Auction Algorithms for Path Planning, Network Transport, and Reinforcement Learning

by

Dimitri P. Bertsekas

Chapter 1
Auction Algorithms - An Introduction

This monograph represents "work in progress," and will be periodically updated. It more than likely contains errors (hopefully not serious ones). Furthermore, the references to the literature are incomplete. Your comments and suggestions to the author at dimitrib@mit.edu are welcome.

March 15, 2025

1

Auction Algorithms - An Introduction

Contents

Network flow problems are one of the most important and frequently encountered class of optimization problems. They arise naturally in the analysis and design of large engineering systems, such as communication, transportation, and manufacturing networks. They can also be used to model important classes of combinatorial problems, such as assignment, shortest path, and traveling salesman problems, which in turn arise in a broad variety of applications. Moreover, they are an integral part of several types of artificial intelligence software, such as those involving knowledge graphs and path planning.

Loosely speaking, network flow problems consist of supply and demand points (also called *sources* and *sinks* respectively), together with several routes that connect these points and are used to transfer the supply to the demand. These routes may contain intermediate transhipment points. Often, the supply, demand, and transhipment points can be modeled by the nodes of a graph, which are connected through arcs. There are costs for traversing the arcs, which encode our preferences for using some of the arcs in place of others. There may also be constraints on the carrying capacities of the arcs. Problems of this type are naturally modeled using network optimization, whereby we aim to select routes that minimize the cost of transfer of the supply to the demand.

We pay special attention to three major mathematical network optimization problems:

(a) The *assignment* problem, which involves matching the elements of two finite sets, on a one-to-one basis, at minimum cost.

(a) The *shortest path* problem, which involves finding a minimum cost path(s) between designated origin(s) and destination(s).

(c) The *transhipment problem*, also known as the *network transport problem*, which involves supply and demand points, but also arc capacity constraints.

The transhipment problem, contains the other two as special cases, but it can also be reformulated as an assignment problem. Moreover the shortest path problem can also be reformulated as an assignment problem. Because of these relations, algorithms that are used to solve one type of problem, can be adapted to solve any other problem. This is a fundamental conceptual point, which will be important for our development.

In this book we focus on a special class of network optimization methods. The starting point is an intuitive algorithm for the assignment problem, the *auction algorithm*, introduced by the author in the 1979 paper [Ber79] and studied together with its variations since then, including in the book [Ber98]. We will adapt this algorithm to solve other network optimization problems in new ways.

Much of the new research in this book relates to adaptations of the auction algorithm for the assignment problem, to apply to path construc-

tion problems, including the classical shortest path problem. Auction algorithms for path construction have an intuitive form, and will be used as the basis for solution of other problems, including max-flow, transportation, and transhipment problems. Auction/path construction algorithms will also be used, in both exact and approximate form, in an important application area: reinforcement learning and sequential decision making, particularly in contexts where the popular Monte Carlo tree search and real-time dynamic programming methods have been used.

The present chapter has an introductory character, and aims at an intuitive introduction of the principal conceptual ideas underlying auction algorithms. In Sections 1.1 and 1.2, we will describe our principal problem formulations and their interrelations. In Section 1.3, we will discuss another fundamental concept, duality, which forms the mathematical foundation for much of our development. Then in Section 1.4, we will outline and justify auction algorithms, and their relation to primal and dual optimization, focusing primarily on our two principal paradigms, assignment and shortest path. In Section 1.4.4 we will provide a preview of ways that auction algorithms can be fruitfully incorporated within the broad framework of the reinforcement learning methodology.

Detailed discussions of auction algorithms for assignment and shortest path problems will be given in Chapters 2 and 3, respectively. The extension of the algorithmic ideas of Chapters 2 and 3 to transhipment and network transport problems will be given in Chapter 4. The application and adaptation of auction algorithms to reinforcement learning will be discussed in Chapter 5. The material in this last chapter is optional and is not needed for the developments of earlier chapters.

The author's 1998 network optimization book [Ber98] (freely available on-line) serves to provide support for some of the more mathematical aspects of the present book, including some proofs and computational complexity analyses. It uses similar notation and terminology, and it discusses a broader range of network algorithms and applications.

1.1 THREE FUNDAMENTAL NETWORK OPTIMIZATION PROBLEMS

The problems discussed in this book involve a directed graph. The arcs of the graph are denoted by (i, j), where i and j are referred to as the *start* and *end* nodes of the arc. The sets of nodes and the set of arcs are denoted by $\mathcal{N}$ and $\mathcal{A}$, respectively, and the graph itself is represented by the pair $(\mathcal{N}, \mathcal{A})$. If (i, j) is an arc, it is possible that (j, i) is also an arc [separate from (i, j)]. No self arcs of the form (i, i) are allowed. For convenience of presentation, we assume that for any two nodes i and j, there is at most one arc with start node i and end node j. For any node i we say that node j is a *downstream neighbor of* i if (i, j) is an arc.

A *path* in a directed graph is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ with $k \geq 2$ and a corresponding sequence of $k-1$ arcs such that the ith arc in the sequence is (n_i, n_{i+1}). The nodes n_1 and n_k are called the *origin* (or *start node*) and the *destination* (or *end node*) of the path, respectively. Note that according to our definition, a path is directed in that all its arcs are oriented in the direction from the origin to the destination.† A *cycle* is a path with more than two nodes for which the start and end nodes are the same ($n_1 = n_k$). Some additional terminology relating to graphs, paths, flows, and other related notions will be introduced as we progress through this chapter.

Graph concepts are fairly intuitive, and can be understood in terms of suggestive figures, but often involve hidden subtleties. An extended presentation is given in the appendix to this chapter; see also the author's network optimization book [Ber98]. We will now introduce three major classes of problems that we will be dealing with.

1.1.1 The Assignment Problem

Suppose that there are n persons and n objects that we have to match on a one-to-one basis. There is a value a_{ij} for matching person i with object j, and we want to assign persons to objects so as to maximize the total value. There is also a restriction that person i can be assigned to object j only if (i, j) belongs to a given set of pairs $\mathcal{A}$. The problem is represented by the graph shown in Fig. 1.1.1.

Mathematically, we want to find a set of person-object pairs

$$(1, j_1), \ldots, (n, j_n)$$

from $\mathcal{A}$ such that the objects $j_1, \ldots, j_n$ are all distinct, and the total value $\sum_{i=1}^{n} a_{ij_i}$ is maximized. Here we have considered the case where the numbers of persons and objects are equal. This is called the *symmetric* assignment problem, to distinguish it from the *asymmetric* assignment problem where the numbers of persons and objects are different.

The assignment problem is important in many practical contexts. The most obvious ones are resource allocation problems, such as assigning employees to jobs, machines to tasks, etc. There are also situations where the assignment problem appears as a subproblem in methods for solving more complex combinatorial problems.

---

† In a more general definition of a path each arc of the path can have either one of the two possible directions. This definition is used in the book [Ber98], where a distinction is made between a path and a *forward* path, whose arcs are all oriented in the forward direction from origin to destination. In the appendix to this chapter, we will introduce this distinction, but for the moment our simpler one-directional definition of a path will suffice.

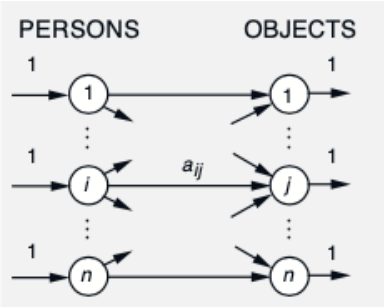


Figure 1.1.1: The graph representation of a symmetric assignment problem. The person nodes can be viewed as sources involving a unit of supply each, while the object nodes can be viewed as sinks involving a unit of demand each. The assignment problem can be viewed as a problem of transporting the unit supplies to the demand points under the additional restriction that each arc can carry flow that is integer (either 0 or 1).

1.1.2 The Shortest Path Problem

Suppose that each arc (i, j) of a graph is assigned a scalar cost a_{ij}, and suppose that we define the cost of a path to be the sum of the costs of its arcs. Given a pair of nodes, the shortest path problem is to find a path that connects these nodes and has minimum cost. An analogy here is made between arcs and their costs, and roads in a transportation network and their lengths, respectively. Within this transportation context, the problem becomes one of finding the shortest route between two geographical points. Based on this analogy, the problem is referred to as the *shortest path problem*, and the arc costs and path costs are also referred to as the *arc lengths* and *path lengths*, respectively.

The shortest path problem arises in a surprisingly large number of contexts. For example in a data communication network, a_{ij} may denote the average delay of a packet to cross the communication link (i, j), in which case a shortest path is a minimum average delay path that can be used for routing the packet from its origin to its destination.

As another example, if p_{ij} is the probability that a given arc (i, j) in a communication network is usable, and each arc is usable independently of all other arcs, then the product of the probabilities of the arcs of a path provides a measure of reliability of the path. With this in mind, it is seen that finding the most reliable path connecting two nodes is equivalent to finding the shortest path between the two nodes with arc lengths $(-\ln p_{ij})$. In this context, the length of the path is equal to minus the logarithm of the product of the probabilities of the arcs of the path. Thus finding a shortest path is equivalent to finding the most reliable path (one for which the product of the corresponding probabilities is maximized).

The shortest path problem also arises often as a subroutine in algorithms that solve other more complicated problems. Within this context, special considerations may come into play, such as for example whether multiple instances of the shortest path problem need to be solved, in real time and with similar data. We will see that auction algorithms are particularly well-suited for applications of this type, as they allow the use of

the solution of a shortest path problem instance as a starting point for the solution of a related problem instance.

1.1.3 The Network Transhipment Problem

To define the transhipment problem, we need the concept of the *flow of an arc*, which can be viewed as a variable that measures the quantity that is moving through the arc. Examples are electric current in an electric circuit, water flow in a hydraulic network, car traffic on a street network, material movement within a manufacturing system, etc. Mathematically, the flow of an arc (i, j) is simply a scalar (real number), which we usually denote by x_{ij}.

Given a graph $(\mathcal{N}, \mathcal{A})$, a set of flows $x = \{x_{ij} \mid (i, j) \in \mathcal{A}\}$ is referred to as a *flow vector* (or simply *flow* when confusion cannot arise). For a given flow vector and a node i, we call the scalar

$$y_i = \sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji}, \tag{1.1}$$

the *excess* of i (the terms *imbalance* of i and *divergence* of i have also been used in the literature). Thus, y_i is the total flow departing from node i minus the total flow arriving at i.

We say that node i is a *source* (respectively, *sink*) for the flow vector x if the corresponding excess satisfies $y_i > 0$ (respectively, $y_i < 0$). If $y_i = 0$ for all $i \in \mathcal{N}$, then x is called a *circulation*. These definitions are illustrated in Fig. 1.1.2. Note that by adding Eq. (1.1) over all $i \in \mathcal{N}$, we obtain

$$\sum_{i \in \mathcal{N}} y_i = 0.$$

Every excess vector must satisfy this equation.

The network transhipment problem is to find a set of arc flows that minimize a linear cost function, subject to the constraints that they produce given node excesses and they lie within some given bounds; that is,

$$\text{minimize} \quad \sum_{(i,j) \in \mathcal{A}} a_{ij} x_{ij} \tag{1.2}$$

subject to the constraints

$$\sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \tag{1.3}$$

$$b_{ij} \leq x_{ij} \leq c_{ij}, \qquad \forall\ (i, j) \in \mathcal{A}, \tag{1.4}$$

where a_{ij}, b_{ij}, c_{ij}, and s_i are given scalars. We call Eq. (1.3) the *conservation of flow equation*. We also use the following terminology:

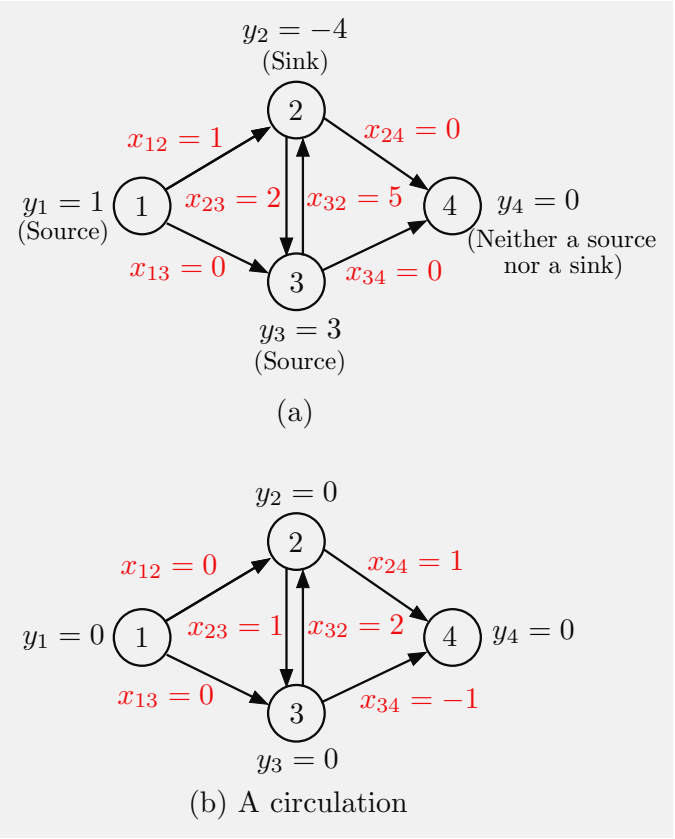


Figure 1.1.2: Illustration of flows x_{ij} and the corresponding excesses y_i. The flow vector of (b) is a circulation because $y_i = 0$ for all i.

a_{ij} the *cost coefficient* (or simply *cost*) of (i, j),

b_{ij} and c_{ij}: the *flow bounds* of (i, j),

$[b_{ij}, c_{ij}]$: the *capacity range* of (i, j),

s_i: the *supply* of node i (when s_i is negative, the scalar $-s_i$ is also called the *demand* of i).

Note the difference in terminology: the excess of a node corresponds to a given flow vector, while the supply of a node is part of the problem data: it is the target excess for the given problem. The conservation of flow equation states that for a feasible flow vector the node excesses should be equal to the node supplies.

For a typical application of the transhipment problem, think of the nodes as locations (cities, warehouses, or factories) where a certain product

is produced or consumed. Think of the arcs as transportation links between the locations, each with transportation cost a_{ij} per unit transported. The problem then is to move the product from the production points to the consumption points at minimum cost while observing the capacity constraints of the transportation links.

The transhipment problem has many applications that are well beyond the transportation context just described. This will be shown in the next section with a variety of examples, which illustrate how some important discrete/combinatorial problems can be modeled as transhipment problems, and highlight the important connection between continuous and discrete network optimization.

Alternative Transhipment Problem Formats - Uncapacitated Transhipment Problem

There is a variant of the transhipment problem that deserves special mention. This variant, called the *uncapacitated format*, arises in cases where there is just a nonnegativity constraint on the flow of each arc (i, j), i.e., the arc flow constraints have the form

$$0 \leq x_{ij}, \qquad \forall\ (i,j) \in \mathcal{A},$$

rather than

$$b_{ij} \leq x_{ij} \leq c_{ij}$$

[cf. Eq. (1.4)]. For example, there are problems where the arc flows are naturally nonnegative and unbounded above, while the upper flow bounds are implied by the conservation of flow Eq. (1.3), in which case we may choose not to introduce them. In particular, the assignment and the shortest path problems can be transformed into both the uncapacitated and the capacitated transhipment formats, as we will see in the next section.

In practice, we may choose to adopt the uncapacitated format for reasons of conceptual and algorithmic simplification. In this regard, it is important to note that a capacitated transhipment problem can be converted into an uncapacitated one with a suitable transformation (see the book [Ber98], Section 4.1). Another possible transformation (also described in [Ber98], Section 4.1) is to reduce the transhipment problem (1.2)-(1.4) into the *circulation format*, whereby all the node supplies s_i are 0.

Other transformations between different problem types and problem formats are possible, and they will be discussed in some detail in the next section. They are important for our purposes as they provide conceptual unification. For example, when a theoretical concept, such as duality (see Section 1.3), is developed for one network problem type, we can be sure that a similar concept has a counterpart for related network problem types. Moreover, once an algorithm, such as auction, has been developed for one problem type, it is reasonable to expect that similar algorithms can be

developed for related problems. The precise form of the transformation from one format to another can be helpful in this algorithmic development process.

1.2 PROBLEM RELATIONS AND TRANSFORMATIONS

In this section, we will illustrate how the three problems, assignment, shortest path, and transhipment, are mathematically interrelated. In particular, in Section 1.2.1 we will introduce some additional special cases and extensions of the transhipment problem, including the classical transportation problem, which provides the connecting link between the assignment and transhipment problems.

In Section 1.2.2, we will discuss the mathematical connections between the different types of problems. In summary, we will show that:

(a) The assignment and shortest path problems (and several other interesting types of network flow problems) can be viewed as special cases of the network transhipment problem (both capacitated and uncapacitated).

(b) The transhipment problem can be reformulated to an equivalent assignment problem, provided the supplies s_i, and the lower and upper capacity bounds b_{ij} and c_{ij} are integer.

(c) The shortest path problem can be reformulated as an equivalent assignment problem.

The details of these transformations are somewhat tedious, and the reader may wish to go through them lightly and return to them later as needed. However, the transformations are conceptually important and practically useful for translating the auction algorithm ideas from the assignment context, where they are most intuitive, to other contexts, such as the shortest path and transhipment problems.

1.2.1 Examples of Network Optimization Problems

We will now describe more formally the types of problems, which collectively delineate the range of problems that we will be dealing with in this book. Auction algorithms for these problems will be outlined in Section 1.4, and will be discussed in greater detail in subsequent chapters.

Example 1.2.1: The Assignment Problem

Let us consider the problem of assignment of n persons to n objects. We may associate any assignment with a vector $x = \{x_{ij} \mid (i,j) \in \mathcal{A}\}$, where $x_{ij} = 1$ if person i is assigned to object j and $x_{ij} = 0$ otherwise. The value of this

assignment is $\sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij}$. The restriction of one object per person can be stated as

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \quad \forall\, i = 1,\ldots,n, \qquad \sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \quad \forall\, j = 1,\ldots,n.$$

We may then formulate the assignment problem as the linear programming problem

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \qquad \forall\, i = 1,\ldots,n, \\ & \sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \qquad \forall\, j = 1,\ldots,n, \\ & 0 \le x_{ij} \le 1, \qquad \forall\, (i,j) \in \mathcal{A}. \end{aligned} \tag{1.5}$$

Note that we have not restricted x_{ij} to be either 0 or 1. As it turns out, this is not necessary: the above linear program has the property that if it has a feasible solution at all, then it has an optimal solution where all x_{ij} are either 0 or 1. Moreover, the set of its optimal solutions includes all the optimal assignments. This can be proved with the help of duality theory, which will be discussed in Section 1.3.

