 2 will be higher for this range of O–D flow, as required by the user equilibrium definition.

The graphical method used to solve this small two-link network example cannot be used to solve large networks. In such networks, the number of paths connecting each O–D pair may be so large that it will be prohibitively expensive to enumerate them all, even by using modern computers. Furthermore, the flow traversing each link results from the assignment of trips between many origins and many destinations. Consequently, as mentioned before, the entire network has to be solved simultaneously.

Outline

The methods used to determine the equilibrium flows and travel times described in this book are based on nonlinear optimization techniques. Part II includes a review of the mathematical background needed for such analysis. At the same time, it presents an application of these methods to the formulation and solution of user equilibrium problems over large networks.

Part III extends the basic user-equilibrium framework in two main dimensions. First, it includes travel choices other than route choice in the analysis. These include the choice of mode of travel, trip destination, and trip frequency. Second, it applies the analysis to cases in which the performance of each link depends on the flows on several (and possibly all) of the network links.

Part IV extends the analysis in yet another dimension by removing a number of the simplifying assumptions leading to the definition of the user equilibrium. This part of the book discusses the concept of stochastic user equilibrium, which was mentioned earlier. It looks at the process of route choice in the context of random utility theory, applying appropriate choice models to this process.

The last part of the book (Part V) focuses on the data needed for equilibrium analysis of urban transportation systems. It explains how O–D matrices

can be derived and presents the necessary background in traffic engineering needed for the derivation of link performance functions.

1.4 SUMMARY

The problem addressed in this book is that of describing the distribution of traffic and transit passenger flows through an urban transportation network. The characteristics of the network, in terms of physical dimensions, control strategies, and operations are assumed to be fixed during the analysis period. Similarly, the pattern of activities that generates the trips in the urban area is assumed to be constant. The analysis offered is therefore descriptive in nature. It should be used for studying various scenarios, each corresponding to a possible state of any one (or a combination of) the aforementioned constant characteristics.

The approach used to solve this problem is based on the notion of equilibrium. Unlike economic equilibrium between supply and demand, the equilibrium discussed here is between performance and demand. Instead of being determined in the quantity/price space, the equilibrium under consideration is being determined in the flow/level of service (or flow/impedance) space. The flow in this case is expressed in terms of the number of travelers (or vehicles) using each component of the transportation network per unit time, while the level of service is expressed in terms of the travel time borne by these travelers. The interdependencies between the components of the network necessitates a system-based analytical approach in which the equilibrium flows and travel times throughout the network have to be determined simultaneously.

The solution approaches are developed with regard to a network representation of the physical structure of streets and transit lines. The network representation includes nodes and directed links. Each link is associated with a single flow variable and a measure of travel time. Nodes are not associated with travel time or any other impedance measure.

The graph representation of the urban network is not unique in that there are many graphs that can represent the same network. The choice of representation depends on the level of detail at which the network is modeled, which, in turn, is a function of the available data and the analysis budget.

Given the origin–destination trip rates, the demand for urban travel can be formulated in terms of the rule by which motorists (and passengers) choose a route from their origin to their destination. Different assumptions regarding these choice mechanisms lead to different equilibrium flow patterns. All route-choice models, however, assume that motorists minimize their travel time through the network. The difference between the two types of equilibria discussed in this book is that user equilibrium assumes that motorists know all link travel times with certainty. Stochastic user equilibrium models are based

on the assumption that each motorist may perceive a different travel time and act accordingly.

1.5 ANNOTATED REFERENCES

The notion of demand/performance equilibrium over urban networks is presented in several transportation planning textbooks, including Morlok (1978) and Manheim (1979). In particular, the early work of Beckman et al. (1956) contains many important conceptual developments. These authors trace a description of an equilibration process over a simple traffic network to Knight (1924). A discussion of transportation networks and their representation is given by Potts and Oliver (1972), Newell (1980), and others.

The operational definition of the user equilibrium given in Section 1.3 was suggested by Wardrop (1952). The concept of stochastic user equilibrium was developed and formalized by Daganzo and Sheffi (1977).

1.6 PROBLEMS

1.1. Consider a raffle in which the prize money is known and the ticket price is constant. The number of people buying a ticket is a function of the winning probability. This probability, in turn, is a function of the number of people buying tickets. Study this situation in the framework of a demand/performance equilibrium. Show, graphically, the equilibrium point and the forces that push toward this point.

1.2. Find the user equilibrium flow patterns and the equilibrium travel time for the network described in Figure P1.1. The performance functions are

$$t_1 = 2 + x_1^2$$

$$t_2 = 1 + 3x_2$$

$$t_3 = 3 + x_3$$

where t_a and x_a denote the travel time and flow, respectively, on link a (where $a = 1, 2, 3$). The origin–destination trip rate is 4 units of flow going from node 1 to node 3.

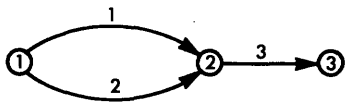


Figure P1.1

1.3. The network in Figure P1.2 includes many links connecting the origin node and the destination node. The performance curve on each link is linear, that is,

$$t_a = A_a + B_a x_a$$

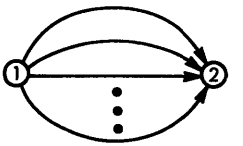
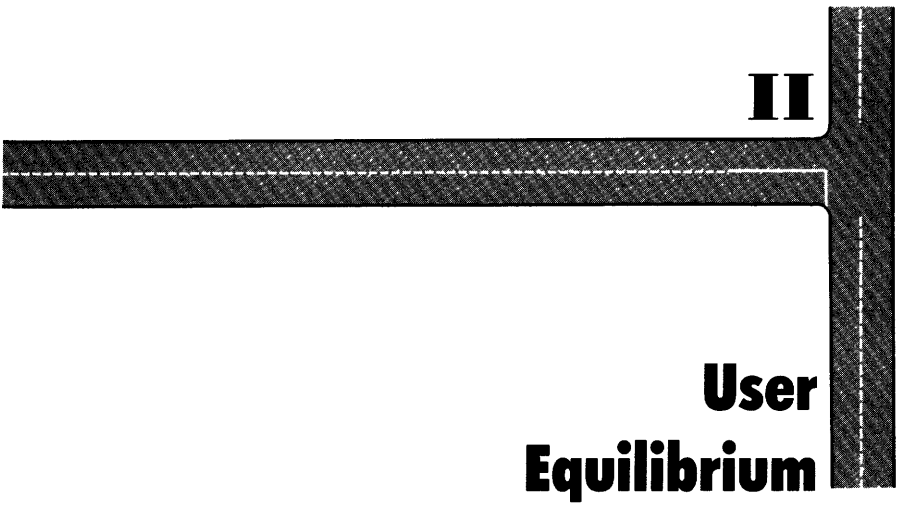


Figure P1.2

where A_a and B_a are constants and, as in Problem 1.2, t_a and x_a represent the travel time and flow, respectively, on link a. The origin–destination trip rate is q.

(a) Assuming that you know that at equilibrium, all links in this network carry flow, develop an expression for the flow on each link.

(b) How would you find the flow on each link in cases where it is not clear that all links are used at equilibrium (i.e., only a subset of the links may be used)?



II

User Equilibrium

Part II formulates and solves the standard user-equilibrium problem. The problem is to find the set of flows that corresponds to the user-equilibrium conditions set forth in Chapter 1. This part contains four chapters. Chapter 2 reviews, in short, some concepts in mathematical programming. These concepts are used in Chapter 3 to formulate the equilibrium problem as a mathematical program. Chapter 4 reviews algorithmic approaches to the solution of mathematical programs, and Chapter 5 concentrates on the solution of the user-equilibrium program.

2

Basic Concepts in Minimization Problems

This chapter reviews some concepts related to the formulation and solution of mathematical minimization programs. The focus of the discussion is on the conditions that characterize the solution of such programs and the conditions under which a solution may be unique.

In solving a mathematical program, the problem is to choose the values of a set of I variables, $x_1, x_2, \ldots, x_I$, that minimize a function (usually termed the *objective function*) $z(x_1, \ldots, x_I)$, yet comply with certain *constraints*. Each constraint can be expressed as an inequality condition on some function, $g(x_1, \ldots, x_I)$, of the variables; the set of all possible values of $x_1, \ldots, x_I$ that comply with all the constraints is known as the *feasible region*. A general minimization program with J constraints can be written (in standard form) as

$$\min z(x_1, \ldots, x_I)$$

subject to

$$\begin{aligned} g_1(x_1, \ldots, x_I) &\geq b_1 \\ g_2(x_2, \ldots, x_I) &\geq b_2 \\ &\vdots \\ g_J(x_1, \ldots, x_I) &\geq b_J \end{aligned}$$

where $g_j(x_1, \ldots, x_I) \geq b_j$ denotes the jth constraint on the values of the variables. These equations can be simplified by using vector notation. Accordingly, let $\mathbf{x}$ denote the array (vector) of decision variables, that is, $\mathbf{x} = (x_1, \ldots, x_I)$.

Using such notation, the program above can be written as

$$\min z(\mathbf{x}) \qquad [2.1a]$$

subject to

$$g_j(\mathbf{x}) \geq b_j; \qquad j = 1, \ldots, J \qquad [2.1b]$$

This is the standard form used in this text to write minimization programs. Note that all constraints are specified using a "greater than or equal to" sign. Constraint of the type "less than or equal to" can be converted to this standard form by multiplying them by -1. Equality constraints of the type $g_j(\mathbf{x}) = b_j$ can be expressed as the pair of constraints $g_j(\mathbf{x}) \geq b_j$ and $g_j(\mathbf{x}) \leq b_j$ (where the latter can be multiplied by -1 to conform to the standard form).

The solution to this program, $\mathbf{x}^*$, is a feasible value of $\mathbf{x}$ that minimizes $z(\mathbf{x})$, that is,

$$z(\mathbf{x}^*) \leq z(\mathbf{x}) \qquad \text{for any feasible } \mathbf{x} \qquad [2.2a]$$

and

$$g_j(\mathbf{x}^*) \geq b_j \qquad \forall\, j \in \mathcal{J} \qquad [2.2b]$$

where $\mathcal{J}$ is the set of constraint indices $(1, 2, \ldots, J)$†. Condition [2.2a] calls for the solution, $\mathbf{x}^*$, to have the lowest possible value of the objective function, while Eq. [2.2b] ensures that this solution satisfies the constraints. Note that the "less than or equal to" condition in Eq. [2.2a] means that there may be some value of $\mathbf{x}$ other than $\mathbf{x}^*$ which solves the program expressed in Eqs. [2.1] (hence the aforementioned concern with the conditions for uniqueness of the solution). Note also that there may be no value of $\mathbf{x}$ that satisfies all the constraints (i.e., Eqs. [2.2b] hold for no $\mathbf{x}$), in which case the program has no solution.

In the following chapter and in the remainder of this book, it is generally assumed that there is always at least one value of $\mathbf{x}$ that satisfies the constraints. Furthermore, this book deals only with programs that possess a finite minimum which occurs at a finite value of $\mathbf{x}$. To ensure the existence of such a minimum, these programs have to satisfy certain regularity conditions.‡ In addition, this book deals only with objective functions and constraints that are all continuously differentiable.

This chapter focuses on the characteristics of the optimal solutions to mathematical programs. Three topics are each discussed separately: single-variable minimization programs, multivariable programs, and some special

†The notation "$\forall$" means "for every." In this case, "$\forall\, j \in \mathcal{J}$" means "for every j in the set $\mathcal{J}$" (i.e., $j = 1, 2, \ldots, J$). When it is clear what the relevant set is, such notation is usually shortened in this book to "$\forall\, j$", meaning "for every j in the relevant set."

‡The objective function has to be continuous over the feasible region, which has to be closed and bounded.

cases of multivariable programs that are used frequently in the analysis of network equilibrium programs.

2.1 PROGRAMS IN ONE VARIABLE

The discussion of single-variable minimization is divided into sections on unconstrained and constrained problems. For each of these cases, included are discussion of the first-order conditions, which characterize the solution, and of the second-order conditions, which deal with its uniqueness. In both cases, it is assumed that the existence and smoothness conditions mentioned above hold (either for any value of the argument, in the case of an unconstrained program, or for the feasible values, in the case of constrained programs).

Unconstrained Minimization Programs

It is well known from elementary calculus that the necessary condition for a differentiable function in one variable, $z(x)$, to have a minimum at $x = x^*$ is that the derivative of $z(x)$ evaluated at x^* equals zero. In other words,

$$\frac{dz(x^*)}{dx} = 0 \qquad [2.3]$$

This is a *first-order* condition (involving only first derivatives) for a minimum.† If there is a minimum at x^*, this condition must hold. As demonstrated in Figure 2.1, however, the first-order condition is not sufficient to ensure that x^* is a minimum of $z(x)$. In this figure $dz(x)/dx = 0$ for four values of x, namely $x = a$, $x = b$, $x = c$, and $x = d$; all of which are *stationary points* of $z(x)$. Note that $x = a$ and $x = d$ are *local minima* (i.e., they minimize the function in their immediate vicinity), while $x = b$ is a point of inflection and $x = c$ is a (local) maximum.

To prove that a stationary point (such as $x = a$ in Figure 2.1) is a minimum, two characteristics have to be demonstrated: (1) that it is a local minimum and not a local maximum or an inflection point, and (2) that it is a global minimum. In other words, the value of $z(x)$ at x^* is lower than $z(x)$ at any other x (including, for example, all the other local minima, such as $x = d$ in Figure 2.1).

A function with a single minimum is called *ditonic* or *unimodal*.‡ A function is ditonic if it is strictly decreasing to the left of the minimum and strictly increasing to the right. Thus, if a function is ditonic, its stationary point is a global minimum.

†Here and later in the book, $dz(x^*)/dx$ denotes the derivative of $z(x)$ evaluated at x^*.

‡The term "unimodal" is sometimes reserved for functions with a single maximum and the term "ditonic" to functions with a single minimum (i.e., the negative of a unimodal function is a ditonic function). In this book these terms are used interchangeably to denote a function with a single minimum.

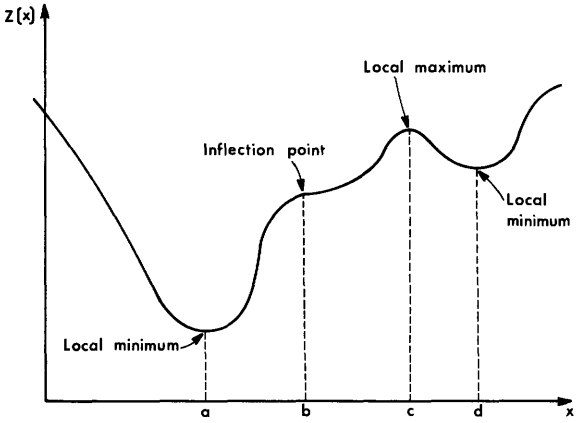


Figure 2.1 Stationary points of $z(x)$.

A sufficient condition for a stationary point to be a local minimum is for the function to be *strictly convex* in the vicinity of the stationary point. Intuitively, strict convexity means that the function is "heavy in the middle," as illustrated in the vicinity of $x = a$ and $x = d$ of Figure 2.1. Formally, strict convexity means that a line segment connecting any two points of the function lies entirely above the function. In other words, a function is strictly convex if

$$z[\theta x_1 + (1 - \theta)x_2] < \theta z(x_1) + (1 - \theta)z(x_2) \qquad [2.4a]$$

for any two distinct points x_1 and x_2 and for any value of θ where $0 < \theta < 1$. These relationships are shown in Figure 2.2a for a strictly convex function. (Note that this condition does not require that the function under consideration be differentiable.) Alternatively, strict convexity of differentiable functions can be determined by testing whether a linear approximation to the function always underestimates the function. Thus $z(x)$ is strictly convex at x_1 if and only if

$$z(x_1) + \frac{dz(x_1)}{dx}(x_2 - x_1) < z(x_2) \qquad [2.4b]$$

for any point x_2 that is not equal to x_1. This condition is demonstrated in Figure 2.2b. Finally, if a function is twice differentiable in the vicinity of a stationary point, the strict convexity condition is equivalent to requiring that the second derivative of $z(x)$ at x^* be positive, that is,

$$\frac{d^2z(x^*)}{dx^2} > 0 \qquad [2.4c]$$

Strict convexity can thus be defined by the line segment condition [2.4a], by the linear approximation condition [2.4b] (for continuously differentiable functions), or by the second derivative condition [2.4c] (for twice-continuously

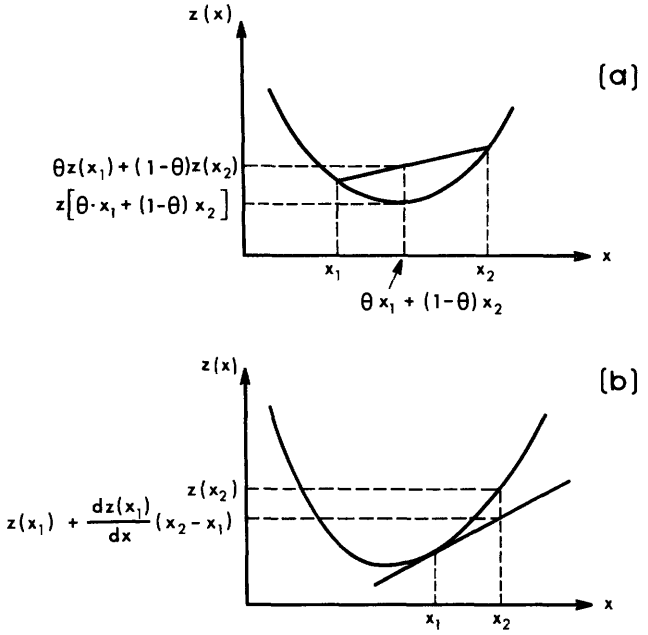


Figure 2.2 Strict convexity can be demonstrated (a) by showing that a line segment connecting any two points lies entirely above the function, or (b) by showing that a linear approximation always underestimates the function.

differentiable functions). If a function is strictly convex in the vicinity of a stationary point, this point is a local minimum.

If a function includes more than one local minimum, it is usually difficult to demonstrate that a particular local minimum is also a global one. Doing so may require identification of all the local minima (a process that may involve repeated solutions) and a comparison of the objective function values at these points. Such a task can be very difficult and is sometimes impossible. If, however, x^* is the only local minimum of $z(x)$, then, naturally, it is also a global minimum. To demonstrate that a local minimum is unique, it is sufficient to show that $z(x)$ is convex for all values of x (i.e. that it is globally convex).

Convexity means that the inequality in Eq. [2.4a] is replaced by a "less than or equal to" sign, meaning that a line segment never lies below the function. Similarly, convexity means that a linear approximation to the function never overestimates it and that the second derivative is nonnegative.

To understand second-order conditions better, note that a function $z(x)$ can have a unique minimum and not be globally convex, and that convexity alone does not guarantee the uniqueness of the minimum. Figure 2.3 illustrates three possible situations: In Figure 2.3a $z(x)$ is convex, but it is not strictly convex in the vicinity of x^*—this function does not have a unique minimum.

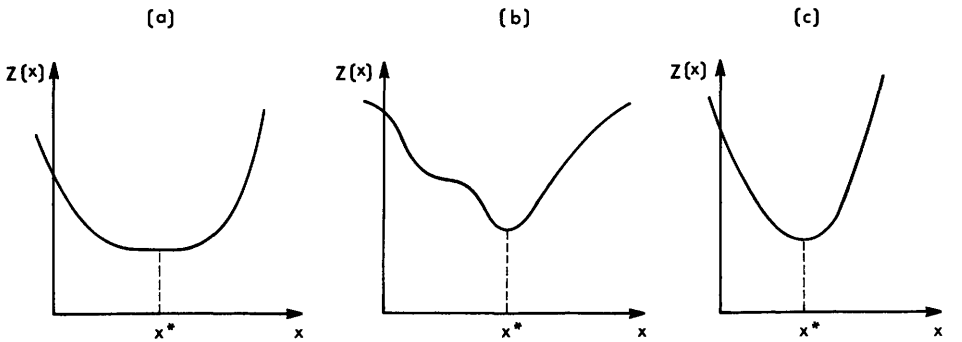


Figure 2.3 Convexity and unimodality. (a) $z(x)$ is convex but not strictly convex; its minimum is not necessarily unique. (b) $z(x)$ is not convex but it is strictly convex at the vicinity of x^*; it is also ditonic and thus x^* is a unique minimum. (c) $z(x)$ is convex everywhere and strictly convex at the vicinity of x^*, which is the unique minimum.

Figure 2.3b depicts a case in which $z(x)$ is not convex even though it is strictly convex in the vicinity of x^*. This function is ditonic and thus its only stationary point is also its unique minimum. In general, however, for a function of this shape it would be hard to demonstrate the uniqueness of the minimum because the function is not convex. Figure 2.3c illustrates a convex function that is strictly convex in the vicinity of its unique minimum, x^*.

In summary, then, it is sufficient to show that a function $z(x)$ is strictly convex at $x = x^*$ and convex elsewhere in order to establish the fact that x^* minimizes that function. A necessary condition, however, for x^* to be a minimum is for the first derivative to vanish at that point. (This is the first-order condition for a minimum.)

This review of programs in a single variable is extended in the next section to constrained minimizations. Again, the focus of the discussion is on the first- and second-order conditions for a minimum.

Constrained Minimization Programs

In constrained minimization it is possible to have a minimum where the first derivative does not vanish. As a result, the first-order conditions have to be generalized in order to include this possibility. Consider, for example, Figure 2.4, showing the function $z(x)$ defined over the feasible region $a' \leq x \leq a''$. In the first case (Figure 2.4a), the minimum point is internal (inside the feasible region) and the condition $dz(x^*)/dx = 0$ holds, while in Figure 2.4b the minimum point is on the boundary, at $x = a'$. Clearly, $dz(a')/dx \neq 0$.

When the minimum is on the boundary of the feasible region, however, the following observation can be made: If the constraint on which the solution lies (the binding constraint) is of the type $x \geq (\cdot)$, then $z(x)$ must be increasing

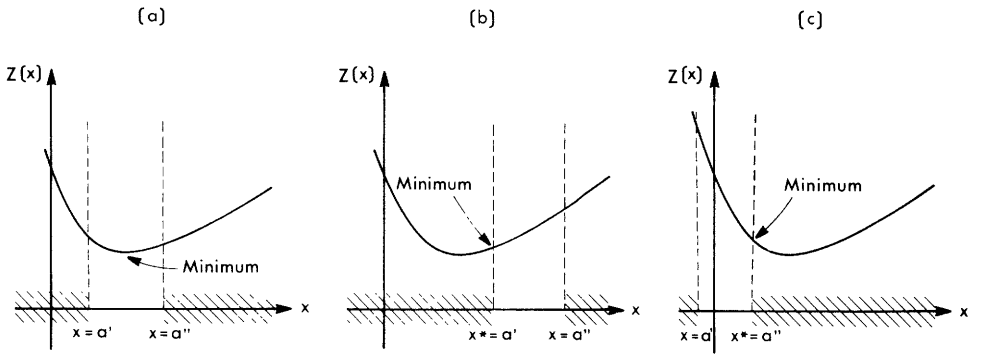


Figure 2.4 Constrained minimum: (a) internal minimum—$dz(x^*)/dx = 0$ inside the feasible region; (b) minimum on the boundary of the feasible region—the binding constraint is $x \geq a$ and $dz(x)/dx \geq 0$; (c) minimum on the boundary—the binding constraint is $x \leq b$ and $dz(x^*)/dx \leq 0$.

(or more precisely nondecreasing) at x^*, as is the case in Figure 2.4b. If, on the other hand, the binding constraint is of the type $x \leq (\cdot)$, then $z(x)$ must be decreasing (actually, nonincreasing) at x^*, as in the case shown in Figure 2.4c.

To characterize these situations mathematically in a statement of first-order conditions, the problem depicted in Figure 2.4 is first written in a standard form, as follows (see Eqs. [2.1]):

$$\min z(x) \qquad [2.5a]$$

subject to

$$x \geq a' \qquad [2.5b]$$

$$-x \geq -a'' \qquad [2.5c]$$

where constraints [2.5b] and [2.5c] are expressed as $g_j(x) \geq b_j$ [in this case $g_1(x) \equiv x$ and $b_1 \equiv a'$, while $g_2(x) \equiv -x$ and $b_2 \equiv -a''$].

When the constraints are expressed in standard form, the aforementioned observation about the slope of $z(x)$ at x^* means that the derivative of $z(x)$ at x^* and the derivative of the binding constraint at this point $[dg(x^*)/dx]$ will always have the same sign (or their product will be zero, that is, they will never have opposite signs). This observation can be used to extend the first-order conditions because it holds whenever a minimum occurs on the boundary of the feasible region as well as for an unconstrained solution. In the example given in Figure 2.4b, this condition can be verified since

$$\frac{dg_1(x^*)}{dx} = 1$$

and from the figure

$$\frac{dz(x^*)}{dx} > 0$$

whereas for Figure 2.4c,

$$\frac{dg_2(x^*)}{dx} = -1 \quad \text{and} \quad \frac{dz(x^*)}{dx} < 0$$

Since both derivatives are scalars, the condition of same signs can be written as

$$\frac{dz(x^*)}{dx} = u_j \frac{dg_j(x^*)}{dx} \qquad \text{for either } j = 1 \quad \text{or} \quad j = 2 \qquad [2.6]$$

where u_j is a nonnegative scalar and $g_j(x)$ is the binding constraint (which can be either the first or the second one).

In order to develop a set of first-order conditions that would apply in all cases (i.e., whether the solution is internal or on the boundary of the feasible region), define a nonnegative variable, u_j, for each of the constraints (not only the binding ones). If the jth constraint is not binding [i.e., if $g_j(x^*) > b_j$], let $u_j = 0$. In other words, the condition

$$[b_j - g_j(x^*)]u_j = 0$$

holds for all the constraints $j \in \mathcal{J}$. (It means that either $u_j = 0$, in which case the jth constraint may or may not be binding, or $u_j \geq 0$ and the jth constraint is binding.) Using this convention, conditions [2.6] can be replaced by the condition

$$\frac{dz(x^*)}{dx} = \sum_j u_j \frac{dg_j(x^*)}{dx}$$

where u_j is a nonnegative scalar and the sum includes all the constraints.† Note that this equation generalizes the condition for an unconstrained minimum (see Eq. [2.3]). An unconstrained minimization problem can be looked upon as a constrained problem with an internal minimum point. In such a case, all $u_j = 0$ (since none of the constraints is binding) and the condition above simply states that $dz(x^*)/dx = 0$.

In summary, then, the first-order conditions for a constrained minimum are

$$\frac{dz(x^*)}{dx} = \sum_j u_j \frac{dg_j(x^*)}{dx} \qquad [2.7a]$$

$$u_j \geq 0, \quad u_j[b_j - g_j(x^*)] = 0, \quad g_j(x^*) \geq b_j \qquad \text{for } j = 1, \ldots, J \qquad [2.7b]$$

These conditions mean that the derivative of the function at the minimum and

†A notational convention used throughout this text is to omit the complete summation notation whenever the summation goes over all the indices (or subscripts) in the relevant set. In this expression "$\sum_j$" is equivalent to "$\sum_{j=1}^{J}$" or "$\sum_{\forall j \in \mathcal{J}}$."

the derivatives of the binding constraints at the minimum have the same signs and that the minimum point is (of course) feasible. Equations [2.7] thus depict the general form of the first-order conditions for the minimum of a single-variable function (see Problem 2.5).

The second-order conditions for a constrained minimum are similar to those for the constrained case, but with one condition added. As in the unconstrained case, strict convexity of $z(x)$ in the vicinity of x^* (where Eqs. [2.7] hold) would ensure that x^* is a local minimum; convexity, however, is not quite sufficient to guarantee the uniqueness of that minimum. Consider, for example, the situation depicted in Figure 2.5, where a strictly convex function $z(x)$ is constrained to the following region: either $0 \leq x \leq a$ or $b \leq x \leq c$. This function depicts two local minima at $x = a$ and at $x = b$ even though $z(x)$ is strictly convex everywhere. Again, proving that a local minimum is also a global one is difficult in the presence of other minima. A condition excluding situations such as the one depicted in Figure 2.5 is therefore added to the uniqueness requirements. This condition requires that the constraints define a convex set of feasible points, or in other words, that the feasible region be convex. The convexity of the feasible region means that a line segment connecting any two points of that feasible region must lie entirely within the region. This condition ensures the existence of only one minimum, which must necessarily be the global minimum. It obviously does not hold for the problem illustrated in Figure 2.5, which does not have a unique minimum.

In conclusion, then, the necessary conditions for a minimum are given by Eqs. [2.7]. The sufficient conditions include strict convexity in the vicinity of the minimum point (this guarantees that it is a local minimum) and convexity of both the objective function and the feasible region (this guarantees that there are no other local minima).

This concludes the discussion of single-variable programs. The concepts introduced in this section are expanded in the following section to the multivariable case.

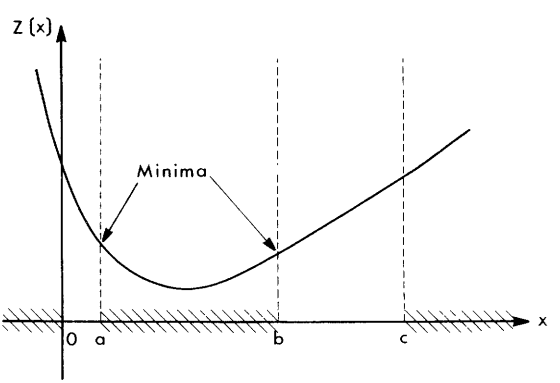


Figure 2.5 Two local minima for a strictly convex function bounded by a constraint set that is not convex.

2.2 MULTIDIMENSIONAL PROGRAMS

In multidimensional problems, the objective function is given by $z(\mathbf{x})$, where $\mathbf{x} = (x_1, \ldots, x_I)$ is a vector of length I (i.e., it is an array of variables $x_1, x_2, \ldots$ up to x_I). The constraints are given by $g_j(\mathbf{x}) \geq b_j$, $\forall\, j \in \mathcal{J}$, where $\mathcal{J}$ is the constraints index set $(1, 2, \ldots, J)$ and b_j is a scalar. Again this discussion of multivariable minimization programs deals separately with unconstrained and constrained problems, including a review of both first- and second-order conditions in each case. As in the preceding section, the regularity conditions on $z(\mathbf{x})$ and $g_j(\mathbf{x})$ are assumed to hold, and a solution therefore always exists.

Unconstrained Minimization Programs

If the function to be minimized is unconstrained, the first-order condition for a minimum at $\mathbf{x} = \mathbf{x}^*$ is that the gradient of $z(\mathbf{x})$ vanish at $\mathbf{x}^*$. The gradient of $z(\mathbf{x})$ with respect to $\mathbf{x}$, $\nabla_{\mathbf{x}} z(\mathbf{x})$, is the vector of partial derivatives, that is,

$$\nabla_{\mathbf{x}} z(\mathbf{x}) = \left(\frac{\partial z(\mathbf{x})}{\partial x_1}, \frac{\partial z(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial z(\mathbf{x})}{\partial x_I} \right) \qquad [2.8]$$

The subscript $\mathbf{x}$ in the gradient notation is usually omitted if the variable with respect to which the partial derivatives are taken is obvious. At every point $\mathbf{x}$, the gradient aims in the direction of the steepest (local) increase in $z(\mathbf{x})$; thus the gradient is the directional derivative of $z(\mathbf{x})$ in the direction of the steepest ascent. As mentioned above, the first-order condition for a minimum is

$$\nabla z(\mathbf{x}^*) = \mathbf{0} \qquad [2.9a]$$

meaning that each component of the gradient (see Eq. [2.8]) has to be equal to zero. In other words,

$$\frac{\partial z(\mathbf{x})}{\partial x_i} = 0 \qquad \text{for } i = 1, 2, \ldots, I \qquad [2.9b]$$

This condition is entirely analogous to the condition $dz(x^*)/dx = 0$ used in the single-variable case. It is only a necessary condition for a minimum; that is, it establishes the fact that $z(\mathbf{x})$ has a stationary point at $\mathbf{x}^*$.

As an example of the application of the first-order conditions, consider the function $z(x_1, x_2)$ where

$$z(x_1, x_2) = x_1^2 + 2x_2^2 + 2x_1x_2 - 2x_1 - 4x_2 \qquad [2.10]$$

The gradient of this function at every point (x_2, x_2) is given by

$$\nabla z(x_1, x_2) = [(2x_1 + 2x_2 - 2), (4x_2 + 2x_1 - 4)] \qquad [2.11]$$

The stationary point is where $\nabla z(x_1, x_2) = 0$, which is the value of $\mathbf{x}$ that solves the equations

$$2x_1 + 2x_2 - 2 = 0 \qquad [2.12a]$$

$$2x_1 + 4x_2 - 4 = 0 \qquad [2.12b]$$

The solution of these equations is $x_1^* = 0$ and $x_2^* = 1$. Thus $\mathbf{x}^* = (0, 1)$ is a stationary point of $z(x_1, x_2)$ in Eq. [2.10].

