8

Complexity of Algorithms

Contents

8.1 Computational Complexity

In this chapter, we see how problems may be classified according to their level of difficulty.

Most problems that we consider in this book are of a general character, applying to all members of some family of graphs or digraphs. By an *instance* of a problem, we mean the problem as applied to one specific member of the family. For example, an instance of the Minimum-Weight Spanning Tree Problem is the problem of finding an optimal tree in a particular weighted connected graph.

An *algorithm* for solving a problem is a well-defined computational procedure which accepts any instance of the problem as *input* and returns a solution to the problem as *output*. For example, the Jarník–Prim Algorithm (6.9) accepts as input a weighted connected graph G and returns as output an optimal tree.

As we have seen, many problems of practical importance can be formulated in terms of graphs. Designing computationally efficient algorithms for solving these problems is one of the main concerns of graph theorists and computer scientists. The two aspects of theoretical interest in this regard are, firstly, to verify that a proposed algorithm does indeed perform correctly and, secondly, to analyse how efficient a procedure it is. We have already encountered algorithms for solving a number of basic problems. In each case, we have established their validity. Here, we discuss the efficiency of these and other algorithms.

By the *computational complexity* (or, for short, *complexity*) of an algorithm, we mean the number of basic computational steps (such as arithmetical operations and comparisons) required for its execution. This number clearly depends on the size and nature of the input. In the case of graphs, the complexity is a function of the number of bits required to encode the adjacency list of the input graph G, a function of n and m. (The number of bits required to encode an integer k is $\lceil \log_2 k \rceil$.) Naturally, when the input includes additional information, such as weights on the vertices or edges of the graph, this too must be taken into account in calculating the complexity. If the complexity is bounded above by a polynomial in the input size, the algorithm is called a *polynomial-time algorithm*. Such an algorithm is further qualified as *linear-time* if the polynomial is a linear function, *quadratic-time* if it is a quadratic function, and so on.

The Class $\mathcal{P}$

The significance of polynomial-time algorithms is that they are usually found to be computationally feasible, even for large input graphs. By contrast, algorithms whose complexity is exponential in the size of the input have running times which render them unusable even on inputs of moderate size. For example, an algorithm which checks whether two graphs on n vertices are isomorphic by considering all $n!$ bijections between their vertex sets is feasible only for small values of n (certainly no greater than 20), even on the fastest currently available computers. The class of problems solvable by polynomial-time algorithms is denoted by $\mathcal{P}$.

The tree-search algorithms discussed in Chapter 6 are instances of polynomial-time algorithms. In breadth-first search, each edge is examined for possible inclusion in the tree just twice, when the adjacency lists of its two ends are scanned. The same is true of depth-first search. Therefore both of these algorithms are linear in m, the number of edges. The Jarník–Prim Algorithm involves, in addition, comparing weights of edges, but it is easily seen that the number of comparisons is also bounded by a polynomial in m.

Unlike the other algorithms described in Chapter 6, the Max-Flow Min-Cut Algorithm is not a polynomial-time algorithm even when all the capacities are integers; the example in Exercise 8.1.1 shows that, in the worst case, the algorithm may perform an arbitrarily large number of iterations before returning a maximum flow. Fortunately, this eventuality can be avoided by modifying the way in which IPS is implemented, as was shown by Edmonds and Karp (1970) and Dinic (1970). Among all the arcs in $\partial(T)$ that qualify for inclusion in T, preference is given to those which are incident to the vertex that entered T the earliest, just as in breadth-first search, resulting in a shortest incrementing path. It can be shown that, with this refinement, the number of iterations of IPS is bounded by a polynomial in n and thus yields a polynomial-time algorithm.

Although our analysis of these algorithms is admittedly cursory, and leaves out many pertinent details, it should be clear that they do indeed run in polynomial time. A thorough analysis of these and other graph algorithms can be found in the books by Aho et al. (1975) and Papadimitriou (1994). On the other hand, there are many basic problems for which polynomial-time algorithms have yet to be found, and indeed might well not exist. Determining which problems are solvable in polynomial time and which are not is evidently a fundamental question. In this connection, a class of problems denoted by $\mathcal{NP}$ (standing for *nondeterministic polynomial-time*) plays an important role. We give here an informal definition of this class; a precise treatment can be found in Chapter 29 of the *Handbook of Combinatorics* (Graham et al. (1995)), or in the book by Garey and Johnson (1979).