We now argue that the assignment/linear program (1.5) is a transhipment problem involving the graph shown in Fig. 1.1.1. Indeed, there are $2n$ nodes: n corresponding to persons and n corresponding to objects. Also, for every possible pair $(i,j) \in \mathcal{A}$, there is an arc connecting person i with object j. The variable x_{ij} is the flow of arc (i,j). The constraint $\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1$ indicates that the excess of person/node i should be equal to 1, while the constraint $\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = 1$ indicates that the excess of object/node j should be equal to -1. Finally, we may view $(-a_{ij})$ as the cost coefficient of the arc (i,j) (by reversing the sign of a_{ij}, we convert the problem from a maximization to a minimization).

It is also possible to eliminate the upper bound constraints $x_{ij} \le 1$ from the problem (1.5), since these constraints are implied by the conservation of flow constraints $\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1$. The resulting assignment problem can be viewed as a special case of the uncapacitated transhipment problem.

Example 1.2.2: The Shortest Path Problem

It is possible to cast the problem of finding a shortest path from an origin node r to destination node t in a directed graph as a transhipment problem by specifying:

(a) The supply of r and the demand of t to be 1, with all other node supplies being 0.

(b) The capacity range of each arc (i, j) to be $[0, 1]$.

(c) The cost coefficient of each arc (i, j) to be a_{ij}, the length of arc (i, j).

This leads to the problem:

$$\begin{aligned} &\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ &\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = \begin{cases} 1 & \text{if } i = r, \\ -1 & \text{if } i = t, \\ 0 & \text{otherwise,} \end{cases} \\ &\qquad 0 \le x_{ij} \le 1, \qquad \forall\ (i, j) \in \mathcal{A}, \end{aligned} \tag{1.6}$$

which we recognize as a transhipment problem.

However, for this problem to be equivalent to the shortest path problem, a certain assumption must be satisfied: *all cycles of the given graph must have nonnegative length* (the sum of their arc lengths must be nonnegative). The reason is that the transhipment problem (1.6) allows feasible solutions with positive flow along a cycle, whereas the shortest path problem does not allow paths with cycles to be feasible solutions. The nonnegative length cycle assumption ensures that the transhipment problem (1.6) has optimal solutions that do not contain cycles with positive flow.

For another view of the relation between the shortest path problem and the transhipment problem (1.6), let us consider any path P from r to t that contains no cycles, and the flow vector x with components given by

$$x_{ij} = \begin{cases} 1 & \text{if } (i, j) \text{ belongs to } P, \\ 0 & \text{otherwise.} \end{cases} \tag{1.7}$$

Then x is feasible for problem (1.6) and its cost is equal to the length of P. Thus, if a flow vector x of the form (1.7) is an optimal solution of problem (1.6), the corresponding path P is shortest.

Conversely, it can be shown that if problem (1.6) has at least one feasible solution, then it has an optimal solution of the form (1.7), with a corresponding path P that is shortest. This is not immediately apparent, but its proof can be traced to a remarkable fact that can be shown in general about transhipment problems with node supplies and arc flow bounds that are integer: such problems, if they have an optimal solution, they have an *integer* optimal solution, that is, a set of optimal arc flows that are integer. From this it can be seen that if problem (1.6) has an optimal solution, it has one with arc flows that are 0 or 1, and which is of the form (1.7) for some path P. This path is shortest because its length is equal to the optimal cost of problem (1.6), so it must be less or equal to the cost of any other flow vector of the form (1.7), and therefore also less or equal to the length of any other path from r to t. Thus the shortest path problem is essentially equivalent with the transhipment problem (1.6).

It is also possible to eliminate the upper bound constraints $x_{ij} \le 1$ from the transhipment problem (1.6), since these constraints are implied by the conservation of flow constraints $\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1$. Then the shortest

path problem can be viewed as a special case of the uncapacitated transhipment problem. Note that eliminating the upper bound constraints $x_{ij} \leq 1$ is essential for variants of the shortest path problem where there are multiple origins, each with supply 1, and a single destination with demand equal to the number of origins. In this case, each arc should be able to carry flows as large as the number of origins, and it is just easier to allow this by discarding any arc capacity constraints.

Example 1.2.3: The Fixed-Flow Problem

The fixed-flow problem is simply the special case of the transhipment problem, where there is a single source/origin, with given supply $s > 0$, a single sink/destination, and all arc costs a_{ij} are 0. Thus the objective here is feasibility, i.e., transfer s units of flow from the origin to the destination, while satisfying the arc capacity constraints; see Fig. 1.2.1 for an illustrative example.

The uncapacitated case ($x_{ij} \geq 0$) where $s = 1$ is the simplest of feasibility problems, which we refer to as the *path construction problem*. Here we are interested to find *some* path from source to sink, without regard to any optimality properties.

Actually any feasibility instance of the transhipment problem (one where all arc costs are 0 and the given supply of node i is s_i) and nonzero lower arc flow bounds b_{ij} can be transformed into a fixed flow problem with zero lower flow bounds. This can be done in two steps:

The first step is to set the lower flow bounds to zero by a translation of variables, that is, by replacing x_{ij} by $x_{ij} - b_{ij}$, and by adjusting the upper flow bounds and the supplies according to

$$c_{ij} := c_{ij} - b_{ij},$$

$$s_i := s_i - \sum_{\{j|(i,j)\in\mathcal{A}\}} b_{ij} + \sum_{\{j|(j,i)\in\mathcal{A}\}} b_{ji}.$$

Optimal flows and the optimal value of the original problem are obtained by adding b_{ij} to the optimal flow of each arc (i, j) and adding $\sum_{(i,j)\in\mathcal{A}} a_{ij}b_{ij}$ to the optimal value of the transformed problem, respectively.

The second step is to introduce an artificial origin node r and destination node t, arcs connecting r with every node i with supply $s_i > 0$ and arcs (i, t) connecting every node i with supply $s_i < 0$ to t, with corresponding capacity constraints

$$0 \leq x_{ri} \leq s_i, \qquad 0 \leq x_{it} \leq -s_i.$$

An example of this transcription is shown in Fig. 1.2.2 for the case of a *matching problem*, the feasibility case of an assignment problem, where the value a_{ij} of any pair (i, j) is 0.

Note that the preceding transformation techniques can also be used to *convert any capacitated transhipment problem to one that has a single source node, a single sink node, and lower arc flow bounds that are equal to 0*.

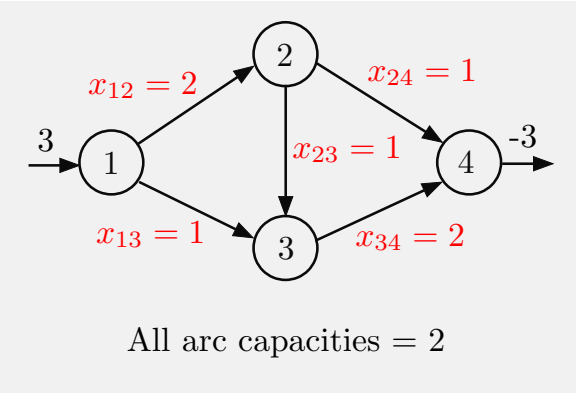


Figure 1.2.1: An example of a feasible solution of a fixed flow problem. There is a single source (node 1), a single sink (node 4), there are no arc costs, and the flow of each arc is constrained to lie between 0 and the arc capacity (2 in this case).

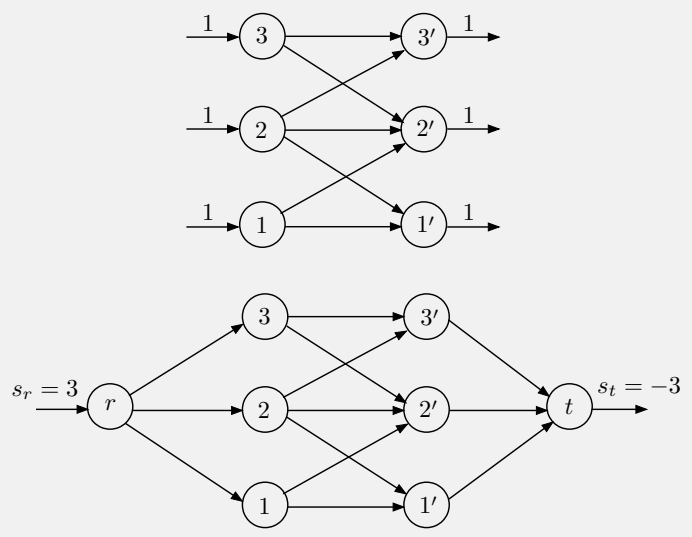


Figure 1.2.2: An example of a problem of matching persons i to objects i', transformed to a fixed flow problem. Artificial supply and demand nodes r and t have been introduced, together with connecting arcs (r, i) and (i', t) whose flow is constrained to lie within the capacity range $[0, 1]$.

Example 1.2.4: The Max-Flow Problem

The max-flow problem is closely related to the fixed flow problem, the only difference being that instead of sending a fixed amount of supply from the

source node r to the sink node t, we try to send the *maximum amount of supply that the arc capacities will allow*. Mathematically, we want to find a flow vector that makes the excess of all nodes other than r and t equal to 0 while maximizing the excess of r. This is equivalent to solving the transhipment problem illustrated in Fig. 1.2.3. As can be expected, algorithms for solving the fixed flow problem can be adapted to solve the max-flow problem. A simple approach for doing so is to solve a sequence of fixed flow problems with ever increasing amounts of supply, up to the point where the supply becomes so large that feasibility is violated. Of course this approach may have to be streamlined for computational efficiency, depending on the context.

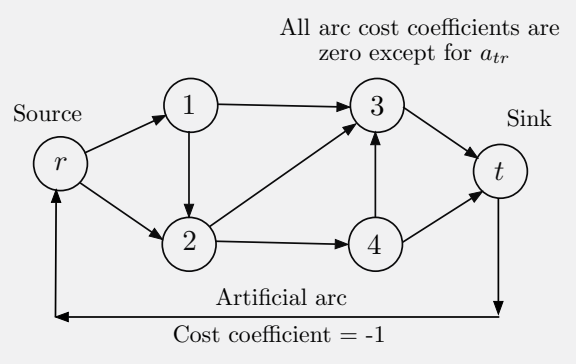


Figure 1.2.3: The representation of a max-flow problem as a transhipment problem. We introduce an artificial arc with cost equal to -1 (or any other negative number) from the sink t to the source r. At the optimum, the flow x_{tr} equals the maximum flow that can be sent from r to t through the subgraph obtained by deleting the artificial arc (t, r).

The max-flow problem arises in several practical contexts, such as calculating the throughput of a highway system or a communication network. It also arises often as a subproblem in more complicated problems or algorithms. In particular, it bears a fundamental connection to the question of existence of a feasible solution of a general transhipment problem (see the discussion in [Ber98], Chapter 3). Moreover, several discrete/combinatorial optimization problems can be formulated as max-flow problems (see the Exercises in [Ber98], Chapter 3).

Example 1.2.5: The Transportation Problem

This problem has played a historically important role in the development of network optimization, and finds wide use in applications, including in the context of network transport (see Example 1.2.7). The transportation problem has a graph structure similar to the one of the assignment problem. However, the node supplies and demands need not be 1, and the numbers of sources and sinks need not be equal. Instead, the supplies and demands are some

positive numbers α_i and β_j, respectively. This in turn implies that the flow of each arc from a source i to a sink j should be no more than α_i and β_j. Mathematically, the problem has the form

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = \alpha_i, \qquad \forall\ i = 1,\ldots,m, \\ & \sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = \beta_j, \qquad \forall\ j = 1,\ldots,n, \\ & 0 \le x_{ij} \le \min\{\alpha_i, \beta_j\}, \qquad \forall\ (i,j) \in \mathcal{A}. \end{aligned} \tag{1.8}$$

The supplies α_i and β_j are given positive scalars, which for feasibility must satisfy

$$\sum_{i=1}^{m} \alpha_i = \sum_{j=1}^{n} \beta_j,$$

(add the conservation of flow constraints).†

Clearly, this problem is a special case of a transhipment problem. In an alternative formulation, the upper bound constraints $x_{ij} \le \min\{\alpha_i, \beta_j\}$ are discarded, since they are implied by the conservation of flow and the nonnegativity constraints, thereby giving rise to a reformulation as an uncapacitated transhipment problem.

For a practical example of a transportation problem that has a combinatorial flavor, suppose that we have m communication terminals, each to be connected to one of n traffic concentrators. We introduce variables x_{ij}, which take the value 1 if terminal i is connected to concentrator j. Assuming that concentrator j can be connected to no more than b_j terminals, we obtain the constraints

$$\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} \le b_j, \qquad \forall\ j = 1,\ldots,n.$$

Also, since each terminal must be connected to exactly one concentrator, we have the constraints

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = 1, \qquad \forall\ i = 1,\ldots,m.$$

---

† The 1975 Nobel prize in economics was awarded to the mathematician L. Kantorovich and the economist T. Koopmans for the formulation and analysis of the transportation problem and its variants. Important related work was done in the 19th century by the mathematician G. Monge, hence the name "Kantorovich-Monge problem" is often used. The transportation problem and a simplex-like method for its solution are also credited to the M.I.T. professor F. L. Hitchcock [Hit41]. According to the network optimization theorist D. R. Fulkerson [Ful56], "The transportation problem was first formulated by the mathematician F. L. Hitchcock in 1941; he also gave a computational procedure, much akin to the general simplex method, for solving the problem." Fulkerson also gives independent credit to Koopmans; he was apparently unaware of Kantorovich's work.

Assuming that there is a cost a_{ij} for connecting terminal i to concentrator j, the problem is to find the connection of minimum cost, that is, to minimize

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} a_{ij}x_{ij}$$

subject to the preceding constraints.

This problem is not yet a transportation problem of the form (1.8) for two reasons:

(a) The arc flows x_{ij} are constrained to be 0 or 1.

(b) The constraints $\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} \leq b_j$ are not equality constraints, as required in problem (1.8).

It turns out, however, that we can ignore the 0-1 constraint on x_{ij}. As discussed in connection with the shortest path and assignment problems, even if we relax this constraint and replace it with the capacity constraint $0 \leq x_{ij} \leq 1$, there is an optimal solution such that each x_{ij} is either 0 or 1.

Furthermore, to convert the inequality constraints to equalities, we can introduce a total of $\sum_{j=1}^{n} b_j - m$ "dummy" terminals that can be connected at zero cost to all of the concentrators. In particular, we introduce a special supply node 0 together with the constraint

$$\sum_{j=1}^{n} x_{0j} = \sum_{j=1}^{n} b_j - m,$$

and we change the inequality constraints $\sum_{j=1}^{n} x_{ij} \leq b_j$ to

$$x_{0j} + \sum_{i=1}^{m} x_{ij} = b_j.$$

The resulting problem has the transportation structure of problem (1.8), and is equivalent to the original problem.

Example 1.2.6: Network Flow Problems with Convex Cost

A more general version of the transhipment problem arises when the cost function is convex rather than linear. An important special case is the problem

$$\begin{aligned} \text{minimize} \quad & \sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}) \\ \text{subject to} \quad & \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i, \qquad \forall\ i \in \mathcal{N}, \\ & x_{ij} \in X_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \end{aligned}$$

where f_{ij} is a convex real-valued function of the flow x_{ij} of arc (i,j), s_i are given scalars, and X_{ij} are convex (bounded or unbounded) intervals of real numbers. We refer to this as the *separable convex cost network flow problem*, because the cost function separates into the sum of cost functions, one per arc.

Convex separable network optimization problems have special properties not shared by more general convex optimization problems, particularly with respect to duality. They admit solution with specialized methods that take advantage of separability, and which we are going to discuss in Chapter 4. A convex cost function may arise naturally for some problems, but it may also be obtained from a linear cost function through some form of regularization. By this we mean the addition of a convex function to the linear cost function, in order to transform the problem to a format that is algorithmically more convenient. This approach is often advocated for the problem discussed in the next example.

Example 1.2.7: The Matrix Balancing/Network Transport Problem

Here the problem is to find an $m \times n$ matrix X that has given row sums and column sums, and approximates a given $m \times n$ matrix M in some optimal manner.

We can formulate such a problem in terms of a graph consisting of m sources and n sinks. In this graph, the set of arcs $\mathcal{A}$ consists of the pairs (i,j) for which the corresponding entry x_{ij} of the matrix X is allowed to be nonzero. The given row sums r_i and the given column sums c_j are expressed as the constraints

$$\sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} = r_i, \qquad i = 1,\ldots,m,$$

$$\sum_{\{i|(i,j)\in\mathcal{A}\}} x_{ij} = c_j, \qquad j = 1,\ldots,n.$$

There may be also bounds for the entries x_{ij} of X. Thus, the structure of this problem is similar to the structure of a transportation problem. The cost function to be optimized has the form

$$\sum_{(i,j)\in\mathcal{A}} f_{ij}(x_{ij}),$$

and expresses the objective of making the entries of X close to the corresponding entries of the given matrix M. A common example is the quadratic function

$$f_{ij}(x_{ij}) = \sum_{(i,j)\in\mathcal{A}} w_{ij}(x_{ij} - m_{ij})^2,$$

where w_{ij} are given nonnegative weights.

Another interesting cost function is the logarithmic

$$f_{ij}(x_{ij}) = x_{ij}\left[\ln\left(\frac{x_{ij}}{m_{ij}}\right) - 1\right],$$

where we assume that $m_{ij} > 0$ for all $(i,j) \in \mathcal{A}$. Note that this function is not defined for $x_{ij} \leq 0$, so to obtain a problem that fits our framework, we must use a constraint interval of the form $X_{ij} = (0, \infty)$ or $X_{ij} = (0, c_{ij}]$, where c_{ij} is a positive scalar.

One of the first applications of this problem arose in telephone communications, where it was used for predicting the distribution matrix X of telephone traffic between m origins and n destinations. Here we are given the total supplies r_i of the origins and the total demands c_j of the destinations, and we are also given some matrix M that defines a nominal traffic pattern obtained from historical data (the name "matrix balancing" originated from this application context). The book by Censor and Zenios [CeZ97] discusses several related applications of the matrix balancing problem, as well as algorithms for its solution.

More recently the problem has received a lot of attention for the case where X is constrained to be a stochastic matrix. Within this context, it can be used to model instances of the famous *Monge-Ampere problem*, which dates from the 19th century, and is also referred in the literature as the *network transport problem*; see e.g., the books by Galichon [Gal16], Peyre and Cuturi [PeC19], Santambrogio [San15], and Villani [Vil09], [Vil21].

The starting points for some of the network transport literature are mathematically more complex infinite-dimensional versions of the problems that we consider here. In particular, the auction algorithms of this book apply to network transport problems only after they have been discretized and have been represented by a graph. Indeed, this is the most common approach in practice, and auction algorithms have been used widely for network transport computations; see e.g., the books noted above, and the papers by Brenier et al. [BFH03], Schmitzer [Sch16], [Sch19], Walsh and Dieci [WaD17], [WaD19], Merigot and Thibert [MeT21].