To show that a stationary point, $\mathbf{x}^*$, is a local minimum, it is sufficient to demonstrate that $z(\mathbf{x})$ is strictly convex in the vicinity of $\mathbf{x} = \mathbf{x}^*$, as was the case with the single-dimensional function. The conditions for strict convexity in the multidimensional case parallel those given in the preceding section for single-variable functions (see Eqs. [2.4]). If a line segment lies entirely above the function or if it is underestimated by a linear approximation, the function is strictly convex. Thus the strict convexity condition is

$$z[\theta\mathbf{x}' + (1 - \theta)\mathbf{x}''] < \theta z(\mathbf{x}') + (1 - \theta)z(\mathbf{x}'') \qquad [2.13a]$$

where $\mathbf{x}'$ and $\mathbf{x}''$ are two different values of $\mathbf{x}$, and θ is a scalar between zero and 1, or†

$$z(\mathbf{x}') + \nabla z(\mathbf{x}') \cdot (\mathbf{x}'' - \mathbf{x}')^T < z(\mathbf{x}'') \qquad [2.13b]$$

where the superscript T denotes the vector (or matrix) transposition operation. Alternatively, a function $z(\mathbf{x})$ can be shown to be strictly convex if some conditions regarding the second derivatives of $z(\mathbf{x})$ hold. To express these conditions mathematically, the second derivatives are usually arranged in a (symmetric) matrix form. [This matrix is known as the *Hessian* of $z(\mathbf{x})$]. The Hessian, denoted by $\nabla^2 z(\mathbf{x})$, is given by

$$\nabla^2 z(\mathbf{x}) \equiv \begin{bmatrix} \dfrac{\partial^2 z(\mathbf{x})}{\partial x_1^2} & \dfrac{\partial^2 z(\mathbf{x})}{\partial x_1\, \partial x_2} & \cdots & \dfrac{\partial^2 z(\mathbf{x})}{\partial x_1\, \partial x_I} \\ \dfrac{\partial^2 z(\mathbf{x})}{\partial x_2\, \partial x_1} & \dfrac{\partial^2 z(\mathbf{x})}{\partial x_2^2} & & \\ \vdots & & & \vdots \\ \dfrac{\partial^2 z(\mathbf{x})}{\partial x_I\, \partial x_1} & & \cdots & \dfrac{\partial^2 z(\mathbf{x})}{\partial x_I^2} \end{bmatrix}$$

The mathematical condition which ensures that $z(\mathbf{x})$ is strictly convex is that the Hessian, $\nabla^2 z(\mathbf{x})$, is positive definite.‡ If the Hessian is positive definite at $\mathbf{x}^*$, this point is a local minimum of $z(\mathbf{x})$. As in the single-dimensional case, it is required that a function be convex everywhere in order to ensure that it has but one local minimum.§

†This condition can be written in expanded notation as follows:

$$z(x_1', \ldots, x_I') + \sum_i \frac{\partial z(x_1', \ldots, x_I')}{\partial x_i} (x_i'' - x_i') < z(x_1'', \ldots, x_I'')$$

‡A matrix, $\mathbf{H}$, is positive definite if the product $\mathbf{h} \cdot \mathbf{H} \cdot \mathbf{h}^T$ is positive for any nonzero vector, $\mathbf{h}$. A symmetric matrix (such as the Hessian) can also be shown to be positive definite if and only if its eigenvalues are positive or if its leading principal minor determinants are positive.

§Convexity is associated with a positive-semidefinite Hessian. A matrix $\mathbf{H}$ is positive semidefinite if and only if $\mathbf{h} \cdot \mathbf{H} \cdot \mathbf{h}^T \geq 0$ for any nonzero vector $\mathbf{h}$. Conditions [2.13a and 2.13b] indicate convexity if the "less than" sign is replaced by a "less than or equal to" sign.

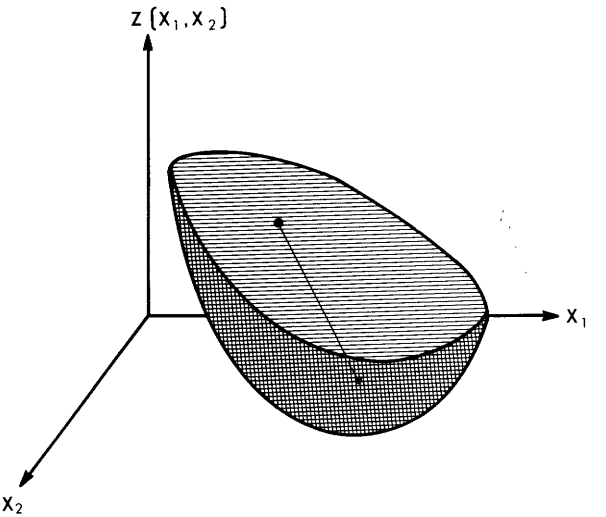


Figure 2.6 Strictly convex function in two dimensions.

It is important to note that the aforementioned conditions for a matrix to be positive definite means that any diagonal matrix (i.e., a matrix in which only the elements along the diagonal are nonzero) with positive elements is positive definite. Many of the objective functions discussed in this book have diagonal Hessians and it is therefore only necessary to check the signs of the diagonal elements to establish that such a Hessian is positive-definite, with the implication that the objective function is strictly convex.

Figure 2.6 shows the shape of a strictly convex function in two dimensions. The figure also shows a line segment connecting two points on this function. As required by the strict convexity condition, this line lies entirely above the function.

It should be noted here that if a line segment connecting any two points of $z(\mathbf{x})$ never lies above the function, $z(\mathbf{x})$ is concave. Concavity (which is a sufficient condition for a local maximum to be unique) can also be identified by a linear approximation of $z(\mathbf{x})$ overestimating (or rather, not underestimating) the function, or by an appropriate condition on the second derivatives.† Consequently, the negative of a concave function is a convex function, and vice versa. Note also that if a (strictly) convex function is multiplied by a positive constant, the product is still a (strictly) convex function, and that the sum of (strictly) convex functions is also a (strictly) convex function (see Problem 2.9). These properties are used later in the text to demonstrate convexity.

To see how the second-order conditions are applied, consider the example of minimizing $z(x_1, x_2)$ in Eq. [2.10]. The stationary point, $x^* = (0, 1)$, is a

†The Hessian of a concave function is negative semidefinite and the Hessian of a strictly concave function is negative definite.

local minimum if the function is strictly convex at that point. If the function is also convex everywhere, then (0, 1) is a global minimum.

The function $z(x_1, x_2)$ in Eq. [2.10] turns out to be strictly convex everywhere. To see this, condition [2.13b] can be used to check if

$$z(x_1, x_2) + \frac{\partial z(x_1, x_2)}{\partial x_1}(y_1 - x_1) + \frac{\partial z(x_1, x_2)}{\partial x_2}(y_2 - x_2) < z(y_1, y_2)$$

for any two points (x_1, x_2) and (y_1, y_2). Substitution of the appropriate functional forms (see Eqs. [2.10] and [2.11]) leads to the inequality

$$[(y_1 - x_1) + (y_2 - x_2)]^2 + (y_2 - x_2)^2 > 0$$

This inequality holds for any two distinct points (x_1, x_2) and (y_1, y_2), meaning that the strict convexity condition [2.13b] is satisfied by $z(x_1, x_2)$ in Eq. [2.10]. This function is, therefore, strictly convex everywhere.† Consequently, it can be concluded that $\mathbf{x}^* = (0, 1)$ is a local as well as a global minimum of $z(x_1, x_2)$. The minimum value of $z(x_1, x_2)$ is $z(0, 1) = -2$.

Constrained Minimization Programs

As in the single-variable case, the focus of the review in this section is on the first- and second-order conditions for a minimum. As in that case, the minimum of a constrained multidimensional program can occur on the boundary of the feasible region, where the condition $\nabla z(\mathbf{x}^*) = \mathbf{0}$ may not hold.

Consider the two-dimensional program depicted in Figure 2.7a, where the feasible region is defined by six constraints and the objective function, $z(\mathbf{x})$, is illustrated by the contour lines. The problem is

$$\min z(x_1, x_2)$$

subject to

$$g_j(x_1, x_2) \geq b_j; \qquad j = 1, 2, \ldots, 6$$

It is apparent from the figure that the solution of this problem, $\mathbf{x}^* = (x_1^*, x_2^*)$, lies at the intersection of constraints 2 and 3. These constraints are therefore binding at the minimum, or, in other words, $g_2(x_1^*, x_2^*) = b_2$ and $g_3(x_1^*, x_2^*) = b_3$.

The direction perpendicular to any curve of the form $g(x_1, x_2) =$ constant, at some point, $\mathbf{x}' = (x_1', x_2')$, is given by the gradient of the curve at

†The strict convexity can also be checked by using the condition on the Hessian, which is given by

$$\mathbf{H} = \begin{pmatrix} 2 & 2 \\ 2 & 4 \end{pmatrix}$$

This matrix is positive definite (its first leading minor equals 2 and its determinant, which is its second leading minor equals $2 \cdot 4 - 2 \cdot 2 = 4$). Both leading minors are positive and $\mathbf{H}$ is, therefore, positive definite, meaning that $z(x_1, x_2)$ is strictly convex everywhere.

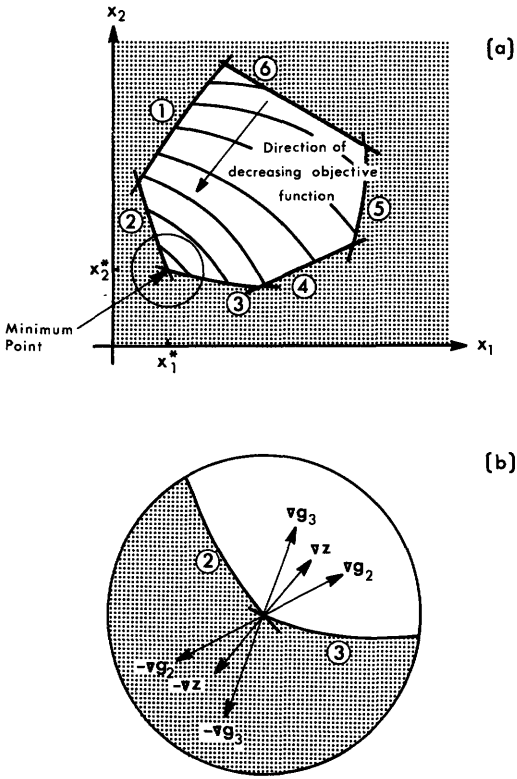


Figure 2.7 Kuhn–Tucker conditions: (a) contours of the objective function and the feasible region; (b) the gradient of the objective function lies within the cone generated by the gradients of the binding constraints.

that point, that is, by the vector

$$\nabla g(\mathbf{x}') = \left(\frac{\partial g(\mathbf{x}')}{\partial x_1}, \frac{\partial g(\mathbf{x}')}{\partial x_2}\right)$$

The enlarged portion shown in Figure 2.7b depicts the point (x_1^*, x_2^*) and the gradients of the two binding constraint functions at this point, $\nabla g_2(\mathbf{x}^*)$ and $\nabla g_3(\mathbf{x}^*)$. The definition of the first-order conditions for constrained problems rests on the observation that the gradient of the objective function at this point, $\nabla z(\mathbf{x}^*)$, must lie between the gradients of the binding constraints. The gradient vector, $\nabla z(\mathbf{x}^*)$, and its opposite vector, $-\nabla z(\mathbf{x}^*)$, as well as $-\nabla g_2(\mathbf{x}^*)$ and $-\nabla g_3(\mathbf{x}^*)$, are all shown in Figure 2.7b, where the aforementioned condition can be intuitively verified. If, for example, $-\nabla z(\mathbf{x}^*)$ were to point below $-\nabla g_3(\mathbf{x}^*)$, $z(\mathbf{x})$ could be further decreased by sliding to the right, along $g_3(\mathbf{x})$ (see Figure 2.7b). Similarly, if $-\nabla z(\mathbf{x}^*)$ were to point above $-g_2(\mathbf{x}^*)$, $z(\mathbf{x})$ could surely be decreased further by sliding upward along $g_2(\mathbf{x}^*)$. Only when $-\nabla z(x^*)$ is between $-\nabla g_2(\mathbf{x}^*)$ and $-\nabla g_3(\mathbf{x}^*)$ can the objective function not be decreased further. This condition is a generalization of the "same signs" condition which was applicable in the single-variable case (see Figure 2.4 and the related argument).

With multidimensional functions, the constraints define some surfaces and the gradient of each of these constraints at a given point is a vector that is perpendicular (normal) to the surface at that point. The observation on which the first-order conditions are based for this case in that the gradients of the binding constraints create an imaginary "cone" within which the gradient of the objective function must lie.

Thus the first-order conditions can be written as a specification of the direction of the gradient of $z(\mathbf{x})$ at $\mathbf{x}^*$ in relation to the directions of the binding constraints. In the multidimensional case, the requirement is that $\nabla z(\mathbf{x}^*)$ be expressed as a linear combination (with nonnegative coefficients) of the gradients of the binding constraints. For the example in Figure 2.7, this condition is

$$\nabla z(\mathbf{x}^*) = u_2 \nabla g_2(\mathbf{x}^*) + u_3 \nabla g_3(\mathbf{x}^*)$$

where u_2 and u_3 are nonnegative scalars associated with the second and third constraints, respectively. The same condition can be written in expanded notation as

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} = u_2 \frac{\partial g_2(\mathbf{x}^*)}{\partial x_i} + u_3 \frac{\partial g_3(\mathbf{x}^*)}{\partial x_i} \qquad \text{for } i = 1, 2$$

In order to generalize this condition to problems of the type

$$\min z(\mathbf{x})$$

subject to

$$g_j(\mathbf{x}) \geq b_j \qquad \forall\, j \in \mathcal{J}$$

an auxiliary variable, u_j, can be defined for each constraint. In a fashion analogous to the single-dimensional case, let u_j be nonnegative if the jth constraint is binding at $\mathbf{x}^*$, and let $u_j = 0$ for any nonbinding constraint.

The first-order conditions can now be written as follows:†

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} = \sum_j u_j \frac{\partial g_j(\mathbf{x}^*)}{\partial x_i} \qquad \text{for } i = 1, 2, \ldots, I \qquad [2.14a]$$

and

$$u_j \geq 0, \quad u_j[b_j - g_j(\mathbf{x}^*)] = 0, \quad g_j(\mathbf{x}^*) \geq b_j \qquad \forall\, j \in \mathcal{J} \qquad [2.14b]$$

Conditions [2.14] are known as the *Kuhn–Tucker* conditions, named after the two mathematicians who first proved that these are the conditions necessary

†Eq. [2.14a] can be written in vector notation as

$$\nabla z(\mathbf{x}^*) = \sum_j u_j \nabla g_j(\mathbf{x}^*)$$

for a constrained minimum. The auxiliary variables u_j are known as *dual variables* as well as *Lagrange multipliers* (since Eq. [2.14a] is identical to the Lagrangian condition in classical optimization theory, as shown in Section 2.3). The condition $u_j[b_j - g_j(\mathbf{x}^*)] = 0, \forall\, j \in \mathcal{J}$, is known as the *complementary slackness* condition.

The Kuhn–Tucker conditions include the observation (Eq. [2.14a]) that the gradient of the objective function, at the minimum, can be expressed as a linear combination, with nonnegative coefficient, of the gradients of the binding constraints. Equations [2.14b] include the condition that these coefficients (which are the dual variables) are, in fact, nonnegative. It also includes the complementary slackness condition, which states that the values of the dual variable associated with each nonbinding constraint is zero. These three conditions mean that if none of the constraints is binding (or if the program is unconstrained), the first-order condition is simply $\nabla z(\mathbf{x}^*) = 0$. The last set of conditions in Eq. [2.14b] ensures that the optimal solution is feasible.

As it turns out, there are some cases, which are seldom encountered in practice, in which the Kuhn–Tucker conditions do not hold at the minimum. If the set of constraints satisfies certain *constraint qualifications*, however, these situations will not arise. For the purposes of this text, it is sufficient to require that the constraint set define a convex feasible region. This condition is explained below in conjunction with the second-order conditions for a constrained minimum of a multidimensional function. The constraint qualification is always satisfied for a convex feasible region.

The values of the dual variables at the solution point provide valuable information about the sensitivity of the value of $z(\mathbf{x})$ at its minimum [i.e., $z(\mathbf{x}^*)$] to the constraints. If the right-hand side of the jth constraint, $g_j(\mathbf{x}) \geq b_j$, is relaxed by some small amount, Δb_j, the minimum value of $z(\mathbf{x})$ will decrease by $u_j\, \Delta b_j$. Thus u_j is a measure of the sensitivity of the minimum value of $z(\mathbf{x})$ to the restriction imposed by the jth constraint. If a constraint is not binding, one would expect that its relaxation would not affect $z(\mathbf{x}^*)$ and indeed, for these constraints $u_j = 0$.

The determination of the first-order conditions of a constrained minimum can be exemplified by using $z(x_1, x_2)$ in Eq. [2.10]. At this point, however, assume that the problem is to minimize the function

$$z(x_1, x_2) = x_1^2 + 2x_2^2 + 2x_1x_2 - 2x_1 - 4x_2 \qquad [2.15a]$$

subject to

$$x_1 + x_2 \geq 2 \qquad [2.15b]$$

Before this problem is attempted, the known unconstrained solution should be tested for feasibility since if it is feasible, it will also be optimal. (Why?) As calculated before, $\mathbf{x}^* = (0, 1)$; this point, unfortunately, is not feasible since constraint [2.15b] is not satisfied.

The Kuhn–Tucker conditions for this problem are as follows:†

$$2x_1^* + 2x_2^* - 2 = u \quad [2.16a]$$

$$4x_2^* + 2x_1^* - 4 = u \quad [2.16b]$$

$$u(2 - x_1^* - x_2^*) = 0 \quad [2.16c]$$

$$x_1^* + x_2^* \geq 2 \quad [2.16d]$$

$$u \geq 0 \quad [2.16e]$$

To solve these equations, note that if $u = 0$, Eqs. [2.16a] and [2.16b] reduce to [2.12a] and [2.12b]. The solution of these equations is $\mathbf{x} = (0, 1)$, a solution that does not satisfy Eq. [2.16d]. Consequently, $u > 0$, meaning that the solution satisfies the three equations

$$2x_1^* + 2x_2^* - 2 - u = 0 \quad [2.17a]$$

$$2x_1^* + 4x_2^* - 4 - u = 0 \quad [2.17b]$$

$$x_1^* + x_2^* - 2 = 0 \quad [2.17c]$$

The solution of this system of equations is

$$x_1^* = 1$$

$$x_2^* = 1$$

$$u^* = 2$$

The value of the constrained minimum is $z(1, 1) = -1$, which is, of course, higher than the value of the unconstrained minimum. (Check this.)

The second-order conditions for constrained multidimensional minimization programs include an additional requirement (in comparison with unconstrained problems), as in the single-dimensional case. For a program to have a unique minimum, the constraint set itself has to define a convex feasible region. Figure 2.8a illustrates two local minima that may occur under a nonconvex feasible region. Figures 2.8b and c illustrate convex feasible regions, where a line segment connecting any two points of each region lies entirely within the region. Hence, to ensure that $\mathbf{x}^*$ is a global minimum, convexity of the feasible region should be required in addition to requiring the convexity of $z(\mathbf{x})$ for all feasible $\mathbf{x}$ and strict convexity of $z(\mathbf{x})$ at $\mathbf{x}^*$.

As in the single-dimensional case, convexity is a relatively strong condition; it is sufficient to require only that a function be ditonic in order to ensure the uniqueness of a minimum. It is, however, easier to prove convexity

†The first two conditions correspond to Eq. [2.14a], stating that equality for each of the two components of the gradient vector. To relate this to Eq. [2.14a], note that the constraint in this problem is expressed in terms of the function $g(x_1, x_2) = x_1 + x_2$, for which $\partial g(\mathbf{x})/\partial x_1 = 1$ and $\partial g(\mathbf{x})/\partial x_2 = 1$. The last three equations above correspond to Eq. [2.14b]. Equation [2.16c] states the complementary slackness condition, Eq. [2.16d] requires the feasibility of the solution, and Eq. [2.16e] states the nonnegativity of the dual variable.

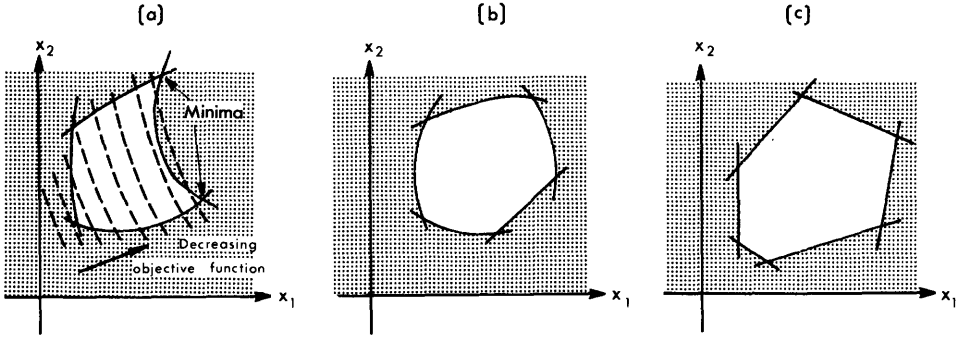


Figure 2.8 Shape of the feasible region: (a) nonconvex feasible region—two local minima are illustrated; (b) convex feasible region; (c) convex feasible region with linear constraints.

and this should always be tried first. Furthermore, in all cases dealt with in this text, the objective function is twice continuously differentiable, meaning that the second derivatives (Hessian) condition can be used to determine convexity. Also, this book deals exclusively with linear constraints and convex feasible regions.

2.3 SOME SPECIAL PROGRAMS

This section deals with several cases of multidimensional constrained minimization programs that are of special interest in the study of equilibrium assignment. Included in this discussion are programs with nonnegativity constraints, programs with equality constraints, programs with both nonnegativity and equality constraints, and linear programs. This section also suggests an alternative approach to the statement of the first-order conditions of any minimization program. This statement is based on the concept of Lagrangians, which is explained in relation to the programs discussed here.

Nonnegativity Constraints

The first-order conditions for the case in which the feasible region includes all nonnegative values of $\mathbf{x}$ can be posed without reference to the dual variables. In the one-dimensional case, the program "min $z(x)$ subject to $x \geq 0$" can result in the two situations shown in Figure 2.9. In Figure 2.9a the solution is at a point where $dz(x^*)/dx = 0$ (and $x^* \geq 0$), while in Figure 2.9b, $x^* = 0$, since the nonnegativity constraint is binding [and $dz(x^*)/dx \geq 0$]. The first-order condition for this problem can be written in a form encompassing both these possible situations, as follows:

$$x^* \frac{dz(x^*)}{dx} = 0 \qquad \text{and} \qquad \frac{dz(x^*)}{dx} \geq 0$$

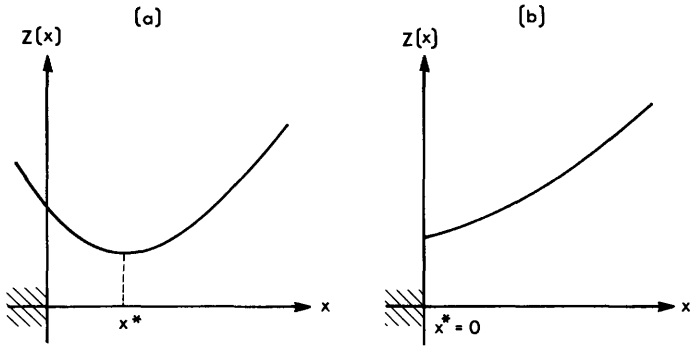


Figure 2.9 First-order conditions for a program with nonnegativity constraints: (a) internal minimum with $dz(x^*)/dx = 0$ and $x^* \geq 0$; (b) constrained minimum with $x^* = 0$ and $dz(x^*)/dx \geq 0$.

Both conditions hold at the minimum point of $z(x)$ (i.e., at x^*). In addition, the constraint $x^* \geq 0$ has to hold as well.

Similarly, in the multidimensional case, the solution of the program

$$\min z(\mathbf{x}) \qquad [2.18a]$$

subject to

$$x_i \geq 0; \qquad i = 1, 2, \ldots, I \qquad [2.18b]$$

can occur either for a positive $\mathbf{x}$ [in which case $\nabla z(\mathbf{x}^*) = \mathbf{0}$] or it can be on the boundary of the feasible region, where some $x_i^* = 0$. Accordingly, the first-order conditions for this problem can be stated as

$$x_i^* \frac{\partial z(\mathbf{x}^*)}{\partial x_i} = 0 \quad \text{and} \quad \frac{\partial z(\mathbf{x}^*)}{\partial x_i} \geq 0 \qquad \text{for } i = 1, 2, \ldots, I \qquad [2.19]$$

Obviously, the condition $x_i^* \geq 0$ has to hold as well for $i = 1, 2, \ldots, I$.

Linear Equality Constraints

One of the most widely used programming problem formulations is the minimization of a convex function subject to a set of equality constraints, that is,

$$\min z(\mathbf{x}) \qquad [2.20a]$$

subject to

$$\sum_i h_{ij} x_i = b_j; \qquad j = 1, 2, \ldots, J \qquad [2.20b]$$

where h_{ij} is a constant.†

†A set of linear equality constraints will always satisfy the aforementioned constraint qualification condition. Furthermore, such a constraint set will define a convex feasible region.

The Kuhn–Tucker conditions for a stationary point of this program are as follows (see Eqs. [2.14]):

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} = \sum_j u_j h_{ij} \qquad \text{for } i = 1, \ldots, I \qquad [2.21a]$$

$$\sum_i h_{ij} x_i^* = b_j \qquad \text{for } j = 1, \ldots, J \qquad [2.21b]$$

At the solution point all the constraints are binding. The complementary slackness conditions are therefore automatically satisfied and the dual variables can have any sign (unlike the case of "greater than or equal to" constraints, in which the dual variables are restricted to be nonnegative).

These first-order conditions can also be derived by the method of Lagrange multipliers. This method involves the specifications of an auxiliary function known as the *Lagrangian.* It includes the original variables $\mathbf{x} = (x_1, \ldots, x_I)$ and the dual variables $\mathbf{u} = (u_1, \ldots, u_J)$, which are also known as Lagrange multipliers. The Lagrangian, $L(\mathbf{x}, \mathbf{u})$, for program [2.20] is given by

$$L(\mathbf{x}, \mathbf{u}) = z(\mathbf{x}) + \sum_j u_j \left[b_j - \sum_i h_{ij} x_i \right] \qquad [2.22]$$

The usefulness of this formulation is that the stationary point of the Lagrangian coincides with the minimum of the constrained optimization [2.20]. Since the Lagrangian is unconstrained, its stationary point can be found by solving for the root of the gradient of the Lagrangian, $\nabla_{\mathbf{x}, \mathbf{u}} L(\mathbf{x}, \mathbf{u})$. Note that the gradient is taken with respect to both types of variables, $\mathbf{x}$ and $\mathbf{u}$. This gradient can be naturally broken into its two types of components, and thus the stationary point of Eq. [2.22] is where

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0} \qquad [2.23a]$$

and

$$\nabla_{\mathbf{u}} L(\mathbf{x}^*, \mathbf{u}^*) = \mathbf{0} \qquad [2.23b]$$

Condition [2.23a] means that

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} - \sum_j u_j^* h_{ij} = 0 \qquad \text{for } i = 1, \ldots, I \qquad [2.24a]$$

and [2.23b] means that

$$b_j - h_j(\mathbf{x}^*) = 0 \qquad \text{for } j = 1, \ldots, J \qquad [2.24b]$$

Thus conditions [2.24] are identical to the Kuhn–Tucker conditions [2.21], with the Lagrangian multipliers $(\ldots, u_j, \ldots)$ playing the same role that the dual variables played in Eqs. [2.21]†. Note that at any feasible point (of the

†The asterisk added to u_j in Eq. [2.24] emphasizes that the u_j's are interpreted here as variables of the Lagrangian. The asterisk denotes the values of these variables at the optimal solution.

original program)

$$L(\mathbf{x}, \mathbf{u}) = z(\mathbf{x})$$

since the added term, $\sum_j u_j[b_j - \sum_i h_{ij} x_i]$, equals zero (each component in the sum is zero). In particular, note that the values of the Lagrangian and the objective function are the same at the minimum point, $\mathbf{x}^*$.

Nonnegativity and Linear Equality Constraints

Many of the problems dealt with in this book include minimization problems with both linear equality constraints and nonnegativity constraints. The general form of these problems is

$$\min z(\mathbf{x}) \qquad [2.25a]$$

subject to

$$\sum_i h_{ij} x_i = b_j; \qquad j = 1, \ldots, J \qquad [2.25b]$$

and

$$x_i \geq 0; \qquad i = 1, \ldots, I \qquad [2.25c]$$

To find the first-order conditions for a minimum for such a problem, the Lagrangian with respect to the equality constraints should be formed first as

$$L(\mathbf{x}, \mathbf{u}) = z(\mathbf{x}) + \sum_j u_j \left[b_j - \sum_i h_{ij} x_i \right] \qquad [2.26a]$$

Then the stationary point of this Lagrangian has to be determined, subject to the constraint

$$x_i \geq 0; \qquad i = 1, \ldots, I \qquad [2.26b]$$

Unlike the case discussed previously, this problem includes nonnegativity constraints. Consequently, the stationary point of program [2.26] has to be determined by the method used to define the first-order condition for such programs, shown in Eqs. [2.19]. For program [2.26] these conditions are the following:

$$x_i^* \frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_i} = 0 \quad \text{and} \quad \frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_i} \geq 0 \qquad \forall\, i \qquad [2.27a]$$

$$\frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_j} = 0 \qquad \forall\, j \qquad [2.27b]$$

Also, $x_i^* \geq 0$, $\forall\, i$, as required by Eq. [2.26b]. Note that in comparison with the first-order conditions for problems with only equality constraints, condition [2.27a] replaces [2.23a] (to account for the nonnegativity of all x_i). Equation [2.27b] requires, simply, that the derivatives of $L(\mathbf{x}, \mathbf{u})$ with respect to $\mathbf{u}$ vanish

at the minimum. No other condition is necessary since the values of $\mathbf{u}$ are not constrained to be nonnegative (or anything else). This condition, then, is identical to [2.23b], specifying the original constraint.

The first-order conditions for the programs with linear equality and nonnegativity constraints can be written explicitly as follows:

$$x_i^* \left(\frac{\partial z(\mathbf{x}^*)}{\partial x_i} - \sum_j u_j^* h_{ij} \right) = 0 \qquad \forall\, i \qquad [2.28a]$$

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} - \sum_j u_j h_{ij} \geq 0 \qquad \forall\, i \qquad [2.28b]$$

$$\sum_i h_{ij} x_i^* = b_j \qquad \forall\, j \qquad [2.28c]$$

$$x_i^* \geq 0 \qquad \forall\, i \qquad [2.28d]$$

These conditions are referred to repeatedly throughout the text. The same conditions can be derived also by applying the Kuhn–Tucker conditions [2.14] directly (see Problem 2.11).

More about Lagrangians†

Lagrangians can be used to derive the first-order conditions for general mathematical programs such as

$$\min z(\mathbf{x}) \qquad [2.29a]$$

subject to

$$g_j(\mathbf{x}) \geq b_j \qquad \forall\, j \in \mathcal{J} \qquad [2.29b]$$

This program can include any type of constraints. The Lagrangian for this program is given by‡

$$L(\mathbf{x}, \mathbf{u}) = z(\mathbf{x}) + \sum_j u_j [b_j - g_j(\mathbf{x})] \qquad [2.30]$$

The dual variables in this formulation are restricted to be nonnegative, due to the "greater than or equal to" type of constraints (as in the case discussed in Section 2.2). This distinguishes this formulation from the case of equality constraints in which the dual variables are unrestricted in sign. (See Problem 2.18.)

As it turns out, the stationary point of the Lagrangian of a convex function is not at a minimum or at a maximum of $L(\mathbf{x}, \mathbf{u})$ but rather at a *saddle point* of the Lagrangian. In fact, $L(\mathbf{x}^*, \mathbf{u}^*)$ minimizes $L(\mathbf{x}, \mathbf{u})$ with respect

†This section can be skipped without loss of continuity. The material in here is needed only as a background to the section on estimation of origin–destination matrices in Chapter 13.

‡Note that Eq. [2.22] is a particular case of this Lagrangian.

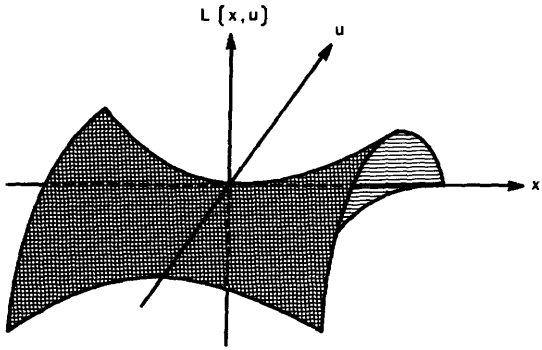


Figure 2.10 Saddle point of a two-argument function. The point (0, 0) maximizes $L(x, u)$ with respect to u and minimizes $L(x, u)$ with respect to x.

to $\mathbf{x}$ and *maximizes* it with respect to $\mathbf{u}$. This condition can be stated as

$$L(\mathbf{x}^*, \mathbf{u}) \leq L(\mathbf{x}^*, \mathbf{u}^*) \leq L(\mathbf{x}, \mathbf{u}^*) \qquad [2.31]$$

Figure 2.10 depicts the shape of saddle point for a function of two variables.

In order to write the first-order conditions of Lagrangian [2.30], note that its minimization is unconstrained with respect to $\mathbf{x}$. The maximization with respect to $\mathbf{u}$, however, is subject to the nonnegativity constraints. The saddle point of $L(\mathbf{x}, \mathbf{u})$ satisfies, then, the following set of first-order conditions:

$$\frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial x_i} = 0 \qquad \forall\, i \qquad [2.32a]$$

$$u_j \frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_j} = 0 \quad \text{and} \quad \frac{\partial L(\mathbf{x}^*, \mathbf{u}^*)}{\partial u_j} \leq 0 \qquad \forall\, j \qquad [2.32b]$$

In addition, it is required that $u_j \geq 0$, $\forall\, j$. Condition [2.32a] states simply that the gradient vanishes at the stationary point. Conditions [2.32b] parallel condition [2.19] but for a maximum of a function (see also Eq. [2.27a]). Since $L(\mathbf{x}, \mathbf{u})$ has to be maximized with respect to $\mathbf{u} = (u_1, \ldots, u_j)$, the maximum of $L(\mathbf{x}, \mathbf{u})$ with respect to u_j can occur† either at a point where $\partial L(\mathbf{x}, \mathbf{u})/\partial u_j = 0$ or at a point where $u_j = 0$. In the latter case, it must be true that $\partial L(\mathbf{x}, \mathbf{u})/\partial u_j \leq 0$. This observation gives rise to conditions [2.32b]. Conditions [2.32] can be written explicitly as

$$\frac{\partial z(\mathbf{x}^*)}{\partial x_i} - \sum_j u_j^* \frac{\partial g_j(\mathbf{x}^*)}{\partial x_i} = 0 \qquad \forall\, i \qquad [2.33a]$$

$$b_j - g_j(\mathbf{x}^*) \leq 0 \qquad \forall\, j \qquad [2.33b]$$

$$u_j[b_j - g_j(\mathbf{x}^*)] = 0 \qquad \forall\, j \qquad [2.33c]$$

†Assuming, of course, that the maximum exists.

and, by definition,

$$u_j^* \geq 0 \qquad \forall j \qquad [2.33d]$$

These conditions are identical to the Kuhn–Tucker conditions [2.14].