The Classes $\mathcal{NP}$ and co-$\mathcal{NP}$

A *decision problem* is a question whose answer is either 'yes' or 'no'. Such a problem belongs to the class $\mathcal{P}$ if there is a polynomial-time algorithm that solves any instance of the problem in polynomial time. It belongs to the class $\mathcal{NP}$ if, given any instance of the problem whose answer is 'yes', there is a certificate validating this fact which can be checked in polynomial time; such a certificate is said to be *succinct*. Analogously, a decision problem belongs to the class *co-$\mathcal{NP}$* if, given any instance of the problem whose answer is 'no', there is a succinct certificate which confirms that this is so. It is immediate from these definitions that $\mathcal{P} \subseteq \mathcal{NP}$, inasmuch as a polynomial-time algorithm constitutes, in itself, a succinct certificate. Likewise, $\mathcal{P} \subseteq$ co-$\mathcal{NP}$. Thus

$$\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$$

Consider, for example, the problem of determining whether a graph is bipartite. This decision problem belongs to $\mathcal{NP}$, because a bipartition is a succinct certificate: given a bipartition (X, Y) of a bipartite graph G, it suffices to check that each edge of G has one end in X and one end in Y. The problem also belongs to co-$\mathcal{NP}$ because, by Theorem 4.7, every nonbipartite graph contains an odd cycle, and any such cycle constitutes a succinct certificate of the graph's nonbipartite character.

It thus belongs to $\mathcal{NP} \cap$ co-$\mathcal{NP}$. In fact, as indicated in Exercise 6.1.3, it belongs to $\mathcal{P}$.

As a second example, consider the problem of deciding whether a graph $G(x, y)$ has k edge-disjoint xy-paths. This problem is clearly in $\mathcal{NP}$, because a family of k edge-disjoint xy-paths is a succinct certificate: given such a family of paths, one may check in polynomial-time that it indeed has the required properties. The problem is also in co-$\mathcal{NP}$ because, by Theorem 7.17, a graph that does not have k edge-disjoint xy-paths has an xy-edge cut of size less than k. Such an edge cut serves as a succinct certificate for the nonexistence of k edge-disjoint xy-paths. Finally, because the maximum number of edge-disjoint xy-paths can be found in polynomial time by applying the Max-Flow Min-Cut Algorithm (7.9) (see Exercise 7.3.5) this problem belongs to $\mathcal{P}$, too.

Consider, now, the problem of deciding whether a graph has a Hamilton cycle.

Problem 8.1 Hamilton Cycle
Given*: a graph G,*
Decide*: Does G have a Hamilton cycle?*

If the answer is 'yes', then any Hamilton cycle would serve as a succinct certificate. However, should the answer be 'no', what would constitute a succinct certificate confirming this fact? In contrast to the two problems described above, no such certificate is known! In other words, notwithstanding that Hamilton Cycle is clearly a member of the class $\mathcal{NP}$, it has not yet been shown to belong to co-$\mathcal{NP}$, and might very well not belong to this class. The same is true of the decision problem for Hamilton paths. These two problems are discussed in detail in Chapter 18.

Many problems that arise in practice, such as the Shortest Path Problem (6.11), are optimization problems rather than decision problems. Nonetheless, each such problem implicitly includes an infinitude of decision problems. For example, the Shortest Path Problem includes, for each real number ℓ, the following decision problem. Given a weighted directed graph (D, w) with two specified vertices x and y, is there a directed (x, y)-path in D of length at most ℓ?

We have noted three relations of inclusion among the classes $\mathcal{P}$, $\mathcal{NP}$, and co-$\mathcal{NP}$, and it is natural to ask whether these inclusions are proper. Because $\mathcal{P} = \mathcal{NP}$ if and only if $\mathcal{P} =$ co-$\mathcal{NP}$, two basic questions arise, both of which have been posed as conjectures.

The Cook–Edmonds–Levin Conjecture

Conjecture 8.2 $\mathcal{P} \neq \mathcal{NP}$

Edmonds' Conjecture

Conjecture 8.3 $\mathcal{P} = \mathcal{NP} \cap co\text{-}\mathcal{NP}$

Conjecture 8.2 is one of the most fundamental open questions in all of mathematics. (A prize of one million dollars has been offered for its resolution.) It is widely (but not universally) believed that the conjecture is true, that there are problems in $\mathcal{NP}$ for which no polynomial-time algorithm exists. One such problem would be Hamilton Cycle. As we show in Section 8.3, this problem, and its directed analogue Directed Hamilton Cycle, are at least as hard to solve as any problem in the class $\mathcal{NP}$; more precisely, if a polynomial-time algorithm for either of these problems should be found, it could be adapted to solve any problem in $\mathcal{NP}$ in polynomial time by means of a suitable transformation. Conjecture 8.2 was, in essence, put forward by J. Edmonds in the mid-1960s, when he asserted that there could exist no 'good' (that is, polynomial-time) algorithm for the Travelling Salesman Problem (Problem 2.6). The conjecture thus predates the formal definition of the class $\mathcal{NP}$ by Cook (1971) and Levin (1973).