1.2.2 Transformations and Equivalences

We have seen that the assignment, shortest path, and transportation problems can be viewed as special cases of the transhipment problem. We will now describe reverse transformations, which can be used to convert a directed graph with general structure to a graph with *bipartite structure* (a graph whose nodes are divided into two subsets such that all arcs are outgoing from one subset and incoming to the other subset, as in assignment and transportation problems). There are two main ideas underlying these transformations:

(a) *Arc splitting*: Here we replace an arc (i,j) with zero arc lower bound ($b_{ij} = 0$) with a node labeled (i,j), and two arcs $\big(i,(i,j)\big)$ and

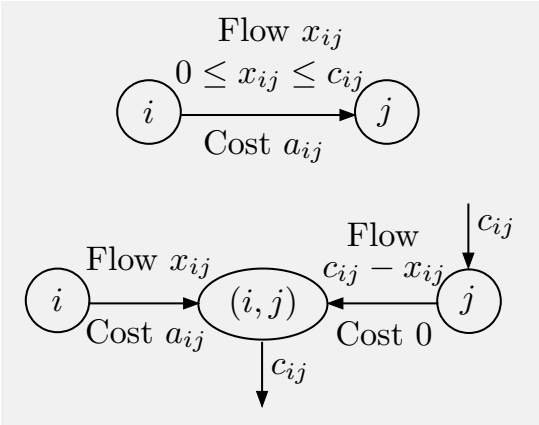


Figure 1.2.4. Illustration of arc splitting for an arc (i,j) with zero arc lower bound ($b_{ij} = 0$). For a given flow x_{ij} of arc (i,j) in the original graph, the flow on the arc $\big(i,(i,j)\big)$ is x_{ij}, and the flow on the arc $\big((i,j),j\big)$ is $c_{ij} - x_{ij}$. Supply and demand c_{ij} is added to node j and to node (i,j), respectively.

$\big(j,(i,j)\big)$ that are incoming to that node as shown in Fig. 1.2.4. Once all arcs have been split as described, the graph takes a bipartite form whereby the nodes are partitioned in two subsets: one that corresponds to the nodes $i \in \mathcal{N}$ of the original graph, and one that corresponds to the arcs $(i,j) \in \mathcal{A}$ of the original graph.

(b) *Node splitting*: Here we replace a zero supply node i ($s_i = 0$) with two nodes, i and i'. We then introduce an arc (i,i') connecting them, and we replace arcs of the form (i,j) by arcs with start node i (the exit point of i) and end node j' (the entry point of j), as shown in Fig. 1.2.5. This creates a bipartite graph structure whereby the nodes are partitioned in two subsets: the original nodes i whose incident arcs are all outgoing, and the artificial nodes i' whose incident arcs are all incoming.

Note that Fig. 1.2.4 and 1.2.5 show the general form of the graph structures involved, and neglect the presence of node supplies. These can be suitably added as needed, while preserving the bipartite structure of the graph that results from the transformation. Generally, arc splitting is used when working with the capacitated transhipment format, and node splitting is used when working with the uncapacitated transhipment format.

From Transhipment to Transportation to Assignment

We will now illustrate how a transhipment problem (capacitated or uncapacitated) can be transformed first to a transportation problem and then to an equivalent assignment problem; see Fig. 1.2.6. Indeed, we have seen

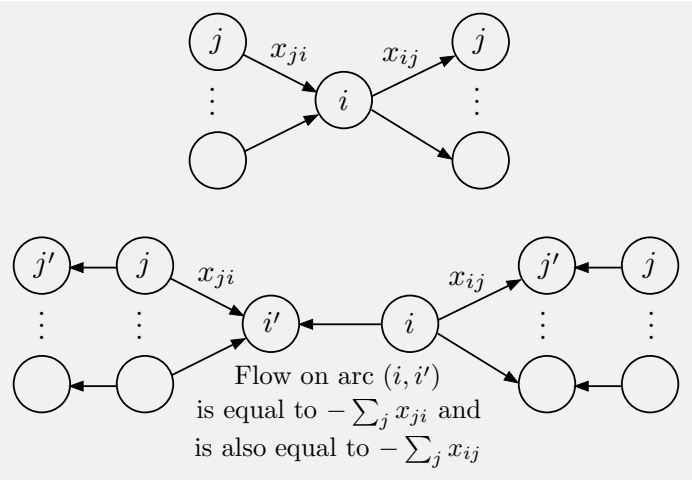


Figure 1.2.5. Illustration of node splitting for a node i that has supply equal to 0 ($s_i = 0$). For a given flow vector x in the original graph, the flow on the arc (i, i') is $-\sum_j x_{ji}$, which in turn is equal to $-\sum_j x_{ij}$.

that the feasibility case of the transhipment problem can be converted to a fixed flow problem with arc flow lower bound equal to 0; cf. Example 1.2.3. To convert this fixed flow problem to an equivalent transportation problem, we use arc splitting. In particular, we replace each arc (i, j) that is not incident to the source or the sink by a node labeled (i, j), and two arcs $\big(i, (i, j)\big)$ and $\big(j, (i, j)\big)$ that are incoming to that node. This is illustrated in Fig. 1.2.6 for the case where the lower flow bounds b_{ij} are equal to 0.

In the more general case of the transhipment problem where there are nonzero arc costs a_{ij}, we set to a_{ij} the cost of each arc $\big(i, (i, j)\big)$ in the transportation problem, and to 0 the cost of each arc $\big(j, (i, j)\big)$. Note that an arc flow x_{ij} in the minimum cost flow problem corresponds to flows equal to x_{ij} and $c_{ij} - b_{ij} - x_{ij}$ on the transportation problem arcs $\big((i, j), j\big)$ and $\big((i, j), i\big)$, respectively.

The transportation problem can in turn be transformed to an assignment problem with multiple identical/similar persons as follows (see Fig. 1.2.6):

(a) Create $\sum_{\{j \mid (j,i) \in \mathcal{A}\}} c_{ji}$ similar persons in place of each node/source $i \neq r, t$ of the transportation problem, and y_r persons in place of the source node r.

(b) Create c_{ij} duplicate objects in place of each arc/sink (i, j), $j \neq t$, of the transportation problem, and y_r duplicate objects in place of the sink node t.

(c) We set to $-a_{ij}$ the value of each pair of the form $\big(i, (i, j)\big)$ in the

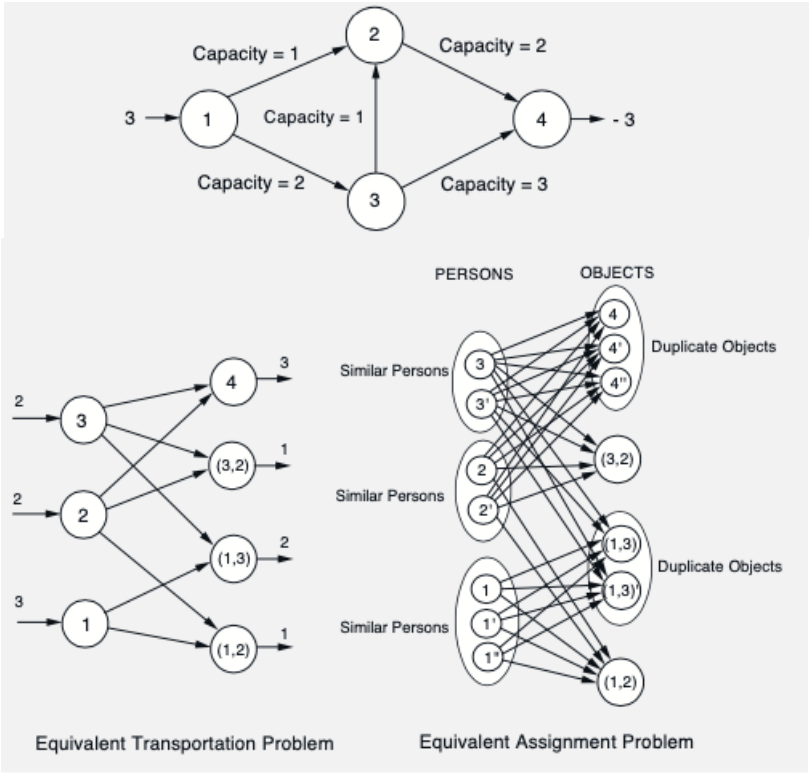


Figure 1.2.6. Transformation of a single-source/single-sink transhipment problem with zero arc flow lower bounds to a transportation problem ($b_{ij} = 0$), and then to an assignment problem.

To convert the transhipment problem to an equivalent transportation problem, we replace each arc (i, j) that is not incident to the source or the sink by a node labeled (i, j), and two arcs that are incoming to that node as shown: $\big(i, (i, j)\big)$ (with cost a_{ij}) and $\big(j, (i, j)\big)$ (with cost 0).

We then convert the transportation problem to an equivalent assignment problem, as shown. This requires that all node supplies of the transportation problem are integer.

assignment problem, and to 0 the value of each pair of the form $\big(j, (i, j)\big)$.

This transformation requires that all node supplies are integer. This is not a significant restriction, since any problem where the node supplies are rational numbers can be converted to a problem with integer supplies by multiplying all supplies and arc capacities with a suitably large number.

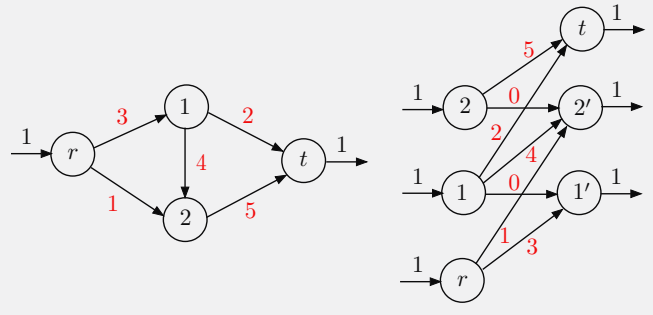


Figure 1.2.7. Transformation of a shortest path problem to an assignment problem from an origin node r to a destination node t. We use node splitting for the nodes 1 and 2. The arc costs are shown next to the arcs. For example, the path $(r, 1, 2, t)$ corresponds to the assignment

$$(r, 1'), (1, 2'), (2, t),$$

while the path $(r, 1, t)$ corresponds to the assignment $(r, 1'), (1, t), (2, 2')$ and the path $(r, 2, t)$ corresponds to the assignment $(r, 2'), (1, 1'), (2, t)$.

From Shortest Path to Assignment

Let us also consider the problem of finding a shortest path from an origin node r to a destination node t; cf. Example 1.2.2. We will show that it can be transformed to an equivalent assignment problem. We will use this transformation to motivate general forms of auction algorithms for the shortest path problem in Section 1.4.

The idea is based on node splitting and is illustrated with an example in Fig. 1.2.7. We split each node $i \neq r, t$ into two nodes, i and i', and connect them with an arc (i, i') of cost 0. Moreover, we replace the outgoing arcs (i, j) from all nodes $i \neq t$ with arcs (i, j'). Then the graph becomes bipartite and the problem is transformed to an assignment problem with persons representing the nodes $i \neq t$, objects representing the nodes j' and t, the costs $a_{ij'}$ being equal to the arc lengths a_{ij}, and the costs a_{it} being equal to the arc lengths a_{it}.

Any path from r to t of the form

$$(r, i_1, i_2, \ldots, i_k, t)$$

in the original shortest path problem corresponds to a feasible assignment, which contains the pairs

$$(r, i'_1), (i_1, i'_2), (i_2, i'_3), \ldots, (i_{k-1}, i'_k), (i_k, t), \tag{1.9}$$

and also possibly contains some additional pairs, such as the pairs (i, i') for $i \neq i_1, \ldots, i_k$. Moreover, it can be seen that the assignment problem has a

feasible solution only if there exists at least one path from r to t, which we will assume in our discussion.

Let us now look at the structure of a feasible assignment in more detail. It must consist of three groups of pairs:

(a) Pairs of the form (1.9), which correspond to a path $(r, i_1, i_2, \ldots, i_k, t)$ that does not contain a cycle. These pairs collectively have cost equal to the path length.

(b) Pairs of the form (i, i') that have cost 0.

(c) "Cycles" of pairs of the form

$$(i_1, i'_2), (i_2, i'_3), \ldots, (i_{k-1}, i'_k), (i_k, i'_1), \tag{1.10}$$

that collectively have cost equal to the lengths of the corresponding cycles

$$(i_1, i_2, \ldots, i_k, i_1).$$

Note that one or both of the sets of pairs in (b) and (c) above may be empty; for example in Fig. 1.2.7, for all the feasible assignments the set of pairs in (c) above is empty.

Consider now a feasible solution of the assignment problem. It must correspond to a path from r to t [as in (a) above], plus pairs of the form (i, i') [as in (b) above], which have zero length, plus pairs of the form (1.10) with corresponding cycles $(i_1, i_2, \ldots, i_k, i_1)$ [as in (c) above]. In this solution, the pairs (1.10) can be replaced by the zero cost pairs $(i_1, i'_1), \ldots, (i_k, i'_k)$ without affecting feasibility, so that the pairs (1.10) must collectively have cost that is either negative or zero in any optimal assignment. Thus, if we assume that all cycles of the original graph have nonnegative length, the pairs (1.10) must collectively have cost 0 in an optimal solution of the assignment problem. It follows that under the nonnegative cycle assumption, by solving the assignment problem we obtain a shortest path, and the cost of an optimal assignment is equal to the length of that path.

If the shortest path problem has a feasible solution but there are cycles of negative length, the corresponding assignment problem will also have an optimal solution that consist of pairs that yield a path from r to t of the form (a) that does not contain a cycle, and pairs of the form (b) and (c) above. However, the path from r to t may not be shortest because the solution of the assignment problem may be biased by the presence of the negative length cycles of the form (c) above.

A Final Word on Transformations

In this section, we have described a wide range of transformations between different types of network optimization problems. A key point has been that the assignment problem is the simplest general model, to which

all others can be reduced (assuming integer supplies, demands, and flow bounds). This fact has a special significance for the purposes of this book: we can start from an algorithm for the assignment problem (such as the intuitive auction algorithm to be described in Section 1.4), and apply it to the transportation or transhipment problem (including the shortest path and max-flow problems). We can then suitably streamline the algorithm to eliminate inefficiencies in the computations that are introduced by the problem transformation. Historically, this is the process through which auction algorithms for transportation, transhipment, shortest paths, and other problems have been discovered, understood, and analyzed; see the papers [Ber86], [Ber75], for the transhipment problem, [BeC88] for the transportation problem, [Ber91], [BeC92], [Ber22] for shortest path-type problems, and [Ber95] for the max-flow problem.

Let us also note that shortest path problems arise as subproblems within some important algorithmic approaches for the transhipment problem, such as the sequential shortest path/primal-dual method, which we will discuss briefly later (Chapter 6 of the book [Ber98] provides a detailed discussion). To solve these shortest path problems, we may use auction algorithms (to be discussed in Section 1.4), thus resulting in additional types of auction-like algorithms for transhipment.

1.3 PRIMAL AND DUAL PROBLEMS

Duality theory deals with the relation between the original network optimization problem and another optimization problem called the *dual*. This is an instance of linear programming duality, which is described in many optimization books; see e.g., Bertsimas and Tsitsiklis [BeT97]. The duality extends to convex network flow problems as well, as we will discuss in Chapter 4. To develop an intuitive understanding of duality, we will first focus on the $n \times n$ assignment problem and consider a closely related economic equilibrium problem.

1.3.1 Duality for the Assignment Problem

In the context of the assignment problem, let us consider matching the n objects with the n persons through a market mechanism, viewing each person as an economic agent acting in his/her own best interest. Suppose that object j has a price p_j and that the person who receives the object must pay the price p_j. Then the net value of object j for person i is $a_{ij} - p_j$, and each person i will logically want to be assigned to an object j_i with maximal value, that is, with

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{1.11}$$

where

$$A(i) = \big\{ j \mid (i,j) \in \mathcal{A} \big\}$$

is the set of objects that can be assigned to person i.

When this condition holds for all persons i, we say that the assignment and the price vector $p = (p_1, \ldots, p_n)$ satisfy *complementary slackness* (CS for short; this name is standard in linear programming). The economic system is then at equilibrium, in the sense that no person would have an incentive to unilaterally seek another object. Such equilibrium conditions are naturally of great interest to economists, but there is also a fundamental relation with the assignment problem.

We have the following classical proposition, which states that if we are able to obtain a feasible assignment and a set of prices that satisfy CS, then the assignment is optimal while the prices solve optimally a certain dual problem.

Proposition 1.3.1: If a feasible assignment and a set of prices satisfy the complementary slackness condition (1.11) for all persons i, then the assignment is optimal. Moreover the prices are an optimal solution of a dual problem, which is to minimize over $p = (p_1, \ldots, p_n)$ the cost function

$$\sum_{i=1}^{n} q_i(p) + \sum_{j=1}^{n} p_j,$$

where the functions q_i are given by

$$q_i(p) = \max_{j \in A(i)} \big\{ a_{ij} - p_j \big\}, \qquad i = 1, \ldots, n.$$

Furthermore, the value of the optimal assignment and the optimal cost of the dual problem are equal.

Proof: Let $\big\{ (i, j_i) \mid i = 1, \ldots, n \big\}$ and $\{ \overline{p}_j \mid j = 1, \ldots, n \}$ be the given assignment and set of prices, which satisfy the CS condition, so that

$$a_{ij_i} - \overline{p}_{j_i} = \max_{j \in A(i)} \{ a_{ij} - \overline{p}_j \}, \qquad i = 1, \ldots, n.$$

By adding this relation over all i, we obtain

$$\sum_{i=1}^{n} a_{ij_i} = \sum_{i=1}^{n} \left(\max_{j \in A(i)} \big\{ a_{ij} - \overline{p}_j \big\} + \overline{p}_{j_i} \right) = \sum_{i=1}^{n} \max_{j \in A(i)} \big\{ a_{ij} - \overline{p}_j \big\} + \sum_{j=1}^{n} \overline{p}_j. \tag{1.12}$$

On the other hand, the total value of any feasible assignment $\{(i,k_i) \mid i=1,\ldots,n\}$ satisfies

$$\sum_{i=1}^{n} a_{ik_i} = \sum_{i=1}^{n}(a_{ik_i} - \overline{p}_{k_i}) + \sum_{j=1}^{n}\overline{p}_j \leq \sum_{i=1}^{n}\max_{j\in A(i)}\left\{a_{ij} - \overline{p}_j\right\} + \sum_{j=1}^{n}\overline{p}_j. \quad (1.13)$$

By combining Eqs. (1.12) and (1.13), we have

$$\sum_{i=1}^{n} a_{ik_i} \leq \sum_{i=1}^{n} a_{ij_i},$$

for every feasible assignment $\left\{(i,k_i) \mid i=1,\ldots,n\right\}$, implying that the assignment $\left\{(i,j_i) \mid i=1,\ldots,n\right\}$ is optimal.