The Lagrangian approach means that constrained minimization programs can be solved as unconstrained problems of finding the saddle point of the Lagrangian. This point can be found by minimizing the Lagrangian with respect to $\mathbf{x}$ given $\mathbf{u}$, and then maximizing over all values of $\mathbf{u}$. This minimization approach is used in Chapter 13. In other parts of the book, the Lagrangian is used only as an aid in the formulation of first-order conditions.

Note that the functional form of the Lagrangian demonstrates why the dual variables can be interpreted as a measure of the sensitivity of the optimal solution to a constraint relaxation, as argued in Section 2.2. At the solution point,

$$L(\mathbf{x}^*, \mathbf{u}^*) = z(\mathbf{x}^*) + \sum_j u_j[b_j - g_j(\mathbf{x}^*)] \qquad [2.34]$$

At this point $L(\mathbf{x}^*, \mathbf{u}^*) = z(\mathbf{x}^*)$. If, however, the kth constraint is relaxed by a small amount, Δb_k, and b_k in [2.34] is replaced by $b_k - \Delta b_k$, the new minimum value of $L(\mathbf{x}^*, \mathbf{u}^*)$ will approximately equal the old value (before the relaxation) minus $u_k \, \Delta b_k$. Thus a relaxation of the kth constraint by Δb_k improves the optimal value of the objective function by, approximately, $u_k \, \Delta b_k$.

Linear Programs

A special case of mathematical programming is linear programming (LP). In a linear minimization program, both the objective function and the constraints are linear functions of $\mathbf{x}$. A linear program can be written as

$$\min \sum_i c_i x_i \qquad [2.35a]$$

subject to

$$\sum_i h_{ij} x_i \geq b_j \qquad \forall j \qquad [2.35b]$$

where c_i and h_{ij} are constants and the summations go from $i = 1$ to $i = I$.

The widespread use of linear programming problems stems from the relative ease of solving them. Large linear programming problems can be solved very efficiently by using modern computers. The ease of solving linear programs results from the fact that their solutions never lie at an internal point but always at the boundary of the feasible region. Furthermore, if a solution exists, it will always be at a "corner" of the feasible region. Thus, instead of searching for a minimum all over the feasible region, only the corner points of this region have to be checked. This property is intuitively apparent in Figure 2.11a, which illustrates a feasible region and the contour lines of an objective function for a linear program in two variables, x_1 and x_2. The simplex method

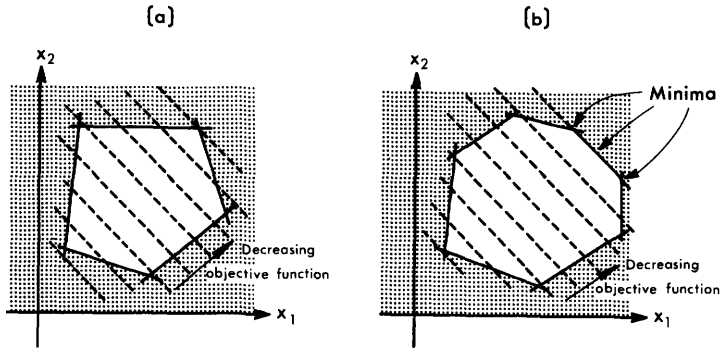


Figure 2.11 Linear programs in two variables: (a) the solution is at a corner of the feasible region; (b) multiple minima.

for solving linear programs exploits this property by progressing through adjacent corners of the feasible region.

In some cases, multiple minima [all with the same value of $z(x^*)$] may exist (since the "strict convexity" condition does not apply to linear programs), as illustrated in Figure 2.11b. Some of these minima, however, will always be at the intersection of several constraints (in other words, at the corners of the feasible region). The minimum of $z(\mathbf{x})$ thus, can still be determined even if only the corners of the feasible region are searched.

2.4 SUMMARY

Chapter 2 reviews the necessary and sufficient conditions for a minimum of a constrained minimization program. The necessary (first-order) conditions are formulated in terms of a set of auxiliary variables known as the dual variables. Each dual variable is associated with a constraint and its value at the solution point indicates the sensitivity of the solution to a small relaxation or tightening of that constraint. Consequently, it is zero for nonbinding constraints and nonnegative for the binding constraints. If the program is unconstrained, these first-order conditions (known as the Kuhn–Tucker conditions) reduce to the requirement that the gradient vanishes. These conditions alone, however, cannot be used to identify the minimum of any program because they hold at any stationary point of the objective function and not only at minima.

To make sure that a stationary point, $\mathbf{x}^*$, is the minimum of some function $z(\mathbf{x})$, two requirements have to be fulfilled: (1) $\mathbf{x}^*$ should be a local minimum, and (2) $\mathbf{x}^*$ should be either the lowest of all the local minima or the only one. In order to meet the first requirement, it is sufficient to show that $z(\mathbf{x})$ is strictly convex in the vicinity of $\mathbf{x}^*$. The second requirement is difficult to meet if there exist multiple minima. It is sufficient, however, to show that $z(\mathbf{x})$ is convex in order to ensure that a local minimum is unique (i.e., that no

other local maxima exist). If a program is constrained, it is also required that the constraint set define a convex feasible region in order to ensure uniqueness. Convexity (or strict convexity) is a second-order condition that can be established with the help of several criteria. The criterion used most often in this book applies when the objective function is twice differentiable; in this case strict convexity can be established by determining that the Hessian of the objective function is positive definite. This condition can easily be checked in cases in which the Hessian is diagonal; if all the entries are positive, the Hessian is positive definite. This implies that the objective function is strictly convex. If the diagonal entries are nonnegative, the Hessian is positive semidefinite, meaning that the objective function is convex.

Two special types of mathematical programs are highlighted in Section 2.3. The first one is a program with only equality and nonnegativity constraints. This type of program is used throughout the book and thus its first-order conditions are derived explicitly. The constraint set of such a program is always convex, meaning that only the objective function has to be checked in order to establish the uniqueness of a minimum.

The first-order conditions for this type of programs, as well as for general maximization programs can be derived through the use of the Lagrangian, which is a function of the original variables and the dual variables of the program. The Lagrangian approach is used repeatedly throughout this book.

The second type of program highlighted is a linear program where both the objective function and the constraints are linear. This type of program is particularly easy to solve and, in fact, the solution techniques to many nonlinear programs consist of repeated solutions of a related linear program.

2.5 ANNOTATED REFERENCES

The material covered in this chapter can be found in any standard mathematical programming text. At the elementary level the reader can consult Chapters 14 and 15 of Wagner (1975), or the texts by Simmons (1975), Bradley et al. (1977), or Wismer and Chattergy (1978). A somewhat more advanced treatment is offered by Zangwill (1969) and Luenberger (1973).

2.6 PROBLEMS

2.1. Show that the first derivative of a continuously differentiable function, $z(x)$, is zero at a local minimum.

***2.2.** Show that if a function is continuously differentiable, it is strictly convex if and only if a linear approximation to the function always underestimates it. (*Hint:* Show that if condition [2.4a] holds, then [2.4b] is true, and if [2.4b] holds, then [2.4a] is true.)

*Problems marked with an asterisk require a somewhat higher level of mathematics than the rest.

*2.3. Show that if a function is twice continuously differentiable, it is strictly convex if and only if the second derivative of the function is positive. (*Hint:* Show that if condition [2.4a] holds, then [2.4c] is true, and if [2.4c] holds, then [2.4a] is true.)

2.4. In which of the following cases does the program min $z(\mathbf{x})$ subject to $g_j(\mathbf{x}) \geq b_j$ with $j = 1, \ldots, J$ have a unique minimum? Explain your answers.
- **(a)** $z(x)$ is convex and the constraint set is convex.
- **(b)** $z(x)$ is ditonic and unconstrained.
- **(c)** $z(x)$ is monotonically increasing and unconstrained.
- **(d)** $z(x)$ is concave and the constraint set is convex.
- **(e)** $z(x)$ is piecewise continuous over a convex constraint set.

2.5. Consider the program min $z(x) = (x - 2)^2$ subject to $0 \leq x \leq 1$.
- **(a)** Write this program in standard form.
- **(b)** Show that the first-order conditions [2.7] hold at the point that minimizes $z(x)$ in the feasible region.

2.6. Plot the contours (in the feasible region only) of a two-dimensional function $z(x_1, x_2)$ subject to $0 \leq x_1 \leq a$, $0 \leq x_2 \leq b$ under the assumption that the function is:
- **(a)** Strictly convex.
- **(b)** Convex with multiple minima.
- **(c)** Linear.
- **(d)** Constrained by $x_1 + x_2 = c$ (in addition to the previous constraints).

*2.7. Show that the gradient always points in the direction of the steepest (local) increase in $z(\mathbf{x})$.

2.8. Calculate the gradient of $z(x_1, x_2) = 2(x_1 - 3)^2 + 3(x_2 - 2)^2$ at $(x_1, x_2) = (2, 3)$ and at $(x_1, x_2) = (3, 2)$.

2.9. Show that the sum of convex functions is a convex function.

2.10. The value of the dual variable used in solving Eqs. [2.15] is $u^* = 2$. Relax the constraint by a small amount and solve the program again. Show that u^* measures the sensitivity of $z(\mathbf{x}^*)$ to the binding constraint.

2.11. **(a)** Using the Kuhn–Tucker conditions, show that Eq. [2.19] holds if the only constraints are nonnegativity constraints.
- **(b)** Derive Eqs. [2.28] directly from the Kuhn–Tucker conditions (Eqs. [2.14]).

2.12. Using the Lagrangian of $z(\mathbf{x})$ subject to $g_j(\mathbf{x}) \geq b_j$ for $j = 1, \ldots, J$, demonstrate why a relaxation of the jth constraint by a small amount, Δb_j, would decrease $z(\mathbf{x}^*)$ by $u_j \, \Delta b_j$.

2.13. Using the Kuhn–Tucker conditions, solve the problem

$$\min z(x_1, x_2) = 4(x_1 - 2)^2 + 3(x_2 - 4)^2$$

subject to

$$x_1 + x_2 \geq 5$$
$$x_1 \geq 1$$
$$x_2 \geq 2$$

Show that the solution is unique.

2.14. Solve the program

$$\min z(x_1, x_2) = 5x_1^2 - 3x_1x_2 + 2x_2^2 + 7$$

subject to

$$2x_1 + x_2 = 5$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

using Lagrangians. Plot the feasible region.

2.15. Solve the program

$$\min z(x_1, x_2) = 2x_1 + x_2$$

subject to

$$3x_1 + x_2 \geq 1$$
$$x_1 + 4x_2 \geq 2$$
$$-x + x_2 \geq 1$$

Plot the feasible region and the contours of the objective function.

2.16. Given the function

$$z(x_1, x_2, x_3) = 3x_1^2 + 2x_1x_2 + 6x_2^2 - x_2x_3 + 5x_3^2 - 4x_1 - 2x_2 - 6x_3 + 9$$

(a) Find the minimum of $z(x_1, x_2, x_3)$.
(b) Show that it is a local minimum.
(c) Show that this local minimum is a global minimum.

2.17. Given that the dual variables indicate the improvement in the objective function value, associated with the relaxation of any constraint, argue why $u_j \geq 0, \forall j$, for a minimization program with standard inequality constraints and why u_j can have any sign for equality constraints.

2.18. Derive the Kuhn–Tucker conditions for the program

$$\min z(\mathbf{x})$$

subject to

$$\sum_i h_{ij}x_i = b_j \qquad \forall j$$

without using Lagrangians. (*Hint:* Express the constraints in standard form.) Show that the dual variables associated with equality constraints are unrestricted in sign.

3

Formulating the Assignment Problem as a Mathematical Program

The traffic assignment or the transportation network equilibrium problem was defined in Chapter 1. As mentioned there, the basic problem is to find the link flows given the origin–destination trip rates, the network, and the link performance functions (see Section 1.3). The solution of the problem is based on the behavioral assumption that each motorist travels on the path that minimizes the travel time† from origin to destination. This choice rule implies that at equilibrium the link-flow pattern is such that the travel times on all used paths connecting any given O–D pair will be equal; the travel time on all of these used paths will also be less than or equal to the travel time on any of the unused paths. At this point, the network is in user equilibrium; no motorist can experience a lower travel time by unilaterally changing routes.

Section 1.3 demonstrated how the user-equilibrium flow pattern can be found for a small network by using graphical methods. Unfortunately, such methods cannot be used to solve for equilibrium over networks with a large number of nodes, links, and O–D pairs. The approach described in this text for solving large problems uses the equivalent minimization method. This approach involves the formulation of a mathematical program, the solution of which is the user-equilibrium flow pattern. This general approach is used often in operations research, in cases in which it is easier to minimize the equivalent program than to solve a set of conditions directly.

For the minimization formulation to be useful, the equivalent mathemat-

†As mentioned in Section 1.3, the term "travel time" can be understood to represent general travel impedance, combining elements of travel costs and other components of travel disutility.

ical program has to have a unique solution, which also satisfies the equilibrium conditions. Furthermore, the program has to be relatively easy to solve. The focus of this chapter is on the formulation of the equivalent minimization program corresponding to the equilibrium traffic assignment problem and on the properties of this program. Solution procedures are then described in Chapters 4 and 5. This chapter is organized as follows: Section 3.1 presents the equivalent minimization formulation, Section 3.2 shows that its solution satisfies the equilibrium conditions, and Section 3.3 proves that this solution is unique. Sections 3.4 and 3.5 explore the nature of the minimization program and the user-equilibrium flow pattern by comparing them to a related flow pattern over the network.

Before the basic formulation is discussed, the following paragraphs present the network notations used in this chapter and throughout this text. The network itself is represented by a directed graph that includes a set of consecutively numbered nodes, $\mathcal{N}$, and a set of consecutively numbered arcs (links), $\mathcal{A}$. In some cases it will be useful to refer to links by their end nodes (i.e., link $m \rightarrow n$ leading from node m to node n), especially in discussing certain algorithms. In these cases the notations will be clarified before the discussion. Let $\mathcal{R}$ denote the set of origin centroids (which are the nodes at which flows are generated) and let $\mathcal{S}$ denote the set of destination centroids (which are the nodes at which flows terminate). The origin node set and the destination node set are not mutually exclusive since nodes can serve as origins and destinations of different trips at the same time (i.e., $\mathcal{R} \cap \mathcal{S} \neq \varnothing$). Each O–D pair r–s is connected by a set of paths (routes) through the network. This set is denoted by $\mathcal{K}_{rs}$ where $r \in \mathcal{R}$ and $s \in \mathcal{S}$.

The origin–destination matrix is denoted by $\mathbf{q}$ with entries q_{rs}. In other words, q_{rs} is the trip rate between origin r and destination s during the period of analysis. Let x_a and t_a represent the flow and travel time, respectively, on link a (where $a \in \mathcal{A}$). Furthermore, $t_a = t_a(x_a)$, where $t_a(\cdot)$ represents the relationship between flow and travel time for link a. In other words, $t_a(x_a)$ is the link performance function (which is also known as the volume–delay curve or the link congestion function). Similarly, let f_k^{rs} and c_k^{rs} represent the flow and travel time, respectively, on path k connecting origin r and destination s ($k \in \mathcal{K}_{rs}$). The travel time on a particular path is the sum of the travel time on the links comprising this path. This relationship can be expressed mathematically as

$$c_k^{rs} = \sum_a t_a \delta_{a,k}^{rs} \qquad \forall\, k \in \mathcal{K}_{rs}, \quad \forall\, r \in \mathcal{R}, \quad \forall\, s \in \mathcal{S} \qquad [3.1a]$$

where $\delta_{a,k}^{rs} = 1$ if link a is a part of path k connecting O–D pair r–s, and $\delta_{a,k}^{rs} = 0$ otherwise. Using the same indicator variable, the link flow can be expressed as a function of the path flow, that is,

$$x_a = \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \qquad \forall\, a \in \mathcal{A} \qquad [3.1b]$$

This equation means that the flow on each arc is the sum of the flows on all

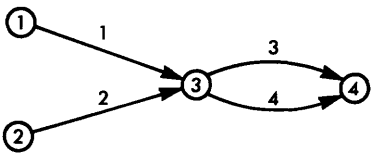


Figure 3.1 Network example with two O–D pairs and four links.

paths going through that arc. Equations [3.1] are known as the *path–arc incidence relationships.*

As an example of the use of the incidence relationships, consider the simple network shown in Figure 3.1. It includes two O–D pairs: 1–4 and 2–4 (node 3 is neither an origin nor a destination point). The link numbers are written on the links. Assume now that the first path from origin node 1 to destination node 4 uses links 1 and 3 and the second one uses links 1 and 4. Similarly, assume that the first path from origin node 2 to destination node 4 uses links 2 and 3 and the second one uses links 2 and 4. For example, $\delta^{14}_{1,1} = 1$ (since link 1 is on path 1 from node 1 to node 4), but $\delta^{24}_{3,2} = 0$ (since link 3 is not on the second path from node 2 to node 4). The incidence relationships for this network means that, for example,

$$\begin{aligned} c_1^{14} &= t_1\delta^{14}_{1,1} + t_2\delta^{14}_{2,1} + t_3\delta^{14}_{3,1} + t_4\delta^{14}_{4,1} \\ &= t_1 + t_3 \end{aligned}$$

Thus Eq. [3.1a] reduces to the anticipated result that the travel time on path 1 between origin 1 and destination 4 is the sum of the travel times on the links comprising this path. Similarly,

$$\begin{aligned} x_3 &= f_1^{14}\delta^{14}_{3,1} + f_2^{14}\delta^{14}_{3,2} + f_1^{24}\delta^{24}_{3,1} + f_2^{24}\delta^{24}_{3,2} \\ &= f_1^{14} + f_1^{24} \end{aligned}$$

Again, Eq. [3.1b] gives the anticipated result here—that the flow on a particular link is the sum of the path flows traversing this link.

Many of the presentations appearing later in the text can be simplified by using vector notation. In most cases this notation is used only to shorten some mathematical expressions [as was the case in Chapter 2, where $z(x_1, \ldots, x_I)$ was denoted by $z(\mathbf{x})$]. Using vector notation, then, let $\mathbf{x} = (\ldots, x_a, \ldots)$, $\mathbf{t} = (\ldots, t_a, \ldots)$, $\mathbf{f}^{rs} = (\ldots, f_k^{rs}, \ldots)$, $\mathbf{f} = (\ldots, \mathbf{f}^{rs}, \ldots)$, $\mathbf{c}^{rs} = (\ldots, c_k^{rs}, \ldots)$, and $\mathbf{c} = (\ldots, \mathbf{c}^{rs}, \ldots)$. Furthermore, let Δ be the link–path incidence matrix with elements $\delta^{rs}_{a,k}$. Typically, this matrix is arranged in block form by O–D pair. In other words, $\Delta = (\ldots, \Delta^{rs}, \ldots)$, where Δ^{rs} is the link–path incidence matrix for O–D pair r–s. The number of rows in the incidence matrix, Δ^{rs}, equals the number of links in the network and the number of columns in this matrix is the number of paths between origin r and destination s. The element in the ath row and kth column of Δ^{rs} is $\delta^{rs}_{a,k}$, in other words $(\Delta^{rs})_{a,k} = \delta^{rs}_{a,k}$. Using vector operations, the incidence relationships can now be written in matrix notation as

$$\mathbf{c} = \mathbf{t} \cdot \Delta \qquad \text{and} \qquad \mathbf{x} = \mathbf{f} \cdot \Delta^T \qquad [3.2]$$

TABLE 3.1 Basic Network Notation

$\mathcal{N}$,	node (index) set
$\mathcal{A}$,	arc (index) set
$\mathcal{R}$,	set of origin nodes; $\mathcal{R} \subseteq \mathcal{N}$
$\mathcal{S}$,	set of destination nodes; $\mathcal{S} \subseteq \mathcal{N}$
$\mathcal{K}_{rs}$,	set of paths connecting O–D pair r–s; $r \in \mathcal{R}$, $s \in \mathcal{S}$
x_a,	flow on arc a; $\mathbf{x} = (\ldots, x_a, \ldots)$
t_a,	travel time on arc a; $\mathbf{t} = (\ldots, t_a, \ldots)$
f_k^{rs},	flow on path k connecting O–D pair r–s; $\mathbf{f}^{rs} = (\ldots, f_k^{rs}, \ldots)$; $\mathbf{f} = (\ldots, \mathbf{f}^{rs}, \ldots)$
c_k^{rs},	travel time on path k connecting O–D pair r–s; $\mathbf{c}^{rs} = (\ldots, c_k^{rs}, \ldots)$; $\mathbf{c} = (\ldots, \mathbf{c}^{rs}, \ldots)$
q_{rs},	trip rate between origin r and destination s; $(\mathbf{q})_{rs} = q_{rs}$
$\delta_{a,k}^{rs}$;	indicator variable: $\delta_{a,k}^{rs} = \begin{cases} 1 & \text{if link } a \text{ is on path } k \text{ between O–D pair } r\text{–}s \\ 0 & \text{otherwise} \end{cases}$
	$(\Delta^{rs})_{a,k} = \delta_{a,k}^{rs}$; $\Delta = (\ldots, \Delta^{rs}, \ldots)$

The incidence matrix for the network example depicted in Figure 3.1 can be written as follows:

$$\begin{array}{c} \\ \text{link} \diagdown \text{path} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} \multicolumn{2}{c}{\text{O–D}} & \multicolumn{2}{c}{\text{O–D}} \\ \multicolumn{2}{c}{\underbrace{}_{\text{1–4}}} & \multicolumn{2}{c}{\underbrace{}_{\text{2–4}}} \\ 1 & 2 & 1 & 2 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

As the reader can check, Eq. [3.2] is identical to Eqs. [3.1] in terms of specifying the relationships between flows and travel times for the network example shown in Figure 3.1.

The network notation introduced here is summarized in Table 3.1. Further notation is introduced as needed.

3.1 THE BASIC TRANSFORMATION

The equilibrium assignment problem is to find the link flows, $\mathbf{x}$, that satisfy the user-equilibrium criterion when all the origin–destination entries, $\mathbf{q}$, have been appropriately assigned. This link–flow pattern can be obtained by solving the following mathematical program:

$$\min z(\mathbf{x}) = \sum_a \int_0^{x_a} t_a(\omega)\, d\omega \qquad [3.3a]$$

subject to

$$\sum_k f_k^{rs} = q_{rs} \qquad \forall\, r, s \qquad [3.3b]$$

$$f_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [3.3c]$$

The definitional constraints

$$x_a = \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} \qquad \forall\, a \qquad [3.3d]$$

are also part of this program.

In this formulation, the objective function is the sum of the integrals of the link performance functions. This function does not have any intuitive economic or behavioral interpretation. It should be viewed strictly as a mathematical construct that is utilized to solve equilibrium problems.

Equation [3.3b] represents a set of flow conservation constraints. These constraints state that the flow on all paths connecting each O–D pair has to equal the O–D trip rate. In other words, all O–D trip rates have to be assigned to the network. The nonnegativity conditions in Eq. [3.3c] are required to ensure that the solution of the program will be physically meaningful.

The objective function of program [3.3], $z(\mathbf{x})$, is formulated in terms of link flows, whereas the flow conservation constraints are formulated in terms of path flows. The network structure enters this formulation through the definitional incidence relationships [3.3d]. These incidence relationships express the link flows in terms of the path flows [i.e., $\mathbf{x} = \mathbf{x}(\mathbf{f})$]. The incidence relationships also mean that the partial derivative of link flow can be defined with respect to a particular path flow.† In other words,

$$\frac{\partial x_a(\mathbf{f})}{\partial f_l^{mn}} = \frac{\partial}{\partial f_l^{mn}} \sum_r \sum_s \sum_k f_k^{rs} \delta_{a,k}^{rs} = \delta_{a,l}^{mn} \qquad [3.4]$$

since $\partial f_k^{rs} / \partial f_l^{mn} = 0$ if r–$s \neq m$–n or $k \neq l$. Equation [3.4] implies that the derivative of the flow on link a with respect to the flow on path l between origin m and destination n equals 1 if the link is a part of that path and zero otherwise. These relationships are used later in the text to investigate the first- and second-order conditions of program [3.3].

It is important to note that this formulation assumes that the travel time on a given link is a function of the flow on that link only and not of the flow on any other link in the network. This somewhat restrictive assumption is discussed in detail and subsequently relaxed in Chapter 8. In addition, the link performance functions are assumed to be positive and increasing. These latter

†The function $x_a(\mathbf{f})$ includes flow summation using the subscripts r, s, and k. To avoid confusion in differentiation, the variable with respect to which the derivative is being taken is subscripted by m, n, and l. Thus f_l^{mn} is the flow on path l between origin m and destination n. Similarly, x_b is used below when the derivative of an expression including a sum over link flows is taken with respect to the flow on a particular link.

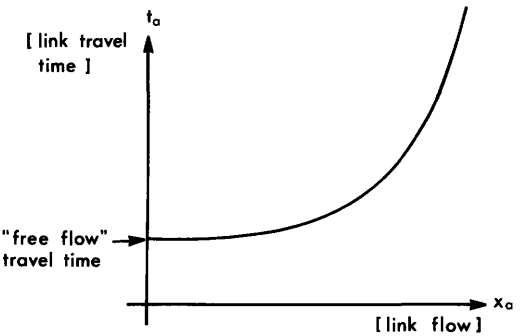


Figure 3.2 Typical link performance function, $t_a(x_a)$.

assumptions are not restrictive in the sense that the congestion effects described by these functions exhibit both characteristics, as mentioned in Section 1.2 (note that these curves are also convex). A typical performance curve is depicted in Figure 3.2 (see also Figure 1.8). The assumptions on the performance curves can be written mathematically as

$$\frac{\partial t_a(x_a)}{\partial x_b} = 0 \qquad \forall\, a \neq b \qquad [3.5a]$$

$$\frac{\partial t_a(x_a)}{\partial x_a} > 0 \qquad \forall\, a \qquad [3.5b]$$

The problem formulation represented by Eqs. [3.3] is known as Beckmann's transformation. It has been evident in the transportation literature since the mid-1950s, but its usefulness became apparent only when solution algorithms for this program were developed in the late 1960s and early 1970s. Some of these algorithms are discussed in Chapter 5.

The following section formally proves that the solution to Beckmann's transformation satisfies the user-equilibrium conditions. This equivalency is first illustrated, however, for a simple situation.

Consider the network depicted in Figure 3.3. This network includes two paths (which are also links), leading from the origin, O, to the destination, D. The volume–delay curves for the two links are given by

$$t_1 = 2 + x_1 \qquad [3.6a]$$

$$t_2 = 1 + 2x_2 \qquad [3.6b]$$

The O–D flow, q, is 5 units of flow, that is,

$$x_1 + x_2 = 5 \qquad [3.6c]$$

The equilibrium condition for this example can be expressed as (see Section 1.3)

$$t_1 \leq t_2 \quad \text{if } x_1 > 0 \qquad \text{and} \qquad t_1 \geq t_2 \quad \text{if } x_2 > 0 \qquad [3.6d]$$

For this example it can be verified by inspection that both paths will be used

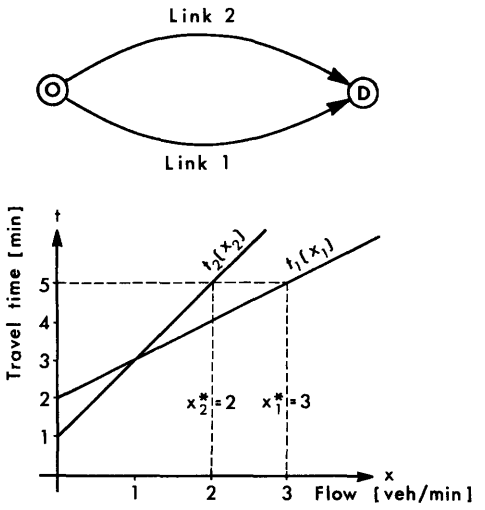


Figure 3.3 Equilibrium example: (a) a two-link network; (b) the performance functions and the equilibrium solution.

at equilibrium and the last equation can therefore be written simply (given that $x_1 > 0$ and $x_2 > 0$) as

$$t_1 = t_2 \qquad [3.6e]$$

The equilibrium problem then, is to solve four equations (the two volume–delay curves [3.6a] and [3.6b], the flow conservation condition [3.6c], and the user-equilibrium condition [3.6e]), in four unknowns: x_1, x_2, t_1, and t_2. The solution to this set of equations is

$$x_1 = 3 \qquad \text{flow units}$$

$$x_2 = 2 \qquad \text{flow units}$$

$$t_1 = t_2 = 5 \qquad \text{time units}$$

When the problem is formulated as a minimization program, the result is the following:

$$\min z(\mathbf{x}) = \int_0^{x_1} (2 + \omega)\, d\omega + \int_0^{x_2} (1 + 2\omega)\, d\omega$$

subject to

$$x_1 + x_2 = 5$$

$$x_1, x_2 \geq 0$$

To set the problem up as a simple one-dimensional unconstrained minimization, $x_2 = 5 - x_1$ can be substituted into the objective function and into the remaining (nonnegativity) constraints to get the problem

$$\min z(x_1) = \int_0^{x_1} (2 + \omega)\, d\omega + \int_0^{5-x_1} (1 + 2\omega)\, d\omega \qquad [3.7a]$$

subject to

$$x_1 \geq 0 \quad \text{and} \quad 5 - x_1 \geq 0 \qquad [3.7b]$$

To solve this program, the constraints can be relaxed and the objective function can be minimized as an unconstrained program. If the solution satisfies the constraints, it is valid for the constrained program as well. Carrying out the integration and collecting similar terms, the objective function becomes

$$z(x_1) = 1.5x_1^2 - 9x_1 + 30$$

This function attains its minimum at $x_1^* = 3$, where $dz(x_1)/dx_1 = 0$. This solution satisfies the two constraints in Eq. [3.7b] and is therefore a minimum of the constrained program [3.7] as well. The original flow conservation constraint guarantees that $x_2^* = 2$ and indeed, the solution of the mathematical program is identical to the solution of the equilibrium equations. This equivalency is demonstrated for the general case in the next section.

3.2 EQUIVALENCY CONDITIONS

To demonstrate the equivalence of the equilibrium assignment problem and program [3.3], it has to be shown that any flow pattern that solves [3.3] also satisfies the equilibrium conditions. This equivalency is demonstrated in this section by proving that the first-order conditions for the minimization program are identical to the equilibrium conditions. From the discussion in Chapter 2, recall that the solution of any mathematical program satisfies its first-order conditions at any local minimum or any stationary point of the program. If the first-order conditions are identical to the equilibrium conditions, the latter hold at any local minimum (or stationary point). Thus, by finding a minimum point of the program, an equilibrium flow pattern is obtained.

To derive the first-order conditions of the Beckmann transformation, observe that it is a minimization problem with linear equality and nonnegativity constraints. A general form of the first-order conditions for such problems was derived in Section 2.3 (see Eqs. [2.25] and the related discussion). Following that section, the Lagrangian of the equivalent minimization problem with respect to the equality constraints [3.3b] can be formulated as†

$$L(\mathbf{f}, \mathbf{u}) = z[\mathbf{x}(\mathbf{f})] + \sum_{rs} u_{rs}\left(q_{rs} - \sum_k f_k^{rs}\right) \qquad [3.8a]$$

where u_{rs} denotes the dual variable associated with the flow conservation constraint for O–D pair r–s in Eq. [3.3b]. The first-order conditions of program [3.3] are equivalent to the first-order conditions of Lagrangian [3.8a],

†The shorthand notation "$\sum_{rs}$" is used throughout the book to abbreviate "$\sum_r \sum_s$."

given that $L(\mathbf{f}, \mathbf{u})$ has to be minimized with respect to nonnegative path flows, that is,

$$f_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [3.8b]$$

The formulation of this Lagrangian is given in terms of path flow, by using the incidence relationships, $x_a = x_a(\mathbf{f})$ in Eq. [3.3d], for every link a. At the stationary point of the Lagrangian, the following conditions have to hold with respect to the path-flow variables:

$$f_k^{rs} \frac{\partial L(\mathbf{f}, \mathbf{u})}{\partial f_k^{rs}} = 0 \quad \text{and} \quad \frac{\partial L(\mathbf{f}, \mathbf{u})}{\partial f_k^{rs}} \geq 0 \qquad \forall\, k, r, s \qquad [3.9a]$$

and the following conditions have to hold with respect to the dual variables:

$$\frac{\partial L(\mathbf{f}, \mathbf{u})}{\partial u_{rs}} = 0 \qquad \forall\, r, s \qquad [3.9b]$$

Also, the nonnegativity constraints have to hold (i.e., $f_k^{rs} \geq 0$, $\forall\, k, r, s$). The asterisks have been omitted from the last expressions (in comparison with Eqs. [2.27]) for clarity of notation. This practice is followed hereafter in this book.

Condition [3.9b] simply states the flow conservation constraints, which obviously, have to hold at equilibrium. The first-order conditions expressed in Eq. [3.9a] can be obtained explicitly by calculating the partial derivatives of $L(\mathbf{f}, \mathbf{u})$ with respect to the flow variables, f_l^{mn}, and substituting the result into [3.9a]. This derivative is given by

$$\frac{\partial}{\partial f_l^{mn}} L(\mathbf{f}, \mathbf{u}) = \frac{\partial}{\partial f_l^{mn}} z[\mathbf{x}(\mathbf{f})] + \frac{\partial}{\partial f_l^{mn}} \sum_{rs} u_{rs} \left(q_{rs} - \sum_k f_k^{rs} \right) \qquad [3.10]$$

The right-hand side of Eq. [3.10] consists of the sum of two types of terms, the first of which is the derivative of the objective function and the second is the derivative of a term involving the constraints. These two types of terms are each calculated separately.