Conjecture 8.3, also proposed by Edmonds (1965c), is strongly supported by empirical evidence. Most decision problems which are known to belong to $\mathcal{NP} \cap$ co-$\mathcal{NP}$ are also known to belong to $\mathcal{P}$. A case in point is the problem of deciding whether a given integer is prime. Although it had been known for some time that this problem belongs to both $\mathcal{NP}$ and co-$\mathcal{NP}$, a polynomial-time algorithm for testing primality was discovered only much more recently, by Agrawal et al. (2004).

Exercises

$\star$**8.1.1**

a) Show that, starting with the zero flow, an application of the Max-Flow Min-Cut Algorithm (7.9) to the network N in Figure 8.1 might execute $2M + 1$ incrementing path iterations before finding a maximum flow.
b) Deduce that this algorithm is not a polynomial-time algorithm.

8.1.2 Show that Fleury's Algorithm (3.3) is a polynomial-time algorithm.

———————≀≀———————

8.1.3 Given a graph $G(x, y)$, consider the problem of deciding whether G has an xy-path of odd (respectively, even) length.

a) Show that this problem:
   i) belongs to $\mathcal{NP}$,

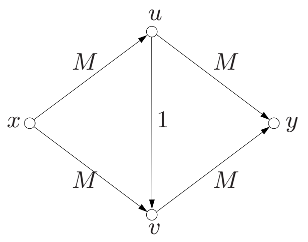


Fig. 8.1. A network on which Algorithm 7.9 might require many iterations

ii) belongs to co-$\mathcal{NP}$.

b) Describe a polynomial-time algorithm for solving the problem.

8.1.4 Describe a polynomial-time algorithm for deciding whether two trees are isomorphic.

8.2 Polynomial Reductions

A common approach to problem-solving is to transform the given problem into one whose solution is already known, and then convert that solution into a solution of the original problem. Of course, this approach is feasible only if the transformation can be made rapidly. The concept of polynomial reduction captures this requirement.

A *polynomial reduction* of a problem P to a problem Q is a pair of polynomial-time algorithms, one which transforms each instance I of P to an instance J of Q, and the other which transforms a solution for the instance J to a solution for the instance I. If such a reduction exists, we say that P is *polynomially reducible* to Q, and write $P \preceq Q$; this relation is clearly both reflexive and transitive. The significance of polynomial reducibility is that if $P \preceq Q$, and if there is a polynomial-time algorithm for solving Q, then this algorithm can be converted into a polynomial-time algorithm for solving P. In symbols:

$$P \preceq Q \text{ and } Q \in \mathcal{P} \Rightarrow P \in \mathcal{P} \tag{8.1}$$

A very simple example of the above paradigm is the polynomial reduction to the Minimum-Weight Spanning Tree Problem (6.8) of the following problem.

Problem 8.4 MAXIMUM-WEIGHT SPANNING TREE
GIVEN*: a weighted connected graph G,*
FIND*: a maximum-weight spanning tree in G.*

In order to solve an instance of this problem, it suffices to replace each weight by its negative and apply the Jarník–Prim Algorithm (6.9) to find an optimal tree in the resulting weighted graph. The very same tree will be one of maximum weight in the original weighted graph. (We remark that one can similarly reduce the problem of finding a longest xy-path in a graph to the Shortest Path Problem (6.11). However no polynomial-time algorithm is known for solving the latter problem when there are negative edge weights.)

Not all reductions are quite as straightforward as this one. Recall that two directed (x, y)-paths are *internally disjoint* if they have no internal vertices in common. Consider the following problem, the analogue for internally disjoint paths of Problem 7.10, the Arc-Disjoint Directed Paths Problem (ADDP).

Problem 8.5 INTERNALLY DISJOINT DIRECTED PATHS (IDDP)
GIVEN*: a digraph* $D := D(x, y)$,
FIND*: a maximum family of internally disjoint directed* (x, y)*-paths in* D.

A polynomial reduction of IDDP to ADDP can be obtained by constructing a new digraph $D' := D'(x, y)$ from D as follows.

- ▷ Split each vertex $v \in V \setminus \{x, y\}$ into two new vertices v^- and v^+, joined by a new arc (v^-, v^+).
- ▷ For each arc (u, v) of D, replace its tail u by u^+ (unless $u = x$ or $u = y$) and its head v by v^- (unless $v = x$ or $v = y$).