Moreover, similar to Eq. (1.13), we have for every set of prices $\{p_j \mid j=1,\ldots,n\}$,

$$\sum_{i=1}^{n} a_{ij_i} = \sum_{i=1}^{n}(a_{ij_i} - p_{j_i}) + \sum_{j=1}^{n} p_j \leq \sum_{i=1}^{n}\max_{j\in A(i)}\left\{a_{ij} - p_j\right\} + \sum_{j=1}^{n} p_j.$$

By combining this relation with Eq. (1.12), we obtain

$$\sum_{i=1}^{n} a_{ij_i} = \sum_{i=1}^{n}\max_{j\in A(i)}\left\{a_{ij} - \overline{p}_j\right\} + \sum_{j=1}^{n}\overline{p}_j \leq \sum_{i=1}^{n}\max_{j\in A(i)}\left\{a_{ij} - p_j\right\} + \sum_{j=1}^{n} p_j.$$

It follows that the set of prices $\{\overline{p}_j \mid j=1,\ldots,n\}$ is optimal for the dual problem, while the optimal values of the primal and dual problems are equal. **Q.E.D.**

The preceding proposition is a basis for a class of methods called *primal-dual*, which iteratively adjust both the assigned pairs (the primal variables) and the prices (the dual variables). Classical methods of this type, such as the Hungarian method due to Kuhn [Kuh55], maintain a set of object prices and an assignment that is not feasible (some persons and objects are unassigned), but satisfy the CS condition. They then try to improve iteratively the feasibility of the assignment, by progressively adding more pairs to the assignment, while maintaining the CS condition. The auction algorithm for the assignment problem is somewhat similar, but it involves a significant exception: it maintains a more relaxed version of the CS condition, called ϵ-*complementary slackness* (ϵ-CS for short), which will be introduced in the next section. While this may seem like a small difference, it leads to much different insights and algorithmic operation.

1.3.2 Duality for the Transhipment Problem

We will now consider the capacitated transhipment problem, and generalize the duality ideas just discussed for the assignment problem. In particular, we will obtain a dual problem by using a procedure that is standard in duality theory. We introduce a Lagrange multiplier p_i for the conservation of flow constraint for node i and we form the corresponding Lagrangian function

$$\begin{aligned} L(x,p) &= \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \left(s_i - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} + \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} \right) p_i \\ &= \sum_{(i,j)\in\mathcal{A}} (a_{ij} + p_j - p_i)x_{ij} + \sum_{i\in\mathcal{N}} s_i p_i. \end{aligned} \tag{1.14}$$

In analogy with the assignment problem, we will also call p_i the *price of node* i. We will also use p to denote the vector whose components are the prices p_i.

Let us now fix p and consider minimizing $L(x,p)$ with respect to x *without the requirement to meet the conservation of flow constraints*. It is seen that p_i may be viewed as a penalty per unit violation of the conservation of flow constraint. If p_i is too small (or too large), there is an incentive for positive (or negative, respectively) violation of the constraint. This suggests that we should search for the correct values p_i for which, when $L(x,p)$ is minimized over all capacity-feasible x, there is no incentive for either positive or negative violation of all the constraints.

We are thus motivated to introduce the dual function value $q(p)$ at a vector p, defined by

$$q(p) = \min_x \big\{ L(x,p) \mid b_{ij} \le x_{ij} \le c_{ij}, (i,j) \in \mathcal{A} \big\}. \tag{1.15}$$

Because the Lagrangian function expression is separable in the arc flows x_{ij}, its minimization decomposes into a separate minimization for each arc (i,j). Each of these minimizations can be carried out in closed form, yielding

$$q(p) = \sum_{(i,j)\in\mathcal{A}} q_{ij}(p_i - p_j) + \sum_{i\in\mathcal{N}} s_i p_i, \tag{1.16}$$

where

$$\begin{aligned} q_{ij}(p_i - p_j) &= \min_{b_{ij}\le x_{ij}\le c_{ij}} (a_{ij} + p_j - p_i)x_{ij} \\ &= \begin{cases} (a_{ij} + p_j - p_i)b_{ij} & \text{if } p_i \le a_{ij} + p_j, \\ (a_{ij} + p_j - p_i)c_{ij} & \text{if } p_i > a_{ij} + p_j. \end{cases} \end{aligned} \tag{1.17}$$

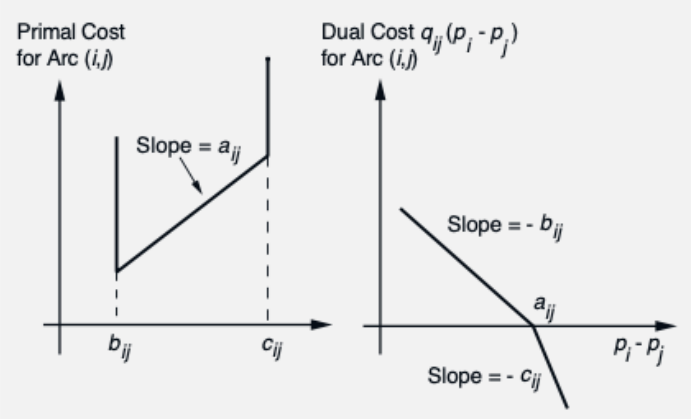


Figure 1.3.1: Form of the dual cost function q_{ij} for arc (i, j).

Figure 1.3.1 illustrates the form of the functions q_{ij}. Since each q_{ij} is piecewise linear, the dual function q is also piecewise linear. The dual function also has some additional interesting structure. In particular, suppose that all node prices are changed by the same amount. Then the values of the functions q_{ij} do not change, since these functions depend on the price differences $p_i - p_j$. If in addition we have $\sum_{i \in \mathcal{N}} s_i = 0$, as we must if the problem is feasible, we see that the term $\sum_{i \in \mathcal{N}} s_i p_i$ also does not change. Thus, the dual function value does not change when all node prices are changed by the same amount, implying that the equal cost surfaces of the dual cost function are unbounded. Figure 1.3.2 illustrates the dual function for a simple example.

Consider now the problem

$$\begin{aligned} &\text{maximize } q(p) \\ &\text{subject to no constraint on } p, \end{aligned}$$

where q is the dual function given by Eqs. (1.16)-(1.17). We call this the *dual problem*, and we refer to the original minimum cost flow problem as the *primal problem*.

To develop our basic duality result for the transhipment problem, we appropriately generalize the notion of complementary slackness, introduced earlier in the context of the assignment problem. In particular, we say that a flow-price vector pair (x, p) satisfies *complementary slackness* (or CS for short) if x is capacity-feasible and

$$p_i - p_j \leq a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } x_{ij} < c_{ij}, \tag{1.18}$$

$$p_i - p_j \geq a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } b_{ij} < x_{ij}. \tag{1.19}$$

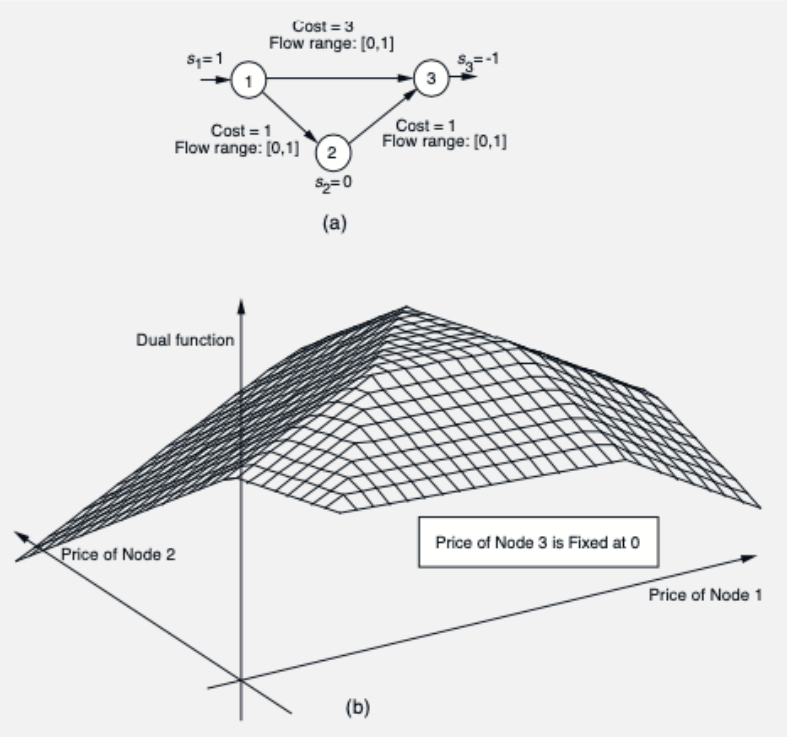


Figure 1.3.2: Form of the dual cost function q for the 3-node problem in (a). The optimal flow is $x_{12} = 1$, $x_{23} = 1$, $x_{13} = 0$. The dual function is

$$q(p_1, p_2, p_3) = \min\{0, 1 + p_2 - p_1\} + \min\{0, 1 + p_3 - p_2\} + \min\{0, 3 + p_3 - p_1\} + p_1 - p_3.$$

Diagram (b) shows the graph of the dual function in the space of p_1 and p_2, with p_3 fixed at 0. For a different value of p_3, say γ, the graph is "translated" by the vector (γ, γ); that is, we have $q(p_1, p_2, 0) = q(p_1 + \gamma, p_2 + \gamma, \gamma)$ for all (p_1, p_2). The dual function is maximized at the vectors p that satisfy CS together with the optimal x. These are the vectors of the form $(p_1 + \gamma, p_2 + \gamma, \gamma)$, where

$$1 \leq p_1 - p_2, \qquad p_1 \leq 3, \qquad 1 \leq p_2.$$

Note that the CS conditions imply that

$$p_i = a_{ij} + p_j, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } b_{ij} < x_{ij} < c_{ij}.$$

An equivalent way to write the CS conditions is that, for all arcs (i, j), we

have $b_{ij} \leq x_{ij} \leq c_{ij}$ and

$$x_{ij} = \begin{cases} c_{ij} & \text{if } p_i > a_{ij} + p_j, \\ b_{ij} & \text{if } p_i < a_{ij} + p_j. \end{cases}$$

Another equivalent way to state the CS conditions is that x_{ij} attains the minimum in the definition of q_{ij}

$$x_{ij} = \arg \min_{b_{ij} \leq z_{ij} \leq c_{ij}} (a_{ij} + p_j - p_i) z_{ij} \tag{1.20}$$

for all arcs (i, j). Figure 1.3.3 provides a graphical interpretation of the CS conditions.

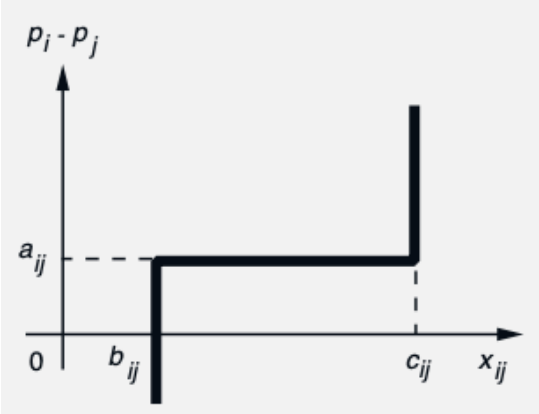


Figure 1.3.3: Illustration of CS for a flow-price pair (x, p). For each arc (i, j), the pair $(x_{ij}, p_i - p_j)$ should lie on the graph shown.

The following proposition is a classical duality theorem, which generalizes Prop. 1.3.1 for the assignment problem. It can also be viewed as a special case of well-known results in linear programming (see e.g., Bertsimas and Tsitsiklis [BeT97]).

Proposition 1.3.2: A feasible flow vector x^* and a price vector p^* satisfy CS for the transhipment problem if and only if x^* and p^* are optimal primal and dual solutions, respectively, and the optimal primal and dual costs are equal.

Proof: To show he forward part of the proposition, we will first show that for any feasible flow vector x and any price vector p, the primal cost of x is

no less than the dual cost of p. We will then use the CS condition to show that the primal value corresponding to x^* and the dual value corresponding to p^* are equal, thus completing the proof.

Indeed, we have from the definitions of L and q,

$$\begin{aligned} q(p) &\leq L(x,p) \\ &= \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij} + \sum_{i\in\mathcal{N}} \left(s_i - \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} + \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} \right) p_i \\ &= \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij}, \end{aligned} \tag{1.21}$$

where the last equality follows from the feasibility of x.

If x^* is feasible and satisfies CS together with p^*, we have by the definition (1.15) of q

$$\begin{aligned} q(p^*) &= \min_x \big\{ L(x,p^*) \mid b_{ij} \leq x_{ij} \leq c_{ij}, (i,j) \in \mathcal{A} \big\} \\ &= L(x^*,p^*) \\ &= \sum_{(i,j)\in\mathcal{A}} a_{ij}x^*_{ij}, \end{aligned} \tag{1.22}$$

where the second equality is true because

(x^*,p^*) satisfies CS if and only if

x^*_{ij} minimizes $(a_{ij} + p^*_j - p^*_i)x_{ij}$ over all $x_{ij} \in [b_{ij}, c_{ij}]$, $\forall\ (i,j) \in \mathcal{A}$,

[cf. Eq. (1.20)], and the last equality follows from the Lagrangian expression (1.14) and the feasibility of x^*. Therefore, Eq. (1.22) implies that x^* attains the minimum of the primal cost on the right-hand side of Eq. (1.21), and p^* attains the maximum of $q(p)$ on the left-hand side of Eq. (1.21), while the optimal primal and dual values are equal.

Conversely, suppose that x^* and p^* are optimal primal and dual solutions, respectively, and the two optimal costs are equal, that is,

$$q(p^*) = \sum_{(i,j)\in\mathcal{A}} a_{ij}x^*_{ij}.$$

We will show that x^* and p^* must satisfy CS. Indeed, we have by definition

$$q(p^*) = \min_x \big\{ L(x,p^*) \mid b_{ij} \leq x_{ij} \leq c_{ij}, (i,j) \in \mathcal{A} \big\},$$

and also, using the Lagrangian expression (1.14) and the feasibility of x^*,

$$\sum_{(i,j)\in\mathcal{A}} a_{ij}x^*_{ij} = L(x^*,p^*).$$

Combining the last three equations, we obtain

$$L(x^*, p^*) = \min_x \big\{ L(x, p^*) \mid b_{ij} \leq x_{ij} \leq c_{ij}, (i,j) \in \mathcal{A} \big\}.$$

Using the Lagrangian expression (1.14), it follows that for all arcs (i,j), we have

$$x^*_{ij} = \arg \min_{b_{ij} \leq x_{ij} \leq c_{ij}} (a_{ij} + p^*_j - p^*_i) x_{ij}.$$

This is equivalent to the pair (x^*, p^*) satisfying CS. **Q.E.D.**

Interpretation of CS and the Dual Problem

Similar to the assignment problem, the CS conditions have a nice economic interpretation. In particular, think of each node i as choosing the flow x_{ij} of each of its outgoing arcs (i,j) from the range $[b_{ij}, c_{ij}]$, on the basis of the following economic considerations: For each unit of the flow x_{ij} that node i sends to node j along arc (i,j), node i must pay a transportation cost a_{ij} plus a storage cost p_j at node j; for each unit of the residual flow $c_{ij} - x_{ij}$ that node i does not send to j, node i must pay a storage cost p_i. Thus, the total cost to node j is $(a_{ij} + p_j)x_{ij} + (c_{ij} - x_{ij})p_i$, or

$$(a_{ij} + p_j - p_i)x_{ij} + c_{ij}p_i.$$

It can be seen that the CS conditions (1.18) and (1.19) are equivalent to requiring that node i act in its own best interest by selecting the flow that minimizes the corresponding costs for each of its outgoing arcs (i,j); that is,

(x,p) satisfies CS if and only if
x_{ij} minimizes $(a_{ij} + p_j - p_i)z_{ij}$ over all $z_{ij} \in [b_{ij}, c_{ij}]$, $\forall\ (i,j) \in \mathcal{A}$,

[cf. Eq. (1.20)].

To interpret the dual function $q(p)$, we continue to view a_{ij} and p_i as transportation and storage costs, respectively. Then, for a given price vector p and supply vector s, the dual function

$$q(p) = \min_{\substack{b_{ij} \leq x_{ij} \leq c_{ij} \\ (i,j) \in \mathcal{A}}} \left\{ \sum_{(i,j) \in \mathcal{A}} a_{ij} x_{ij} + \sum_{i \in \mathcal{N}} \left(s_i - \sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij} + \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji} \right) p_i \right\} \tag{1.23}$$

is the minimum total transportation and storage cost to be incurred by the nodes, by choosing flows that satisfy the capacity constraints.

Suppose now that we introduce an organization that sets the node prices, and collects the transportation and storage costs from the nodes. We see that if the organization wants to maximize its total revenue (given that the nodes will act in their own best interest), it must choose prices that solve the dual problem optimally.

Duality for Uncapacitated Transhipment and Shortest Path Problems

In the case of the uncapacitated transhipment problem, a similar duality development is possible. It is based on introducing the Lagrangian function

$$L(x,p) = \sum_{(i,j)\in\mathcal{A}} (a_{ij} + p_j - p_i)x_{ij} + \sum_{i\in\mathcal{N}} s_i p_i,$$

of Eq. (1.14), and defining the dual function as

$$q(p) = \min_x \big\{ L(x,p) \mid 0 \leq x_{ij}, (i,j) \in \mathcal{A} \big\}. \tag{1.24}$$

However, in the absence of the upper bound arc flow constraints, this minimization will yield $-\infty$ unless the price vector satisfies the constraint

$$p_i - p_j \leq a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}.$$

There are also variants of CS and duality results for the uncapacitated transhipment problem. In particular, the CS conditions take the form

$$p_i - p_j \leq a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A}, \tag{1.25}$$

$$p_i - p_j = a_{ij}, \qquad \forall\ (i,j) \in \mathcal{A} \text{ with } 0 < x_{ij}, \tag{1.26}$$

(see Fig. 1.3.4). This is also the form of the CS conditions for the uncapacitated formulation of the shortest path problem where the constraints $x_{ij} \leq 1$ are discarded.

The counterpart of the duality result of most interest to us is the following. Its proof is similar to the ones we have given for the assignment problem in Prop. 1.3.1 and in Prop. 1.3.2 for the capacitated transhipment problem.

Proposition 1.3.3: For the uncapacitated transhipment problem, a feasible flow vector x^* and a price vector p^* satisfy the CS conditions (1.25)-(1.26) if and only if x^* and p^* are optimal primal and dual solutions, respectively, and the optimal primal and dual costs are equal.

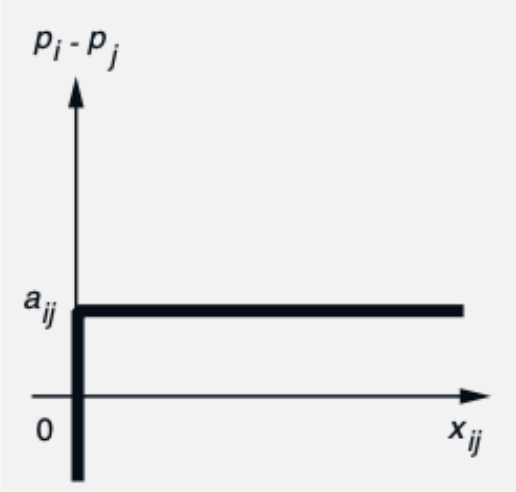


Figure 1.3.4: Illustration of CS for a flow-price pair (x, p) in the uncapacitated transhipment problem. The pair $(x_{ij}, p_i - p_j)$ should lie on the graph shown.