The first term on the right-hand side of Eq. [3.10] is the derivative of the objective function [3.3a] with respect to f_l^{mn}. This derivative can be evaluated by using the chain rule:

$$\frac{\partial z[\mathbf{x}(\mathbf{f})]}{\partial f_l^{mn}} = \sum_{b \in \mathscr{A}} \frac{\partial z(\mathbf{x})}{\partial x_b} \frac{\partial x_b}{\partial f_l^{mn}} \qquad [3.11]$$

Each term in the sum on the right-hand side of this equation is the product of two quantities. The first quantity is $\partial z(\mathbf{x})/\partial x_b$, which can be easily calculated since the travel time on any link is a function of the flow on that link only. Hence

$$\frac{\partial z(\mathbf{x})}{\partial x_b} = \frac{\partial}{\partial x_b} \sum_a \int_0^{x_a} t_a(\omega)\, d\omega = t_b \qquad [3.12a]$$

The second quantity in the product is the partial derivative of a link flow with

respect to the flow on a particular path. As shown in Section 3.1 (see Eq. [3.4]),

$$\frac{\partial x_b}{\partial f_l^{mn}} = \delta_{b,l}^{mn} \qquad [3.12b]$$

Substituting the last two expressions into Eq. [3.11], the derivative of the objective function with respect to the flow on a particular path becomes

$$\frac{\partial z[\mathbf{x}(\mathbf{f})]}{\partial f_l^{mn}} = \sum_b t_b \delta_{b,l}^{mn} = c_l^{mn} \qquad [3.13]$$

In other words, it is the travel time on that particular path.

The second type of term in Eq. [3.10] is even simpler to calculate since

$$\frac{\partial f_k^{rs}}{\partial f_l^{mn}} = \begin{cases} 1 & \text{if } r = m,\ s = n \text{ and } k = l \\ 0 & \text{otherwise} \end{cases}$$

Thus (since q_{rs} is a constant and u_{rs} is not a function of f_l^{mn}) this term becomes

$$\frac{\partial}{\partial f_l^{mn}} \sum_{rs} u_{rs}\left(q_{rs} - \sum_k f_k^{rs}\right) = -u_{mn} \qquad [3.14]$$

Substituting both [3.13] and [3.14] into Eq. [3.10], the partial derivative of the Lagrangian becomes

$$\frac{\partial}{\partial f_l^{mn}} L(\mathbf{f}, \mathbf{u}) = c_l^{mn} - u_{mn} \qquad [3.15]$$

The general first-order conditions (Eqs. [3.9]) for the minimization program in Eqs. [3.3] can now be expressed explicitly as

$$f_k^{rs}(c_k^{rs} - u_{rs}) = 0 \qquad \forall\ k, r, s \qquad [3.16a]$$

$$c_k^{rs} - u_{rs} \geq 0 \qquad \forall\ k, r, s \qquad [3.16b]$$

$$\sum_k f_k^{rs} = q_{rs} \qquad \forall\ r, s \qquad [3.16c]$$

$$f_k^{rs} \geq 0 \qquad \forall\ k, r, s \qquad [3.16d]$$

Conditions [3.16c] and [3.16d] are simply the flow conservation and nonnegativity constraints, respectively. These constraints, naturally, hold at the point that minimizes the objective function. The following discussion is focused on the nature of the first two conditions expressed in Eqs. [3.16a] and [3.16b]. These conditions hold for each path between any O–D pair in the network. For a given path, say path k connecting origin r and destination s, the conditions hold for two possible combinations of path flow and travel time. Either the flow on that path is zero (i.e., $f_k^{rs} = 0$ and Eq. [3.16a] holds), in which case the travel time on this path, c_k^{rs}, must be greater than or equal to the O–D specific Lagrange multiplier, u_{rs} (as required by Eq. [3.16b]); or the

flow on the kth path is positive, in which case $c_k^{rs} = u_{rs}$ and both Eqs. [3.16a] and [3.16b] hold as equalities.† In any event, the Lagrange multiplier of a given O–D pair is less than or equal to the travel time on all paths connecting this pair. Thus u_{rs} equals the minimum path travel time between origin r and destination s.

With this interpretation, it is now clear that Eqs. [3.16], in fact, state the user-equilibrium principle. The paths connecting any O–D pair can be divided into two categories: those carrying flow, on which the travel time equals the minimum O–D travel time; and those not carrying flow, on which the travel time is greater than (or equal to) the minimum O–D travel time. If the flow pattern satisfies these equations, no motorist can be better off by unilaterally changing routes. All other routes have either equal or higher travel times. The user-equilibrium criteria are thus met for every O–D pair.

The equivalence between the UE conditions and the first-order conditions of program [3.3] means that the UE conditions are satisfied at any local minimum or, in fact, at any stationary point of this program. Accordingly, this program is usually referred to as the UE program or the UE equivalent minimization. The next section shows that the UE program has only one stationary point, which is a minimum.

3.3 UNIQUENESS CONDITIONS

In order to show that the UE equivalent minimization program has only one solution, it is sufficient to prove that objective function [3.3a] is strictly convex in the vicinity of $\mathbf{x}^*$ (and convex elsewhere) and that the feasible region (defined by constraints [3.3b] and [3.3c]) is convex. The convexity of the feasible region is assured for linear equality constraints, as mentioned in Section 2.3. The addition of the nonnegativity constraints does not alter this characteristic. The focus of this section, then, is on the properties of the objective function.

The convexity of the objective function is proved here with respect to link flows; path flows are treated later. This section demonstrates that the function

$$z(\mathbf{x}) = \sum_a \int_0^{x_a} t_a(\omega)\, d\omega$$

is convex under the aforementioned assumptions on the link performance functions (namely, that $\partial t_a(\cdot)/\partial x_b = 0$ for $a \neq b$ and $dt_a(x_a)/dx_a > 0$, $\forall\ a$—see Eqs. [3.5]). This is done by proving that the Hessian [the matrix of the second derivatives of $z(\mathbf{x})$ with respect to $\mathbf{x}$] is positive definite, thus ensuring that $z(\mathbf{x})$ is, in fact, strictly convex everywhere.

The Hessian is calculated by using a representative term of the matrix.

†All this holds at the minimum point (the solution of the minimization program); recall that the asterisks are omitted from Eqs. [3.16] for presentation purposes only.

The derivative of $z(\mathbf{x})$ is therefore taken with respect to the flow on the mth and nth links. First,

$$\frac{\partial z(\mathbf{x})}{\partial x_m} = t_m(x_m)$$

as in Eq. [3.12a], and second,

$$\frac{\partial^2 z(\mathbf{x})}{\partial x_m \, \partial x_n} = \frac{\partial t_m(x_m)}{\partial x_n} = \begin{cases} \dfrac{dt_n(x_n)}{dx_n} & \text{for } m = n \\ 0 & \text{otherwise} \end{cases} \qquad [3.17]$$

because of condition [3.5a]. This means that all the off-diagonal elements of the Hessian, $\nabla^2 z(\mathbf{x})$, are zero and all the diagonal elements are given by $dt_a(x_a)/dx_a$. In other words,†

$$\nabla^2 z(\mathbf{x}) = \begin{bmatrix} \dfrac{dt_1(x_1)}{dx_1} & 0 & 0 & \cdots \\ 0 & \dfrac{dt_2(x_2)}{dx_2} & 0 & \cdots \\ 0 & 0 & \ddots & \\ \vdots & \vdots & & \dfrac{dt_A(x_A)}{dx_A} \end{bmatrix} \qquad [3.18]$$

This matrix is positive definite since it is a diagonal matrix with strictly positive entries (all entries are positive because of condition [3.5b]). The objective function is thus strictly convex, and since the feasible region is convex as well, the UE program has a unique minimum.

This result means that there is only one flow pattern that minimizes program [3.3]. As shown in the last section, this minimum is a user-equilibrium flow pattern and consequently, the UE flow pattern can be found by minimizing this program.

The strict convexity of the UE program was established above with respect to the link flows. This program, however, is not convex with respect to path flows and, in fact, the equilibrium conditions themselves are not unique with respect to path flows. This point is demonstrated in the simple network example depicted in Figure 3.4. The network includes two origin–destination pairs connected by two paths each, as shown in the figure. The figure also depicts the link performance functions and the O–D trip rates ($q_{15} = 2$ and $q_{25} = 3$).

The equilibrium link flows for this example are given by $x_1 = 2$, $x_2 = 3$,

†Note that Eq. [3.18] is the Jacobian of the link-travel-time vector, $\mathbf{t}$, with respect to the link flows, $\mathbf{x}$. The Jacobian matrix of a given vector is the matrix of first derivatives of each of the vector components with respect to the arguments of these components.

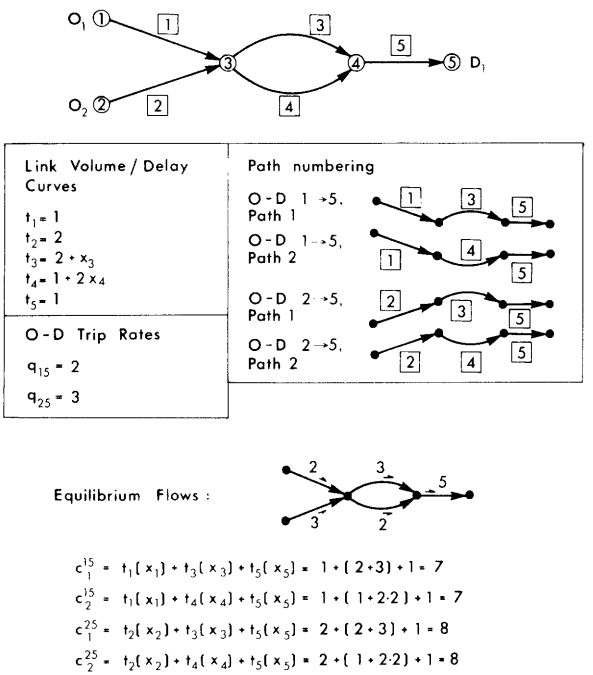


Figure 3.4 Equilibrium flows and path travel times in a network with two O–D pairs and five links.

$x_3 = 3$, $x_4 = 2$, and $x_5 = 5$, as shown in Figure 3.4. These equilibrium flows can be achieved by many combinations of path flows; for example:

$$f_1^{15} = 0, \quad f_2^{15} = 2, \quad f_1^{25} = 3, \quad \text{and} \quad f_2^{25} = 0$$

or

$$f_1^{15} = 2, \quad f_2^{15} = 0, \quad f_1^{25} = 1, \quad \text{and} \quad f_2^{25} = 2$$

Both these path-flow patterns generate the same link-flow pattern shown in Figure 3.4. In fact, any path-flow pattern that satisfies

$$f_1^{15} = 2\alpha, \quad f_2^{15} = 2(1 - \alpha), \quad f_1^{25} = 3 - 2\alpha, \quad \text{and} \quad f_2^{25} = 2\alpha$$

for any value of α such that $0 \leq \alpha \leq 1$ will generate the equilibrium link-flow pattern in this example. Thus there are, in principle, an infinite number of equilibrium path flows.

The convexity of the UE equivalent minimization program with respect to link flows can be established without analyzing the Hessian of $z(\mathbf{x})$, by making use of the properties of convex functions. The objective function of the UE minimization consists of a sum each element of which is an integral of an increasing function. Such an integral is always strictly convex (prove it) and the sum of strictly convex functions is always strictly convex. Thus $z(\mathbf{x})$ is a strictly convex function which has only one minimum. This point is also the

user-equilibrium solution for the network, as shown by the first-order condition.

The next two chapters focus on solution algorithms for convex minimization problems in general and the UE program in particular. Before discussing these algorithms, however, the next two sections of this chapter deal with some related minimization programs.

3.4 THE SYSTEM-OPTIMIZATION FORMULATION

As mentioned in Section 3.1, the UE minimization program is a mathematical construct that lacks an intuitive interpretation. It is merely an efficient method for solving the user equilibrium equations. These equations describe the flow pattern resulting from each motorist's choice of the shortest travel-time route from origin to destination.

This section examines a related minimization program including an objective function that has a straightforward interpretation and is subject to the same constraints as the UE equivalent program. The objective function of this program is the total travel time spent in the network. The flow pattern that solves this program minimizes this objective function while satisfying the flow conservation constraints (i.e., all the O–D trip rates are assigned to the network). This program can be expressed as follows:

$$\min \tilde{z}(\mathbf{x}) = \sum_a x_a t_a(x_a) \qquad [3.19a]$$

subject to

$$\sum_k f_k^{rs} = q_{rs} \qquad \forall\, r, s \qquad [3.19b]$$

$$f_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [3.19c]$$

As in the UE program, the objective function is formulated in terms of link flows while the constraints are formulated in terms of path flows.

Program [3.19] is known as the system-optimization (SO) program. The flow pattern that minimizes this program does not generally represent an equilibrium situation. Except in special cases, it can result only from joint decisions by all motorists to act so as to minimize the total system travel time rather than their *own*. In other words, at the SO flow pattern, drivers may be able to decrease their travel time by unilaterally changing routes. Such a situation is unlikely to sustain itself and consequently the SO flow pattern is not stable and should not be used as a model of actual behavior and equilibrium. (The only cases in which the SO flow pattern can be used to represent equilibrium are those special cases in which the SO solution is identical to the UE solution.)

The significance of the SO formulation and the resulting flow pattern is that the value of the SO objective function may serve as a yardstick by which

different flow patterns can be measured. Indeed, total (systemwide) travel time is a common measure of performance of a network under a given scenario (see Section 1.1). This measure can be computed in a straightforward manner given the equilibrium flows, and it does not require any data in addition to those required for the equilibrium analysis itself. Thus the flow pattern associated with any proposed project can be measured in terms of the total travel time associated with it relative to the minimum possible total travel time. By definition this minimum is obtained by solving the SO program.†

The necessary conditions for a minimum for the SO program are given by the first-order conditions for a stationary point of the following Lagrangian program:

$$\tilde{L}(\mathbf{f}, \tilde{\mathbf{u}}) = \tilde{z}[\mathbf{x}(\mathbf{f})] + \sum_{rs} \tilde{u}_{rs}\left(q_{rs} - \sum_k f_k^{rs}\right) \qquad [3.20a]$$

where $\tilde{L}(\mathbf{f}, \tilde{\mathbf{u}})$ has to be minimized with respect to $\mathbf{f}$, subject to the set of nonnegativity conditions

$$f_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [3.20b]$$

The variable $\tilde{u}_{rs}$ is the Lagrange multiplier (or the dual variable) associated with the flow conservation constraint of O–D pair r–s (Eq. [3.19b]). The first-order conditions for a stationary point of Eqs. [3.20] are (see Eqs. [3.9])

$$f_k^{rs} \frac{\partial \tilde{L}(\mathbf{f}, \tilde{\mathbf{u}})}{\partial f_k^{rs}} = 0 \quad \text{and} \quad \frac{\partial \tilde{L}(\mathbf{f}, \tilde{\mathbf{u}})}{\partial f_k^{rs}} \geq 0 \qquad \forall\, k, r, s \qquad [3.21a]$$

as well as

$$\frac{\partial \tilde{L}(\mathbf{f}, \tilde{\mathbf{u}})}{\partial \tilde{\mathbf{u}}_{rs}} = 0 \qquad \forall\, r, s \qquad [3.21b]$$

and

$$f_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [3.21c]$$

Again, the asterisks denoting the optimal solution in terms of $\mathbf{f}^*$ and $\tilde{\mathbf{u}}^*$ have been omitted from these equations, as in Eqs. [3.9].

As in the case of the equivalent UE program, conditions [3.21b] and [3.21c] simply restate the flow conservation and nonnegativity constraints, respectively, and are therefore not discussed further. Conditions [3.21a] can be expressed explicitly by deriving the partial derivatives of the Lagrangian with respect to the path flows. These derivatives are given by

$$\frac{\partial \tilde{L}(\mathbf{f}, \tilde{\mathbf{u}})}{\partial f_l^{mn}} = \frac{\partial}{\partial f_l^{mn}} \tilde{z}[\mathbf{x}(\mathbf{f})] + \frac{\partial}{\partial f_l^{mn}} \sum_{rs} \tilde{u}_{rs}\left(q_{rs} - \sum_k f_k^{rs}\right) \qquad \forall\, m, n, l \qquad [3.22]$$

The partial derivative of Lagrangian [3.22] consists of two terms, which

†In addition, the SO program is used as a bound in many mathematical programs dealing with network design.

are calculated separately below. The second term is similar to the second term of Eq. [3.10] and therefore (see Eq. [3.14])

$$\frac{\partial}{\partial f_l^{mn}} \sum_{rs} \tilde{u}_{rs}\left(q_{rs} - \sum_k f_k^{rs}\right) = -\tilde{u}_{mn} \qquad \forall\ l, m, n \qquad [3.23]$$

The first term on the right-hand side of Eq. [3.22] includes the partial derivative of the SO objective functions with respect to the flow on path l between origin m and destination n. This derivative is given by

$$\begin{aligned}\frac{\partial}{\partial f_l^{mn}} \tilde{z}[\mathbf{x}(\mathbf{f})] &= \sum_b \frac{\partial \tilde{z}(\mathbf{x})}{\partial x_b} \frac{\partial x_b}{\partial f_l^{mn}} = \sum_b \frac{\partial \tilde{z}(\mathbf{x})}{\partial x_b} \delta_{b,l}^{mn} \\ &= \sum_b \delta_{b,l}^{mn} \frac{\partial}{\partial x_b} \sum_a x_a t_a(x_a) \\ &= \sum_b \delta_{b,l}^{mn}\left[t_b(x_b) + x_b \frac{dt_b(x_b)}{dx_b}\right] \qquad \forall\ l, m, n \qquad [3.24]\end{aligned}$$

This derivative can be interpreted, intuitively, by letting

$$\tilde{t}_a(x_a) = t_a(x_a) + x_a \frac{dt_a(x_a)}{dx_a} \qquad \forall\ a$$

The travel time $\tilde{t}_a$ can be interpreted as the marginal contribution of an additional traveler† on the ath link to the total travel time on this link. It is the sum of two components: $t_a(x_a)$ is the travel time experienced by that additional traveler when the total link flow is x_a, and $dt_a(x_a)/dx_a$ is the additional travel-time burden that this traveler inflicts on each one of the travelers already using link a (there are x_a of them). Using the new travel time variable, $\tilde{t}_a$, Eq. [3.24] can be written as

$$\begin{aligned}\frac{\partial}{\partial f_l^{mn}} \tilde{z}[\mathbf{x}(\mathbf{f})] &= \sum_b \delta_{b,l}^{mn} \tilde{t}_b \\ &= \tilde{c}_l^{mn} \qquad \forall\ l, m, n \qquad [3.25]\end{aligned}$$

where $\tilde{c}_l^{mn}$ is interpreted in a fashion analogous to $\tilde{t}_a$. It is the marginal total travel time on path l connecting O–D pair m–n.

The first-order conditions for the SO program can now be written as

$$f_l^{mn}(\tilde{c}_l^{mn} - \tilde{u}_{mn}) = 0 \qquad \forall\ l, m, n \qquad [3.26a]$$

$$\tilde{c}_l^{mn} - \tilde{u}_{mn} \geq 0 \qquad \forall\ l, m, n \qquad [3.26b]$$

$$\sum_l f_l^{mn} = q_{mn} \qquad \forall\ m, n \qquad [3.26c]$$

$$f_l^{mn} \geq 0 \qquad \forall\ l, m, n \qquad [3.26d]$$

†More accurately, an additional infinitesimal flow unit.

Equations [3.26a] and [3.26b] state that, at optimality, the marginal total travel times on all the used paths connecting a given O–D pair are equal. This result is reminiscent of classical optimization results in which the marginals of the functions minimized have to be equal at the solution point. The flow on a given route is zero only if the marginal total travel time on this route is greater than or equal to the marginal total travel time on the used routes. The value of the dual variable $\tilde{u}_{mn}$ at optimality is the marginal travel time on all the used paths between m and n.

For the solution of the program SO to be unique, it is sufficient to show that the Hessian of $\tilde{z}(\mathbf{x})$ is positive definite. A typical term of this Hessian can be obtained by taking the derivative of the objective function with respect to x_m and x_n, that is,

$$\frac{\partial \tilde{z}(\mathbf{x})}{\partial x_b} = \frac{\partial}{\partial x_b} \sum_a x_a t_a(x_a)$$

$$= t_b(x_b) + x_b \frac{dt_b(x_b)}{dx_b}$$

and

$$\frac{\partial^2 \tilde{z}(\mathbf{x})}{\partial x_b \, \partial x_a} = \begin{cases} 2 \dfrac{dt_a(x_a)}{dx_a} + x_a \dfrac{d^2 t_a(x_a)}{dx_a^2} & \text{for } b = a \\ 0 & \text{otherwise} \end{cases} \qquad \forall\, a, b \qquad [3.27]$$

As in the UE program, this result represents a diagonal Hessian with the nonzero terms $\partial^2 \tilde{z}(\mathbf{x})/\partial x_a^2$ given by Eq. [3.27]. This Hessian is positive definite if all those terms are positive, which is the case if all the link performance functions are convex. All performance functions dealt with in this text have the general shape shown in Figure 3.2, which is convex, and thus $dt_a^2(x_a)/dx_a^2 > 0$ and the diagonal terms are positive. Consequently, the SO program has a unique minimum in terms of link flows.

The next section compares some aspects of the SO problem to the UE program in order to illustrate the nature of the corresponding flow patterns.

3.5 USER EQUILIBRIUM AND SYSTEM OPTIMUM

It is interesting to note that when congestion effects are ignored, both UE and SO programs will produce identical results. Imagine a network where $t_a(x_a) = t'_a$. In other words, each link travel time is not a function of the flow on that (or any other) link. In this case, the SO objective function would be

$$\tilde{z}(\mathbf{x}) = \sum_a x_a t'_a \qquad [3.28a]$$

and the UE objective function

$$z(\mathbf{x}) = \sum_a \int_0^{x_a} t'_a \, d\omega$$
$$= \sum_a x_a t'_a \qquad [3.28b]$$

which is identical to the SO objective function, $\tilde{z}(\mathbf{x})$.

Minimizing the objective function shown in Eqs. [3.28] subject to the flow constraints [3.19b] and [3.19c] is an easier task than solving either the SO or UE problems. The reason is that in this case the link travel times are not a function of the link flows, whereas, in general, both the SO and UE problems assume that t_a does vary with x_a. The problem here is to find the flow pattern that minimizes the total travel time over the network, given the (fixed and known) values of the link travel times and the O–D matrix. The solution of this problem is conceptually straightforward—all the flow for a given O–D pair r–s, q_{rs}, is assigned to the minimum-travel-time path connecting this pair. All other paths connecting this O–D pair do not carry flow. Consequently, this traffic assignment procedure is known as the "all-or-nothing" assignment. The resulting flow pattern is both an equilibrium situation (since no user will be better off by switching paths) and an optimal assignment (since the total travel time in the system is obviously minimized).

The study of this special case is important in the development of solution algorithms for the more general problem of equilibrium assignment. This study is undertaken in Chapter 5, while the remainder of this section deals with both the UE and SO formulations.

The similarity in the structures of the SO and UE programs can be expressed in various ways. For example, if the travel times over the network are expressed in terms of $\tilde{t}_a(x_a)$, the solution of the UE program with these travel times will produce an SO flow pattern. Similarly, the SO formulation with link travel-time functions given by

$$\hat{\tilde{t}}_a(x_a) = \frac{1}{x_a} \int_0^{x_a} t_a(\omega) \, d\omega \qquad [3.29]$$

will result in a UE flow pattern (see Problem 3.10).

In cases in which the flow over the network is relatively low, the network links are not congested. The marginal travel time on each link, at this point, is very small due to the shape of the link performance functions (see Figure 3.2); the slope of this function is very small for low flow. In this case, the UE and SO flow patterns are similar since the travel times are almost insensitive to additional flows. The situation, then, is close to the fixed-travel-time case described in Eqs. [3.28].

As the flows between origins and destinations increase, the UE and SO patterns become increasingly dissimilar. Large flows mean that some links carry an amount of flow which is near their capacity. In that range the marginal travel time, $\tilde{t}_a(x_a)$, becomes very large though the travel time itself remains

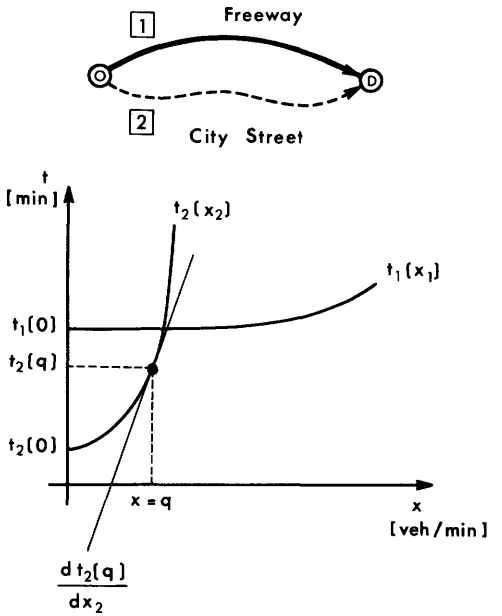


Figure 3.5 User equilibrium and system optimization. The UE solution to the two-link network equilibrium problem (with $x_1^* = 0$ and $x_2^* = q$) is not a system optimizing solution since $t_1(x_1^*) + x_1^* \, dt_1(x_1^*)/dx_1 \neq t_2(x_2^*) + x_2 \, dt_2(x_2^*)/dx_2$.

bounded. Thus the differences between solving the same problem with the set of marginal travel times $\{\tilde{t}_a\}$, versus solving it with the travel times themselves, $\{t_a\}$, increase as the network becomes more congested.

To understand this better, consider the network example shown in Figure 3.5. This network includes one O–D pair connected by two links (paths). Assume that the top link (number 1 in the figure) is a freeway, while the other link represents a city street. Hypothetical performance curves for such links are shown in the figure. If the total O–D flow is q, the UE solution will be the one shown in the figure with $x_1 = 0$ and $x_2 = q$. In this case, $t_2(q) < t_1(0)$, and no user will choose to travel on the freeway. Note, however, that the derivative of $t_2(x_2)$ at $x = q$, $dt_2(q)/dx_2$, is relatively large and it is therefore possible that $\tilde{t}_1(0) < \tilde{t}_2(q)$. In other words, this solution is not optimal from the system perspective. The SO solution to this problem may include some flow using the top route as well. If one driver shifts from route 2 to route 1 (at the UE flow pattern shown in the figure), his or her own travel time will increase—the driver will experience $t_1(0)$ instead of $t_2(q)$. The travel time experienced by each of the remaining drivers on route 2 will, however, decrease by $dt_2(q)/dx_2$. Thus, from the system perspective, an increase of $[t_1(0) - t_2(q)]$ may be more than offset by a decrease of $(q - 1)\, dt_2(q)/dx_2$. The SO flow pattern is achieved only when

$$t_1(x_1) + x_1 \frac{dt_1(x_1)}{dx_2} = t_2(x_2) + x_2 \frac{dt_2(x_2)}{dx_2} \qquad [3.30]$$

(Problem 3.12 includes a numerical example intended to demonstrate this

point.) Note that flow should still be treated as a continuous quantity; the mention of an integer flow unit (one driver) above was made only to explain the concept.

A failure to realize the fundamental difference between the normative SO flow pattern and the descriptive UE flow pattern can lead to pseudo-paradoxical scenarios. The most famous of these is known as "Braess's paradox," which is described below.

Figure 3.6a depicts a simple network including one O–D pair connected by two paths and four links. The figure shows the two paths (numbered 1 and 2) and the congestion curves for each of the four links. Assume that 6 units of flow are to travel between O and D (i.e., $q = 6$). The user equilibrium flow pattern can be solved for this network by inspection (due to the travel time symmetry between the paths). Obviously, half the flow would use each path and the solution would be

$$f_1 = 3, \quad f_2 = 3 \qquad \text{flow units}$$

or, in terms of link flows,

$$x_1 = 3, \quad x_2 = 3, \quad x_3 = 3, \quad x_4 = 3 \qquad \text{flow units}$$

The associated link travel times are:

$$t_1 = 53, \quad t_2 = 53, \quad t_3 = 30, \quad t_4 = 30 \qquad \text{time units}$$

and the path times are

$$c_1 = 83, \quad c_2 = 83 \qquad \text{time units}$$

in accordance with the UE criterion. The total travel time on the network is 498 (flow · time) units.

Assume now that the local transportation authority decides to expand the transportation network in order to improve flow and reduce delay (as well as save energy and reduce pollution). This is to be accomplished by building a new highway connecting the two intermediate nodes as shown in Figure 3.6b. The figure shows the added (fifth) link, the performance function for this link, and the new path (number 3) generated as a result of the link addition.

The old UE flow pattern is no longer an equilibrium solution since under that flow pattern

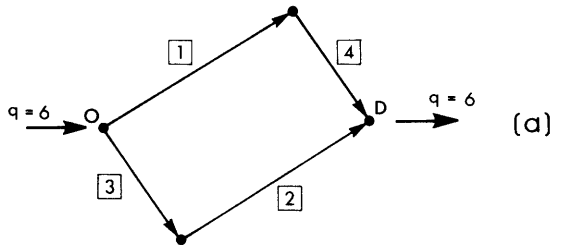
$$x_1 = 3, \quad x_2 = 3, \quad x_3 = 3, \quad x_4 = 3, \quad x_5 = 0 \qquad \text{flow units}$$

with path travel times

$$c_1 = 83, \quad c_2 = 83, \quad c_3 = 70 \qquad \text{time units}$$

The travel time on the unused path (number 3) is lower than the travel time on the two used paths, meaning that this cannot be an equilibrium solution. An equilibrium flow pattern for the new network is given by the solution

$$x_1 = 2, \quad x_2 = 2, \quad x_3 = 4, \quad x_4 = 4, \quad x_5 = 2 \qquad \text{flow units}$$



Performance Data

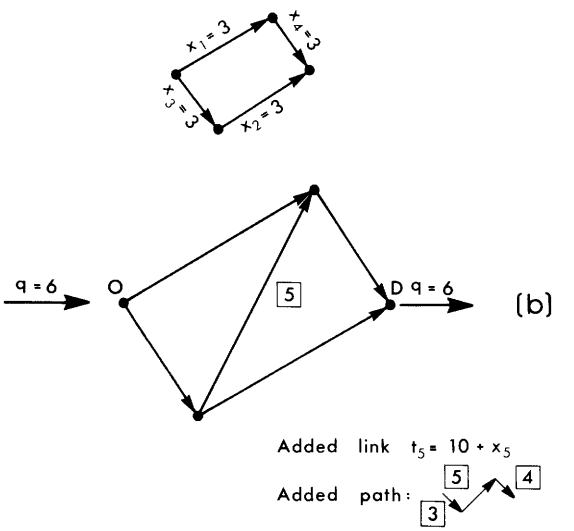
$t_1(x_1) = 50 + x_1$
$t_2(x_2) = 50 + x_2$
$t_3(x_3) = 10x_3$
$t_4(x_4) = 10x_4$

Path Definition

Path 1 [1] [4]

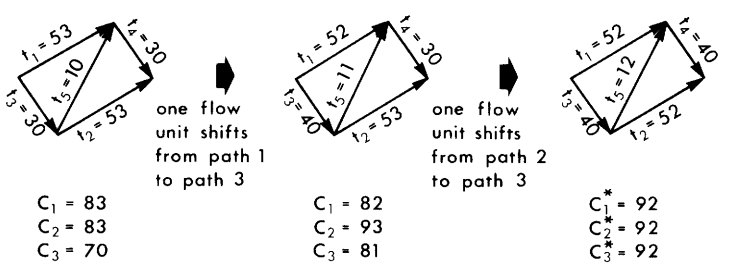
Path 2 [3] [2]

User Equilibrium Solution



Added link $t_5 = 10 + x_5$

Added path: [5] [4] [3]



$C_1 = 83$
$C_2 = 83$
$C_3 = 70$

$C_1 = 82$
$C_2 = 93$
$C_3 = 81$

$C_1^* = 92$
$C_2^* = 92$
$C_3^* = 92$

Figure 3.6 Braess's paradox example: (a) a user-equilibrium solution for a four-link network (link numbers are shown in the squares); (b) an additional link creates a UE solution with higher individual and total cost.

with path flows

$$f_1 = f_2 = f_3 = 2 \qquad \text{flow units}$$

and path travel times

$$c_1 = c_2 = c_3 = 92 \qquad \text{time units}$$

Figure 3.6b depicts a possible sequence of assignment of flow units that would generate this equilibrium from the old one.

The important point to note here is that the total travel time in the network is now 552 (flow · time) units as compared to 498 (flow · time) units before the link addition. Not only did the total travel time go up, but the travel time experienced by each traveler in the network increased from 83 time units to 92 time units. The additional link seems to have made the situation worse—congestion and delays increased instead of decreasing. This (seemingly) counter-intuitive result is known as Braess's paradox.

This "paradox" can, of course, be readily explained. The increase in travel time is rooted in the essence of the user equilibrium, where each motorist minimizes his or her own travel time. The individual choice of route is carried out with no consideration of the effect of this action on other network users. There is no reason, therefore, to expect the total travel time to decrease.

Looking at it from a mathematical programming point of view, the supply action (adding a link) was taken with the intention of reducing the SO objective function. The flow, however, is distributed according to the UE objective function, so the resulting pattern does not necessarily reduce the SO objective function. Had the flow before and after the link addition been assigned according to the SO objective function, the total travel time could not have increased with the addition of the new link. (Prove it.) Note that the value of the UE objective function did go down from a value of 399 before the link addition, to a value of 386, after the link addition.

From a more general perspective, Braess's paradox underscores the importance of a careful and systematic analysis of investments in urban networks. Not every addition of capacity can bring about all the anticipated benefits and in some cases, the situation may be worsened. In fact, traffic engineers have known for a long time that restrictions of travel choices and reductions in capacity may lead to better overall flow distribution patterns. This, for instance, is the underlying principle behind many traffic control schemes, such as ramp metering on freeway entrances.†

†Ramp metering is the process of restricting the entry of flow onto a freeway, usually by installing a traffic light at the entrance ramp. This light controls the number of cars allowed to enter the freeway during certain times of the day (typically, the peak traffic period). In other words, it effectively lowers the capacity of one of the network links (the ramp). The metering causes drivers to use alternative routes by raising the travel time associated with the freeway entrance. This reduction in capacity leads to better overall conditions and lower total delay compared with the situation associated with no controls.