This construction is illustrated in Figure 8.2.

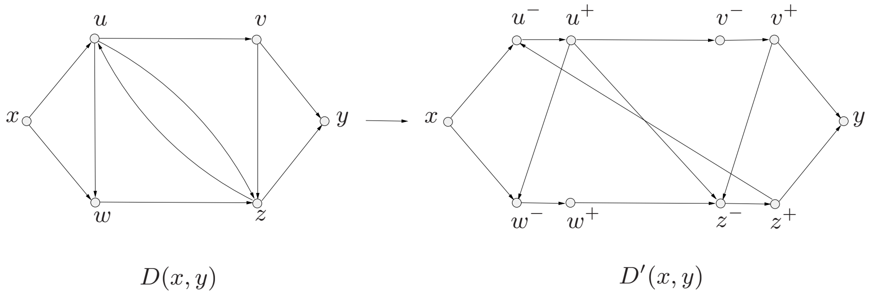


Fig. 8.2. Reduction of *IDDP* to *ADDP*

It can be seen that there is a bijection between families of internally disjoint directed (x, y)-paths in D and families of arc-disjoint directed (x, y)-paths in D'. Thus, finding a maximum family of internally disjoint directed (x, y)-paths in $D(x, y)$ amounts to finding a maximum family of arc-disjoint directed (x, y)-paths in $D'(x, y)$. This transformation of the instance $D(x, y)$ of IDDP to the instance $D'(x, y)$ of ADDP is a polynomial reduction because $v(D') = 2v(D) - 2$ and $a(D') = a(D) + v(D) - 2$. Hence $IDDP \preceq ADDP$.

The Max-Flow Min-Cut Algorithm (7.9) is a polynomial-time algorithm for solving ADDP. Therefore $ADDP \in \mathcal{P}$. Because $IDDP \preceq ADDP$, we may conclude that $IDDP \in \mathcal{P}$, also.

Most problems concerning paths in undirected graphs can be reduced to their analogues in directed graphs by the simple artifice of considering the associated digraph. As an example, let $G := G(x, y)$ be an undirected graph and let $D := D(G)$ be its associated digraph. There is an evident bijection between families of internally disjoint xy-paths in G and families of internally disjoint directed (x, y)-paths in D. Thus $IDP \preceq IDDP$, where IDP is the problem of finding a maximum family of internally disjoint xy-paths in a given graph $G(x, y)$. We showed above that $IDDP \in \mathcal{P}$. It now follows from the transitivity of the relation $\preceq$ that $IDP \in \mathcal{P}$.

Exercises

8.2.1 Consider a network in which a nonnegative integer $m(v)$ is associated with each intermediate vertex v. Show how a maximum flow f satisfying the constraint $f^-(v) \leq m(v)$, for all $v \in I$, can be found by applying the Max-Flow Min-Cut Algorithm to a suitably modified network.

8.2.2 Consider the following problem.

Problem 8.6 DISJOINT PATHS
GIVEN: *a graph G, a positive integer k, and two k-subsets X and Y of V,*
DECIDE: *Does G have k disjoint (X, Y)-paths?*

Describe a polynomial reduction of this problem to IDP (INTERNALLY DISJOINT PATHS).

8.3 $\mathcal{NP}$-Complete Problems

THE CLASS $\mathcal{NPC}$

We have just seen how polynomial reductions may be used to produce new polynomial-time algorithms from existing ones. By the same token, polynomial reductions may also be used to link 'hard' problems, ones for which no polynomial-time algorithm exists, as can be seen by writing (8.1) in a different form:

$$P \preceq Q \text{ and } P \notin \mathcal{P} \Rightarrow Q \notin \mathcal{P}$$

This viewpoint led Cook (1971) and Levin (1973) to define a special class of seemingly intractable decision problems, the class of $\mathcal{NP}$-complete problems. Informally, these are the problems in the class $\mathcal{NP}$ which are 'at least as hard to solve' as any problem in $\mathcal{NP}$.

Formally, a problem P in $\mathcal{NP}$ is *$\mathcal{NP}$-complete* if $P' \preceq P$ for every problem P' in $\mathcal{NP}$. The class of $\mathcal{NP}$-complete problems is denoted by $\mathcal{NPC}$. It is by no means obvious that $\mathcal{NP}$-complete problems should exist at all. On the other hand, once one such problem has been found, the $\mathcal{NP}$-completeness of other problems may be established by means of polynomial reductions, as follows.