Let us now apply this result to a shortest path problem. It suggests the following sufficiency condition for optimality.

Proposition 1.3.4: Consider the shortest path problem. If we have a path $P = (r, i_1, \ldots, i_m, t)$ that starts at the origin r and ends at the destination t, and a price vector p satisfying the CS condition

$$p_i - p_j \leq a_{ij}, \quad \forall\ (i,j) \in \mathcal{A}, \qquad \text{and} \qquad p_i - p_j = a_{ij}, \quad \forall\ (i,j) \in P,$$

then P is a shortest path. Moreover, the shortest distance/length of P is equal to the price differential $p_r - p_t$.

Proof: Consider any path $P' = (r, i'_1, \ldots, i'_m, t)$ that starts at r and ends at t. By adding the CS conditions

$$\begin{gathered} p_r - p_{i'_1} \leq a_{ri'_1} \\ p_{i'_1} - p_{i'_2} \leq a_{i'_1 i'_2} \\ \cdots\cdots \\ p_{i'_{m-1}} - p_{i'_m} \leq a_{i'_{m-1} i'_m} \\ p_{i'_m} - p_t \leq a_{i'_m t} \end{gathered}$$

we obtain

$$p_r - p_t \leq \text{Length of } P'.$$

When applied to the path P, the preceding CS conditions hold with equality, so that we obtain

$$p_r - p_t = \text{Length of } P \leq \text{Length of } P',$$

for all paths P' that start at r and end at t. This proves the desired result. **Q.E.D.**

1.4 AUCTION ALGORITHMS

In this section, we will provide an introduction to auction algorithms. We will start with the assignment context, where the auction algorithm ideas are most intuitive, and then adapt these ideas to the shortest path problem. Both the assignment and the shortest path auction algorithms will play an important role in the development of various auction algorithms for the transhipment problem and some of its special cases, which will be discussed in Chapter 4.

1.4.1 The Naive Auction Algorithm

Let us return to the assignment problem, involving n persons and n objects, as described in Example 1.2.1. We will first consider a natural process for finding an equilibrium assignment and price vector. We will call this process the *"naive" auction algorithm*, because it has a serious flaw, as we will see shortly. Nonetheless, this flaw will help motivate a more sophisticated and correct algorithm.

The naive auction algorithm proceeds in iterations and generates a sequence of price vectors and partial assignments. By a *partial assignment* we mean an assignment where only a subset of the persons have been matched with objects. A partial assignment should be contrasted with a *feasible* or *complete* assignment where all the persons have been matched with objects on a one-to-one basis. At the beginning of each iteration, the CS condition

$$a_{ij_i} - p_{j_i} = \max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{1.27}$$

[cf. Eq. (1.11)], is satisfied for all pairs (i, j_i) of the partial assignment. If all persons are assigned, the algorithm terminates. Otherwise some person who is unassigned, say i, is selected. This person finds the "best" object j_i

$$j_i = \arg \max_{j \in A(i)} \{a_{ij} - p_j\},$$

i.e., one that offers maximal value, and then:

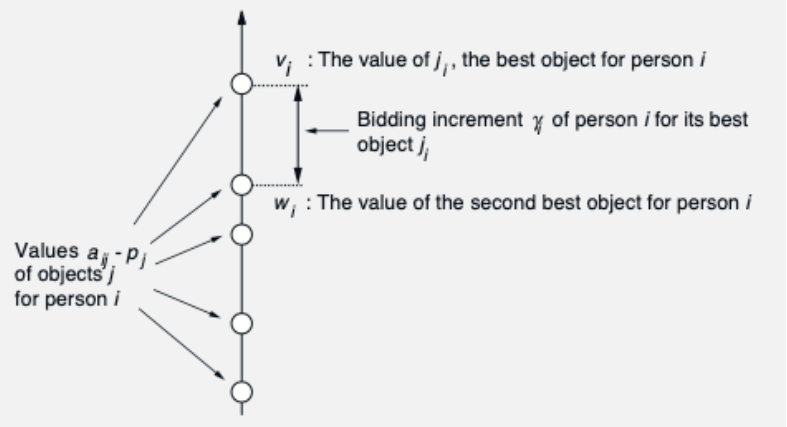


Figure 1.4.1: In the naive auction algorithm, even after the price of the best object j_i is increased by the bidding increment γ_i, j_i continues to be the best object for the bidder i, so the CS condition (1.27) is satisfied at the end of the iteration. However, we have $\gamma_i = 0$ if there is a tie between two or more objects that are most preferred by i.

(a) Gets assigned to the best object j_i; the person who was assigned to j_i at the beginning of the iteration (if any) becomes unassigned.

(b) Sets the price of j_i to the level at which he/she is indifferent between j_i and the second best object; that is, he/she sets p_{j_i} to

$$p_{j_i} + \gamma_i, \tag{1.28}$$

where

$$\gamma_i = v_i - w_i, \tag{1.29}$$

v_i is the best object value,

$$v_i = \max_{j \in A(i)} \{a_{ij} - p_j\}, \tag{1.30}$$

and w_i is the second best object value,

$$w_i = \max_{j \in A(i),\, j \neq j_i} \{a_{ij} - p_j\}. \tag{1.31}$$

This process is repeated in a sequence of iterations until each person has been assigned to an object.

We may view this process as an auction where at each iteration the bidder i raises the price of a preferred object by the *bidding increment* γ_i. The choice γ_i is illustrated in Fig. 1.4.1. Note that γ_i, as given by Eq. (1.29), is nonnegative and it is the largest increment by which p_{j_i} can be increased, while maintaining the property that j_i offers maximal value to i. Just as in a real auction, bidding increments and price increases spur competition by making the bidder's own preferred object less attractive to other potential bidders. As Fig. 1.4.1 illustrates, the CS condition (1.27) is satisfied at the end of a naive auction iteration.

Unfortunately, the naive auction algorithm does not always work. The difficulty is that the bidding increment γ_i is 0 when two or more objects are tied in offering maximum value for the bidder i. As a result, a situation may be created where several persons contest a smaller number of equally desirable objects without raising their prices, thereby creating a never ending cycle. An example is shown in Fig. 1.4.2.

In the next section we will modify the naive auction algorithm to correct this flaw. We note, however, that despite its convergence difficulties, the naive auction algorithm is an excellent initialization procedure for other methods, as we will discuss in more detail in Chapter 2. The reason is that, with proper implementation, it is typically capable of producing very quickly a partial assignment that contains all but very few persons and objects, which can in turn be assigned by switching to a primal-dual method that has guaranteed convergence. This approach was first proposed and validated experimentally in the author's paper [Ber81], and can also be supported by computation complexity analysis that will be noted in Chapter 2. The idea of using naive auction for initialization followed by a primal dual method was pursued by Jonker and Volgenant in the paper [JoV87], who (working from the author's assignment code) made publicly available a coded implementation, often referred to as the JV code, which is widely used at present.

1.4.2 ϵ-Complementary Slackness and the Auction Algorithm

In this section, we will introduce a perturbation mechanism to deal with the cycling phenomenon illustrated in Fig. 1.4.2. It is motivated by real auctions where each bid for an object must raise its price by a minimum positive increment, and bidders must on occasion take risks to win their preferred objects. In particular, let us fix a scalar $\epsilon > 0$, and say that a partial assignment and a price vector p satisfy *ϵ-complementary slackness (ϵ-CS for short)* if

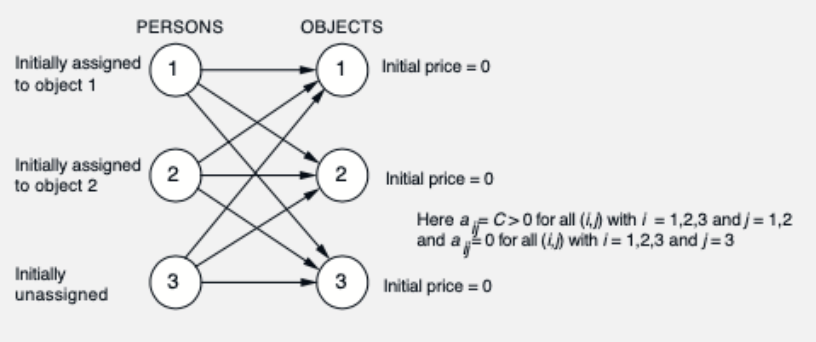
$$a_{ij} - p_j \geq \max_{k \in A(i)} \{a_{ik} - p_k\} - \epsilon$$

for all assigned pairs (i, j) within the given partial assignment. In words, to satisfy ϵ-CS, all assigned persons of the partial assignment must be assigned to objects that are within ϵ of being best.

We now reformulate the previous auction process so that the bidding increment is always at least equal to ϵ, while ϵ-CS (rather than CS) is maintained. The resulting method, the *auction algorithm*, is the same as the naive auction algorithm, except that the bidding increment γ_i is

$$\gamma_i = v_i - w_i + \epsilon \tag{1.32}$$

rather than $\gamma_i = v_i - w_i$ as in Eq. (1.29). With this choice, the ϵ-CS condition is satisfied, as illustrated in Fig. 1.4.3. The particular increment



At Start of Iteration #	Object Prices	Assigned Pairs	Bidder	Preferred Object	Bidding Increment
1	0,0,0	(1,1), (2,2)	3	2	0
2	0,0,0	(1,1), (3,2)	2	2	0
3	0,0,0	(1,1), (2,2)	3	2	0

Figure 1.4.2: Illustration of how the naive auction algorithm may never terminate for a problem involving three persons and three objects. Here objects 1 and 2 offer value $C > 0$ to all persons, and object 3 offers value 0 to all persons. The algorithm cycles as persons 2 and 3 alternately bid for object 2 without changing its price because they prefer equally object 1 and object 2.

$\gamma_i = v_i - w_i + \epsilon$ used in the auction algorithm is the maximum amount with this property. Smaller increments γ_i would also work as long as $\gamma_i \geq \epsilon$, but using the largest possible increment accelerates the convergence of the algorithm. This is consistent with experience from real auctions, which tend to terminate faster when the bidding is aggressive.

It can be shown that this reformulated auction process terminates, necessarily with a feasible assignment and a set of prices that satisfy ϵ-CS. To get an intuitive sense of this, note that if an object receives a bid during m iterations, its price must exceed its initial price by at least $m\epsilon$. Thus, for sufficiently large m, the object will become "expensive" enough to be judged "inferior" to some object that has not received a bid so far. It follows that an object can receive a bid only for a limited number of iterations, while some other object still has not yet received any bid. On the other hand, once every object has received at least one bid, the auction terminates. (This argument assumes that any person can bid for any object, but it can be generalized to the case where the set of feasible person-object pairs is limited, as long as at least one feasible assignment exists; see Chapter 2.) Figure 1.4.4 shows how the auction algorithm, based

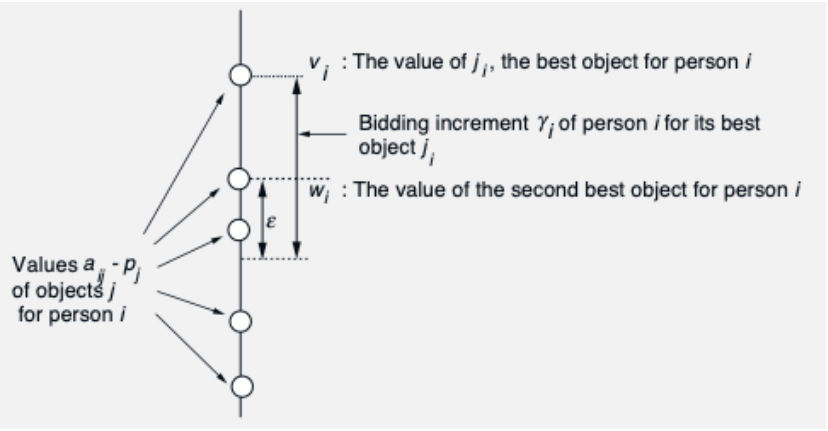


Figure 1.4.3: In the auction algorithm, even after the price of the preferred object j_i is increased by the bidding increment γ_i, j_i will be within ϵ of being most preferred, so the ϵ-CS condition holds at the end of the iteration.

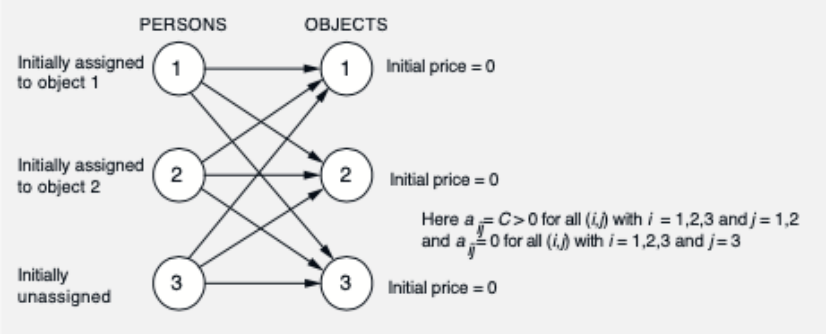
on the bidding increment $\gamma_i = v_i - w_i + \epsilon$ of Eq. (1.32) overcomes the cycling difficulty in the example of Fig. 1.4.2.

When the auction algorithm terminates, we have an assignment satisfying ϵ-CS, but is this assignment optimal? The answer depends strongly on the size of ϵ. In a real auction, a prudent bidder would not place an excessively high bid for fear of winning the object at an unnecessarily high price. Consistent with this intuition, we can show that if ϵ is small, then the final assignment will be "almost optimal." In particular, we will show that *the total value of the final assignment is within $n\epsilon$ of being optimal*. The idea is that a feasible assignment and a set of prices satisfying ϵ-CS may be viewed as satisfying CS for a *slightly different* problem, where all values a_{ij} are the same as before except the values of the n assigned pairs, which are modified by no more than ϵ.

Proposition 1.4.1: A feasible assignment satisfying ϵ-CS, together with some price vector, attains within $n\epsilon$ the optimal primal value. Furthermore, the price vector attains within $n\epsilon$ the optimal dual cost.

Proof: Let A^* be the optimal total assignment value

$$A^* = \max_{\substack{k_i,\, i=1,\ldots,n \\ k_i \neq k_m \text{ if } i \neq m}} \sum_{i=1}^{n} a_{ik_i}$$



At Start of Iteration #	Object Prices	Assigned Pairs	Bidder	Preferred Object	Bidding Increment
1	0,0,0	(1,1), (2,2)	3	2	ϵ
2	0,ϵ,0	(1,1), (3,2)	2	1	2ϵ
3	2ϵ,ϵ,0	(2,1), (3,2)	1	2	2ϵ
4	2ϵ,3ϵ,0	(1,2), (2,1)	3	1	2ϵ
5	4ϵ,3ϵ,0	(1,2), (3,1)	2	2	2ϵ
6	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

Figure 1.4.4: Illustration of how the auction algorithm, by making the bidding increment at least ϵ, overcomes the cycling difficulty for the example of Fig. 1.4.2. The table shows one possible sequence of bids and assignments generated by the auction algorithm, starting with all prices equal to 0 and with the partial assignment $\{(1,1),(2,2)\}$. At each iteration except the last, the person assigned to object 3 bids for either object 1 or 2, increasing its price by ϵ in the first iteration and by 2ϵ in each subsequent iteration. In the last iteration, after the prices of 1 and 2 reach or exceed C, object 3 receives a bid and the auction terminates.

and let D^* be the optimal dual cost (cf. Prop. 1.3):

$$D^* = \min_{\substack{p_j \\ j=1,\ldots,n}} \left\{ \sum_{i=1}^{n} \max_{j \in A(i)} \{a_{ij} - p_j\} + \sum_{j=1}^{n} p_j \right\}.$$

If $\{(i, j_i) \mid i = 1, \ldots, n\}$ is the given assignment satisfying the ϵ-CS condition together with a price vector $\overline{p}$, we have

$$\max_{j \in A(i)} \{a_{ij} - \overline{p}_j\} - \epsilon \leq a_{ij_i} - \overline{p}_{j_i}.$$

By adding this relation over all i, we see that

$$D^* \leq \sum_{i=1}^{n} \left(\max_{j \in A(i)} \{a_{ij} - \overline{p}_j\} + \overline{p}_{j_i} \right) \leq \sum_{i=1}^{n} a_{ij_i} + n\epsilon \leq A^* + n\epsilon.$$

Since we showed in Prop. 1.3.1 that $A^* = D^*$, it follows that the total assignment value $\sum_{i=1}^{n} a_{ij_i}$ is within $n\epsilon$ of the optimal value A^*, while the dual cost of $\overline{p}$ is within $n\epsilon$ of the optimal dual cost. **Q.E.D.**

Suppose now that the values a_{ij} are all integer, which is the typical practical case. (If a_{ij} are rational numbers, they can be scaled up to integer by multiplication with a suitable common number.) Then the total value of any assignment is integer, so if $n\epsilon < 1$, any complete assignment that is within $n\epsilon$ of being optimal must be optimal. It follows that *if $\epsilon < \frac{1}{n}$ and the values a_{ij} are all integer, then the assignment obtained upon termination of the auction algorithm is optimal*.

Figure 1.4.5 shows the sequence of generated object prices for the example of Fig. 1.4.4 in relation to the contours of the dual cost function. It can be seen from this figure that each bid has the effect of setting the price of the object receiving the bid nearly equal (within ϵ) to the price that minimizes the dual cost with respect to that price, with all other prices held fixed (this will be shown in Section 2.1). Successive minimization of a cost function along single coordinates is a central feature of coordinate descent and relaxation methods, which are popular for unconstrained minimization of smooth functions and for solving systems of smooth equations. Thus, the auction algorithm can be interpreted as an *approximate coordinate descent method* for solving the dual problem, as we will discuss in Chapter 2.

ϵ-Scaling

Figure 1.4.5 also illustrates a generic feature of auction algorithms. The amount of work needed to solve the problem can depend strongly on the value of ϵ and on the maximum absolute object value

$$C = \max_{(i,j)\in\mathcal{A}} |a_{ij}|.$$

Basically, for many types of problems, the number of iterations up to termination tends to be proportional to C/ϵ. This can be seen from the figure, where the total number of iterations is roughly C/ϵ, starting from zero initial prices. Thus the algorithm can be quite slow, with a nonpolynomial complexity, a potential inefficiency that we will aim to correct with an approach that we will now describe.

As a first step in this direction, we note that there is a dependence on the initial prices: if these prices are "near optimal," we expect that the number of iterations needed to solve the problem will be relatively small. This can be seen from Fig. 1.4.5; if the initial prices satisfy $p_1 \approx p_3 + C$ and $p_2 \approx p_3 + C$, the number of iterations up to termination is quite small.