3.6 SUMMARY

The focus of this chapter is on the formulation of the traffic assignment problem as a mathematical program. This program is given by Eqs. [3.3]. To demonstrate that the solution of this program is equivalent to the solution of the UE conditions, it is shown that the equilibrium equations are, in fact, the first-order conditions of the program. This guarantees that the equilibrium conditions hold at any stationary point of the program. Next, it is shown that this program is strictly convex, meaning that it has only one stationary point which is a minimum. All this proves that instead of solving the equilibrium equations directly, the equilibrium flows can be determined by minimizing the equivalent mathematical program. This approach to the solution of equilibrium problems is adopted throughout this book.

The nature of the user equilibrium and its equivalent minimization program are illustrated in this chapter by contrasting this formulation with the system-optimization formulation. The SO formulation calls for the flow pattern that minimizes the total travel time in the system. It is generally not an equilibrium flow pattern, since under it some travelers can be better off by unilaterally changing routes. It represents, however, the flow pattern for which the total systemwide travel time is the lowest. The difference between the UE equivalent minimization and the SO program is that the former is devised to describe motorists' behavior, whereas the latter is normative in nature. A failure to understand these differences may lead to "paradoxical" situations when travel choices are added to the network but all users are made worse off. Such situations highlight the noncooperative behavior represented by the user equilibrium.

3.7 ANNOTATED REFERENCES

The formulation of the user-equilibrium problem as a mathematical program was first developed by Beckmann et al. (1956), who proved the equivalency and the existence and uniqueness of the solution. Boyce (1981), in an editorial note, traces the historical development of the concept of equilibrium assignment, including many algorithms and problem formulations. The differences between the user-equilibrium and system-optimizing flow patterns over a transportation network are discussed by Potts and Oliver (1972) and Newell (1980), from whom the example shown in Figure 3.5 was taken. The paradox presented in Section 3.5 was discussed and explained by Murchland (1970) following Braess (1968). A case in which this phenomenon actually took place is reported by Knödel (1969). Newell (1980) pointed out the evident nature of this phenomenon in the context of standard traffic engineering practice.

3.8 PROBLEMS

3.1. Consider the network in Figure P3.1. It includes two O–D pairs ($1 \rightarrow 4$ and $2 \rightarrow 4$) and five links. (The link numbers are shown on the respective links.) Devise a numbering system for the paths and identify the values of all the indicator variables. Write the link–path incidence matrix for this network. Use this matrix to get x_3 from **f**.

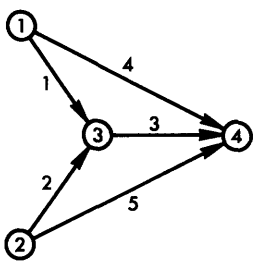


Figure P3.1

3.2. Solve the network example in Figure 3.3 by setting up the equivalent minimization and by using Lagrange multipliers. Comment on the value of the multipliers at the solution point.

3.3. Find the user-equilibrium flow and travel times for the network shown in Figure P3.2, where

$$t_1 = 2 + x_1^2$$

$$t_2 = 3 + x_2$$

$$t_3 = 1 + 2x_3^2$$

$$t_4 = 2 + 4x_4$$

(The link numbers are shown on the respective links.) The O–D trip rate is 4 units of flow between nodes 1 and 3.

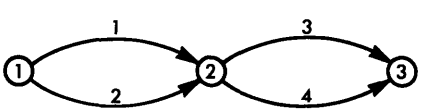


Figure P3.2

3.4. Solve for the equilibrium flows in the network example depicted in Figure 3.4. Set up the UE program and solve the first-order conditions.

***3.5.** **(a)** Express the Hessian of the UE objective function in terms of path flow over the network depicted in Figure 3.4. Show that it is not a positive-definite matrix.

(b) Show that the path-flow Hessian of the UE objective function is usually not a positive-definite matrix. For what type of network will this Hessian be positive definite?

3.6. Show that the integral of an increasing function is always a convex function.

3.7. Find the system-optimizing flow pattern for the network example in Figure 3.3.

Compare this flow pattern to the UE flow pattern and comment on the differences.

3.8. Find the system-optimizing flow pattern for the network example discussed in Problem 3.3. Compare it to the UE flow pattern.

3.9. Does the SO program have a unique solution in terms of path flows? Explain your answer.

3.10. **(a)** Show that the solution of the SO program with travel times $\hat{\hat{t}}_a(x_a)$ (see Eq. [3.29]) is an UE flow pattern.

(b) Show that the solution of the UE program with travel times $\tilde{t}_a(x_a)$ (see Eq. [3.24] and the definition that follows) is an SO flow pattern.

3.11. Show, analytically, that as the flows over the network become smaller (and congestion decreases), the SO flow pattern tends to grow in similarity to the UE flow pattern.

3.12. Consider the network of two routes (a freeway and a city street connecting one O–D pair) shown in Figure 3.5. Let

$$t_1 = 3 + 0.5x$$

$$t_2 = 1 + x_2$$

$$q = 1.5$$

(a) Find the UE solution and demonstrate that it is not an SO solution.

(b) Find the SO solution and demonstrate that it is not an UE solution.

3.13. Solve for the system-optimizing flow pattern over the network depicted in Figure 3.6 before and after the link addition. Show that the total travel time decreases. Explain why the total travel time would never increase if a link is added and the flow pattern follows the SO rule.

3.14. Assume that a transportation planner can set a toll on each one of the paths connecting the origins to the destination shown in Figure P3.3.

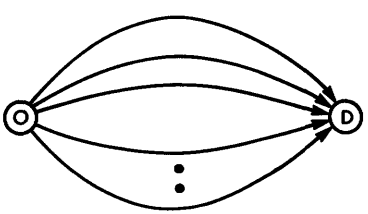


Figure P3.3

(a) Given that the O–D flow is q and the performance function on each link is $t_a(x_a)$, how should the tolls be set so that the total travel time in the network is minimized?

(b) Assuming that the link performance functions are linear, derive a closed-form expression for these tolls.

(c) Comment about the feasibility of this approach in real urban networks.

4

Review of Some Optimization Algorithms

This chapter describes some of the most common minimization algorithms. It is not a comprehensive review, but rather a presentation of methods that are applicable to the solution of minimization programs of the type formulated in this text as well as methods that bring out some important principles. The discussion deals separately with the minimization of a function of a single variable and the minimization of multidimensional functions.

The focus of the discussion is on the mechanics of the algorithms and, therefore, proofs of convergence are not given explicitly. Such proofs can be found in most of the nonlinear programming books mentioned at the end of this chapter. The last section is devoted exclusively to one algorithm—the convex combinations method. This algorithm is the basis for solving many equilibrium problems.

4.1 ONE-DIMENSIONAL MINIMIZATION

This section deals with the minimization of a nonlinear function of a single variable, $z(x)$. The regularity conditions mentioned in Chapter 2 are still assumed to hold; that is, it is assumed that x lies within some finite interval $[a, b]$ and that $z(x)$ is continuous and uniquely defined everywhere in that interval. These requirements ensure the existence of a finite minimum of $z(x)$ for some x in the interval of interest. For the purposes of this discussion it is assumed that $z(x)$ is ditonic over the interval $[a, b]$, implying that it has only a single, unique minimum in that interval.

The study of one-dimensional optimization methods is important mainly because such an optimization (or *line search*) is in many cases a part of an algorithm designed to find the minimum of multivariate functions. Furthermore, some of the principles imbedded in the algorithms described below are used in minimizing multivariate functions as well.

This section includes two basic approaches to single-dimensional minimization. The first is known as interval reduction and includes the golden section and the bisection methods. The second utilizes quadratic curve fitting and includes Newton's search, the false position method, and the quadratic approximation method.

Interval Reduction Methods

Interval reduction methods involve iterative procedures in which each iteration is focused on a current interval. The current interval in the nth iteration is a portion of $[a, b]$, denoted $[a^n, b^n]$, which was determined to include the minimum point, x^*. At each iteration this interval is examined and divided into two parts: the part in which the minimum *cannot* lie and the current interval for the next iteration. The part in which the minimum cannot lie is discarded and the procedure is repeated for the new current interval. These procedures start by designating $[a, b]$ as the first current interval (i.e., $a^0 = a$ and $b^0 = b$). The interval is then reduced at each successive iteration until a good approximation (a small enough current interval) for x^* is obtained.

To understand this reduction process better, let the size of the current interval at iteration n be denoted by I_n and let $r_n = I_{n+1}/I_n$ denote the interval reduction ratio for the nth iteration. Since there is no reason to believe that any of these algorithms would work better (in terms of interval reduction) in any given iteration, the reduction ratio is usually a constant, (i.e., $r_n = r$ for every n).

Interval reduction algorithms are typically terminated when the size of the interval of interest is less than a predetermined constant. The estimate of x^* is then the midpoint, $\hat{x}$, of the interval remaining after N iterations, I_N [i.e., $\hat{x} = (a^N + b^N)/2$]. If the optimum has to be estimated with a tolerance of $\pm\epsilon$ (i.e., x^* must lie within $\hat{x} \pm \epsilon$), then the number of required iterations can be calculated as a function of the length, I_0, of the initial interval (see Problem 4.1). The number is

$$N = \text{INT}\left[\frac{\log 2\epsilon - \log I_0}{\log r} + 1\right] \qquad [4.1]$$

where INT $[\cdot]$ means the integer part of the argument.

The various interval reduction algorithms differ from each other only in the rules used to examine the current interval and to decide which portion of it can be discarded.

Golden section method. The interval reduction strategy of the golden section search is based on a comparison of the values of $z(x)$ at two points, x_L^n and x_R^n (where $x_L^n < x_R^n$). These points are within the interval of interest $[a^n, b^n]$ at the nth iteration. The choice rule for selecting the interior points is the unique feature of this method; it is explained below, following an explanation of the interval discarding process.

The discarding mechanism is demonstrated in Figure 4.1, depicting a ditonic function, $z(x)$, which has to be minimized in the interval $[a^n, b^n]$. The top drawing (denoted "iteration n") shows the two interior points at the nth iteration x_L^n and x_R^n. In this case, $z(x_L^n) > z(x_R^n)$. Since the function is ditonic, the optimum must lie "to the right" of x_L^n (i.e., $x^* \geq x_L^n$), and thus the interval $[a^n, x_L^n]$ can be discarded. This completes the nth iteration. The new current interval [for the $(n+1)$st iteration] is $[a^{n+1}, b^{n+1}]$, where $a^{n+1} = x_L^n$ and $b^{n+1} = b^n$. The interval reduction process continues with two new interior points, x_L^{n+1} and x_R^{n+1}, as shown in the bottom drawing of Figure 4.1 (labeled "iteration $n+1$"). Note that if the function was such that $z(x_R^n)$ was greater than $z(x_L^n)$, then the interval $[x_R^n, b^n]$ would have been discarded at the nth iteration. This is the case in iteration $n+1$, where $z(x_L^{n+1}) < z(x_R^{n+1})$. In this case the interval $[x_R^{n+1}, b^{n+1}]$ is discarded and the new current interval is $[a^{n+1}, x_R^{n+1}]$ (since x^* cannot be "to the right" of x_R^{n+1}).

The essence of the golden section method is in the rule used for choosing x_L^n and x_R^n, given an interval $[a^n, b^n]$. This rule is designed to minimize the number of function evaluations. Given a new current interval the algorithm

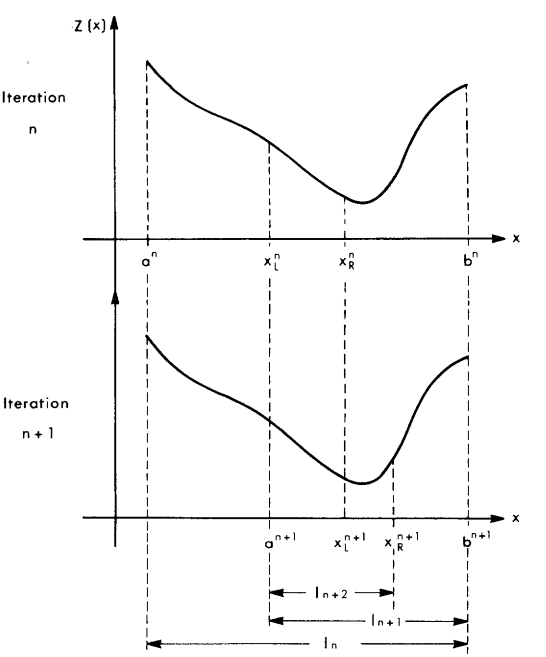


Figure 4.1 The interval reduction sequence followed by the golden section method.

needs two interior points to continue. The golden section procedure makes use of one of the interior points from the last interval (where the function value is already known). Only one new point where the function value needs to be evaluated is therefore added at each iteration.

This arrangement is illustrated in Figure 4.1, where, when going from the nth to the $(n+1)$st iteration, only one interior point has to be added and evaluated (x_R^{n+1} in the bottom drawing); the other is available from the nth iteration since $x_L^{n+1} = x_R^n$ [so $z(x_L^{n+1})$ need not be evaluated because it equals $z(x_R^n)$]. Such a sequence of interior points can be obtained, while keeping a constant reduction ratio, by using $r = 0.618$ [more precisely, $r = \frac{1}{2}(\sqrt{5} - 1)$]. Accordingly, the interior points are selected so that x_R^n is 0.618 of the interval length to the right of a^n and x_L^n is 0.618 of the interval length to the left of b^n (or 0.382 of the interval length to the right of a^n). The quantity $\frac{1}{2}(\sqrt{5} - 1)$ is known as the "golden section," from which the method derives its name.

Note that such a sequence of intervals leads to a situation in which

$$I_n = I_{n+1} + I_{n+2} \qquad [4.2]$$

as shown in Figure 4.1 (see Problem 4.2).

A flowchart of the algorithm is presented in Figure 4.2.† This algorithm takes as input the function to be minimized, the interval end points, and the accuracy required. The output includes the estimate of the optimum, $\hat{x}$, the number of iterations performed, N, and the accuracy actually obtained, $\frac{1}{2}(b^N - a^N)$. Note that given r, the number of iterations needed to achieve a given degree of accuracy, can be determined a priori by using Eq. [4.1].

A well-known search method similar to the golden section is the Fibonacci algorithm mentioned in the references given in Section 4.5. It is slightly more efficient than the golden section procedure, due to better positioning of the first two interior points. From a practical point of view, however, the small advantage offered by this algorithm does not justify its detailed study (see Problem 4.5) herein.

Bisection method. In many cases, the derivative of the function to be minimized can be evaluated easily and the search for the minimum can therefore be expedited. The method of interval bisection exploits the fact that a ditonic function is monotonic on each side of the minimum. In other words, the derivative of the function to be minimized, $dz(x)/dx$, is negative for $x < x^*$ and positive for $x > x^*$.

The algorithm computes the derivative of $z(x)$ at the midpoint of the current interval, $[a^n, b^n]$. Denote this point by $\hat{x}^n$. If $dz(\hat{x}^n)/dx < 0$, then $x^* > \hat{x}^n$, meaning that the interval $[a^n, \hat{x}^n]$ can be discarded. The next current interval will thus be $[\hat{x}^n, b^n]$. If $dz(\hat{x}^n)/dx > 0$, then $x^* < \hat{x}^n$, and the search can

†The notation ":=" used in the figure indicates the equality sign in programming languages (i.e., the assignment of a variable's value). For example, $n := n + 1$ means an increment of the counter n by 1.

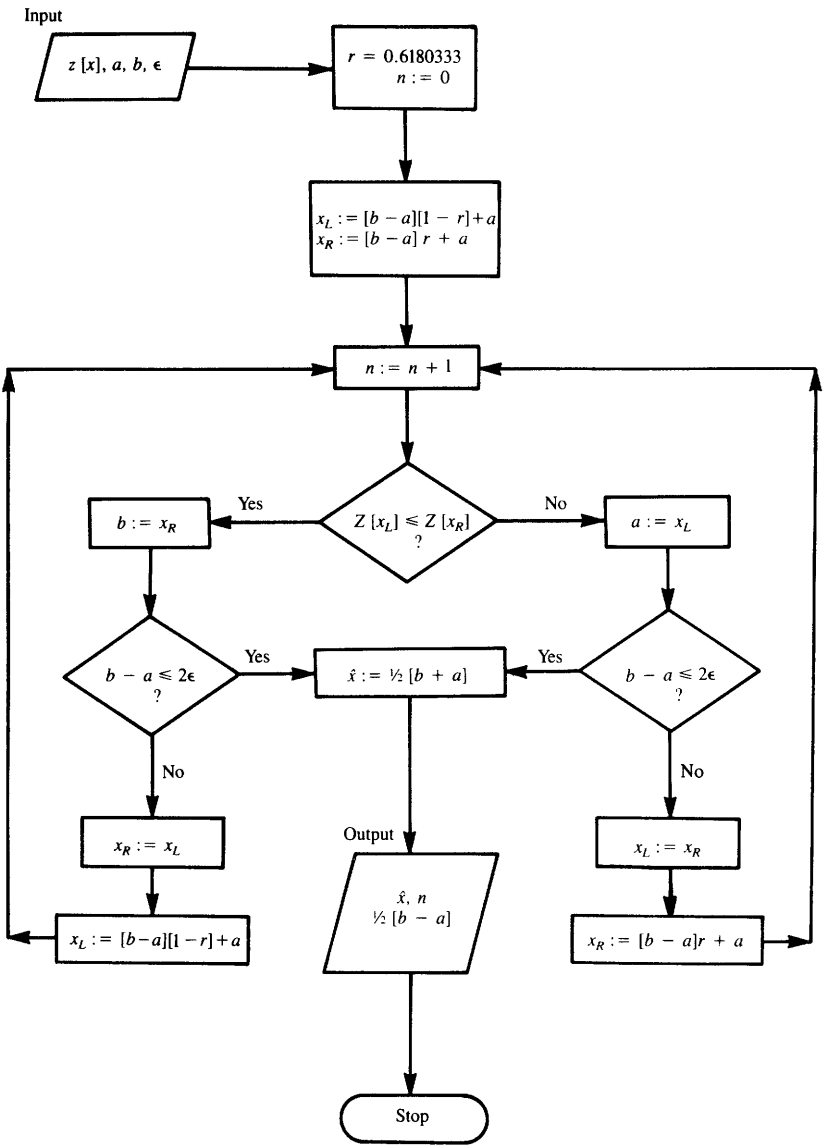


Figure 4.2 Flowchart of the golden section algorithm.

focus on the interval $[a^n, \hat{x}^n]$. Figure 4.3 depicts a flowchart of the bisection method (known also as Bolzano search).

As with the golden section method, a convergence criterion can be specified, causing the precedure to terminate (the estimate of x^* is taken at the middle of the remaining interval) when this criterion is met. The reduction ratio for this method is $r = 0.5$, and Eq. [4.1] can be used to determine the number of iterations required for a given accuracy. For example, only six

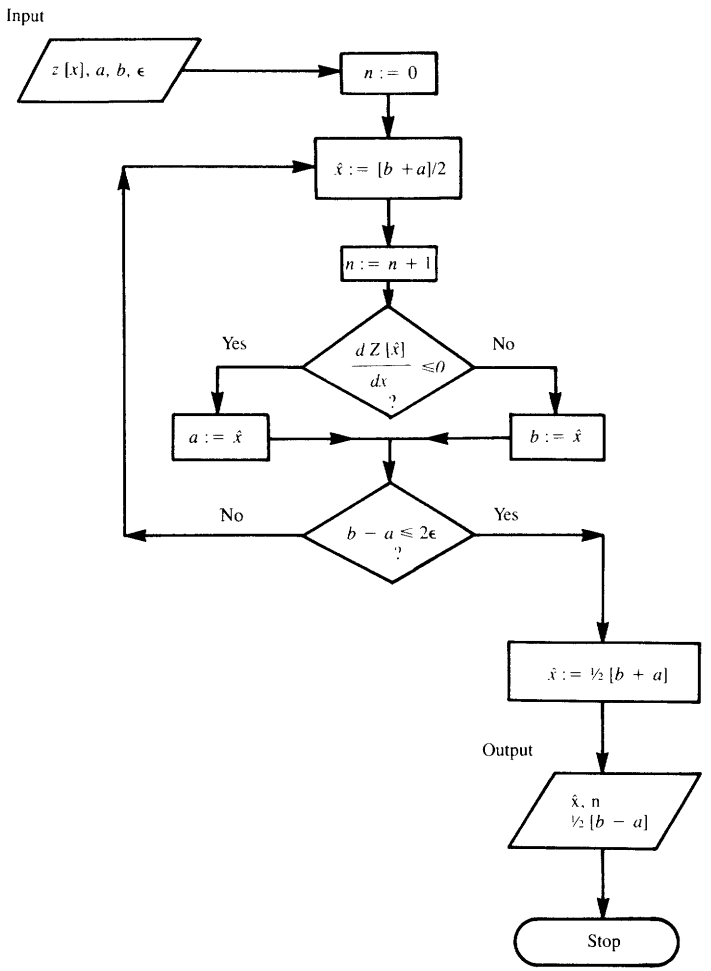


Figure 4.3 Flowchart of the bisection algorithm.

iterations of the bisection method are required to determine the minimum of $z(x)$ with an accuracy of $\pm 1\%$ of the length of the original interval. The golden section method, by way of comparison, would require nine iterations, involving 10 evaluations of $z(x)$ to reach this degree of accuracy. Table 4.1 shows the ratio of the length of the current interval and the original interval, after N function evaluations† for each of the two interval reduction methods discussed in this section. The table demonstrates that the convergence rate of the bisection method is almost twice that of the golden section method. The bisection method requires, however, that the derivative of $z(x)$ be evaluated in every iteration. This may not be easy in some cases, and thus this algorithm

†Note that the first iteration of the golden section method requires two evaluations of $z(x)$.

TABLE 4.1 Convergence Rate for Interval Reduction Methods (I_N/I_0 versus N for each algorithm)

Number of Function Evaluations, N	Size of Current Interval as a Fraction of the Initial Interval	
	Golden Section	Bisection
2	0.6180	0.2500
3	0.3819	0.1250
4	0.2361	0.0625
5	0.1459	0.0313
6	0.0902	0.0156
7	0.0557	0.0078
8	0.0344	0.0039
9	0.0213	0.0020
10	0.0132	0.0010

should be preferred to the golden section algorithm only in cases in which the calculation of the derivative is not much more difficult than calculating the function itself.

All the interval reduction methods produce, after a given number of iterations, a final interval $[a^N, b^N]$ which contains the minimum. If the minimized function is convex, the optimal value of $z(x)$ can be bounded. As mentioned in Section 2.1, a linear approximation to a convex function will always underestimate it. Thus

$$z(x^*) \geq z(a^N) + \frac{dz(a^N)}{dx}(x^* - a^N) \qquad [4.3a]$$

and

$$z(x^*) \geq z(b^N) + \frac{dz(b^N)}{dx}(x^* - b^N) \qquad [4.3b]$$

Since the value of x^* is not known, the value of b^N can be substituted for x^* in Eq. [4.3a], and a^N can be substituted for x^* in Eq. [4.3b]. The higher of the two lower bounds suggested by these linear approximations is a tighter lower bound for $z(x^*)$. Alternatively, the two approximations can be solved simultaneously for an even tighter bound.

Curve-Fitting Methods

When the function to be minimized, $z(x)$, is not only ditonic but also relatively smooth, such smoothness can be exploited by algorithms that are more efficient than the aforementioned interval reduction methods. Curve-fitting methods work by iteratively improving a current solution point. The characteristics of the function at the last point (or points) are utilized to generate a smooth approximation for $z(x)$. (All the methods described in this

section use a parabola for this purpose.) The new point is taken to be at the minimum of this approximation as explained below in the discussion of specific algorithms. These algorithms differ from each other in the technique used to generate the approximation of the objective function.

Curve-fitting methods exhibit many of the characteristics of general minimization methods in that (unlike interval reduction methods) they generate a series of points $x^1, \ldots, x^N, \ldots$ that converge to the minimum, x^*. Each point is generated by applying some algorithmic procedure to the previous point, and the algorithm terminates when a convergence criterion is met.

The convergence criterion for curve-fitting methods can be based on various rules. These rules are typically based on the required accuracy of the solution or on "cost-effectiveness" considerations. For example, the algorithm can be terminated when the marginal contribution of an iteration to the improvement in the solution becomes small. In other words, if

$$z(x^{n-1}) - z(x^n) \leq \kappa$$

where κ is a predetermined constant (tolerance), the algorithm terminates. Alternatively, a dimensionless constant based on the relative change between successive solutions (i.e., $[z(x^{n-1}) - z(x^n)]/z(x^{n-1})$ can be used to test for convergence.† In many cases the algorithms can terminate on the basis of the change in the variable value (e.g., when $|x^n - x^{n-1}| \leq \kappa'$ where κ' is some other predetermined constant). The closeness of any particular solution to the minimum can also be tested by checking the derivative of $z(x^n)$ at x^n. If this derivative is close to zero, the algorithm terminates.

The choice of convergence criterion is based on the function to be minimized, the particular algorithms used, and the context in which the problem is solved. This point is discussed in later chapters in connection with some specific problems. The next paragraphs present some of the common curve-fitting algorithms. As mentioned above these algorithms differ from each other in the method used to approximate $z(x)$.

Newton's method. Newton's method approximates $z(x)$ at each iteration with a quadratic fit at the current point, x^n. The fitted curve, $\hat{z}(x)$, is given by

$$\hat{z}(x) = z(x^n) + \frac{dz(x^n)}{dx}(x - x^n) + \frac{1}{2}\frac{d^2z(x^n)}{dx^2}(x - x^n)^2 \qquad [4.4a]$$

This curve, $\hat{z}(x)$, is a parabola that agrees with $z(x)$ at x^n up to second derivatives. The next solution, x^{n+1}, is located at the point that minimizes $\hat{z}(x)$ [i.e.,

†These rules have been devised for descent method, that is, for algorithms in which $z(x^1) > z(x^2) > \cdots > z(x^n) > \cdots \geq z(x^*)$. While curve-fitting algorithms are not strictly descent methods (why?), they tend to behave as if they were after the first few iterations, and thus the convergence criteria above do apply.

where $d\hat{z}(x)/dx = 0$]. This point is given by

$$x^{n+1} = x^n - \frac{dz(x^n)/dx}{d^2z(x^n)/dx^2} \qquad [4.4b]$$

Equation [4.4b] specifies, then, the algorithmic step, that is, the rule by which x^{n+1} is obtained from x^n. (Equation [4.4a] is included here only to explain the basis of this rule; it need not be evaluated when the algorithm is executed.)

Newton's method is very efficient, but it requires, of course, that $z(x)$ be twice differentiable. It also requires that the first and second derivatives be evaluated at every iteration. In some cases this may be very costly in terms of computational effort.

False position method. The false position method is similar to Newton's search. The only difference is that, instead of using the second derivative at x^n, it uses the first derivative at x^{n-1} and at x^n to approximate the second derivative. The approximation is

$$\frac{d^2z(x^n)}{dx^2} \simeq \frac{dz(x^{n-1})/dx - dz(x^n)/dx}{x^{n-1} - x^n} \qquad [4.5]$$

This expression can be substituted for the second derivative in Eq. [4.4b] to obtain the algorithmic step of the false position method. This method can be used when the second derivatives are unavailable or difficult to evaluate.

Quadratic fit method. A third curve-fitting method uses no derivatives at all. The quadratic fit is based on three points, x_1, x_2, and x_3. At the nth iteration, these points are a subset of the series of solution points $x^1, \ldots, x^n$. As with the other curve-fitting methods, the new solution point is chosen at the minimum of the fitted curve. This point is given by

$$x^{n+1} = \frac{1}{2}\frac{(x_2^2 - x_3^2)z(x_1) + (x_3^2 - x_1^2)z(x_2) + (x_1^2 - x_2^2)z(x_3)}{(x_2 - x_3)z(x_1) + (x_3 - x_1)z(x_2) + (x_1 - x_2)z(x_3)} \qquad [4.6]$$

The set of three points used in the next iteration includes x^{n+1}, and the two points out of x_1, x_2, and x_3 with the smallest value of $z(x)$. These points are labeled x_1, x_2, and x_3 and the procedure is repeated (see Problem 4.9).

For a given problem, the choice among the various curve-fitting methods should be guided by the difficulty of calculating the derivatives. If this calculation can be easily accomplished, Newton's method is the appropriate algorithm, whereas the quadratic fit method is preferred when even the first derivative is difficult to evaluate. In general, curve-fitting methods are faster than interval reduction methods when applied to functions that are relatively smooth. Curve-fitting methods should also be used in cases where the interval within which the minimum lies is not obvious (e.g., if the function to be minimized is unconstrained).

This concludes the discussion of algorithms for minimizing single-variable functions. The next section describes some approaches to the minimization of multivariable functions.

4.2 MULTIVARIATE MINIMIZATION†

This section deals with the minimization of a nonlinear convex function of several variables. The algorithms included here are all descent methods. In each case, the algorithm generates a point $\mathbf{x}^{n+1} = (x_1^{n+1}, \ldots, x_I^{n+1})$ from $\mathbf{x}^n = (x_1^n, \ldots, x_I^n)$, so that $z(\mathbf{x}^{n+1}) < z(\mathbf{x}^n)$. The focus of the discussion is on some principles of minimization. Particular procedures are mentioned only to illustrate these principles.‡

The core of any algorithmic procedure is the calculation of $\mathbf{x}^{n+1}$ from $\mathbf{x}^n$. This algorithmic step can be written in standard form as

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha_n \mathbf{d}^n \qquad [4.7a]$$

where $\mathbf{d}^n$ is a descent direction vector $(d_1^n, \ldots, d_I^n)$ and α_n is a nonnegative scalar known as the *move size* (or "step size"). This formula means that at point $\mathbf{x}^n$ a direction in which the function is (locally) decreasing, $\mathbf{d}^n$, is identified. The move size, α_n, determines how far along $\mathbf{d}^n$ the next point, $\mathbf{x}^{n+1}$, will be.§ As Eq. [4.7a] is executed, each component of the variable vector is updated, that is,

$$x_i^{n+1} = x_i^n + \alpha_n d_i^n \qquad \text{for } i = 1, 2, \ldots, I \qquad [4.7b]$$

Consider, for example, the function $z(x_1, x_2) = (x_1 - 6)^2 + 3(x_2 - 4)^2$ depicted in Figure 4.4 and assume the current solution to be $\mathbf{x}^n = (3, 1)$. The dashed line in the figure is a tangent to the contour of $z(x)$ at $\mathbf{x}^n$. This line separates the directions of descent at $\mathbf{x}^n$ from the directions of ascent. The figure depicts several descent directions (denoted $\mathbf{d}^n$) emanating from $\mathbf{x}^n$, and also shows the direction of the gradient [denoted by $\nabla z(3, 1)$] at this point. The value of $z(\mathbf{x})$ at any point, $\mathbf{x}$, located a small distance away from $\mathbf{x}^n$ along one of the descent directions, is smaller than $z(\mathbf{x}^n)$, the value of the function at $\mathbf{x}^n$. Note that the gradient vector is perpendicular to the tangent of the contour line at $\mathbf{x}^n$. Consequently, any descent direction can be characterized by the inequality $\nabla \mathbf{z} \cdot \mathbf{d} < 0$. In other words, the cosine of the angle between the

†Readers who are not interested in the principles of minimization algorithms may skip this section and go directly to the description of the convex combinations algorithm in Section 4.3.

‡Proofs of convergence are not within the scope of this text. Several of the references mentioned in Section 4.4 contain such proofs. In particular, Zangwill's "global convergence theorem" specifies the necessary and sufficient conditions for algorithmic convergence.

§The term "move size" for α_n assumes that the direction vector is normalized so that $\sqrt{d_1^2 + d_2^2 + \cdots + d_I^2} = 1$. This normalization does not have to be performed in practice, meaning that the size of the move will actually be the product of α_n and the magnitude of the direction vector.

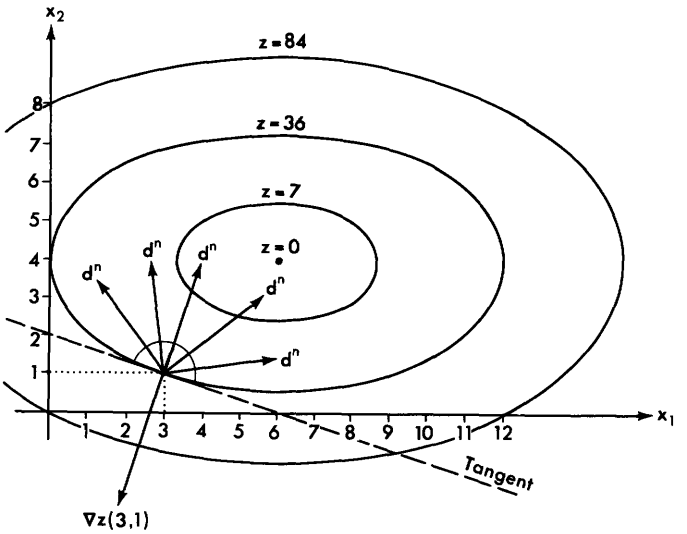


Figure 4.4 Contour lines of $z(x_1, x_2) = (x_1 - 6)^2 + 3(x_2 - 4)^2$ and the range of descent directions at $(x_1, x_2) = (3, 1)$.

gradient direction and the descent direction is always negative. Any direction vector, **d**, is a descent direction if and only if this inequality holds.

The magnitude of the step along the chosen direction is determined by the move size, α_n. If, for example, the chosen direction is $\mathbf{d}^n = (1/\sqrt{2}, 1/\sqrt{2})$, a move of $3\sqrt{2}$ would lead to the optimum in one iteration. (Check this.) If the descent direction is $\mathbf{d}^n = (0, 1)$ [pointing straight up from (3, 1)], any step size that is smaller than $\alpha_n = 6$ will cause a decrease in the objective function [as compared to $z(3, 1) = 36$]. The minimum value of $z(x_1, x_2)$ along this direction is at (3, 4). This point will be attained by using a move size, $\alpha_n = 3$ (verify this graphically).

The algorithmic iteration described in Eq. [4.7] can be used to summarize all descent methods for unconstrained minimization. These methods differ mostly in the rule used for finding the descent direction. The move size is typically chosen so that the objective function is minimized along the descent direction.

If the function to be minimized is constrained, the minimization algorithms have to be modified so that a desired but infeasible direction can be changed according to some appropriate rule. Similarly, if the move size leads outside the feasible region, it has to be truncated so that feasibility is maintained.†

The remainder of this section describes some issues related to mini-

†Note that conserving feasibility is not mandatory, and many problems can be solved without it. The algorithms discussed in this text, though, ensure that the final (optimal) solution is feasible by ensuring that the entire sequence of solution points is feasible.

mization techniques as well as specific descent algorithms for unconstrained and for constrained problems. These algorithms contain principles and ideas that are important to the analysis of network equilibium. The programs to which these algorithms are applied are assumed herein to be convex (that is, having a convex objective function defined over a convex feasible region).

Unconstrained Minimization Algorithms

The convergence criteria used to terminate the algorithmic iterations are an extension of the methods used in one-dimensional search algorithms. For instance, the convergence criterion can be based on the marginal contribution of successive iterations, that is, the closeness of $z(\mathbf{x}^n)$ and $z(\mathbf{x}^{n-1})$. In other words, terminate if

$$[z(\mathbf{x}^{n-1}) - z(\mathbf{x}^n)] \leq \kappa \qquad [4.8a]$$

Alternatively, the algorithm can be terminated if the elements of the gradient vector are close to zero, for example, if†

$$\max_i \left\{ \left| \frac{\partial z(\mathbf{x}^n)}{\partial x_i} \right| \right\} \leq \kappa \qquad [4.8b]$$

In some cases, criteria that are based on the change in the variables between successive iterations are used. These include, for example,

$$\max_i \left\{ \frac{|x_i^n - x_i^{n-1}|}{x_i^{n-1}} \right\} \leq \kappa \qquad [4.8c]$$

or

$$\sum_i (x_i^n - x_i^{n-1})^2 \leq \kappa \qquad [4.8d]$$

In these criteria (Eqs. [4.8]), κ is a predetermined tolerance (different for each criterion) selected especially for each problem based on the desired degree of accuracy.

The focus of the following discussion is on the commonly used steepest descent algorithm.

The method of steepest descent. The method of steepest descent is perhaps the most intuitively logical minimization procedure. The direction of search is opposite to the gradient direction, while the move size is chosen so as to minimize $z(\mathbf{x})$ in that direction. This means that each move is made in the direction in which $z(\mathbf{x})$ decreases (locally) the most (i.e., where the function is steepest). The length of each move is determined by the points (along this direction) where the value of $z(\mathbf{x})$ stops decreasing and starts increasing.

†The notation $\max_i \{\cdot\}$ stands for the maximum, over all possible values of i, of the arguments in the braces.

The basic iteration is given by

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha_n[-\nabla z(\mathbf{x}^n)] \qquad [4.9]$$

where α_n is the optimal move size. The gradient can be found either analytically or, when the derivatives are difficult to evaluate, by using numerical methods. If $\nabla z(\mathbf{x})$ can be derived analytically, it has to be supplied as an input to the minimization routine. Alternatively, it can be computed numerically by approximating the partial derivative of $z(\mathbf{x})$ with respect to each x_i. In other words,

$$\frac{\partial z(\mathbf{x}^n)}{\partial x_i} \simeq \frac{z(\ldots, x_{i-1}^n, x_i^n + \Delta x_i, x_{i+1}^n, \ldots) - z(\ldots, x_i^n, \ldots)}{\Delta x_i} \qquad [4.10]$$

where Δx_i is a small interval associated with the ith component of $\mathbf{x}$. The evaluation of the gradient at $\mathbf{x}^n$ therefore involves I computations of $z(\mathbf{x})$ in addition to $z(\mathbf{x}^n)$.

In order to find the value of α_n, the function

$$z[\mathbf{x}^n + \alpha(-\nabla z(\mathbf{x}^n))] \qquad [4.11]$$

has to be minimized with respect to α subject to the constraint that $\alpha > 0$. The value of α that minimizes Eq. [4.11] is α_n. Finding α_n is a one-dimensional minimization problem that can be solved by using any of the techniques mentioned in Section 4.1. If the problem is simple (and convex), α_n can be determined analytically by solving†

$$\frac{d}{d\alpha} z[\mathbf{x}^n + \alpha(-\nabla z(\mathbf{x}^n))] = 0$$

rather than using a numerical method. Note that the interval reduction methods mentioned in the preceding section cannot be naturally applied here since the problem is not constrained and therefore no initial interval exists. Any one of the curve-fitting methods, however, would be appropriate for this minimization.

The steepest descent algorithm is a descent method, meaning that the objective function value decreases at every iteration. For objective functions satisfying certain regularity conditions this method converges to a local minimum, which would naturally be a global one for strictly convex functions.

Figure 4.5 depicts a typical sequence of solutions generated by the steepest descent algorithm. An interesting property of this method is that the successive search directions are perpendicular to each other, that is, $\mathbf{d}^n \cdot \mathbf{d}^{n+1} = 0$. This follows from the fact that $\mathbf{x}^{n+1}$ is located at a point where the rate of change in $z(\mathbf{x})$ with respect to movement in the direction $\mathbf{d}^n$ is zero, due to the optimization in the line search. Thus the gradient (and its opposite direction) at $\mathbf{x}^{n+1}$ is perpendicular to the direction of $\mathbf{d}^n$.

For a general quadratic form (the contour lines of which are ellipsoids),

†Note that the point $\alpha = 0$ has to be checked as well.

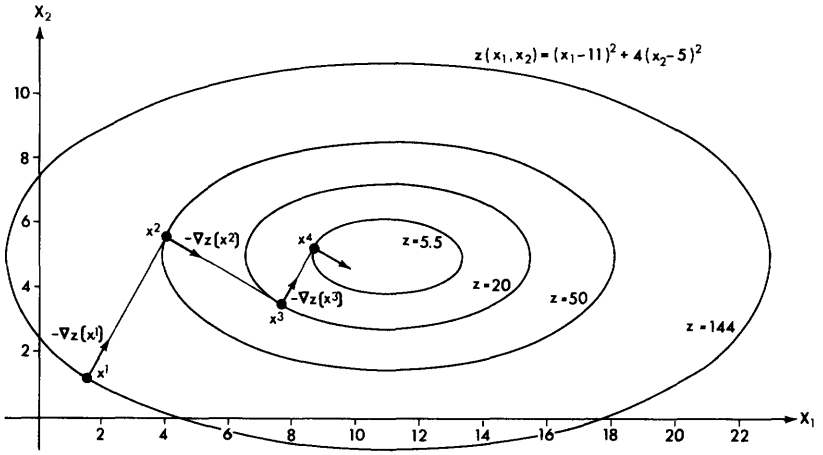


Figure 4.5 Convergence pattern of the steepest descent algorithm; note the zigzagging of the consecutive descent directions.

this "zigzagging" becomes more pronounced as the eccentricity (the ratio of the axes) increases. On the other hand, if the contours are circular, the steepest descent method converges in a single step. The zigzagging means that the steepest descent method requires a relatively large number of iterations. Several heuristic procedures to alleviate this problem are suggested in the references mentioned in Section 4.5. These methods generally choose a direction other than the opposite gradient every few iterations.

Other methods. The operations research literature includes many other minimization approaches including second order methods in which the search direction is determined by using information regarding the second derivatives of the objective function. Most of these methods are not particularly applicable to the solution of the problems arising in the course of studying urban networks. These problems are typically large (i.e., including many variables), a characteristic that often precludes certain calculations such as the determination of the Hessian (which is the matrix of second derivatives) or even approximations of this Hessian. The following paragraphs, then, are provided mostly for completeness.

One of the most efficient minimization algorithms is Newton's method. This method is a straightforward extension of the corresponding line search procedure (described in Section 4.1). The objective function $z(\mathbf{x})$ is approximated by a second-order Taylor series. This approximation is then minimized and the new solution is taken to be the point that minimizes the approximation. The resulting algorithmic step is given by

$$\mathbf{x}^{n+1} = \mathbf{x}^n - \nabla z(\mathbf{x}^n) \cdot [\mathbf{H}(\mathbf{x}^n)]^{-1} \qquad [4.12]$$

where $\mathbf{H}(\mathbf{x}^n)$ is the Hessian of $z(\mathbf{x})$ at $\mathbf{x} = \mathbf{x}^n$, and $[\cdot]^{-1}$ denotes the matrix

inversion operator. This method can be expressed in the standard form [4.7a] by defining $\mathbf{d}^n = -\nabla z(\mathbf{x}^n) \cdot [H(\mathbf{x}^n)]^{-1}$ and $\alpha_n = 1, \forall\, n$.†

Newton's method is more efficient than the steepest descent method because its search direction uses more information about the shape of $z(\mathbf{x})$ at $\mathbf{x}^n$. It requires, however, the calculation and inversion of the Hessian matrix at each iteration, a task that is computationally intensive even for medium-size problems, and totally impractical for large ones.

The approach underlying quasi-Newton methods‡ is to try to construct an approximation to the inverse Hessian by using information gathered in the minimization process. The current approximation is used at each step to modify the gradient direction and the process is usually coupled with a line search to determine the optimal move size. The particulars of these methods can be found in the references mentioned at the end of this chapter.

Constrained Minimization Algorithms

The study of transportation network equilibrium problems involves equivalent mathematical programs which include exclusively linear constraints. The discussion of constrained minimization is limited, therefore, to these types of constraints. The techniques reviewed in this section are applicable to both equality and inequality constraints, even though most network problems involve only equality constraints (apart from the nonnegativity constraints).

The basic framework outlined in the preceding section for unconstrained minimization (i.e., finding a descent direction and advancing by an optimal amount along this direction) can be used to describe algorithms for constrained optimization as well. The added difficulty in constrained optimization is to maintain feasibility. In other words, the search direction has to point toward a region where some feasible solutions lie, and the search for the optimal step size has to be constrained to feasible points only. The focus of this section is on these topics, which are common to all *feasible direction* methods. The emphasis on this minimization approach stems from its importance in equilibrium analysis. Most current approaches to minimizing equivalent equilibrium programs can be cast as feasible direction methods. In particular, the convex combinations algorithm, which is the basic tool for many equilibrium analyses, is such a method. This algorithm is described in detail in Section 4.3.

The discussion in this section does not mention explicitly convergence criteria since those can be similar to the ones used for unconstrained mini-

†Note that in order to ensure that Newton's is a descent method, especially at the initial iterations, the move size has to be optimized. In such cases α_n will not equal 1.

‡These methods are also called variable metric approaches. The name stems from a particular interpretation of the technique used to define the search direction—it can be seen as an effort to change the scale of the axes of the vector $\mathbf{x}$ so that $z(\mathbf{x})$ is close to a spherical shape and the gradient points in the right direction.

mization (see Eqs. [4.8]). Note only that the minimum of a constrained program is not necessarily a stationary point of the objective function, and thus a gradient-based convergence criterion (such as Eq. [4.8b]) is not applicable.

As mentioned above, the two issues associated with maintaining feasibility are that the search direction has to point in a direction in which at least some feasible points lie and that, given such a search direction, the step size would not cause infeasibility. These two issues are described below in reverse order. First the problem of maintaining feasibility given the search direction and then the problem of finding a feasible direction.

Maintaining feasibility in a given direction. Consider the standard form of a minimization program (see Chapter 2)

$$\min z(\mathbf{x}) \qquad [4.13a]$$

subject to

$$\sum_i h_{ij} x_i \geq b_j \qquad \forall j \in \mathcal{J} \qquad [4.13b]$$

where the constraints are all linear (the nonnegativity constraints are included in Eqs. [4.13]). Assume further that the current solution is $\mathbf{x}^n$, a point that may lie on the boundary of the feasible region. In this case some of the constraints are binding, that is,

$$\sum_i h_{ij} x_i^n = b_j \qquad \forall j \in \mathcal{J}^n \qquad [4.14a]$$

where $\mathcal{J}^n$ is the index set of the binding constraints at the nth iteration. The other constraints are not binding, that is,

$$\sum_i h_{ij} x_i^n > b_j \qquad \forall j \notin \mathcal{J}^n \qquad [4.14b]$$

Once a feasible descent direction, $\mathbf{d}^n$, is obtained, the maximum move size that can be taken without violating any constraints should be determined. This maximum can then be used to bracket the search along the descent direction, so the new solution will necessarily be feasible. Note that only the constraints which are not currently binding (i.e., the constraints indexed by $j \notin \mathcal{J}^n$) might pose a problem in this regard (why?—see Problem 4.17) and thus only the inequality constraints (all $j \notin \mathcal{J}^n$) need be considered. For $\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha_n \mathbf{d}^n$ to be feasible, the (currently) nonbinding constraints must not be violated by the new solution, $\mathbf{x}^{n+1}$. In other words, the following must hold:

$$\sum_i h_{ij}(x_i^n + \alpha_n d_i^n) = \sum_i h_{ij} x_i^n + \alpha_n \sum_i h_{ij} d_i^n \geq b_j \qquad \forall j \notin \mathcal{J}^n$$

Obviously, if $\sum_i h_{il} d_i^n \geq 0$ for a given constraint l, there is no problem in satisfying the above mentioned requirement, since the current solution is feasible (i.e., $\sum_i h_{il} x_i^n \geq b_l$). (Intuitively, this means that the constraints for which $\sum_i h_{ij} d_i^n \geq 0$ will not be violated by the search direction.) Only in the cases

where $\sum_i h_{ij} d_i^n < 0$ would there be a possible violation. Thus the maximum possible value of α_n, $\alpha_n^{\max}$, is given oy

$$\alpha_n^{\max} = \min_{\forall j \in \bar{\mathcal{J}}^n} \frac{b_j - \sum_i h_{ij} x_i^n}{\sum_i h_{ij} d_i^n} \qquad [4.15]$$

where the set $\bar{\mathcal{J}}^n$ includes all the constraints that are nonbinding and for which $\sum_i h_{ij} d_i^n < 0$ at $\mathbf{x}^n$. In other words, the maximum move size is determined by the first constraint to be violated when moving along the descent direction. The optimal move size, α_n, can now be determined by solving

$$\min_{\alpha} z(\mathbf{x}^n + \alpha \mathbf{d}^n) \qquad [4.16a]$$

subject to

$$0 \le \alpha \le \alpha_n^{\max} \qquad [4.16b]$$

with any of the interval reduction methods of Section 4.1.

Figure 4.6 demonstrates the various types of constraints involved in this situation. It shows a feasible region bounded by five constraints (numbered 1 through 5). The current solution is on the boundary defined by constraint 1 and the descent direction points into the feasible region as shown in the figure. The nonbinding constraints are 2 through 5 but constraints 2 and 5 need not be considered since in the given search direction they will not be violated. (These are constraints for which $\sum_i h_{ij} d_i^n \ge 0$.) The move size is determined by checking which of the remaining constraints (3 or 4) will be violated first. In this case, it is constraint 4 that sets the limit, $\alpha_n^{\max}$, for the line search that follows.

Finding a feasible descent direction. Consider now the problem of determining a feasible descent direction. At some iterations this may not be a problem since a search direction identified by the algorithm used (which may

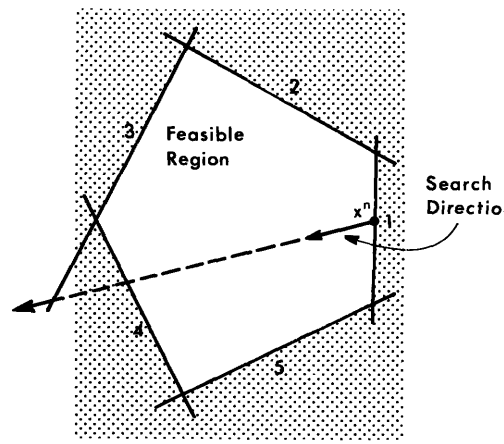


Figure 4.6 Maintaining feasibility along a search direction by bracketing the range of the line search.

be steepest descent, Newton, quasi-Newton, or any other method) may be feasible. In other cases, however, where the current solution is on the boundary of the feasible region, the best descent direction may be pointing outside that region. In such cases the direction has to be modified and the best (in some sense) *feasible* descent direction should be used to continue the search.

In looking for a feasible direction of descent from $\mathbf{x}^n$, only the binding constraints need be considered. Those constraints that are not currently binding would not be violated by a small (infinitesimal) move in any direction and should not, therefore, be considered. Given a candidate descent direction $\mathbf{d} = (\ldots, d_i, \ldots)$, the following must hold in order to ensure feasibility:

$$\sum_i h_{ij} x_i^n + \Delta\alpha \sum_i h_{ij} d_i \geq b_j \qquad \forall j \in \mathcal{J}^n \qquad [4.17a]$$

where $\Delta\alpha$ is a small move size. Since $\sum_i h_{ij} x_i^n = b_j$ for all $j \in \mathcal{J}^n$, expression [4.17a] reduces to

$$\sum_i h_{ij} d_i \geq 0 \qquad \forall j \in \mathcal{J}^n \qquad [4.17b]$$

Condition [4.17b] should be satisfied by any feasible direction.

The direction offering the steepest feasible descent can be found by solving the program

$$\min \nabla z(\mathbf{x}^n) \cdot \mathbf{d}^T \qquad [4.18a]$$

subject to

$$\sum_i h_{ij} d_i \geq 0 \qquad \forall j \in \mathcal{J}^n \qquad [4.18b]$$

$$\sum_i d_i^2 = 1 \qquad [4.18c]$$

The objective function [4.18a] is the cosine of the angle between the gradient and the descent direction.† This program then finds the descent direction that is closest to the opposite gradient direction (the cosine is minimized for an angle of 180°) yet satisfies the feasibility requirements. The last constraint normalizes the descent direction vector in order to ensure a unique solution. It is, unfortunately, a nonlinear constraint, meaning that program [4.18] may be difficult to solve.‡

The solution of this program is the steepest feasible descent vector. This vector defines the direction in which the search for the next solution, $\mathbf{x}^{n+1}$, should be conducted. (As explained above, this search may have to be bracketed first to ensure the feasibility of $\mathbf{x}^{n+1}$.)

The literature includes many other algorithms for determining a "good"

†Since $\mathbf{d}$ is constrained to be a unit vector, the objective function can be interpreted as the slope of $z(\mathbf{x})$ at $\mathbf{x}^n$, in the direction $\mathbf{d}$.

‡The references mentioned in Section 4.5 include some descriptions of approximate methods to determine the best feasible direction. For example, one method approximates constraint [4.18c] by the constraints $-1 \leq d_i \leq 1, \forall\, i$, which is linear and therefore easier to deal with.

feasible descent method. As an example of such methods, consider the gradient projection method, which obtains the next solution by "sliding" along the binding constraints at the current point. This strategy is, of course, followed only if the current descent direction (which is the direction opposite the gradient) points out of the feasible region (otherwise, there is no problem and the algorithm proceeds along the opposite gradient direction). The sliding direction is identified by projecting the opposite gradient direction on the binding constraints. The projection procedure itself includes some matrix manipulations involving the binding constraints. The move size is optimized along the descent direction, subject to the requirement that none of the (currently nonbinding) constraints is violated.

4.3 CONVEX COMBINATIONS METHOD

The convex combination algorithm was orginally suggested by Frank and Wolfe in 1956 as a procedure for solving quadratic programming problems with linear constraints and is known also as the Frank–Wolfe (FW) method. The method is especially useful for determining the equilibrium flows for transportation networks, which is why it is emphasized in this chapter.

Algorithm

The convex combinations algorithm is a feasible direction method. Unlike the general procedure for feasible direction methods described in the preceding section, the bounding of the move size does not require a separate step (such as Eq. [4.15]) with this algorithm. The bounding is accomplished as an integral part of the choice of descent direction. The direction-finding step of the convex combinations algorithm is explained in this section from two angles. First, this step is explained by using the logic of the general procedure for feasible direction methods outlined in the preceding section, and second, this step is presented as a linear approximation method.

To look at the direction-finding step of the convex combination algorithm as that of a feasible direction method, consider the convex program

$$\min z(\mathbf{x}) \qquad [4.19a]$$

subject to

$$\sum_i h_{ij} x_i \geq b_j \qquad \forall j \in \mathcal{J} \qquad [4.19b]$$

Assume now that at the nth iteration, the current solution is $\mathbf{x}^n$. Most feasible direction methods use information about the shape of the objective function in the vicinity of $\mathbf{x}^n$ to determine the descent direction. Consequently, the descent direction is based on the opposite gradient direction or the direction of the (locally) steepest feasible descent.

The convex combinations method selects the (feasible) descent direction not only on the basis of how steep each candidate direction is in the vicinity of $\mathbf{x}^n$, but also according to how far it is possible to move along this direction. If, for example, only a small move is feasible in a certain direction, effort might be wasted in actually performing an iteration based on such a move. Even though the direction may be associated with a steep *local* decrease in $z(\mathbf{x})$, the overall reduction in the objective function from $z(\mathbf{x}^n)$ to $z(\mathbf{x}^{n+1})$ may not be significant. On the other hand, a direction of change in which the local rate of improvement is modest but where the feasible region allows a considerable movement may achieve a larger reduction. The criterion for choosing directions in the convex combinations method is therefore based on the product of the rate of descent in the vicinity of $\mathbf{x}^n$ in a given direction and the length of the feasible region in that direction. This product, known as the "drop," is an upper bound to the possible reduction in the objective function value which can be achieved by moving in this direction. The algorithm uses the direction that maximizes the drop.

To find a descent direction, the algorithm looks at the entire feasible region for an auxiliary feasible solution, $\mathbf{y}^n = (y_1^n, \ldots, \ldots y_1^n)$, such that the direction from $\mathbf{x}^n$ to $\mathbf{y}^n$ provides the maximum drop. The direction from $\mathbf{x}^n$ to any feasible solution, $\mathbf{y}$, is the vector $(\mathbf{y} - \mathbf{x}^n)$ [or the unit vector $(\mathbf{y} - \mathbf{x}^n)/\| \mathbf{y} - \mathbf{x}^n \|$].† The slope of $z(\mathbf{x}^n)$ in the direction of $(\mathbf{y} - \mathbf{x}^n)$ is given by the projection of the opposite gradient $[-\nabla z(\mathbf{x}^n)]$ in this direction, that is,

$$-\nabla z(\mathbf{x}^n) \cdot \frac{(\mathbf{y} - \mathbf{x}^n)^T}{\| \mathbf{y} - \mathbf{x}^n \|}$$

The drop in the objective function in the direction $(\mathbf{y} - \mathbf{x}^n)$ is obtained by multiplying this slope by the distance from $\mathbf{x}^n$ to $\mathbf{y}$, $\| \mathbf{y} - \mathbf{x}^n \|$. The result is

$$-\nabla z(\mathbf{x}^n) \cdot (\mathbf{y} - \mathbf{x}^n)^T$$

This expression has to be maximized (in $\mathbf{y}$) subject to the feasibility of $\mathbf{y}$. Alternatively, the expression can be multiplied by (-1) and minimized, resulting in the program

$$\min \nabla z(\mathbf{x}^n) \cdot (\mathbf{y} - \mathbf{x}^n)^T = \sum_i \frac{\partial z(\mathbf{x}^n)}{\partial x_i} (y_i - x_i^n) \qquad [4.20a]$$

subject to

$$\sum_i h_{ij} y_i \geq b_j \qquad \forall j \in \mathcal{J} \qquad [4.20b]$$

where the constraints [4.20b] are equivalent to the original constraint set [4.19b] expressed in $\mathbf{y}$. The solution of program [4.20] is $\mathbf{y}^n$ and the descent direction is the vector pointing from $\mathbf{x}^n$ to $\mathbf{y}^n$, that is, $\mathbf{d}^n = (\mathbf{y}^n - \mathbf{x}^n)$, or in expanded form, $d_i^n = y_i^n - x_i^n$, $\forall\, i$.

†The notation $\| \mathbf{v} \|$ means the norm (or length) of the vector $\mathbf{v}$, that is, $\sqrt{\mathbf{v} \cdot \mathbf{v}} = \sqrt{\sum_i v_i^2}$.

Finding the descent direction, then, involves a minimization of a linear program, which is given in Eqs. [4.20]. As mentioned in the beginning of this section, this algorithmic step can also be motivated by looking at the convex combination method as a linear approximation method. This approach is based on finding a descent direction by minimizing a linear approximation to the function (instead of the function itself) at the current solution point. Minimizing this linearized function subject to a linear constraint set is a linear programming problem which has its solution at a corner of the feasible region (see Section 2.3). The line connecting the current solution points, $\mathbf{x}^n$, with the solution of the linearized problem (denoted $\mathbf{y}^n$) is the direction of search. This approach results in the same program as Eqs. [4.20]. To see this, let $z_L^n(\mathbf{y})$ denote the linear approximation of the value of the objective function at some point $\mathbf{y}$, based on its value at $\mathbf{x}^n$. This approximation is given by

$$z_L^n(\mathbf{y}) = z(\mathbf{x}^n) + \nabla z(\mathbf{x}^n) \cdot (\mathbf{y} - \mathbf{x}^n)^T \qquad [4.21]$$

This linear function of $\mathbf{y}$ has to be minimized subject to the constraints of the original problem, that is,

$$\min z_L^n(\mathbf{y}) = z(\mathbf{x}^n) + \nabla z(\mathbf{x}^n) \cdot (\mathbf{y} - \mathbf{x}^n)^T \qquad [4.22a]$$

subject to

$$\sum_i h_{ij} y_i \geq b_j \qquad \forall j \in \mathcal{J} \qquad [4.22b]$$

Note that, at the point $\mathbf{x} = \mathbf{x}^n$, the value of the objective function is constant and thus $z(\mathbf{x}^n)$ can be dropped from Eq. [4.22a]. The problem that remains is that of minimizing $\nabla z(\mathbf{x}^n) \cdot (\mathbf{y} - \mathbf{x}^n)^T$ subject to constraints [4.22b]. This program is identical to program [4.20], and its solution (again) can be used to construct a direction for the convex combinations algorithm. Thus both points of view can be used to motivate the algorithmic step of the convex combinations method.

The objective function of the linearized problem, can, however, be simplified even further by noting that $\nabla z(\mathbf{x}^n)$ is constant at $\mathbf{x}^n$ and the term $\nabla z(\mathbf{x}^n) \cdot (\mathbf{x}^n)^T$ can therefore be dropped from the linearized program (see Eq. [4.20a]), which can then be written as

$$\min z^n(\mathbf{y}) = \nabla z(\mathbf{x}^n) \cdot \mathbf{y}^T = \sum_i \left(\frac{\partial z(\mathbf{x}^n)}{\partial x_i} \right) y_i \qquad [4.23a]$$

subject to

$$\sum_i h_{ij} y_i \geq b_j \qquad \forall j \in \mathcal{J} \qquad [4.23b]$$

The variables of this linear program are $y_1, y_2, \ldots, y_I$ and the objective function coefficients are $\partial z(\mathbf{x}^n)/\partial x_1$, $\partial z(\mathbf{x}^n)/\partial x_2, \ldots, \partial z(\mathbf{x}^n)/\partial x_I$. These coefficients are the derivatives of the original objective function at $\mathbf{x}^n$, which are known at this point. The solution of program [4.23], $\mathbf{y}^n = (y_1^n, y_2^n, \ldots, y_I^n)$, is used to define the descent direction (i.e., $\mathbf{d}^n = \mathbf{y}^n - \mathbf{x}^n$), as explained above.

Once the descent direction is known, the other algorithmic steps involve the determination of the move size and a convergence test.

As in many other descent methods, the move size in the direction of $\mathbf{d}^n$ equals the distance to the point along $\mathbf{d}^n$ which minimizes $z(\mathbf{x})$. As mentioned before, the convex combinations methods does not require a special step to bracket the search for an optimal move size (such as Eq. [4.15]) in order to maintain feasibility. The new solution, $\mathbf{x}^{n+1}$, must lie between $\mathbf{x}^n$ and $\mathbf{y}^n$ (since $\mathbf{y}^n$, being a solution of a linear program, naturally lies at the boundary of the feasible region). In other words, the search for a descent direction automatically generates a bound for the line search by accounting for all the constraints (not only the binding ones) when the descent direction is determined. Since the search interval is bracketed, then, any one of the interval reduction methods would be suitable for the minimization of $z(\mathbf{x})$ along $\mathbf{d}^n = (\mathbf{y}^n - \mathbf{x}^n)$, that is, for solving

$$\min z[\mathbf{x}^n + \alpha(\mathbf{y}^n - \mathbf{x}^n)] \qquad [4.24a]$$

subject to

$$0 \leq \alpha \leq 1 \qquad [4.24b]$$

Once the optimal solution of this line search, α_n, is found, the next point can be generated with the usual step,

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha_n(\mathbf{y}^n - \mathbf{x}^n) \qquad [4.25]$$

Note that Eq. [4.25] can be written as $\mathbf{x}^n = (1 - \alpha_n)\mathbf{x}^n + \alpha_n \mathbf{y}^n$. The new solution is thus a *convex combination* (or weighted average) of $\mathbf{x}^n$ and $\mathbf{y}^n$.

The convergence criterion used in conjunction with the convex combinations algorithm can be any one of the criteria commonly used for feasible descent methods. Thus the convergence criterion can be based on the similarity of two successive solutions or the reduction of the objective function values between successive iterations.

Two general comments are in order here. First, as explained above, the convex combinations algorithm involves a minimization of a linear program as part of the direction-finding step. The convex combinations algorithm, then, is useful only in cases in which this linear program can be solved relatively easily. It is also useful when algorithms which are generally more efficient than the convex combination method (e.g., Newton and quasi-Newton methods) cannot be utilized due to the size of the problem. Many of the minimization problems described in this book possess both properties: they include a large number of variables, yet the linear program associated with the direction finding step can be solved with particular ease. The reason is that this program, in the cases dealt with in this text, has a special structure to it—that of a network—which can be exploited to facilitate the solution of the linear program.

The second comment concerns an interesting characteristic of the convex

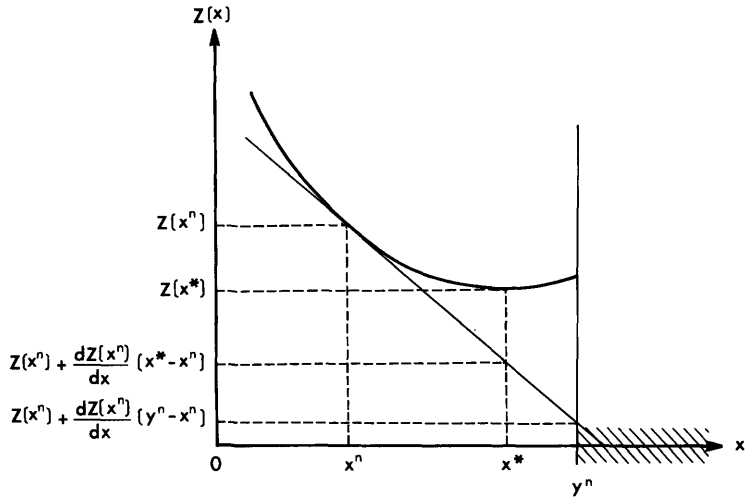


Figure 4.7 The bounds generated by a linear approximation to a function $z(x)$ at a point x^n.

combinations algorithm. In this method, the value of the linearized objective function at its solution, $z_L^n(\mathbf{y}^n)$, is a lower bound to the optimal value of the objective function itself, $z(\mathbf{x}^*)$. To see this, recall that by convexity,

$$z(\mathbf{x}^*) \geq z(\mathbf{x}^n) + \nabla z(\mathbf{x}^n) \cdot (\mathbf{x}^* - \mathbf{x}^n)^T = z_L^n(\mathbf{x}^*) \qquad [4.26a]$$

However, $\mathbf{y}^n$ minimizes $z_L^n(\mathbf{y})$ for any feasible $\mathbf{y}$. Consequently,

$$z_L^n(\mathbf{x}^*) \geq z_L^n(\mathbf{y}^n) = z(\mathbf{x}^n) + \nabla z(\mathbf{x}^n) \cdot (\mathbf{y}^n - \mathbf{x}^n)^T \qquad [4.26b]$$

Thus $z_L^n(\mathbf{y}^n)$ is a lower bound for $z(\mathbf{x}^*)$ at every iteration. Note that this lower bound generated by the convex combinations method is a straightforward extension of the lower bound mentioned in Section 4.1 for interval reduction methods. Figure 4.7 illustrates the relationships in Eqs. [4.26] for a one-dimensional convex function, $z(x)$. Note also that this lower bound is not monotonically increasing from iteration to iteration and therefore cannot be used in a straightforward fashion to create a convergence criterion (see Problem 4.21).

Given a current feasible solution, $\mathbf{x}^n$, the nth iteration of the convex combinations algorithm can be summarized as follows:

Step 1: *Direction finding.* Find $\mathbf{y}^n$ that solves the linear program [4.23].

Step 2: *Step-size determination.* Find α_n that solves

$$\min_{0 \leq \alpha \leq 1} z[\mathbf{x}^n + \alpha(\mathbf{y}^n - \mathbf{x}^n)]$$

Step 3: *Move.* Set $\mathbf{x}^{n+1} = \mathbf{x}^n + \alpha_n(\mathbf{y}^n - \mathbf{x}^n)$.

Step 4: *Convergence test.* If $z(\mathbf{x}^n) - z(\mathbf{x}^{n+1}) \leq \kappa$, stop.† Otherwise, let $n := n + 1$ and go to step 1.

Starting with a feasible solution, $\mathbf{x}^0$, the algorithm will converge after a finite number of iterations.