This suggests the idea of *ϵ-scaling*, which was noted in the original proposal of the auction algorithm for the assignment problem [Ber79]. It consists of applying the algorithm several times, with each application providing good initial prices for the next application. In the case when a_{ij}

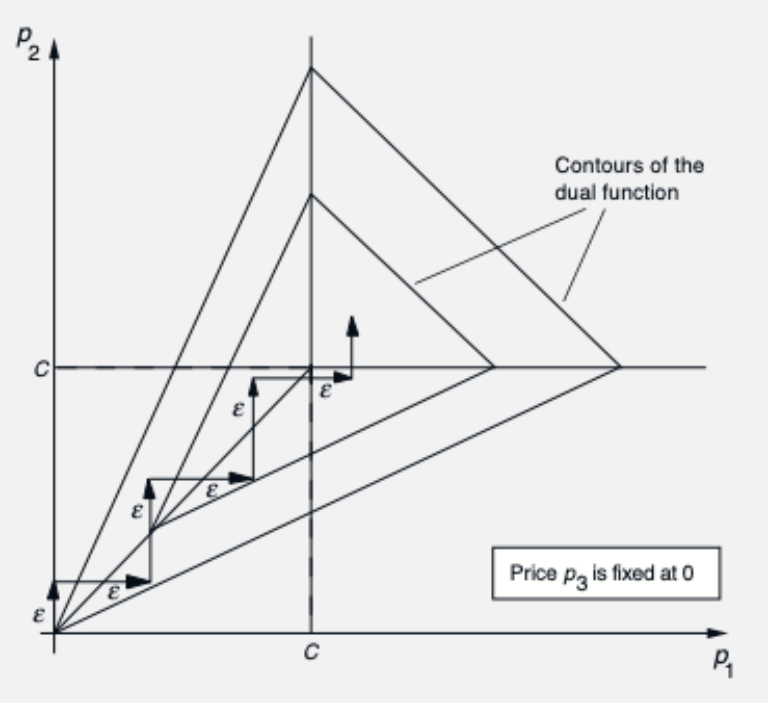


Figure 1.4.5: A sequence of prices p_1 and p_2 generated by the auction algorithm for the example of Figs. 1.4.2 and 1.4.4. The figure shows the equal dual cost surfaces in the space of p_1 and p_2, with p_3 fixed at 0. The arrows indicate the price iterates as given by the table of Fig. 1.4.4. Termination occurs when the prices reach an ϵ-neighborhood of the point (C, C), and object 3 becomes "sufficiently inexpensive" to receive a bid and to get assigned. The total number of iterations is roughly C/ϵ, starting from zero initial prices.

are integer, reducing ϵ until it is less than $1/n$ yields an optimal solution. Moreover, ϵ-scaling also allows the option of stopping the algorithm, with a less refined solution, if the allotted time for computation is limited.

In practice, ϵ-scaling is typically beneficial, and sometimes dramatically so. It often accelerates the termination of the algorithm, particularly for problems with a sparse structure (the set of possible pairs $\mathcal{A}$ is relatively small), and it also leads to better (polynomial) complexity estimates, as we will show in Chapter 2.

1.4.3 Auction Algorithms for Path Planning

We will now discuss auction-based algorithmic ideas for finding a shortest path from an origin r to a destination t; cf. Example 1.2.2. These ideas are based on the shortest path to assignment transformation described in Fig. 1.2.7, which uses node splitting for all nodes $i \neq r, t$, i.e., for each $i \neq r, t$

we introduce a node i', and an arc (i, i') of cost 0. Moreover, we replace the outgoing arcs (i, j) from all nodes $i \neq t$ with arcs (i, j'). This creates a bipartite graph structure and an assignment problem to which the auction algorithm can be applied.

One possibility is to start the auction algorithm with the partial assignment that consists of all the pairs (i, i'), $i \neq r, t$, along with node prices that satisfy ϵ-CS. Another possibility is to consider the equivalent assignment problem, and to simply start the auction algorithm with the empty assignment and arbitrary node prices.

These two possibilities lead to somewhat different algorithms, but they both entail a very similar structure. In particular, the resulting algorithm involves the idea of maintaining node prices, and an acyclic path that starts at the origin r and is iteratively extended or contracted by adding a new node at the end of the path, or deleting the terminal node of the path. The reader may verify that the extension and contraction operations are related to the bidding operations of the auction algorithm, as applied to the equivalent assignment problem illustrated in Fig. 1.2.7. Rather than focus on the details of the corresponding algorithmic relations and equivalences, we will simply adapt the auction process directly to the shortest path problem, taking advantage of its intuitive character.

In particular, auction algorithms for the shortest path problem maintain a path that starts at the origin and is iteratively extended or contracted. The decision to extend or to contract is based on a set of price variables, one for each node. Roughly speaking, the price of a node is viewed as a measure of the desirability of revisiting and advancing from that node in the future (low-price nodes are viewed as more desirable). Once the destination becomes the terminal node of the path, the algorithm stops with a path from origin to destination, which is near optimal or exactly optimal under certain conditions.

To get an intuitive sense of the algorithm, think of a mouse searching through a graph-like maze, trying to reach the destination. The mouse criss-crosses the maze, either advancing or backtracking along its current path, guided by "learned" experience; see Fig. 1.4.7 for an illustration.

Our algorithm, called *auction shortest path* algorithm (ASP for short), uses a "price" for each node, which provides a measure of desirability for including the node into a shortest path. We will provide details later, but roughly speaking, the mouse advances forward from high price to low price nodes, going from a node to a downstream neighbor node only if that neighbor has lower price (or equal price under some conditions). It backtracks when it reaches a node whose downstream neighbors all have higher price. In this case, it also suitably increases the price of that node, thus marking the node as less desirable for future exploration, and providing an incentive to explore alternative paths to the destination. An important side benefit is that the prices provide the means to "transfer knowledge," in the sense that good learned prices from previous searches can be used as initial prices

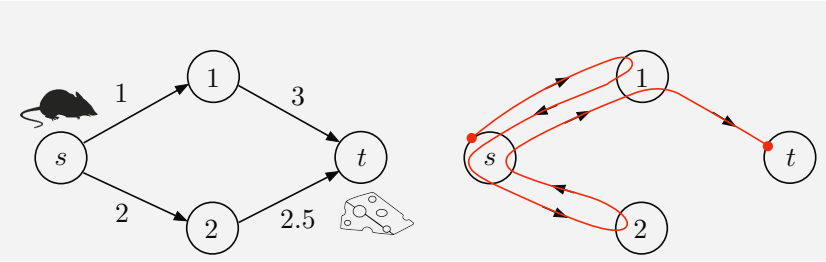


Figure 1.4.7: In this example the shortest path from s to t is $(s, 1, t)$ with shortest distance 4. The mouse starts by going greedily to node 1, but also making a mental note that there is an alternative choice to go to node 2 at cost 2. Upon reaching node 1, the mouse sees that going to t involves an additional cost 3. So the mouse returns to s to explore the possibility that going through node 2 may be better, but also makes a mental note that the path $(s, 1, t)$ has length 4. The mouse then tries node 2, and discovers that the path to t through node 2 has length 4.5, which is greater than the length of the path $(s, 1, t)$ already discovered. The mouse then returns to s and then moves to node 1 and then to the destination t. The precise details of the algorithm, using a system of node prices to guide the search, will be given later in this section; see Fig. 1.4.9.

for subsequent related searches, with an attendant computational speedup.

We will now introduce our ASP algorithm. We assume that *all cycles have nonnegative length*. By this we mean that for every cycle $(i, n_1, \ldots, n_k, i)$ we have

$$a_{in_1} + a_{n_1 n_2} + \cdots + a_{n_{k-1} n_k} + a_{n_k i} \geq 0. \tag{1.33}$$

This is a common assumption in shortest path problems, since a negative length cycle can be incorporated an arbitrarily large times within a feasible path to reduce its length to $-\infty$. We also assume for simplicity that every node $i \neq t$ is not deadend, in the sense that its set of outgoing arcs, $\big\{(i, j) \mid j \in \mathcal{N}\big\}$, is nonempty.

The ASP Algorithm

The ASP algorithm maintains and updates a path with no cycles that starts at the origin,

$$P = (r, n_1, \ldots, n_k),$$

and a price p_i for each node i. The algorithm terminates when n_k becomes the destination t.

Let us introduce some terminology. If P is not the degenerate path $P = (r)$, its last node n_k is called the *terminal node* of P, and its node n_{k-1} is denoted by

$$\text{pred}(n_k) = n_{k-1}$$

[in the case where $P = (s, n_1)$, we let $\text{pred}(n_1) = s$]. We denote by $\text{succ}(n_k)$ a downstream neighbor j of n_k for which $a_{n_k j} + p_j$ is minimized:

$$\text{succ}(n_k) \in \arg \min_{\{j \mid (n_k, j) \in \mathcal{A}\}} \{a_{n_k j} + p_j\};$$

[if multiple downstream neighbors of n_k attain the minimum, the algorithm designates arbitrarily one of these neighbors as $\text{succ}(n_k)$].

We say that under the current set of prices and lengths an arc (i, j) is:

(a) *Downhill*: If $p_i > a_{ij} + p_j$.

(b) *Level*: If $p_i = a_{ij} + p_j$.

(c) *Uphill*: If $p_i < a_{ij} + p_j$.

We will now describe the rules by which the path and the prices are updated. At any one iteration the algorithm starts with a path P that starts at the origin and a scalar price p_i for each node i. At the end of the iteration a new path $\overline{P}$ is obtained from P through a contraction or an extension as earlier. For iterations where the algorithm starts with the degenerate path $P = (s)$, only an extension is possible, i.e., $P = (s)$ is replaced by a path of the form $\overline{P} = (s, n_1)$. Also the price of the terminal node of P is increased just before a contraction, and in some cases, just before an extension.

The algorithm starts with the degenerate path $P = (s)$, and terminates when the destination becomes the terminal node of P. It makes use of a positive parameter ϵ, which plays a similar role to the one of the corresponding paparemeter for the aufction algorithm for the assignment problem. The rules by which the path P and the prices p_i are updated at each iteration are as follows.

ASP Iteration for Shortest Path Construction:

We distinguish two cases.

(a) $P = (s)$: We then set the price p_s to

$$\max\{p_s,\, a_{s\text{succ}(s)} + p_{\text{succ}(s)} + \epsilon\},$$

and extend P to $\text{succ}(s)$.

(b) $P = (s, n_1, \ldots, n_k)$ *with* $n_k \neq s$. We consider the following two cases.

(1)

$$p_{\text{pred}(n_k)} > a_{\text{pred}(n_k) n_k} + a_{n_k \text{succ}(n_k)} + p_{\text{succ}(n_k)}.$$

We then extend P to $\text{succ}(n_k)$ and set p_{n_k} to any price level that makes the arc $\big(\text{pred}(n_k), n_k\big)$ level or downhill and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill. [Setting

$$p_{n_k} = p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k},$$

thus raising p_{n_k} to the maximum possible level, is a possibility, in which case the arc $\big(\text{pred}(n_k), n_k\big)$ becomes level.]

(2)

$$p_{\text{pred}(n_k)} \leq a_{\text{pred}(n_k)n_k} + a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)}.$$

We then contract P to $\text{pred}(n_k)$ and raise the price of n_k to

$$a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)} + \epsilon$$

[thus making the arcs $\big(\text{pred}(n_k), n_k\big)$ and $\big(n_k, \text{succ}(n_k)\big)$ uphill and downhill, respectively].

Figure 1.4.8 illustrates the price change operations of the algorithm under an extension and under a contraction. A key fact that can be easily verified is that P and the prices p_i satisfy the following downhill path property at the start of each iteration for which $P \neq (s)$.

Downhill Path Property:

All arcs of the path $P = (s, n_1, \ldots, n_k)$ maintained by the ASP algorithm are level or downhill. Moreover, the last arc (n_{k-1}, n_k) of P is downhill following an extension to n_k.

The significance of the downhill path property is that *when an extension occurs, a cycle cannot be created*, in the sense that the terminal node n_k is different than all the predecessor nodes $s, n_1, \ldots, n_{k-1}$ on the path P. The reason is that the downhill path property implies that following an extension, we must have

$$p_s \geq a_{sn_1} + p_{n_1},$$

$$p_{n_1} \geq a_{n_1n_2} + p_{n_2},$$

$$\ldots$$

$$p_{n_{k-2}} \geq a_{n_{k-2}n_{k-1}} + p_{n_{k-1}},$$

$$p_{n_{k-1}} > a_{n_{k-1}n_k} + p_{n_k}.$$

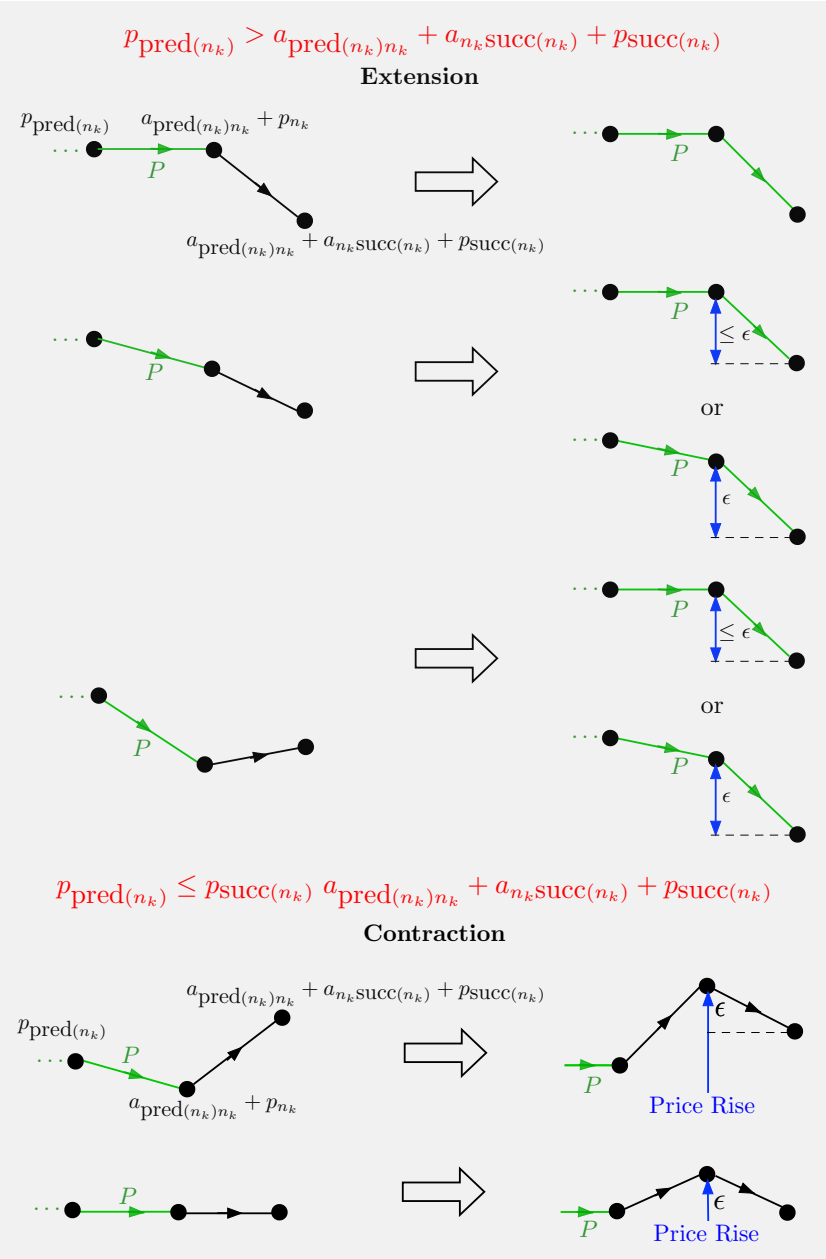


Figure 1.4.8 Illustration of an iteration of the ASP algorithm for the case where $P = (s, n_1, \ldots, n_k)$ with $n_k \neq s$. The figure shows the levels

$$p_{\text{pred}(n_k)}, \quad a_{\text{pred}(n_k)n_k} + p_{n_k}, \quad a_{\text{pred}(n_k)n_k} + a_{n_k\text{succ}(n_k)} + p_{\text{succ}(n_k)},$$

before and after an extension (top figure), and before and after a contraction (bottom figure).

This shows that the terminal node n_k following an extension cannot be equal to any of the preceding nodes n_i of P: if it were, by adding the preceding inequalities along the cycle $(n_i, n_{i-1}, \ldots, n_k)$, it follows that the length of this cycle is negative, which a contradiction.

In addition to maintaining the downhill path property, the algorithm is structured so that following a contraction of a nondegenerate path $P = (s, n_1, \ldots, n_k)$, the price of its terminal node n_k is increased by a positive amount. In conjunction with the fact that P never contains a cycle, this implies that either the algorithm terminates, or some node prices will increase to infinity. This is the key idea that underlies the validity of the algorithm, and forms the basis for its proof of termination, which will be formally shown in Chapter 3.

The parameter ϵ in the ASP algorithm is used to regulate the size of price rises, similar to the auction algorithm for the assignment problem. In particular, ϵ is used to provide an important tradeoff between the ability of the algorithm to construct paths with near-minimum length, and its rate of convergence. Generally, as ϵ becomes smaller the quality of the path produced improves, as we will show with examples and analysis in what follows. On the other hand a small value of ϵ tends to slow down the algorithm. Thus the role of ϵ is the ASP algorithm is similar to the role of ϵ in the auction algorithm. This is not surprising, since the ASP algorithm can be viewed as essentially a special case of the auction algorithm. We will elaborate on these issues in Chapter 3. Figure 1.4.9 illustrates the ASP algorithm with the example of Fig. 1.4.7.

Shortest Distances and Error Bounds

As noted earlier, the ASP algorithm need not produce a shortest path. The deviation from optimality of depends on the initial prices, as well as the parameter ϵ. In Section 3.2, we will elaborate on this issue and quantify this dependence. Particular, we will show that if the ASP algorithm terminatea with a path P, and P' is any other path from s to t, we have

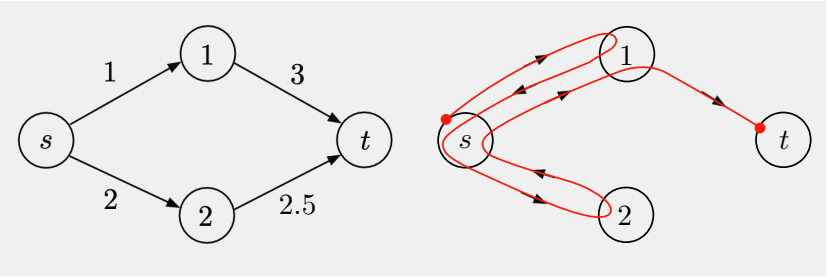
$$L_P + \sum_{(i,j)\in P} d_{ij} \leq L_{P'} + \sum_{(i,j)\in P'} d_{ij}, \tag{1.34}$$

where L_P and $L_{P'}$ are the total lengths of P and P', and d_{ij} are given by

$$d_{ij} = \max\{0,\, p_i - a_{ij} - p_j\}, \qquad (i,j) \in \mathcal{A}. \tag{1.35}$$

The scalars d_{ij} will be referred to as the *discrepancies* of the arcs (i,j), and they quantify the error from optimality of the path P generated by the algorithm. In particular, if all the arc discrepancies d_{ij}, $(i,j) \in \mathcal{A}$, are zero, the path P is shortest.