Example

Consider the following mathematical program:

$$\min z(\mathbf{x}) = x_1^2 + 2x_2^2 - 2x_1x_2 - 10x_2$$

subject to

$$0 \leq x_1 \leq 4$$

$$0 \leq x_2 \leq 6$$

Application of the convex combinations method to this program can be facilitated by some preliminary calculations. The gradient of $z(\mathbf{x})$ at $\mathbf{x}^n$ is given by

$$\nabla z(x_1, x_2) = [(2x_1^n - 2x_2^n), (4x_2^n - 2x_1^n - 10)]$$

The linear program at the nth iteration is thus

$$\min z^n(\mathbf{y}) = (2x_1^n - 2x_2^n)y_1 + (4x_2^n - 2x_1^n - 10)y_2$$

subject to

$$0 \leq y_1 \leq 4$$

$$0 \leq y_2 \leq 6$$

Since this problem is small, the optimal move size can be determined analytically by setting

$$\frac{dz[\mathbf{x}^n + \alpha(\mathbf{y}^n - \mathbf{x}^n)]}{d\alpha} = 0$$

Given $\mathbf{x}^n$ and $\mathbf{y}^n$, the optimal move size is, as the reader can verify,

$$\alpha_n = \frac{(y_1^n - x_1^n)(x_1^n - x_2^n) + (y_2^n - x_2^n)(2x_2^n - x_1^n - 5)}{2(y_1^n - x_1^n)(y_2^n - x_2^n) - 2(y_2^n - x_2^n)^2 - (y_1^n - x_1^n)^2}$$

This concludes the preliminary calculations and the algorithm can now be executed. The convergence criterion is set, for example, at 0.1, terminating the procedure when $z(\mathbf{x}^n) - z(\mathbf{x}^{n+1}) \leq 0.1$. The algorithmic steps are depicted in Figure 4.8 in relation to the feasible region and the objective function. These steps are explained below.

†Other convergence criteria could also be used.

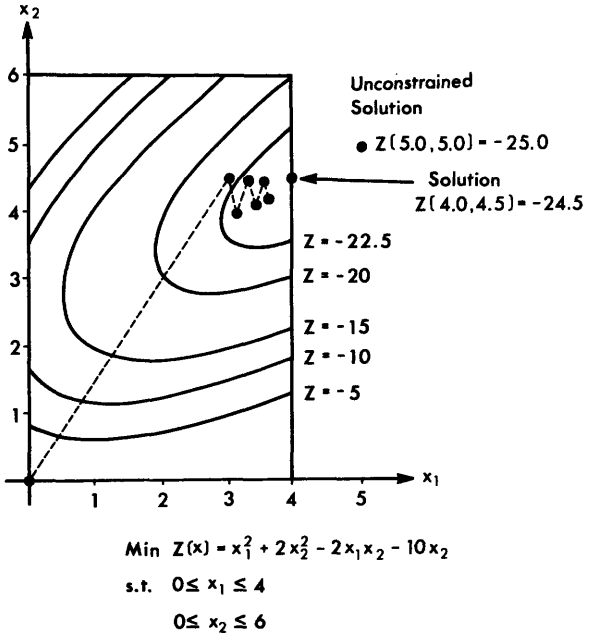


Figure 4.8 Convergence pattern of the convex combinations algorithm for a quadratic function with linear constraints.

Starting with the feasible solution $\mathbf{x}^1 = (0, 0)$ and $z(0, 0) = 0$, the first iteration goes as follows:

First iteration:

Step 1: The gradient is $\nabla z(0, 0) = (0, -10)$. The linear program is

$$\min \nabla z(\mathbf{x}^1) \cdot \mathbf{y} = -10y_2$$

subject to

$$0 \leq y_1 \leq 4$$
$$0 \leq y_2 \leq 6$$

Solution (by inspection of Figure 4.8): $\mathbf{y}^1 = (0, 6)$ or $\mathbf{y}^1 = (4, 6)$. Choose $\mathbf{y}^1 = (4, 6)$.

Step 2:

$$\alpha_1 = \frac{(4-0)(0-0) + (6-0)(2\cdot 0 - 0 - 5)}{2(4-0)(6-0) - 2(6-0)^2 - (4-0)^2} = 0.750$$

Step 3:

$$x_1^2 = 0 + 0.750(4 - 0) = 3$$
$$x_2^2 = 0 + 0.750(6 - 0) = 4.5$$

Step 4:

$$z(3, 4.5) = -22.5$$
$$z(\mathbf{x}^1) - z(\mathbf{x}^2) = 0 - (-22.5) = 22.5$$

Second iteration:

Step 1: $\nabla z(3, 4.5) = (-3.0, 2.0)$. The linear program is

$$\min \nabla z(\mathbf{x}^2) \cdot \mathbf{y} = -3y_1 + 2y_2$$

subject to

$$0 \le y_1 \le 4$$
$$0 \le y_2 \le 6$$

Solution (by inspection): $\mathbf{y}^2 = (4, 0)$.

Step 2: $\alpha_2 = 0.119$.
Step 3:

$$x_1^3 = 3.0 + 0.119(4.0 - 3.0) = 3.119$$
$$x_2^3 = 4.5 + 0.119(0.0 - 4.5) = 3.966$$

Step 4:

$$z(3.119, 3.966) = -23.213$$
$$z(\mathbf{x}^2) - z(\mathbf{x}^3) = 0.713$$

Obviously, the convergence criterion is not met and the algorithm continues. Table 4.2 shows the sequence of iterations of the convex combinations algorithm for this example. As evident in the table, the convergence criterion is

TABLE 4.2 Iterations of the Convex Combinations Algorithm (see Figure 4.8)

n	$\nabla z(\mathbf{x}^n)$	$\mathbf{y}^n$	$z_L^n(\mathbf{y}^n)$	α_n	$\mathbf{x}^{n+1}$	$z(\mathbf{x}^{n+1})$	$z(\mathbf{x}^n) - z(\mathbf{x}^{n+1})$
0					(0.000, 0.000)		
1	(0.000, −10.000)	(4, 6)	−60.000	0.750	(3.000, 4.500)	−22.500	22.500
2	(−3.000, 2.000)	(4, 0)	−35.213	0.119	(3.119, 3.969)	−23.213	0.713
3	(−1.693, −0.376)	(4, 6)	−24.211	0.206	(3.301, 4.385)	−23.446	0.233
4	(−2.169, 0.939)	(4, 0)	−29.257	0.063	(3.344, 4.111)	−23.622	0.176
5	(−1.833, −0.295)	(4, 6)	−25.196	0.144	(3.439, 4.383)	−23.728	0.106
6	(−1.889, 0.656)	(4, 0)	−27.752	0.045	(3.464, 4.186)	−23.816	0.089

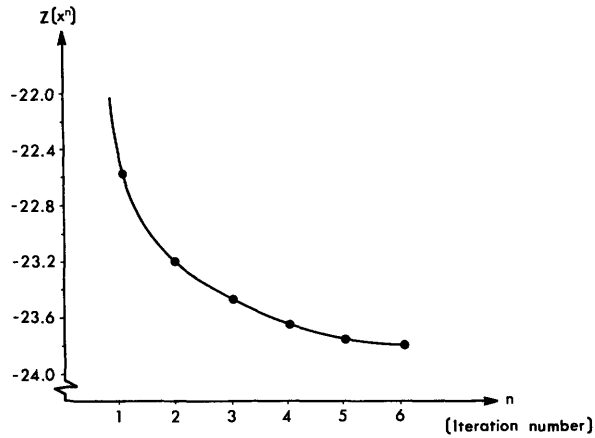


Figure 4.9 The asymptotic reduction in the objective function value as the iterations of the convex combinations algorithm proceed.

met after six iterations and the algorithm terminates. Note that, as mentioned before, the lower bound generated at each step, $z_L^n(\mathbf{y}^n)$, is not monotonically increasing.

Figure 4.8 depicts the pattern of convergence toward the minimum. This convergence pattern is demonstrated in Figure 4.9 in terms of the reduction in the value of the objective function from iteration to iteration. The asymptotic pattern shown in this figure is typical of the convex combinations algorithm. The marginal contribution of each successive iteration becomes smaller and smaller as the algorithm proceeds (this property was the basis for the convergence criterion used in the example).

4.4 SUMMARY

This chapter reviews numerical minimization methods. It begins with a study of line search algorithms, that is, methods for minimizing a function of a single argument. The presentation covers two groups of methods: interval reduction algorithms and quadratic fit approaches. Quadratic fit methods are usually faster if the objective function is relatively smooth. Interval reduction methods are naturally useful in minimizing a function over a given interval.

The review of minimization algorithms for multivariable programs deals separately with unconstrained and constrained minimizations. Most of the methods available for minimizing an unconstrained function of several variables are based on choosing a descent direction at a current solution and moving along this direction to the next solution point. The steepest descent method uses the opposite gradient as the descent direction and then chooses the next solution at the point that minimizes the objective function along this direction. This method often requires a large number of iterations to converge

due to its "zigzagging" tendency. Many of the algorithms that require smaller number of iterations cannot be used for the problems covered in this book since they require computation of the Hessian (or an approximation of it). Such requirements are computationally prohibitive for large problems of the type encountered in traffic assignment analyses.

When minimizing constrained functions, the choice of descent direction and move size cannot be governed by considering only the objective function, since feasibility has to be maintained. The descent vector has to point in a direction that includes at least some feasible points. Similarly, the move size has to be constrained so that the next point is within the feasible region.

The last section of this chapter describes the convex combinations algorithm. This algorithm is a feasible descent method that is based on a linear approximation of the objective function at every iteration. Consequently, a linear program has to be solved at every iteration in order to determine the descent direction. The move size is determined by minimizing the objective function along the descent direction. This algorithm is efficient when the linear program can be solved easily.

Convergence of any of the algorithms described in this chapter can be measured by the similarity of successive solutions or by the rate of reduction in the objective function from one iteration to the next.

4.5 ANNOTATED REFERENCES

Much of the material covered in this chapter can be found in the references mentioned at the end of Chapter 2. In particular, the reader who is interested in convergence proofs should consult the texts of Zangwill (1969) and Luenberger (1973). The former includes many original contributions, and the latter summarizes the global convergence conditions for the various algorithms mentioned in this text. Good descriptions of feasible direction methods can be found in Simmons (1975) (including several methods for expediting convergence of steepest descent algorithms by modifying the search direction every few steps) and Wagner (1975). More detailed descriptions of the Fibonacci method can be found in the texts of Wilde (1964) and Wismer and Chattergy (1978). The convex combinations method first appeared in the literature in the paper by Frank and Wolfe (1956). A general proof of convergence for this method is offered by Zangwill in the aforementioned text.

4.6 PROBLEMS

4.1. Derive formula [4.1] for interval reduction methods.

4.2. Show that if an interval reduction method with a constant reduction ratio has to satisfy Eq. [4.2], then the reduction ratio is the golden section. Show also that $r = \frac{1}{2}(\sqrt{5} - 1)$ leads to Eq. [4.2].

4.3. Use the golden section method to find the minimum of the function $z(x) = 3x^2 - 4x + 1$ within $x^* \pm 0.1$ in the interval [0, 2]. How many function evaluations are needed? How many function evaluations would be needed to find the solution within $x^* \pm 0.001$?

4.4. Use the golden section to find, within ± 0.15 of the optimum, the value of x that minimizes the function

$$z(x) = \max\left\{\left(2 - \frac{x}{3}\right), (x-3)^2, \frac{x}{5}\right\}$$

such that $0 \leq x \leq 8$.

4.5. Using a mathematical programming text, describe the Fibonacci search algorithm. Compare its mechanism and its efficiency to that of the golden section method and the bisection method.

4.6. Use the bisection method to find the minimum of $z(x) = 3x^2 - 4x + 1$ over the interval [0, 2]. Determine the optimal value of x within 5% of the initial interval. How many function evaluations are needed to get within $x^* \pm 0.001$?

4.7. Find the minimum of $z(x) = 2x^3 + 3x^2 - 12x + 5$ in the interval [0, 4] using:
- **(a)** Newton's method (use $x^0 = 0$).
- **(b)** The false position method (use $x^0 = 0$, $x^1 = 0.5$).
- **(c)** The quadratic fit method (use $x_1 = 0$, $x_2 = 0.5$, $x_3 = 1$).

***4.8.** **(a)** Show why the curve-fitting algorithms described in the text are not truly descent methods. How can the quadratic fit method be modified to be a descent method.
- **(b)** Show why Newton's method for minimizing multivariate functions (Eq. [4.12]) is not a descent method and how should it be modified to be a descent method.

4.9. Develop the algorithmic iteration for the quadratic fit method (i.e., derive formula [4.6]).

4.10. Write a computer subroutine that accepts a function, an interval, and a convergence criterion and performs a golden section search.

4.11. Write a computer subroutine that executes a line search using the bisection method.

4.12. Suggest a procedure for amending an interval reduction method so that it can be used for the line search phase when minimizing an unconstrained function.

***4.13.** Prove that the steepest descent algorithm is a descent method.

4.14. Using the steepest descent method, find the minimum of the quadratic function

$$z(x_1, x_2) = (x_1 - 2)^2 + 4(x_2 - 3)^2$$

Starting from $\mathbf{x}^0 = (0, 0)$ plot the algorithmic moves and verify the zigzag property of the algorithm.

4.15. Perform four iterations of the steepest descent method to solve

$$\min z(\mathbf{x}) = x_1^2 + 2x_2^2 + x_3^2 - 2x_1 - 3x_2 - 2x_3$$

Start at the point $x = (0, 0, 0)$.

***4.16.** Find an approximate analytical expression for the optimal step size along the negative gradient direction for convex functions. [*Hint:* Use a second-order expansion of $z(x)$ about $z(\mathbf{x}^n)$.]

4.17. Consider the mechanics of feasible descent algorithms.

(a) Why do only the nonbinding constraints need be considered in maintaining feasibility along the (feasible) descent direction?

(b) Why do only the binding constraints need be considered in finding the best feasible descent direction?

4.18. Show that if $z(\mathbf{x})$ is a convex function of $\mathbf{x}$, $z(\mathbf{x}^n + \alpha \mathbf{d}^n)$ is a convex function of α.

4.19. Show that the direction used by the convex combinations method is always a descent direction.

4.20. Use the convex combinations algorithm to solve the program:

$$\min f(x_1, x_2) = 4(x_1 - 10)^2 + (x_2 - 4)^2$$

subject to

$$x_1 - x_2 \leq 10$$

$$\tfrac{1}{5}x_1 - x_2 \geq 3$$

$$x_1 \geq 0$$

Use a plot of the feasible region to solve the linearized problem by inspection. Plot the algorithmic steps in this space.

4.21. The convex combinations algorithm generates a lower bound to the objective function at every iteration. Use this bound to develop a convergence criterion for this algorithm.

5

Solving for User Equilibrium

The problem of finding the user equilibrium over a transportation network was introduced in Chapter 1. It involves the assignment of O–D flows to the network links so that travel time on all used paths for any O–D pair equals the minimum travel time between the origin and destination. This problem was formulated as a mathematical program (the equivalent UE minimization) in Chapter 3. Chapter 4 reviewed a general class of algorithms for solving mathematical programs and focused on the convex combinations method. This chapter describes the application of this method to the solution of the user-equilibrium minimization program.

Before this application of the algorithm is explained, however, two of the most common heuristic methods for finding the user-equilibrium flow pattern are outlined in Section 5.1. These approaches had been extensively used before solution algorithms for the UE program were developed and are still widely applied today. Section 5.2 then demonstrates how the convex combinations algorithm can be applied to the equivalent user-equilibrium program. The last section of this chapter concentrates on an algorithm for finding the minimum-travel-time path between two network nodes. The determination of minimum paths is a major component of the algorithmic solution of the UE program (and of the heuristic algorithms mentioned in Section 5.1).

5.1 HEURISTIC EQUILIBRATION TECHNIQUES

The heuristic approaches to the user-equilibrium problem reviewed in this section include *capacity restraint* methods and *incremental assignment* techniques. At the core of these methods lies the *network loading* mechanism.

Network loading is the process of assigning the O–D entries to the network for specific (constant) link travel times. The process follows the route-choice criterion, which underlies any traffic assignment model. As argued in Chapter 1, the UE flow pattern is the result of each motorist using the minimum travel time path between his origin and his destination. Accordingly, the network loading mechanism used in all the algorithms designed to solve the UE problem assigns each O–D flow to the shortest travel time path connecting this O–D pair. As mentioned in Chapter 3 (see discussion following Eqs. [3.28]), this procedure is known as the "all-or-nothing" assignment.

In the all-or-nothing procedure, each O–D pair r–s is examined in turn and the O–D flow, q_{rs}, is assigned to every link that is on the minimum-travel-time path connecting origin r to destination s. All other paths connecting this O–D pair are not assigned any flow. During this process, the link travel times are assumed to be fixed (i.e., not flow dependent) at some value. Accordingly, if there is a performance curve, $t_a(x_a)$, corresponding to each link in the network, any mention of all-or-nothing assignment (or any other network loading mechanism) should be made in conjunction with a specific travel time. For example, the all-or-nothing assignment can be applied to an empty network, that is, using the travel times $t_a = t_a(0)$ for every link, a. The only computational difficulty with the all-or-nothing procedure involves the identification of the minimum-travel-time paths connecting each O–D pair. This, however, is a well-researched problem in graph theory and several algorithms for its solution are readily available. Section 5.3 describes one of the more efficient of these procedures, one that is particularly applicable to transportation networks.

Many of the early urban transportation studies have used the all-or-nothing procedure (based on empty network times) as the traffic assignment procedure. This assignment method does not recognize, of course, the dependence between flows and travel time, thus, in effect, it ignores the equilibrium problem altogether.

Capacity Restraint

In an attempt to capture the equilibrium nature of the traffic assignment problem, transportation planners have devised an iterative scheme known as capacity restraint. This method involves a repetitive all-or-nothing assignment in which the travel times resulting from the previous assignment are used in the current iteration. The algorithm can be summarized as follows:

Step 0: *Initialization.* Perform all-or-nothing assignment based on $t_a^0 = t_a(0), \forall\ a$. Obtain a set of link flows $\{x_a^0\}$. Set iteration counter $n := 1$.

Step 1: *Update.* Set $t_a^n = t_a(x_a^{n-1}), \forall\ a$.

Step 2: *Network loading.* Assign all trips to the network using all-or-nothing based on travel times $\{t_a^n\}$. This yields a set of link flows $\{x_a^n\}$.

TABLE 5.1 Capacity Restraint Algorithm Applied to the Network in Figure 5.1

Iteration Number	Algorithmic Step	Link 1	Link 2	Link 3
0	Initialization	$t_1^0 = 10$ $x_1^0 = 10$	$t_2^0 = 20$ $x_2^0 = 0$	$t_3^0 = 25$ $x_3^0 = 0$
1	Update Loading	$t_1^1 = 947$ $x_1^1 = 0$	$t_2^1 = 20$ $x_2^1 = 10$	$t_3^1 = 25$ $x_3^1 = 0$
2	Update Loading	$t_1^2 = 10$ $x_1^2 = 10$	$t_2^2 = 137$ $x_2^2 = 0$	$t_3^2 = 25$ $x_3^2 = 0$
3	Update Loading	$t_1^3 = 947$ $x_1^3 = 0$	$t_2^3 = 20$ $x_2^3 = 10$	$t_3^3 = 25$ $x_3^3 = 0$
		⋮	⋮	⋮

Step 3: *Convergence test.* If $\max_a \{|x_a^n - x_a^{n-1}|\} \leq \kappa$,† stop (the current set of link flows is the solution). Otherwise, set $n := n + 1$ and go to step 1.

Table 5.1 demonstrates an application of this procedure to the example network depicted in Figure 5.1, shown on page 114. Starting with travel times corresponding to an empty network, the initial solution is determined and the iterations follow the above-mentioned algorithm. Note that the algorithm does not converge, as the flow "flip-flops" between links 1 and 2, whereas link 3 does not get loaded at all.

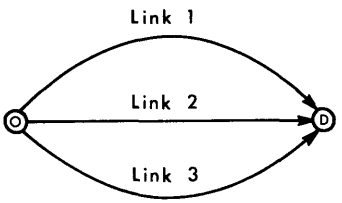
To remedy this situation, the algorithm can be modified as follows. First, instead of using the travel time obtained in the previous iteration for the new loading, a combination of the last two travel times obtained is used. This introduces a "smoothing" effect. Second, the failure to converge is recognized explicitly and the algorithm is terminated after a given number of iterations, N. The equilibrium flow pattern is then taken to be the average flow for each link over the last four iterations (obviously, N should never be less than 4). This form of the algorithm was adopted by the U.S. Federal Highway Administration (FHWA) as part of its transportation planning package. The steps of the modified capacity restraint algorithm (using weights of 0.75 and 0.25 for the averaging process) are as follows:

Step 0: *Initialization.* Perform an all-or-nothing assignment based on $t_a^0 = t_a(0)$. Obtain $\{x_a^0\}$. Set $n := 1$.

Step 1: *Update.* Set $\tau_a^n = t_a(x_a^{n-1})$, $\forall\ a$.

Step 2: *Smoothing.* Set $t_a^n = 0.75t_a^{n-1} + 0.25\tau_a^n$, $\forall\ a$.

†This convergence test is based on the maximum change in link flow between successive iterations. Other criteria can also be used.



$t_1 = 10[1+0.15[\frac{x_1}{2}]^4]$ time units

$t_2 = 20[1+0.15[\frac{x_2}{4}]^4]$ time units

$t_3 = 25[1+0.15[\frac{x_3}{3}]^4]$ time units

$x_1 + x_2 + x_3 = 10$ flow units

Figure 5.1 Network example, with three links and one O–D pair.

Step 3: *Network loading.* Perform all-or-nothing assignment based on travel times $\{t_a^n\}$. This yields $\{x_a^n\}$.

Step 4: *Stopping rule.* If $n = N$, go to step 5. Otherwise, set $n := n + 1$ and go to step 1.

Step 5: *Averaging.* Set $x_a^* = \frac{1}{4}\sum_{i=0}^{3} x_a^{n-i}\ \forall\ a$ and stop. ($\{x_a^*\}$ are the link flows at equilibrium.)

The smoothing is accomplished by creating a temporary link-travel-time variable, τ_a^n, which is not used as the travel time for the next iteration (see step 1). Instead, it is averaged together with the travel time used in the last iteration, t_a^{n-1}, to obtain the link travel time for the current iteration, t_a^n. This is done in step 2. This algorithm differs from the original capacity restraint algorithm in the addition of the smoothing step and the averaging step. Note that step 4 is called a stopping rule rather than a convergence test, since there is no reason to expect this algorithm to converge to the equilibrium solution, in spite of these changes.

An application of this algorithm to the network example of Figure 5.1 is demonstrated in Table 5.2. Note that it produces a solution that is not an equilibrium flow pattern since, even though all paths are used, t_1^* is substantially different from t_2^* and t_3^*. The table shows three iterations in addition to the initialization, meaning that four minimum-path computations are represented. For large networks, this is the main computational burden, and thus traffic assignment algorithms should be compared in terms of efficiency for a given number of minimum-path computations (i.e., a given number of all-or-nothing assignments).

TABLE 5.2 Modified Restraint Algorithm Applied to the Network in Figure 5.1

Iteration Number	Algorithmic Step	Link 1	Link 2	Link 3
0	Initialization	$t_1^0 = 10$	$t_2^0 = 20$	$t_3^0 = 25$
		$x_1^0 = 10$	$x_2^0 = 0$	$x_3^0 = 0$
1	Update	$\tau_1^1 = 947$	$\tau_2^1 = 20$	$\tau_3^1 = 25$
	Smoothing	$t_1^1 = 244$	$t_2^1 = 20$	$t_3^1 = 25$
	Loading	$x_1^1 = 0$	$x_1^1 = 10$	$x_3^1 = 0$
2	Update	$\tau_1^2 = 10$	$\tau_2^2 = 137$	$\tau_3^2 = 25$
	Smoothing	$t_2^2 = 186$	$t_2^2 = 49$	$t_3^2 = 25$
	Loading	$x_1^2 = 0$	$x_2^2 = 0$	$x_3^2 = 10$
3	Update	$\tau_1^3 = 10$	$\tau_2^3 = 20$	$\tau_3^3 = 488$
	Smoothing	$t_1^3 = 142$	$t_2^3 = 42$	$t_3^3 = 141$
	Loading	$x_1^3 = 0$	$x_2^3 = 10$	$x_3^3 = 0$
Average		$x_1^* = 2.5$	$x_2^* = 5.0$	$x_3^* = 2.5$
		$t_1^* = 13.7$	$t_2^* = 27.3$	$t_3^* = 26.8$

Incremental Assignment

Another heuristic method for attaining the user-equilibrium solution assigns a portion of the origin–destination matrix at each iteration. The travel times are then updated and an additional portion of the O–D matrix is loaded onto the network. In this manner, the general shape of the link performance functions can be "traced" with the successive assignments. This procedure, which is known as incremental assignment, is outlined below. (In this description, w_a^n denotes the flow on link a resulting from the assignment of the nth increment of the O–D matrix onto the network.)

Step 0: *Preliminaries.* Divide each origin–destination entry into N equal portions (i.e. set $q_{rs}^n = q_{rs}/N$). Set $n := 1$ and $x_a^0 = 0$, $\forall\ a$.

Step 1: *Update.* Set $t_a^n = t_a(x_a^{n-1})$, $\forall\ a$.

Step 2: *Incremental loading.* Perform all-or-nothing assignment based on $\{t_a^n\}$, but using only the trip rates q_{rs}^n for each O–D pair. This yields a flow pattern $\{w_a^n\}$.

Step 3: *Flow summation.* Set $x_a^n = x_a^{n-1} + w_a^n$, $\forall\ a$.

Step 4: *Stopping rule.* If $n = N$, stop (the current set of link flows is the solution); otherwise, set $n := n + 1$ and go to step 1.

In some versions of this algorithm, the incremental all-or-nothing procedure in step 2 is modified and origin–destination pairs are selected in random order,

TABLE 5.3 Incremental Assignment Algorithm Applied to the Network in Figure 5.1

Iteration (Increment)	Algorithmic Step	Link 1	Link 2	Link 3
1	Update	$t_1^1 = 10$	$t_2^1 = 20$	$t_3^1 = 25$
	Incremental loading	$w_1^1 = 2.5$	$w_2^1 = 0$	$w_3^1 = 0$
	Summation	$x_1^1 = 2.5$	$x_2^1 = 0$	$x_3^1 = 0$
2	Update	$t_1^2 = 14$	$t_2^2 = 20$	$t_3^2 = 25$
	Incremental loading	$w_1^2 = 2.5$	$w_2^2 = 0$	$w_3^2 = 0$
	Summation	$x_1^2 = 50$	$x_2^2 = 0$	$x_3^2 = 0$
3	Update	$t_1^3 = 69$	$t_2^3 = 20$	$t_3^3 = 25$
	Incremental loading	$w_1^3 = 0$	$w_2^3 = 2.5$	$w_3^3 = 0$
	Summation	$x_1^3 = 5.0$	$x_2^3 = 2.5$	$x_3^3 = 0$
4	Update	$t_1^4 = 69$	$t_2^4 = 20.5$	$t_3^4 = 25$
	Incremental loading	$w_1^4 = 0$	$w_2^4 = 2.5$	$w_3^4 = 0$
	Summation	$x_1^4 = 5.0$	$x_2^4 = 5.0$	$x_3^4 = 0$
	Travel time at convergence	$t_1^* = 69$	$t_2^* = 27.3$	$t_3^* = 25$

with a flow summation phase (as in step 3) and travel-time update (as in step 4) following each partial assignment (i.e., after each O–D entry is loaded).

Table 5.3 demonstrates the application of this algorithm to the network example of Figure 5.1. The flow pattern resulting from an incremental assignment with four increments is $\mathbf{x} = (5, 5, 0)$. The last row in the table $[t_a^* = t_a(x_a^4)$ for $a = 1, 2, 3]$ is provided so that the reader can judge the closeness of the solution to an equilibrium flow pattern. As evident from this table the two used paths (links 1 and 2) do not exhibit equal travel times. Furthermore, these travel times are higher than that of the unused path (link 3).

In conclusion, it is clear that the heuristic methods reviewed in this section either do not converge or produce a set of flows that is not in agreement with the user-equilibrium criterion. It may be reasonable to believe, though, that in general, as the number of increments grows, the incremental assignment algorithm may generate a flow pattern closer to the user-equilibrium condition. A very large number of increments (associated with a considerable computational effort) may be required, however, and even then the method will not always produce the user-equilibrium flow pattern.

5.2 APPLYING THE CONVEX COMBINATIONS METHOD

As the examples of the preceding section demonstrate, the heuristic methods discussed thus far may not converge to the equilibrium solution. It is this failure to converge which motivates the overall approach taken in this book—

formulating the user-equilibrium problem as a mathematical program and solving this program. Chapter 3 dealt with the program formulation and showed that the flow pattern which minimizes the UE equivalent program is, in fact, a user-equilibrium solution. This program includes a convex (nonlinear) objective function and a linear constraint set. Chapter 4 reviewed various feasible direction methods for solving such programs, emphasizing the convex combinations method. This method is especially suitable for solving the equivalent UE program since the direction-finding step can be executed relatively efficiently. This step involves the solution of a linear program, which, in the case of the UE program, has a special structure that simplifies its solution.

Using the notation introduced in Chapter 3, the UE objective function is given by

$$\min z(\mathbf{x}) = \sum_a \int_0^{x_a} t_a(\omega)\, d\omega \qquad [5.1a]$$

subject to

$$\sum_k f_k^{rs} = q_{rs} \qquad \forall\, r, s \qquad [5.1b]$$

$$f_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [5.1c]$$

where x_a is the flow on link a, f_k^{rs} is the flow on path k connecting origin r with destination s, and the incidence relationships $x_a = \sum_{rs} \sum_k f_k^{rs} \delta_{a,k}^{rs}$, $\forall\, a$, hold.

Applying the convex combinations algorithm to the minimization of the UE program requires, at every iteration, a solution of the linear program (LP)

$$\min z^n(\mathbf{y}) = \nabla z(\mathbf{x}^n) \cdot \mathbf{y}^T = \sum_a \frac{\partial z(\mathbf{x}^n)}{\partial x_a} y_a \qquad [5.2]$$

over all feasible values of $\mathbf{y} = (\ldots, y_a, \ldots)$ (see Eqs. [4.23]). The gradient of $z(\mathbf{x})$ with respect to the link flows at the nth iteration is the link travel-time vector, since $\partial z(\mathbf{x}^n)/\partial x_a = t_a^n$ (see Eq. [3.12a]). The LP objective function at the nth iteration (given by Eq. [5.2]) thus becomes

$$\min z^n(\mathbf{y}) = \sum_a t_a^n y_a \qquad [5.3a]$$

subject to

$$\sum_k g_k^{rs} = q_{rs} \qquad \forall\, r, s \qquad [5.3b]$$

$$g_k^{rs} \geq 0 \qquad \forall\, k, r, s \qquad [5.3c]$$

where $y_a = \sum_{rs} \sum_k g_k^{rs} \delta_{a,k}^{rs}$, $\forall\, a$, and $t_a^n = t_a(x_a^n)$. In this linear program, y_a is the auxiliary variable representing the flow on link a, while g_k^{rs} is the auxiliary flow variable for path k connecting O–D pair r–s. This program calls for minimizing the total travel time over a network with fixed (not flow-dependent) travel times, t_a^n. The total travel time spent in the network will be minimized by assigning all motorists to the shortest travel-time path connecting their

origin to their destination (see Section 3.5). Such an assignment is performed by the all-or-nothing network loading procedure mentioned earlier. Consequently, the program in Eqs. [5.3] is known as the all-or-nothing program. The core of the all-or-nothing procedure is the determination of the shortest paths between all origins and all destinations. Section 5.3 outlines an efficient method for finding these paths between all the network nodes. At this point, it is sufficient to mention that program [5.3] can be solved efficiently.

To see that the solution of program [5.3] does not involve more than an all-or-nothing assignment, the linearization step of the convex combinations method can be derived by taking the gradient of objective function [5.1a] with respect to path flows (instead of link flow, as in [5.2]). The linearized program then becomes

$$\min z^n(\mathbf{g}) = \nabla_f z[\mathbf{x}(\mathbf{f}^n)] \cdot \mathbf{g}^T = \sum_{rs} \sum_{k} c_k^{rsn} g_k^{rs} \qquad [5.4a]$$

subject to

$$\sum_{rs} \sum_{k} g_k^{rs} = q_{rs} \qquad \forall\ r, s \qquad [5.4b]$$

$$g_k^{rs} \geq 0 \qquad \forall\ k, r, s \qquad [5.4c]$$

where c_k^{rsn} is the travel time on path k connecting r and s, at the nth iteration of the algorithm. This program can be decomposed by O–D pair since the path travel times are fixed. The resulting subproblem for pair r–s is given by

$$\min z^n(\mathbf{g}^{rs}) = \sum_{k} c_k^{rs} g_k^{rs} \qquad [5.5a]$$

subject to

$$\sum_{k} g_k^{rs} = q_{rs} \qquad [5.5b]$$

$$g_k^{rs} \geq 0 \qquad \forall\ k \qquad [5.5c]$$

This program is obviously minimized by finding the path, m, with the smallest travel time among all paths connecting r and s, and assigning all the flow to it. In other words,

$$g_m^{rs} = q_{rs} \qquad \text{if } c_m^{rs} \leq c_k^{rs} \quad \forall\ k \qquad [5.6a]$$

and

$$g_k^{rs} = 0 \qquad \text{for all other paths} \qquad [5.6b]$$

In case two or more paths are tied for the minimum, any one of them can be chosen for flow assignment.

Once the path flows $\{g_k^{rsn}\}$ are found, the link flows can be calculated by using the incidence relationships, that is,

$$y_a^n = \sum_{rs} \sum_{k} g_k^{rsn} \delta_{a,k}^{rs} \qquad \forall\ a$$

This solution defines the descent direction $\mathbf{d}^n = \mathbf{y}^n - \mathbf{x}^n$.

The remaining algorithmic steps of the convex combinations method are similar to those outlined in Section 4.3. The initial solution can be usually determined by applying an all-or-nothing network loading procedure to an empty network, in a way similar to the initialization of the capacity restraint methods discussed in Section 5.1.

The line search for the optimal move size can be performed with any of the interval reduction methods, but the bisection method (Bolzano search) may be particularly applicable. The reason is that the derivative of the objective function $z[\mathbf{x}^n + \alpha(\mathbf{y}^n - \mathbf{x}^n)]$ with respect to α is given by

$$\frac{\partial}{\partial \alpha} z[\mathbf{x}^n + \alpha(\mathbf{y}^n - \mathbf{x}^n)] = \sum_a (y_a^n - x_a^n) t_a[x_a^n + \alpha(y_a^n - x_a^n)] \qquad [5.7]$$

which can be easily calculated for any value of α.

The stopping criterion for solving the UE program could be based on the values of the objective function. As mentioned in Section 3.1, however, this function is merely a mathematical construct that lacks behavioral or economic meaning. Consequently, the convergence criterion should be based on the relevant figures of merit, which in this case consist of the flows and the travel times. A possible measure of the closeness of a particular solution to equilibrium is the similarity of successive O–D travel times. Letting u_{rs}^n denote the minimum path travel time between O–D pair r–s at the nth iteration, the algorithm can terminate if, for example,

$$\sum_{rs} \frac{|u_{rs}^n - u_{rs}^{n-1}|}{u_{rs}^n} \leq \kappa \qquad [5.8a]$$

Alternatively, a criterion that is based on the change in flows can be used. For example, the algorithm can terminate if

$$\frac{\sqrt{\sum_a (x_a^{n+1} - x_a^n)^2}}{\sum_a x_a^n} \leq \kappa' \qquad [5.8b]$$

Other convergence criteria can be used as well. The algorithm itself, when applied to the solution of the UE problem, can be summarized as follows:

Step 0: *Initialization.* Perform all-or-nothing assignment based on $t_a = t_a(0)$, $\forall\ a$. This yields $\{x_a^1\}$. Set counter $n := 1$.

Step 1: *Update.* Set $t_a^n = t_a(x_a^n)$, $\forall\ a$.

Step 2: *Direction finding.* Perform all-or-nothing assignment based on $\{t_a^n\}$. This yields a set of (auxiliary) flows $\{y_a^n\}$.

Step 3: *Line search.* Find α_n that solves

$$\min_{0 \leq \alpha \leq 1} \sum_a \int_0^{x_a^n + \alpha(y_a^n - x_a^n)} t_a(\omega)\, d\omega$$

Step 4: *Move.* Set $x_a^{n+1} = x_a^n + \alpha_n(y_a^n - x_a^n)$, $\forall\, a$.

Step 5: *Convergence test.* If a convergence criterion is met, stop (the current solution, $\{x_a^{n+1}\}$, is the set of equilibrium link flows); otherwise, set $n := n + 1$ and go to step 1.

Note the similarity between this algorithm and the capacity restraint method reviewed in Section 5.1. In fact, if the move size, α_n, is fixed at $\alpha_n = 1$ for all n, the resulting algorithm is identical to the capacity restraint method. This observation is important because it means that existing computer programs that use the capacity restraint method can be easily modified to give a convergent algorithmic solution to the UE problem (by inserting step 3).

To demonstrate the convergence of this algorithm to the equilibrium solution, it is applied to the three-link network example depicted in Figure 5.1. The iterations of this algorithm are shown in Table 5.4. The results in this table should be compared to the convergence patterns of the heuristic methods shown in Tables 5.1 to 5.3. From Table 5.4 it is evident that after five iterations (six minimum-path calculations, including the initialization process) the flows are close to equilibrium; the travel times on all three routes are very similar.

A comparison of this algorithm with the previous (heuristic) methods, in

TABLE 5.4 Convex Combinations Algorithm Applied to the Network in Figure 5.1

Iteration Number	Algorithmic Step	Link 1	Link 2	Link 3	Objective Function	Step Size
0	Initialization	$t_1^0 = 10.0$	$t_2^0 = 20.0$	$t_3^0 = 25.0$		
		$x_1^1 = 10.00$	$x_2^1 = 0.00$	$x_3^1 = 0.00$		
1	Update	$t_1^1 = 947.0$	$t_2^1 = 20.0$	$t_3^1 = 25.0$	$z(\mathbf{x}) = 1975.00$	
	Direction	$y_1^1 = 0$	$y_2^1 = 10$	$y_3^1 = 0$		$\alpha_1 = 0.596$
	Move	$x_1^2 = 4.04$	$x_2^2 = 5.96$	$x_3^2 = 0.00$		
2	Update	$t_1^2 = 35.0$	$t_2^2 = 35.0$	$t_3^2 = 25.0$	$z(\mathbf{x}) = 197.00$	
	Direction	$y_1^2 = 10$	$y_2^2 = 0$	$y_3^2 = 0$		$\alpha_1 = 0.161$
	Move	$x_1^3 = 3.39$	$x_2^3 = 5.00$	$x_3^3 = 1.61$		
3	Update	$t_1^3 = 22.3$	$t_2^3 = 27.3$	$t_3^3 = 35.3$	$z(\mathbf{x}) = 189.98$	
	Direction	$y_1^3 = 10$	$y_2^3 = 0$	$y_3^3 = 0$		$\alpha = 0.035$
	Move	$x_1^4 = 3.62$	$x_2^4 = 4.83$	$x_3^4 = 1.55$		
4	Update	$t_1^4 = 26.1$	$t_2^4 = 26.3$	$t_3^4 = 25.3$	$z(\mathbf{x}) = 189.44$	
	Direction	$y_1^4 = 0$	$y_2^4 = 0$	$y_3^4 = 10$		$\alpha = 0.020$
	Move	$x_1^5 = 3.54$	$x_2^5 = 4.73$	$x_3^5 = 1.72$		
5	Update	$t_1^5 = 24.8$	$t_2^5 = 25.8$	$t_3^5 = 25.4$	$z(\mathbf{x}) = 189.33$	
	Direction	$y_1^5 = 10$	$y_2^5 = 0$	$y_3^5 = 0$		$\alpha = 0.007$
	Move	$x_1^6 = 3.59$	$x_2^6 = 4.70$	$x_3^6 = 1.71$		
	Update	$t_1^6 = 25.6$	$t_2^6 = 25.7$	$t_3^6 = 25.4$	$z(\mathbf{x}) = 189.33$	

terms of computational efficiency, can be made on the basis of the flow pattern at the end of the third iteration (this is equivalent to four all-or-nothing assignments in terms of computational effort). At this point, the travel times generated by the convex combinations algorithm on the three paths are $\mathbf{t}^4 =$ (26.1, 26.3, 25.3). This situation is much closer to equilibrium than the path travel times generated by the modified capacity restraint method, $\mathbf{t}^* =$ (13.7, 27.3, 26.8), or the ones produced by the incremental assignment method, $\mathbf{t}^* =$ (69.0, 27.3, 25.0).

As Table 5.4 shows, flow is taken away from congested paths and assigned to less congested paths, at each iteration of the convex combinations method. This process equalizes the travel times among all paths and brings the system toward equilibrium. The reduction in the objective function value illustrated in Table 5.4 follows the pattern demonstrated in Figure 4.9. Again, the marginal contribution of each successive iteration to the reduction in the objective function value is decreasing.

In solving the UE program over a large network, each iteration involves a significant computational cost, due primarily to the effort required to calculate the shortest paths in the direction-finding step. It is important, then, that a good answer is achieved after a relatively small number of iterations.

In practice, this is not a major problem because of two reasons. First, the convergence pattern of the convex combinations algorithm is such that the first few iterations are the most "cost effective." In other words, the flow pattern after only a few iterations is not very far from equilibrium. Second, the convergence criteria used in practice are not very stringent and thus convergence can be achieved after only a small number of iterations. This is because the accuracy of the input data does not warrant the effort needed to obtain an extremely accurate equilibrium flow pattern.

The number of iterations required for convergence is significantly affected by the congestion level on the network. In relatively uncongested networks, a single iteration may suffice since the link flows may be in the range where the performance functions are almost flat. This means that the updated travel times are very close to the initial ones, generating a set of link flows that is quite similar to the initial solution. As congestion builds up, more iterations are required to equilibrate the network. This effect is demonstrated in Figure 5.2, which depicts the convergence pattern of the convex combinations method for a medium-sized network, for three congestion levels. The convergence measure depicted in this figure is based on the objective function values. The three curves in the figure correspond to low, medium, and high levels of congestion over the network. As the figure shows, the congestion effect on the convergence rate can be quite pronounced.

In actual applications, only four to six iterations are usually sufficient to find the equilibrium flow pattern over large urban networks. This number reflects common practice in terms of trade-offs among analytical accuracy, data limitations, and budget, given typical congestion levels.

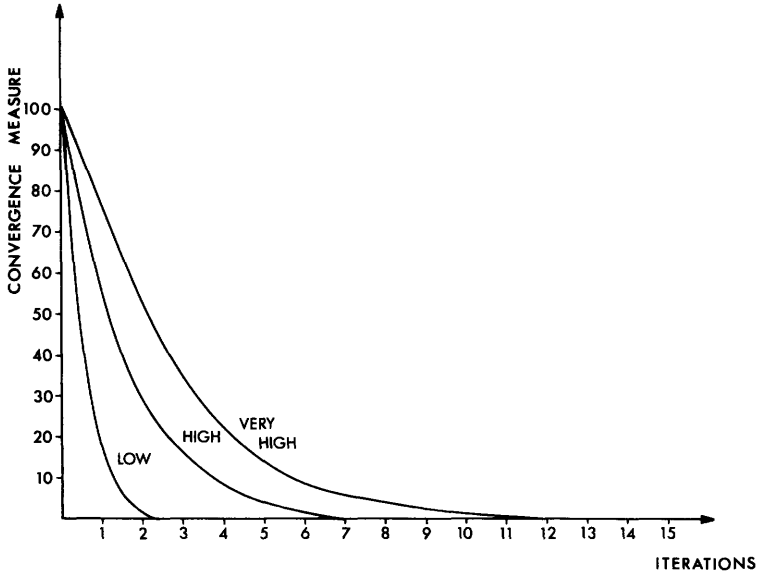


Figure 5.2 Convergence rate of the convex combinations algorithm for various congestion levels. Congestion is measured in terms of $\sum_a x_a t_a(x_a)/\sum_a x_a t_a(0)$, where x_a and t_a are the equilibrium flow and travel times. Convergence is measured by the value of the objective function, normalized between the initial iteration and the 30th one. At iteration n this measure is $\{1 - [z(\mathbf{x}^0) - z(\mathbf{x}^n)]/[z(\mathbf{x}^0) - z(\mathbf{x}^{30})]\}100$.

5.3 SHORTEST PATHS OVER A NETWORK

As demonstrated in this chapter, both the optimization and heuristic approaches to the solution of the UE problem require an iterative all-or-nothing assignment. This assignment includes loading the trips between each origin–destination pair on to the shortest travel time path connecting this pair. The problem is thus that of finding the minimum-travel-time paths connecting each O–D pair for a given set of link travel times. The solution algorithm has to be very efficient since, in the course of finding the user equilibrium, the minimum-path problem has to be solved over and over again for each O–D pair at different flow levels.

The notation used up to this point will be slightly modified for the purposes of the discussion in this section. Computer memory, when storing a network, does not keep separate lists of nodes and links, but rather a list of nodes only. Links are identified by their end nodes. If i and j are nodes (i.e., $i, j \in \mathcal{N}$), the link pointing from i to j is indexed by ij (i.e., $ij \in \mathcal{A}$). Accordingly, x_{ij} and t_{ij} denote the flow and travel time, respectively, on link ij.

Shortest-Path Algorithm

The algorithm presented here is known in the operations research literature as the label-correcting method. It finds the shortest path from a given origin (root) node to all other nodes in the network. Accordingly, it has to be

used for each origin in turn for a complete all-or-nothing assignment to be performed. The explanation that follows pertains to the calculation of such a minimum-path *tree*, that is, the shortest path from one root node to all other nodes.

The algorithm essentially scans the network nodes in an iterative manner. At each iteration the algorithm tries to find a path from the root to the node being scanned that is better (shorter) than the current path. The algorithm terminates when no better path can be found from the root to any of the other nodes in the network.

Assume that the network is stored in the computer as a list of links identified by their end nodes. A travel time (or length) t_{ij} is associated with each link ij. In addition, two pieces of information are stored for every node i in the network: 1) the (current) *label* of this node, l_i, and 2) its (current) *predecessor* node, p_i. The label of node i is the distance from the root node to node i along the (current) shortest path. The predecessor of node i is the node just preceding node i along the (current) shortest path. A list of the predecessor nodes (the p_i's) is continuously updated so that the minimum paths can be traced once the algorithm is terminated. (These paths are traced backward from every node to the root.) The algorithm requires an examination of all the network nodes at least once. To help manage and keep track of the nodes, the algorithm uses an additional list called the sequence list. This list includes all the nodes that have yet to be examined as well as the nodes requiring further examination.

The algorithm is initialized by setting all labels (which are arranged in a label list) to infinity (or, in practice, to a very large number), setting all predecessor nodes (in the predecessor list) to zero, and placing the origin node, r, on the sequence list with label $l_r := 0$.

Each iteration starts with the selection of a node (say, i) from the sequence list for examination. (At the first iteration, the root node is the only one on the sequence list and therefore is examined first.) All nodes, j, that can be reached from i by traversing only a single link are tested in the examination process. If the minimum path to j through i is shorter than the previous path to j, then l_j is updated. In other words, if

$$l_i + t_{ij} < l_j \qquad [5.9]$$

then the current shortest path from the root node to j can be improved by going through node i. To reflect this change, the label list is updated by setting $l_j := l_i + t_{ij}$, the predecessor list is updated by setting $p_j := i$, and the sequence list is updated by adding j to it (since this change may affect nodes that can be reached from j). Once all the nodes j (that can be reached from i) are tested, the examination of node i is complete and it is deleted from the sequence list. The algorithm terminates when the sequence list is empty. At this point, the shortest path from the root to any other node can be found by tracing the predecessor list back to the root node.

Note that the algorithm "fans out" from the root node. It first places all

the nodes that can be reached from the origin by traversing only a single link on the sequence list, and eliminates the origin itself from the list. The algorithm then keeps adding nodes to the list if the label test is met and deleting nodes already examined. Note that a node eliminated from the sequence list can reappear on it in a later stage.

Example

Consider the simple network depicted in Figure 5.3. This network includes one O–D pair and four links. (The number shown next to each link indicates the travel time associated with it.) The label-correcting algorithm is used below to find the shortest path from node 1 to node 4.

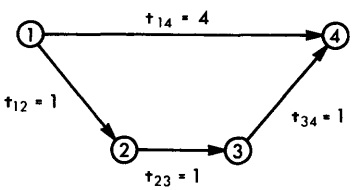


Figure 5.3 Network example for the minimum-path algorithm.

First, all labels are initialized to ∞ and all predecessors to zero. Next, the label of node 1 is changed to zero and it is placed on the sequence list. The sequence list is then scanned and node 1 is chosen for examination (at this point there are no other choices). Two nodes can be reached from node 1 (2 or 4); assume that 4 is considered first. Now, since $l_1 + t_{14} = 4 < l_4 = \infty$, the label of node 4 is changed to 4 and this node is placed on the sequence list. Node 1 is not yet erased from the list, since link $1 \rightarrow 2$ must be considered next. This is then followed by considering links $2 \rightarrow 3$ and $3 \rightarrow 4$. The contents of the label list, the predecessor list, and the sequence list for these iterations are given in Table 5.5. At the fourth iteration, the label of node 4 is changed from 4 to 3. The fifth iteration only verifies that no link emanates from node 4. This node is then removed from the sequence list and the algorithm terminates (since the sequence list is empty). The minimum path can now be traced backward from node 4 by using the predecessor list.

TABLE 5.5 Contents of the Label, Prediction, and Sequence Lists

		Label List				Predecessor List				
Iteration	Link Tested	Node 1	Node 2	Node 3	Node 4	Node 1	Node 2	Node 3	Node 4	Sequence List
Initialization	—	0	∞	∞	∞	0	0	0	0	1
1	$1 \rightarrow 4$	0	∞	∞	4	0	0	0	1	1, 4
2	$1 \rightarrow 2$	0	1	∞	4	0	1	0	1	2, 4
3	$2 \rightarrow 3$	0	1	2	4	0	1	2	1	3, 4
4	$3 \rightarrow 4$	0	1	2	3	0	1	2	3	4
5	—	0	1	2	3	0	1	2	3	

Note on Computer Implementation†

The label-correcting algorithm discussed here can efficiently determine the minimum-travel-time path from an origin node to all other nodes in a network. The algorithm can be applied to very large networks, but care must be taken in coding the algorithm. This section discusses some computer implementation issues and gives some guidelines for efficient coding of the label-correcting method.

The first issue of concern is the method of storing the network. In describing the algorithm, it was assumed that a list of the network links is available. Consider, for example, the network depicted in Figure 5.4 on page 126. This network can be described by the list of its links identified by their end nodes. For example:

"From" Node	"To" Node
1	2
2	3
3	6
1	4
4	5
5	6
1	5
5	2
2	6
2	5

In addition, each link is associated with some travel time, as shown in the figure. At each iteration of the algorithm, all links emanating from a particular node have to be tested. Thus, to avoid repeated searches of the link list, it is advantageous to sort the links so that all links emanating from the same node are stored adjacently: that is,

"From" Node	"To" Node
1	2
1	4
1	5
2	3
2	6
2	5
3	6
4	5
5	2
5	6

†This section can be skipped without loss of continuity.

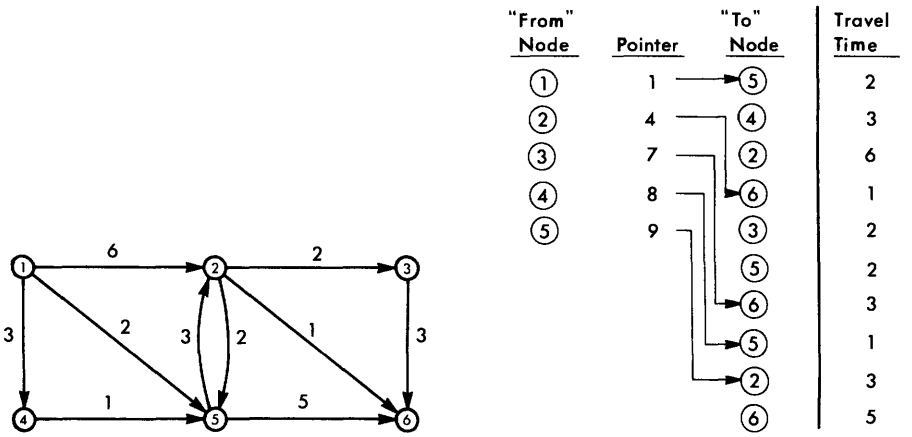


Figure 5.4 Network example demonstrating a forward star network presentation. (The numbers on the links show the links' travel times.)

where "from" nodes are arranged in ascending order. To save space in computer memory, this list can be stored in a "forward star" form, where each arc is represented only by its ending node. A pointer is kept for each node, indicating the position, in the link list, of links beginning with this node. The right-hand side of Figure 5.4 demonstrates this pointer representation. For the example under consideration, this pointer can be thought of as an array, **B**, of length 5 with the following entries:

$$B(1) = 1$$

$$B(2) = 4$$

$$B(3) = 7$$

$$B(4) = 8$$

$$B(5) = 9$$

Instead of keeping two link-length arrays,† the network can thus be stored with one link-length and one node-length array. Using this forward star form, all the links emanating from each node can be processed easily.

The management of the sequence list is another area in which significant computational gains can be made. First, note that *all* the nodes that represent links emanating from a particular node under examination should be tested consecutively. This keeps the sequence list small and avoids unnecessary searches for the location of a particular link.

To avoid duplicating the computation, nodes should be added to the sequence list only if they are not already on it. As it turns out, it is advanta-

†A link-length array includes as many components as the number of links in the network. Similarly, a node-length array includes as many entries as there are nodes in the network.

geous to process the sequence list (i.e., to choose candidate nodes for examination) in a particular order. In general, the sequence list is processed from the top down, and nodes are added to the list by placing them at the bottom. (In other words, the sequence is treated as a queue.) If, however, a node to be added has already been on the list (and examined and removed from it), it should be placed on the top (meaning that it would be examined next). This strategy of managing the sequence list has been shown to be most effective for computing shortest paths over transportation networks.

Figure 5.5 depicts a flowchart of the label-correcting algorithm which utilizes the forward star form and uses the aforementioned strategy for managing the sequence list. Note that, in the computer, this list is not managed by physically moving all nodes in order to place a node at the top of the list. Instead, the sequence list is a node-length array, say s, containing the following information regarding the status of each node in the sequence list:

$$s(i) = \begin{cases} -1 & \text{if node } i \text{ was previously on the list but is no longer on it} \\ 0 & \text{if node } i \text{ has never been on the list} \\ +j & \text{if node } i \text{ is on the list and } j \text{ is the next node on the list} \\ +\infty & \text{if node } i \text{ is on the list and it is the last node on the list} \end{cases}$$

In addition, the top and bottom of the list are identified by special pointers. Placing a node, j, at the top of the list means only that $s(j) := m$ if m was the previous first node on the list, and the top pointer is changed to point at j. Placing a node, j, at the bottom of the list is done by setting $s(j) := \infty$, $s(n) := j$ if n was the previous last node on the list, and the bottom pointer is changed to point at j.

This concludes the description of the shortest path algorithm. The remainder of this section discusses some related numerical issues regarding the solution of user-equilibrium problems.

Transportation networks are characterized by a number of links/number of nodes ratio of approximately 4. For such networks, the computational effort associated with the identification of each minimum-path tree grows linearly (on the average) with the size of the network analyzed. In other words, the computer CPU time† needed is proportional to the number of nodes (and links) in the network. This observation can be used to assess the computational effort associated with the solution of the UE program (using the algorithm outlined in Section 5.2).

Finding the minimum paths is the most computation-intensive component of each iteration of most UE solution procedures (and, in particular, the convex combinations algorithm). The other components (loading the mini-

†CPU (central processing unit) time is a measure of the effort required by the computer to execute a program.

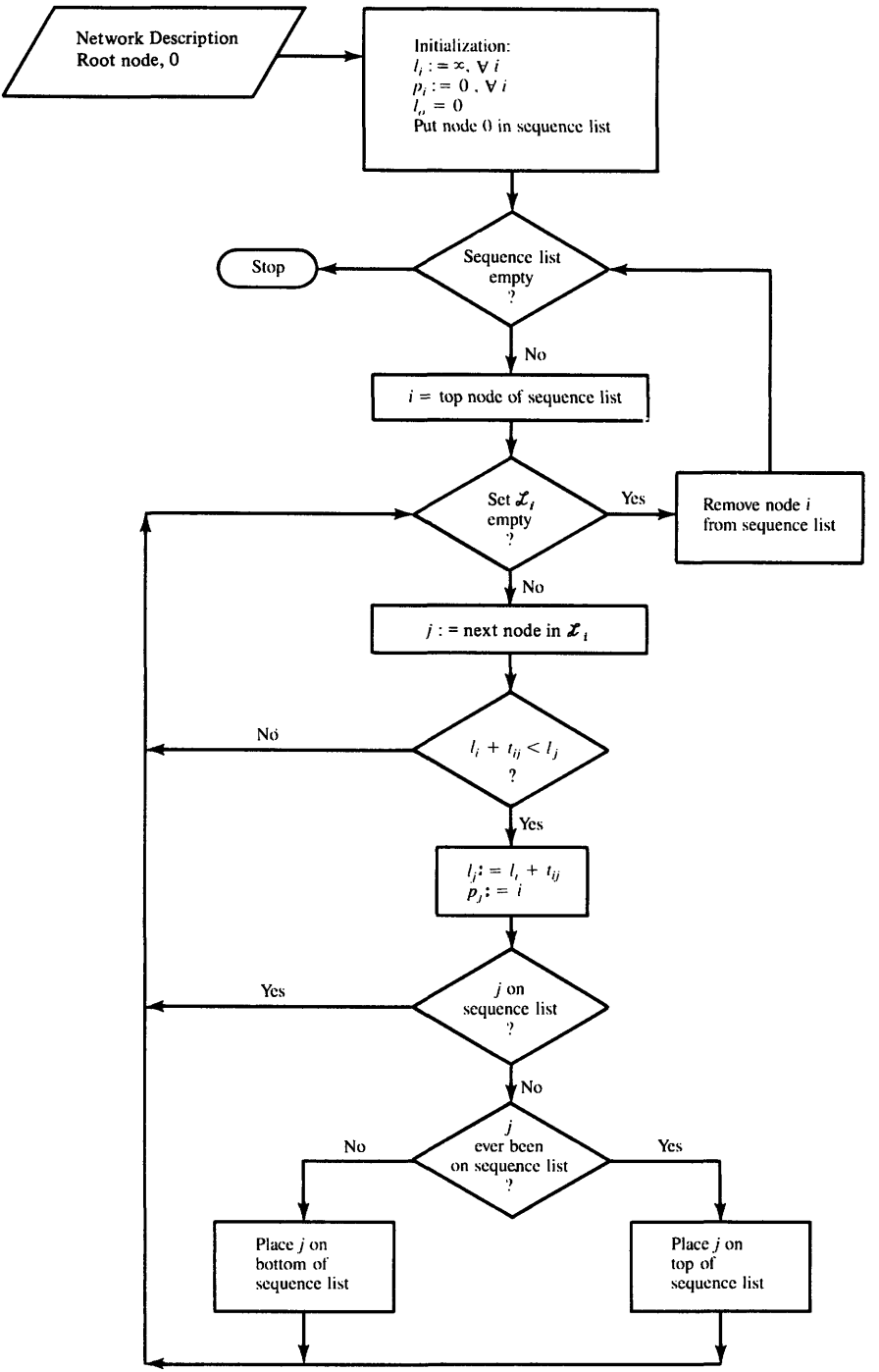


Figure 5.5 Flowchart of the label-correcting shortest-path algorithm. (The set $\mathscr{L}_i$ includes all nodes that can be reached from i by traversing a single link.)

mum paths, the line search, the updates, and the convergence checks) do not require more than a few percentages of the total CPU time. Consequently, the computational effort associated with this application of the convex combinations algorithm is proportional to the product of the number of iterations, the number of origins† (which determines the number of minimum-path trees to be calculated at each iteration) and the number of nodes (which determines the effort needed to calculate each tree). In other words,

$$\begin{pmatrix}\text{computational}\\ \text{costs}\end{pmatrix} = K\begin{pmatrix}\text{number of}\\ \text{iterations}\end{pmatrix}\begin{pmatrix}\text{number of}\\ \text{origins}\end{pmatrix}\begin{pmatrix}\text{number of}\\ \text{nodes}\end{pmatrix} \quad [5.10]$$

where K is a constant of proportionality that is computer specific.

5.4 SUMMARY

The focus of this chapter is on finding the user-equilibrium flow pattern over a transportation network. The first section reviews the two most widely used heuristic techniques for solving this network equilibrium problem. These include capacity restraint methods, which are not guaranteed to converge, and incremental assignment methods, which may "converge" to a nonequilibrium solution. The inadequate performance of these heuristics has motivated the development of the equivalent minimization approach used in this text.

The convex combinations algorithm described in Section 4.3 can be easily applied to the solution of the UE equivalent minimization program. The solution of the linear program associated with each step of this algorithm requires merely an all-or-nothing network loading. Such a loading involves the assignment of all flow between each O–D pair to the minimum-travel-time path connecting this O–D pair. The problem of finding the set of all relevant minimum paths is a well-known problem in operations research for which many efficient algorithms are available.

One of the most efficient algorithms for finding the minimum path from a single node to all other network nodes is the label-correcting algorithm described in Section 5.3. This algorithm is extremely efficient and its execution can be expedited even further by careful coding and list-processing procedures.

The use of the convex combinations method in conjunction with the label-correcting (or any other minimum-path) algorithm for the direction-finding step provides an easy and efficient approach to the minimization of the equivalent UE program. The convergence of the convex combinations method is asymptotic in nature; the marginal contribution of each additional iteration

†If the number of destinations is smaller than the number of origins, the all-or-nothing assignment can be carried out by rooting the minimum-path trees at the destinations. Each tree would then give the minimum path from each of the network nodes to the destination root. The total computational costs associated with solving the UE program are, then, proportional to the product of the number of iterations, the number of nodes, and the minimum of the number of origins and number of destinations.

to the reduction in the value of the objective function is decreasing. The number of iterations required for convergence is primarily a function of the congestion over the network. The computational effort needed for each iteration is proportional to the number of origins (or number of destinations, if it is smaller) and the size of the network.

5.5 ANNOTATED REFERENCES

The two heuristic method presented in Section 5.1 are embedded in many of the early transportation planning computer packages. The U.S. Federal Highway Administration (FHWA) includes a modified capacity restraint method in its Urban Transportation Planning Package (Federal Highway Administration, 1977). The incremental assignment method was used in The DODOTRANS package developed at M.I.T. by Manheim and Ruiter (1970). Both types of procedures were criticized by a number of researchers on the grounds mentioned in this chapter. For example, Sheffi and Daganzo (1978) showed the divergence property of the capacity restraint method for a contrived network, while Ferland et al. (1975) demonstrated the problems associated with the use of incremental assignment. The example used in Section 5.1 to demonstrate both the problems associated with the use of heuristic methods and the convergence of the convex combinations method follows the exposition by Eash et al. (1979). Currently, the network assignment package UROAD, which is part of the UMTA Transportation Planning System (UTPS) supported by the U.S. Urban Mass Transportation Administration, includes the convex combinations algorithm.

The application of Frank and Wolfe's convex combinations method to the solution of transportation network equilibrium was first suggested by Bruynooghe et al. (1968) and applied by Murchland (1969). Shortly thereafter it was used by LeBlanc et al. (1975), who coded and tested the algorithm for a small city. At the same time, Nguyen (1974b) suggested the use of the convex simplex method for solving the UE equivalent minimization program. Nguyen (1974a) also suggested the use of the reduced gradient method and a modified reduced gradient method for this purpose. In some side-by-side comparative experiments, Florian and Nguyen (1976) found that even though the convex simplex method converges somewhat faster than the convex combinations method, it requires more computer memory. Consequently, the overall computational effort required by both methods is similar. The latter reference also includes an interesting validation study, where the equilibrium flows were found to be in satisfactory agreement with ground counts in the city of Winnipeg, Canada. Bovy and Jansen (1981) also demonstrate a high level of agreement between ground counts taken in the city of Eindhoven, the Netherlands, and the flows resulting from a user-equilibrium assignment. Figure 5.2 is taken from Mimis (1984), who used a "hub and spokes" simulated network to study some numerical issues in conjunction with equilibrium assignment algorithms.

The minimum-path algorithm presented in Section 5.3 is the label-correcting algorithm suggested by Moore (1957), with the modification for handling the sequence list suggested by Pape (1974). The presentation of the algorithm and the discussion of the computer implementation issues follow Dial et al. (1977), who tested a number of the most widely used minimum-path algorithms and concluded that the Moore–Pape algorithm is the most efficient for transportation networks. This reference includes a discussion and detailed presentations of many of these algorithms.

5.6 PROBLEMS

5.1. Show how the convex combinations algorithm can be used to solve the system optimization assignment problem discussed in Section 3.4. Describe the algorithm in detail. Should the objective function values be used as the basis for a convergence criterion?

5.2. Find the user-equilibrium flow pattern over the network shown in Figure P5.1. The O–D flows are:

$$1 \rightarrow 2 \qquad 2 \text{ flow units}$$

$$3 \rightarrow 2 \qquad 2 \text{ flow units}$$

and the volume–delay curves are $t_a = 1 + 0.15(x/a)^4$, where a is the link's number shown in the figure. Perform three iterations of the convex combinations method.

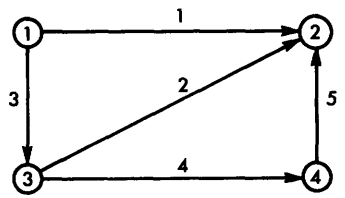


Figure P5.1

5.3. Streets and roadways cannot carry more than a certain amount of vehicular flow known as "capacity."

(a) Formulate the UE problem assuming that each network link is associated with a capacity c_a.

(b) Using the ideas outlined in the description of the feasible direction methods in Section 4.2, devise a modification to the convex combinations method described in Section 5.2 so that the algorithm would solve the capacity-constrained UE problem. You can assume that an initial *feasible* solution is given and that the performance curves are asymptotic to the capacity flow, for example, $t_a(x_a) = 1/(c_a - x_a)$.†

(c) Devise a method for finding an initial feasible solution for your algorithm.

†Mathematically, it is required that the performance curves would satisfy

$$\lim_{x_a \rightarrow c_a} \int_0^{x_a} t_a(\omega)\, d\omega = \infty$$

as shown by Daganzo (1977c).

5.4. Program [5.4] defines the linearization step of the convex combinations algorithm in terms of path flows. Outline all the other algorithmic steps associated with the application of this algorithm to the UE equivalent program, in terms of path flows. Can the algorithm be executed in terms of path flows?

5.5. In transportation planning applications the analyst may delete and add links to the network. Discuss the problems that the forward star representation poses for easy addition and deletion of links. How can these problems be overcome?

5.6. Use the label-correcting algorithm to find the minimum path from node 1 to all other nodes in the network depicted in Figure 5.4.

5.7. Explain why the particular strategy described in the text for managing the sequence list is advantageous for transportation networks. (As an aside, note that it is not particularly advantageous for other types of networks, such as communication networks.)

5.8. Write a subroutine that would take as input a network in forward star representation (transferred in a COMMON block or through GLOBAL variables) and an origin node (transferred in the calling statement), and would compute the minimum paths from the origin to all other nodes. The result is to be returned in another COMMON block or a set of GLOBAL variables (of predecessors).

5.9. Write an input subroutine that would read network link information (including starting node, end node, and performance function parameters) in random order and store it in a forward star representation. Include a flowchart.

5.10. Use the results of Problems 5.8, 5.9, and 4.10 (or 4.11) to write a computer code that will find the equilibrium flows over a transportation network.

5.11. The operation research literature includes algorithms for finding the minimum path from every node of a network to every other node. Indicate under which circumstances such algorithms will be useful and explain why they are not used in transportation planning applications.

III

Extensions of User Equilibrium

Part III extends in several directions the UE framework discussed in previous chapters. Chapter 6 focuses on the variable-demand problem, in which the O–D trip rates are not assumed to be fixed but rather a function of the equilibrium travel times. The concepts developed there are used to formulate a model in which the network includes the transit mode, in addition to the automobile network, and the modal split is modeled explicitly. Chapter 7 extends the original UE framework in another direction by assuming that the total number of trips leaving each origin node is known but that their destination is to be determined in conjunction with the equilibration process. Chapter 8 removes the assumption used throughout Part II that links are independent of each other, and Chapter 9 develops a framework in which many travel choice dimensions (i.e., whether to take a trip, where to go, by which mode, and what route to use) can be modeled simultaneously.