An interesting empirical observation is that when the algorithm creates a new downhill arc (i,j) that lies outside P, the corresponding discrepancy d_{ij} becomes equal to ϵ or a small multiple of ϵ. A reasonable



Iteration #	Path P prior to iteration	Price vector (p_s, p_1, p_2, p_t) prior to iteration	Type of iteration
1	(s)	$(\underline{0}, 0, 0, 0)$	Extension to 1
2	$(s, 1)$	$(1+\epsilon, \underline{0}, 0, 0)$	Contraction to s
3	(s)	$(\underline{1+\epsilon}, 3+\epsilon, 0, 0)$	Extension to 2
4	$(s, 2)$	$(2+\epsilon, 3+\epsilon, \underline{0}, 0)$	Contraction to s
5	(s)	$(\underline{2+\epsilon}, 3+\epsilon, 2.5+\epsilon, 0)$	Extension to 1
6	$(s, 1)$	$(4+2\epsilon, \underline{3+\epsilon}, 2.5+\epsilon, 0)$	Extension to t
7	$(s, 1, t)$	$(4+2\epsilon, 3+2\epsilon, 2.5+\epsilon, \underline{0})$	Termination

Figure 1.4.9: A four-node example, with the arc lengths shown next to the arcs in the left-side figure. The right-side figure and the table trace the steps of the ASP algorithm starting with $P = (s)$ and all initial prices equal to 0. In the extension case (c1), we raise the price level of n_k to the maximum possible level, which is

$$p_{\text{pred}(n_k)} - a_{\text{pred}(n_k)n_k}.$$

The price of the terminal node of the path is underlined. The trajectory of the ASP algorithm shown in the table corresponds to values of $\epsilon \leq 3$. The final path obtained is the shortest path $(s, 1, t)$. If instead $\epsilon > 3$, the algorithm will still find the shortest path and faster: it will first perform an extension to 1, setting $p_s = 1 + \epsilon$. It will then perform an extension to t, since the condition

$$1 + \epsilon = p_{\text{pred}(1)} > a_{\text{pred}(1)1} + a_{1\text{succ}(1)} + p_{\text{succ}(1)} = 1 + 3 + 0 = 4$$

is satisfied, and terminate. The final path will be $P = (s, 1, t)$ and the final price vector will be

$$(p_s, p_1, p_2, p_t) = (1 + \epsilon, 1 + \epsilon, 0, 0),$$

with the arc $(s, 1)$ being balanced and the arc $(1, t)$ being downhill. Note also that generally, there is no guarantee that the ASP algorithm will find a shortest path for all initial prices and values of ϵ. Conditions that provide such guarantees will be given later.

conjecture is that if all the discrepancies d_{ij} are initially bounded by a small multiple of ϵ, then the path produced by the algorithm upon termination is shortest to within a small multiple of $n\epsilon$, where n is the number of nodes. This bears similarity to the auction algorithms for the assignment problem, where the solution obtained can be proved to be optimal to within $n\epsilon$. Further discussion and analysis of this issue will be provided in Section 3.2.

ϵ-Complementary Slackness and ϵ-Scaling

There is an ϵ-complementary slackness notion that parallels the one given for the case of the assignment problem. It is defined as follows.

ϵ-Complementary Slackness:

For a given $\epsilon > 0$, the prices $\{p_i \mid i \in \mathcal{N}\}$ and the path P satisfy

$$p_i \leq a_{ij} + p_j + \epsilon, \qquad \text{for all arcs } (i,j),$$

i.e., every arc is uphill, or level, or downhill by at most ϵ, and

$$p_i \geq a_{ij} + p_j, \qquad \text{for all arcs } (i,j) \text{ of the path } P,$$

i.e., every arc of P is level or downhill (by at most ϵ).

An important observation is that when ϵ-CS holds, the discrepancies d_{ij} of Eq. (1.35) are at most equal to ϵ, so *if the ASP algorithm maintains ϵ-CS throughout its operation, it produces a path that is suboptimal by at most* $(n+1)\epsilon$, and hence also optimal for ϵ sufficiently small [$(n+1)\epsilon$ should be less than the difference between the 2nd shortest path distance and the shortest path distance]. Thus maintaining ϵ-CS is desirable.

The ASP algorithm, as described earlier in this section, need not maintain ϵ-CS throughout its operation, because the increase of p_{n_k} prior to an extension may violate the ϵ-CS inequality $p_{n_k} \leq a_{n_k j} + p_j + \epsilon$ for $j = \text{succ}(n_k)$ and possibly for j equal to some other downstream neighbors of n_k. A simple remedy is to choose the price increase prior to an extension in a specific way. In particular, in case (b1) of the ASP algorithm, we raise the price p_{n_k} to the largest value that satisfies ϵ-CS, while extending P to $\text{succ}(n_k)$, rather than setting p_{n_k} to any value that makes the arc $\big(\text{pred}(n_k), n_k\big)$ level or downhill and the arc $\big(n_k, \text{succ}(n_k)\big)$ downhill.

In Chapter 3, we will discuss and analyze further shortest path algorithms that maintain ϵ-CS. We will also discuss ϵ-scaling, which is implemented by running the algorithm with a relatively large value of ϵ to

estimate "good" prices, at least for a subset of "promising" transit nodes from s to t, and then progressively refining the assessment of the "promise" of these nodes. This is done by rerunning the algorithm with smaller values of ϵ, while using as initial prices at each run the final prices of the previous run. Similar to the case of the assignment problem, we will demonstrate in Chapter 3 that ϵ-scaling improves the computational complexity of the ASP algorithm.

1.4.4 Connections to Reinforcement Learning

Reinforcement learning (RL for short) is a popular approximation methodology for a large variety of sequential decision and control problems that can in principle be dealt with by dynamic programming (DP for short). It is well-known that every finite-state deterministic DP problem can be posed as a shortest path problem over an acyclic graph, with the origin node corresponding to the initial state of the DP problem. It is therefore natural to expect that the shortest path algorithms of the preceding section can find substantial application within the context of RL.

In this section, we will outline some of the connections of auction algorithms (particularly for shortest paths) with the broad RL methodology of *approximation in value space*, which is based on replacing the optimal cost function in the DP algorithm by an approximation. We will provide a more detailed discussion in Chapter 5.

Approximation in value space with one-step or multistep lookahead minimization lies at the heart of many prominent artificial intelligence successes, including AlphaZero and other related game programs. It is also representative of the methods of rolling and receding horizon control, including model predictive control, which have been used with success for many years in control system design and operations research applications. Generally, in such problems we have a dynamic system that generates a sequence of transitions between states under the influence of decision/control over a finite or infinite number of steps, and with a cost for every transition. The objective is to select the decisions to minimize the sum of all the transition costs. For example in the s-to-t shortest path problem, the states are the nodes, with s and t being the initial and final states, respectively, the decision/control at a node is the choice of a downstream neighbor node, and the transition cost is the length of the corresponding arc.

A useful viewpoint, which has been emphasized in the author's recent books [Ber20a] and [Ber22], is to think of approximation in value space schemes as consisting of two components:

(a) *The off-line training algorithm*, which "learns" a value function and possibly a default policy by using data, either externally given or self-generated by simulation. The value function provides an estimate of the cost of starting at any one state, while the default policy supplies a (suboptimal) decision/control at any one state.

(b) *The on-line play algorithm*, which generates decisions in real time using the value function and possibly the policy that has been obtained by off-line training. This algorithm is invoked to select a decision at any state of the DP problem, once this state is generated in real time.

It is argued in the book [Ber22a] (and also more formally in the book [Ber20a]) that the on-line play algorithm amounts to a step of Newton's method for solving Bellman's equation, while the starting point for the Newton step is determined by the results of off-line training. This supports a conceptual idea that applies in great generality and is central in the books [Ber20a] and [Ber22a], namely that *the performance of an off-line trained policy can be greatly improved by on-line play*.

We next discuss how our path construction algorithms of this paper can be blended into approximation in value space schemes for deterministic DP problems, as well as into some schemes that apply to stochastic DP problems.

Path Construction in the Context of Off-Line Training

The analysis of Section 1.4.3 suggests that the initial prices p_i in the ASP algorithm should be chosen to be close to the shortest distances D_i^*, or more accurately, they should be chosen in a way that keeps the arcs nearly level or uphill, and minimizes the arc discrepancies given by Eq. (1.35). Of course we do not know the exact values D_i^*, but in a given application we may be able to use as initial prices approximate values, which can be obtained off-line through a computationally inexpensive heuristic or other machine learning methods. Collectively, these approximate values constitute a value function obtained by off-line training, to be used subsequently by the on-line play algorithm that is based on ASP.

In one possible approach we may use data to train a neural network or other approximation architecture to learn approximations to the shortest distances D_i^*. The data may be obtained by using a shortest path algorithm and arc lengths that are similar to the ones of the given problem. The training should also aim to produce prices for which the discrepancies d_{ij} are small. This objective can and should be encoded into the training problem. It is also possible to train multiple neural networks to use for different patterns of arc lengths.

Path Construction in the Context of On-Line Play

There are also possibilities for using the ASP algorithm during on-line play, since several RL methods rely on the computation of (nearly) shortest paths on-line. An important such context arises in *rollout algorithms*. This is a popular class of RL methods that has received a lot of attention as an effective and easily implementable (suboptimal) methodology; see the books [Ber19], [Ber20a]. In a rollout algorithm, at each encountered

state, we minimize over the decisions of the current stage, and treat the future stages approximately, through a relatively fast heuristic, called the *base heuristic*. An auction algorithm, including ASP, could be a suitable base heuristic. As an example, the paper [Ber20b] illustrates applications of a combined auction/rollout algorithm for solution of multidimensional assignment problems.

Generally, in a rollout algorithm the idea is to use as value function the cost function of the base heuristic. The key property is that the performance of the rollout algorithm improves on the performance of the base heuristic. This is in the spirit of the fundamental DP method of policy iteration, which is intimately connected to rollout (the book [Ber20a] has a special focus on rollout and related methods that also apply to multiagent problems).

On-line play schemes often use a *multistep lookahead search* for path construction through an acyclic decision graph. The search involves a graph traversal algorithm to reach the leaves of the graph, starting from the root node, which corresponds to the current state of the DP problem being solved on-line. It also uses a terminal cost at the leaves of the graph, which is obtained by using an off-line trained value function or by using a base heuristic on-line. The graph traversal may be done by using (nearly) shortest path calculations (see RL books such as the author's [Ber19], [Ber20a], [Ber22b], as well as the books by Sutton and Barto [SuB18], and Lattimore and Szepesvari [LaS20]). The techniques of real-time dynamic programming, described in the papers by Korf [Kor90], and Barto, Bradtke, and Singh [BBS95], among many others, are relevant in this context.

A popular class of methods for on-line play with multistep lookahead is *Monte Carlo tree search* (MCTS for short). These methods evaluate approximately the leaves of a tree of state transitions with root at the current state, combine the results of the evaluations by backwards propagation to the root of the tree, and progressively expand the depth of the tree by adding new leaves. A standard way to describe MCTS (see the surveys [BPW12] and [SGS21], and the book [LaS20]) is in terms of four components: *selection, expansion, simulation, and backup*. Selection refers to choosing a leaf node of the tree, to improve its evaluation, and possibly to add its descendants to the tree. Often in MCTS the selection is done by various criteria that try to balance exploration and exploitation, such as the statistics-based UCB (upper confidence bound) criterion. Expansion refers to the method used for tree enlargement, and may be based on the UCB criterion or other more traditional iterative deepening techniques (searching to adequate precision at a given level of lookahead before starting to search at a deeper level of lookahead). Simulation refers to the approximate evaluation of a leaf node by one or more stochastic Monte Carlo simulation runs. Finally, backup refers to the backwards propagation of the results of the leaf node evaluations to the root of the tree. The decision to be applied at the current state is the one corresponding to the

best backed up evaluation. Note that while MCTS inherently assumes a stochastic decision environment, it has been applied to deterministic problems as well by using problem-dependent heuristics for tree pruning and expansion. Moreover, MCTS algorithms has been developed for adversarial contexts and games (even deterministic such as chess), in conjunction with techniques of minimax search such as alpha-beta pruning and others.

The problem that is solved approximately by MCTS is a shortest path problem with the origin being the root of the tree (the current state of the DP problem), and the destination being an artificial node to which all leaf nodes are connected with arcs that have the leaf evaluations as lengths. Thus for deterministic problems, one may consider the use of the ASP algorithm as an alternative to MCTS for solving this shortest path problem. In particular, the selection and backup processes are replaced by the extension/contraction mechanism of the ASP algorithm, while the simulation process may be performed by a deterministic base heuristic. The interim leaf evaluation results may be used for tree expansion in some more or less heuristic way. Each tree expansion may be followed by suitable price modifications to enforce an ϵ-CS condition, similar to the scheme discussed in Section 3.4 for acyclic graphs. One may also use ϵ-scaling at appropriate points to refine the quality of the solution. At some point the tree search is terminated, possibly upon reaching the limit of the computational budget. The decision at the current state is chosen to be the first arc of the final path generated by the ASP algorithm. The analysis and implementation of the ASP algorithm within search contexts where MCTS has traditionally been applied is an interesting subject for further research.

The ASP algorithm is inherently deterministic, but it can also be applied in stochastic multistep lookahead contexts, where the Monte Carlo tree search methods have been used widely. This can be done by replacing all steps of a multistep lookahead *except for the first* by deterministic approximations through the use of certainty equivalence (replacing stochastic quantities by fixed deterministic substitutes; see e.g., the book [Ber19]). The deterministic shortest path optimizations following the first step of lookahead involve an acyclic graph, and can be handled with the ASP algorithm. There is good reason for taking into account the stochastic nature of the first step without approximation, in order to maintain the connection of the lookahead minimization with Newton's method for solving the Bellman equation, as has been explained in the book [Ber22b], Section 3.2.

Let us also mention a possibility that arises in a time-varying environment where some of the arc lengths may be changing, possibly because some arcs may become unavailable and new arcs may become available, while new instances of shortest path problems arise. This is also typical in problems of adaptive control. An interesting possibility may then be to update the initial prices using machine learning methodology, and a combination of off-line and on-line training with data.

In this regard, we should mention that the use of reinforcement learn-

ing (RL) methods in conjunction with our path construction algorithms is facilitated by the fact that the initial prices are unrestricted. This makes our algorithms well-suited for large-scale and time-varying environments, such as data mining and transportation, where requests for solution of path construction problems arise continuously over time. Addressing the special implementation and machine learning issues in the context of such environments is an interesting subject for further research. In conclusion, there are several potentially fruitful possibilities to mesh the ASP algorithm within the RL methodology. The key property is that these algorithms will produce a feasible path starting with arbitrary prices. This path will be near optimal if the starting prices are close to the true (unknown) shortest distances or if they satisfy an ϵ-CS condition with ϵ relatively small. Moreover, it is plausible that better paths can be obtained by more closely approximating the shortest distances using heuristics and training with data. This conjecture is supported by experience with related auction algorithms, but remains to be established empirically.

In conclusion, there are several potentially fruitful possibilities to mesh the ASP algorithm within the RL methodology, which will be discussed in greater detail in Chapter 5. The key property is that these algorithms will produce a feasible path starting with arbitrary prices. This path will be near optimal if the starting prices are close to the true (unknown) shortest distances or if they satisfy an ϵ-CS condition with ϵ relatively small. Moreover, it is plausible that better paths can be obtained by more closely approximating the shortest distances using heuristics and training with data. This conjecture is supported by experience with related auction algorithms.

1.5 NOTES AND SOURCES

In this chapter we have provided an introduction to our principal network problem formulations, their interrelations, and their connection with primal and dual optimization. We have focused primarily on two principal paradigms, assignment and shortest path, and we have described in intuitive terms the application of auction algorithm and its market-based mechanism. In Section 1.4.4, we have provided a preview of ways that auction algorithms can be fruitfully incorporated within the broad framework of the reinforcement learning methodology. This subject will be revisited in Chapter 5.

Projected Additions

The material of the following 2024 paper on auction algorithms will be incorporated in time in this chapter:

D. P. Bertsekas, "New Auction Algorithms for the Assignment Problem and Extensions," Results in Control and Optimization, Vol. 14, 2024.

Abstract: We consider the classical linear assignment problem, and we introduce new auction algorithms for its optimal and suboptimal solution. The algorithms are founded on duality theory, and are related to ideas of competitive bidding by persons for objects and the attendant market equilibrium, which underlie real-life auction processes. We distinguish between two fundamentally different types of bidding mechanisms: *aggressive* and *cooperative*. Mathematically, aggressive bidding relies on a notion of approximate coordinate descent in dual space, an ϵ-complementary slackness condition to regulate the amount of descent approximation, and the idea of ϵ-scaling to resolve efficiently the price wars that occur naturally as multiple bidders compete for a smaller number of valuable objects. Cooperative bidding avoids price wars through detection and cooperative resolution of any competitive impasse that involves a group of persons.

We discuss the relations between the aggressive and the cooperative bidding approaches, we derive new algorithms and variations that combine ideas from both of them, and we also make connections with other primal-dual methods, including the Hungarian method. Furthermore, our discussion points the way to algorithmic extensions that apply more broadly to network optimization, including shortest path, max-flow, transportation, and minimum cost flow problems with both linear and convex cost functions.

1.6 APPENDIX: BACKGROUND ON GRAPHS, PATHS AND FLOWS

In this appendix, we introduce some of the basic definitions relating to graphs, paths, flows, and other related notions. Graph concepts are fairly intuitive, and can be understood in terms of suggestive figures, but often involve hidden subtleties.

The material in this appendix provides more detailed background than what is provided in Section 1.1. It will be used primarily in Chapter 4, and also for some of the theoretical discussions in Chapters 2 and 3. For the moment, the reader may wish to just skim through this appendix, and revisit it later as needed.

A *directed graph*, $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, consists of a set $\mathcal{N}$ of *nodes* and a set $\mathcal{A}$ of pairs of distinct nodes from $\mathcal{N}$ called *arcs*. The numbers of nodes and arcs are denoted by N and A, respectively, and it is assumed throughout that $1 \leq N < \infty$ and $0 \leq A < \infty$. An arc (i, j) is viewed as an ordered pair, and is to be distinguished from the pair (j, i). If (i, j) is an arc, we say that (i, j) is *outgoing* from node i and *incoming* to node j; we also say that j is an *outward neighbor* of i and that i is an *inward neighbor* of j. We say that arc (i, j) is *incident* to i and to j, and that i is the *start* node and j is the *end* node of the arc. We also say that i and j are the *end nodes* of arc (i, j). The *degree* of a node i is the number of arcs that are incident to

i. A graph is said to be *complete* if it contains all possible arcs; that is, if there exists an arc for each ordered pair of nodes.

We do not exclude the possibility that there is a separate arc connecting a pair of nodes in each of the two directions. However, we do not allow more than one arc between a pair of nodes in the same direction, so that we can refer unambiguously to the arc with start i and end j as arc (i, j). This is done for notational convenience.† Our analysis can be simply extended to handle multiple arcs with start i and end j; the extension is based on modifying the graph by introducing for each such arc, an additional node, call it n, together with the two arcs (i, n) and (n, j). On occasion, we will pause to provide examples of this type of extension.

We note that much of the literature of graph theory distinguishes between *directed* graphs where an arc (i, j) is an ordered pair to be distinguished from arc (j, i), and *undirected* graphs where an arc is associated with a pair of nodes regardless of order. One may use directed graphs, even in contexts where the use of undirected graphs would be appropriate and conceptually simpler. For this, one may need to replace an undirected arc (i, j) with two directed arcs (i, j) and (j, i) having identical characteristics. We have chosen to deal exclusively with directed graphs because in our development there are only a few occasions where undirected graphs are convenient. Thus, *all our references to a graph implicitly assume that the graph is directed*. In fact we often omit the qualifier "directed" and refer to a directed graph simply as a *graph*.

1.6.1 Paths and Cycles

A *path* P in a directed graph is a sequence of nodes $(n_1, n_2, \ldots, n_k)$ with $k \geq 2$ and a corresponding sequence of $k-1$ arcs such that the ith arc in the sequence is either (n_i, n_{i+1}) (in which case it is called a *forward* arc of the path) or (n_{i+1}, n_i) (in which case it is called a *backward* arc of the path). Nodes n_1 and n_k are called the *start node* (or *origin*) and the *end node* (or *destination*) of P, respectively. A path is said to be *forward* (or *backward*) if all of its arcs are forward (respectively, backward) arcs. We denote by P^+ and P^- the sets of forward and backward arcs of P, respectively.

A *cycle* is a path for which the start and end nodes are the same. A path is said to be *simple* if it contains no repeated arcs and no repeated nodes, except that the start and end nodes could be the same (in which case the path is called a *simple cycle*). A *Hamiltonian cycle* is a simple forward cycle that contains all the nodes of the graph. These definitions are illustrated in Fig. 1.6.1. We mention that some authors use a slightly

† Some authors use a single symbol, such as a, to denote an arc, and use something like $s(a)$ and $e(a)$ to denote the start and end nodes of a, respectively. This notational method allows the existence of multiple arcs with the same start and end nodes, but is also more cumbersome and less suggestive.

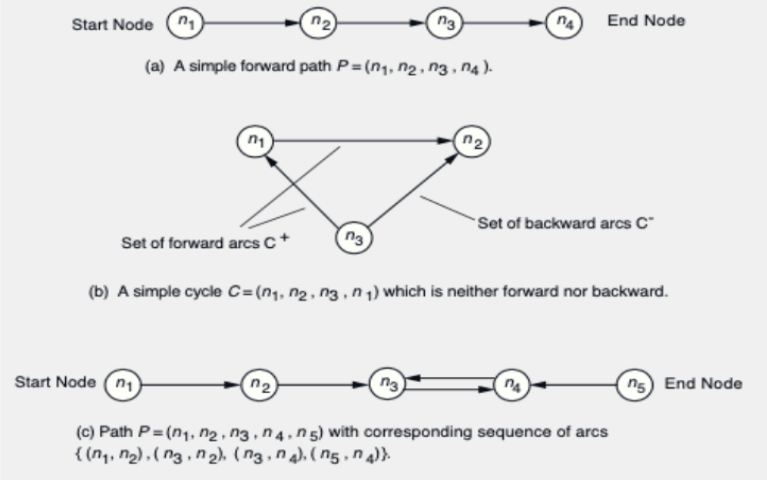


Figure 1.6.1: Illustration of various types of paths and cycles. The cycle in (b) is not a Hamiltonian cycle; it is simple and contains all the nodes of the graph, but it is not forward. Note that for the path (c), in order to resolve ambiguities, it is necessary to specify the sequence of arcs of the path (rather than just the sequence of nodes) because both (n_3, n_4) and (n_4, n_3) are arcs.

different terminology: they use the term "walk" to refer to a path and they use the term "path" to refer to a simple path.

Note that the sequence of nodes $(n_1, n_2, \ldots, n_k)$ is not sufficient to specify a path; the sequence of arcs may also be important, as Fig. 1.6.1(c) shows. The difficulty arises when for two successive nodes n_i and n_{i+1} of the path, both (n_i, n_{i+1}) and (n_{i+1}, n_i) are arcs, so there is ambiguity as to which of the two is the corresponding arc of the path. If a path is known to be forward or is known to be backward, it is uniquely specified by the sequence of its nodes. Otherwise, however, the intended sequence of arcs must be explicitly defined.

A graph that contains no simple cycles is said to be *acyclic*. We say that a graph $\mathcal{G}' = (\mathcal{N}', \mathcal{A}')$ is a *subgraph* of a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ if $\mathcal{N}' \subset \mathcal{N}$ and $\mathcal{A}' \subset \mathcal{A}$. A *tree* is a acyclic graph with a special node r, called the *root*, such that for every node $n \neq r$, there is a unique path starting at r and ending at n. A *spanning tree* of a graph $\mathcal{G}$ is a subgraph of $\mathcal{G}$, which is a tree and includes all the nodes of $\mathcal{G}$. It can be shown that a subgraph is a spanning tree if and only if it is connected and it contains $N - 1$ arcs.

1.6.2 Flow and Excess

In many applications involving graphs, it is useful to introduce a variable that measures the quantity flowing through each arc, like for example, electric current in an electric circuit, or water flow in a hydraulic network.

We refer to such a variable as the *flow of an arc*. Mathematically, the flow of an arc (i, j) is simply a scalar (real number), which we usually denote by x_{ij}. It is convenient to allow negative as well as positive values for flow. In applications, a negative arc flow indicates that whatever is represented by the flow (material, electric current, etc.), moves in a direction opposite to the direction of the arc. We can always change the sign of a negative arc flow to positive as long as we change the arc direction, so in many situations we can assume without loss of generality that all arc flows are nonnegative. For the development of a general methodology, however, this device is often cumbersome, which is why we prefer to simply accept the possibility of negative arc flows.

Given a graph $(\mathcal{N}, \mathcal{A})$, a set of flows $\{x_{ij} \mid (i, j) \in \mathcal{A}\}$ is referred to as a *flow vector*. The *excess vector* y associated with a flow vector x is the N-dimensional vector with coordinates

$$y_i = \sum_{\{j \mid (i,j) \in \mathcal{A}\}} x_{ij} - \sum_{\{j \mid (j,i) \in \mathcal{A}\}} x_{ji}, \qquad \forall\ i \in \mathcal{N}. \tag{1.36}$$

Thus, y_i is the total flow departing from node i less the total flow arriving at i; it is referred to as the *excess of* i.

We say that node i is a *source* (respectively, *sink*) for the flow vector x if $y_i > 0$ (respectively, $y_i < 0$). If $y_i = 0$ for all $i \in \mathcal{N}$, then x is called a *circulation*. These definitions are illustrated in Fig. 1.6.2. Note that by adding Eq. (1.36) over all $i \in \mathcal{N}$, we obtain

$$\sum_{i \in \mathcal{N}} y_i = 0.$$

Every excess vector y must satisfy this equation.

The flow vectors x that we will consider will often be constrained to lie between given lower and upper bounds of the form

$$b_{ij} \leq x_{ij} \leq c_{ij}, \qquad \forall\ (i, j) \in \mathcal{A}.$$

Given a flow vector x that satisfies these bounds, we say that a path P is *unblocked with respect to* x if, roughly speaking, we can send some positive flow along P without violating the bound constraints; that is, if flow can be increased on the set P^+ of the forward arcs of P, and can be decreased on the set P^- of the backward arcs of P:

$$x_{ij} < c_{ij}, \quad \forall\ (i, j) \in P^+, \qquad b_{ij} < x_{ij}, \quad \forall\ (i, j) \in P^-.$$

For example, in Fig. 1.6.2(a), suppose that all arcs (i, j) have flow bounds $b_{ij} = -2$ and $c_{ij} = 2$. Then the path consisting of the sequence of nodes $(1, 2, 4)$ is unblocked, while the reverse path $(4, 2, 1)$ is not unblocked.

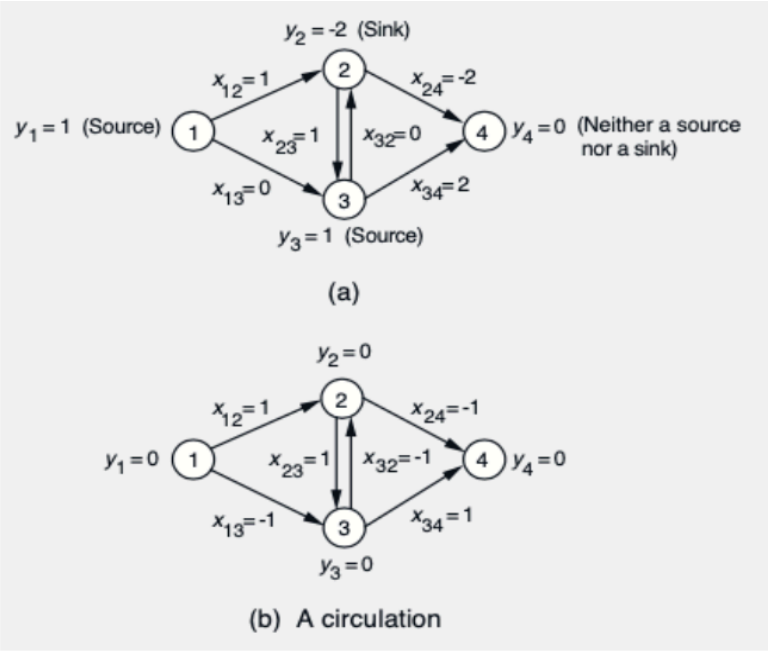


Figure 1.6.2: Illustration of flows x_{ij} and the corresponding excesses y_i. The flow in (b) is a circulation because $y_i = 0$ for all i.

1.6.3 Path Flows and Conformal Decomposition

A *simple path flow* is a flow vector that corresponds to sending a positive amount of flow along a simple path; more precisely, it is a flow vector x with components of the form

$$x_{ij} = \begin{cases} a & \text{if } (i,j) \in P^+, \\ -a & \text{if } (i,j) \in P^-, \\ 0 & \text{otherwise,} \end{cases} \tag{1.37}$$

where a is a positive scalar, and P^+ and P^- are the sets of forward and backward arcs, respectively, of some simple path P. Note that the path P may be a cycle, in which case x is also called a *simple cycle flow*.

It is often convenient to break down a flow vector into the sum of simple path flows. This leads to the notion of a conformal realization, which we proceed to discuss.

We say that a path P *conforms* to a flow vector x if $x_{ij} > 0$ for all forward arcs (i,j) of P and $x_{ij} < 0$ for all backward arcs (i,j) of P, and furthermore either P is a cycle or else the start and end nodes of P are a source and a sink of x, respectively. Roughly, a path conforms to a flow vector if it "carries flow in the forward direction," i.e., in the direction

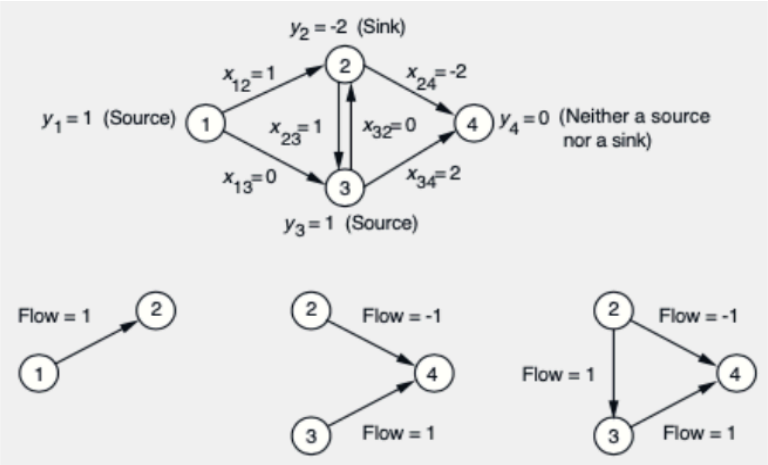


Figure 1.6.3: Decomposition of a flow vector x into three simple path flows conforming to x. Consistent with the definition of conformance of a path flow, each arc (i, j) of the three component paths carries positive (or negative) flow only if $x_{ij} > 0$ (or $x_{ij} < 0$, respectively). The first two paths $[(1, 2)$ and $(3, 4, 2)]$ are not cycles, but they start at a source and end at a sink, as required. Arcs $(1, 3)$ and $(3, 2)$ do not belong to any of these paths because they carry zero flow. In this example, the decomposition is unique, but in general this need not be the case.

from the start node to the end node. In particular, for a forward cycle to conform to a flow vector, all its arcs must have positive flow. For a forward path which is not a cycle to conform to a flow vector, its arcs must have positive flow, and in addition the start and end nodes must be a source and a sink, respectively; for example, in Fig. 1.6.2(a), the path consisting of the sequence of arcs (1,2), (2,3), (3,4) does not conform to the flow vector shown, because node 4, the end node of the path, is not a sink.

We say that a simple path flow x^s *conforms* to a flow vector x if the path P corresponding to x^s via Eq. (1.37) conforms to x. This is equivalent to requiring that

$$0 < x_{ij} \qquad \text{for all arcs } (i, j) \text{ with } 0 < x^s_{ij},$$

$$x_{ij} < 0 \qquad \text{for all arcs } (i, j) \text{ with } x^s_{ij} < 0,$$

and that either P is a cycle or else the start and end nodes of P are a source and a sink of x, respectively.

An important fact is that any flow vector can be decomposed into a set of conforming simple path flows, as illustrated in Fig. 1.6.3. We state this as a proposition. The proof is based on an algorithm that can be used to construct the conforming components one by one; see [Ber98], Exercise 1.2 (with solution).

Proposition 1.6.1: (Conformal Realization Theorem) A nonzero flow vector x can be decomposed into the sum of t simple path flow vectors $x^1, x^2, \ldots, x^t$ that conform to x, with t being at most equal to the sum of the numbers of arcs and nodes $A + N$. If x is integer, then $x^1, x^2, \ldots, x^t$ can also be chosen to be integer. If x is a circulation, then $x^1, x^2, \ldots, x^t$ can be chosen to be simple cycle flows, and $t \leq A$.

Proof: Assume first that x is a circulation. Consider the following procedure by which given x, we obtain a simple cycle flow x' that conforms to x and satisfies

$$\begin{aligned} &0 \leq x'_{ij} \leq x_{ij} && \text{for all arcs } (i,j) \text{ with } 0 \leq x_{ij}, \\ &x_{ij} \leq x'_{ij} \leq 0 && \text{for all arcs } (i,j) \text{ with } x_{ij} \leq 0, \\ &x_{ij} = x'_{ij} && \text{for at least one arc } (i,j) \text{ with } x_{ij} \neq 0; \end{aligned}$$

(see Fig. 1.6.4). Choose an arc (i,j) with $x_{ij} \neq 0$. Assume that $x_{ij} > 0$. (A similar procedure can be used when $x_{ij} < 0$.) Construct a sequence of node subsets $T_0, T_1, \ldots$, as follows: Take $T_0 = \{j\}$. For $k = 0, 1, \ldots$, given T_k, let

$$T_{k+1} = \Big\{ n \notin \cup_{p=0}^{k} T_p \mid \text{ there is a node } m \in T_k, \text{ and either an arc } (m,n) \text{ such that } x_{mn} > 0 \text{ or an arc } (n,m) \text{ such that } x_{nm} < 0 \Big\},$$

and mark each node $n \in T_{k+1}$ with the label "(m,n)" or "(n,m)," where m is a node of T_k such that $x_{mn} > 0$ or $x_{nm} < 0$, respectively. The procedure terminates when T_{k+1} is empty.

At the end of the procedure, trace labels backward from i until node j is reached. (How do we know that i belongs to one of the sets T_k?) In particular, let "(i_1, i)" or "(i, i_1)" be the label of i, let "(i_2, i_1)" or "(i_1, i_2)" be the label of i_1, etc., until a node i_k with label "(i_k, j)" or "(j, i_k)" is found. The cycle $C = (j, i_k, i_{k-1}, \ldots, i_1, i, j)$ is simple, it contains (i,j) as a forward arc, and is such that all its forward arcs have positive flow and all its backward arcs have negative flow. Let $a = \min_{(m,n) \in C} |x_{mn}| > 0$. Then the simple cycle flow x', where

$$x'_{ij} = \begin{cases} a & \text{if } (i,j) \in C^+, \\ -a & \text{if } (i,j) \in C^-, \\ 0 & \text{otherwise,} \end{cases}$$

has the required properties.

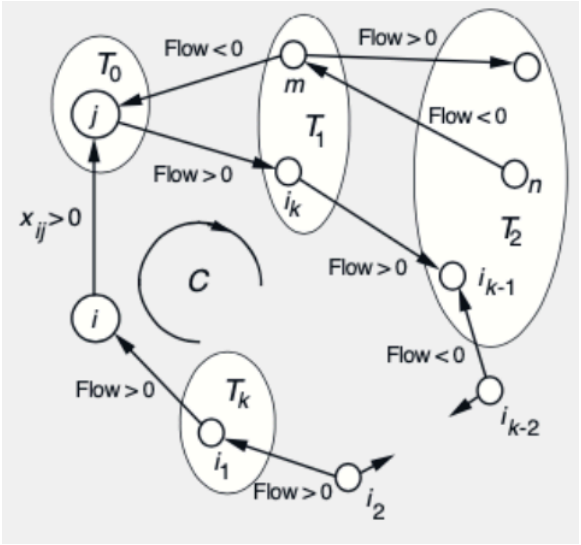


Figure 1.6.4: Construction of a cycle of arcs with nonzero flow used in the proof of the conformal realization theorem.

Now subtract x' from x. We have $x_{ij} - x'_{ij} > 0$ only for arcs (i,j) with $x_{ij} > 0$, $x_{ij} - x'_{ij} < 0$ only for arcs (i,j) with $x_{ij} < 0$, and $x_{ij} - x'_{ij} = 0$ for at least one arc (i,j) with $x_{ij} \neq 0$. If x is integer, then x' and $x - x'$ will also be integer. We then repeat the process (for at most A times) with the circulation x replaced by the circulation $x - x'$ and so on, until the zero flow is obtained.

If x is not a circulation, we form an enlarged graph by introducing a new node s and by introducing for each node $i \in \mathcal{N}$ an arc (s,i) with flow x_{si} equal to the divergence y_i. The resulting flow vector is seen to be a circulation in the enlarged graph (why?). This circulation, by the result just shown, can be decomposed into at most $A + N$ simple cycle flows of the enlarged graph, conforming to the flow vector. Out of these cycle flows, we consider those containing node s, and we remove s and its two incident arcs while leaving the other cycle flows unchanged. As a result we obtain a set of at most $A + N$ path flows of the original graph, which add up to x. These path flows also conform to x, as required. **Q.E.D.**