In order to prove that a problem Q in $\mathcal{NP}$ is $\mathcal{NP}$-complete, it suffices to find a polynomial reduction to Q of some known $\mathcal{NP}$-complete problem P. Why is this so? Suppose that P is $\mathcal{NP}$-complete. Then $P' \preceq P$ for all $P' \in \mathcal{NP}$. If $P \preceq Q$, then $P' \preceq Q$ for all $P' \in \mathcal{NP}$, by the transitivity of the relation $\preceq$. In other words, Q is $\mathcal{NP}$-complete. In symbols:

$$P \preceq Q \ \text{ and } \ P \in \mathcal{NPC} \Rightarrow Q \in \mathcal{NPC}$$

Cook (1971) and Levin (1973) made a fundamental breakthrough by showing that there do indeed exist $\mathcal{NP}$-complete problems. More precisely, they proved that the satisfiability problem for boolean formulae is $\mathcal{NP}$-complete. We now describe this problem, and examine the theoretical and practical implications of their discovery.

Boolean Formulae

A *boolean variable* is a variable which takes on one of two values, 0 ('false') or 1 ('true'). Boolean variables can be combined into *boolean formulae*, which may be defined recursively as follows.

- ▷ Every boolean variable is a boolean formula.
- ▷ If f is a boolean formula, then so too is $(\neg f)$, the *negation* of f.
- ▷ If f and g are boolean formulae, then so too are:
  - $(f \vee g)$, the *disjunction* of f and g,
  - $(f \wedge g)$, the *conjunction* of f and g.

These three operations may be thought of informally as 'not f', 'f or g', and 'f and g', respectively. The negation of a boolean variable x is often written as $\overline{x}$. Thus the expression

$$(\neg(x_1 \vee \overline{x_2}) \vee x_3) \wedge (x_2 \vee \overline{x_3}) \tag{8.2}$$

is a boolean formula in the variables x_1, x_2, x_3. Note that the parentheses are needed here to avoid ambiguity as to the order of execution of the various operations. (For ease of reading, we omit the outer pair of parentheses.)

An assignment of values to the variables of a boolean formula is called a *truth assignment*. Given a truth assignment, the value of the formula may be computed according to the following rules.

$\neg$	
0	1
1	0

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

For instance, if $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$, the value of formula (8.2) is:

$$(\neg(1 \vee \overline{0}) \vee 1) \wedge (0 \vee \overline{1}) = (\neg(1 \vee 1) \vee 1) \wedge (0 \vee 0) = (\overline{1} \vee 1) \wedge 0 = (0 \vee 1) \wedge 0 = 1 \wedge 0 = 0$$

Two boolean formulae are *equivalent* (written $\equiv$) if they take the same value for each truth assignment of the variables involved. It follows easily from the above rules that negation is an *involution*:

$$\neg(\neg f) \equiv f$$

and that disjunction and conjunction are *commutative*, *associative*, and *idempotent*:

$$\begin{aligned} f \vee g &\equiv g \vee f, & f \wedge g &\equiv g \wedge f \\ f \vee (g \vee h) &\equiv (f \vee g) \vee h, & f \wedge (g \wedge h) &\equiv (f \wedge g) \wedge h \\ f \vee f &\equiv f, & f \wedge f &\equiv f. \end{aligned}$$

Furthermore, disjunction and conjunction together satisfy the *distributive laws*:

$$f \vee (g \wedge h) \equiv (f \vee g) \wedge (f \vee h), \qquad f \wedge (g \vee h) \equiv (f \wedge g) \vee (f \wedge h)$$

and interact with negation according to *de Morgan's laws*:

$$\neg(f \vee g) \equiv (\neg f) \wedge (\neg g), \qquad \neg(f \wedge g) \equiv (\neg f) \vee (\neg g)$$

Finally, there are the *tautologies*:

$$f \vee \neg f = 1, \qquad f \wedge \neg f = 0.$$

Boolean formulae may be transformed into equivalent ones by applying these laws. For instance:

$$\begin{aligned} (\neg(x_1 \vee \overline{x_2}) \vee x_3) \wedge (x_2 \vee \overline{x_3}) &\equiv ((\overline{x_1} \wedge x_2) \vee x_3) \wedge (x_2 \vee \overline{x_3}) \\ &\equiv ((\overline{x_1} \vee x_3) \wedge (x_2 \vee x_3)) \wedge (x_2 \vee \overline{x_3}) \\ &\equiv (\overline{x_1} \vee x_3) \wedge ((x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3})) \\ &\equiv (\overline{x_1} \vee x_3) \wedge (x_2 \vee (x_3 \wedge \overline{x_3})) \\ &\equiv (\overline{x_1} \vee x_3) \wedge x_2 \end{aligned}$$

Satisfiability of Boolean Formulae

A boolean formula is *satisfiable* if there is a truth assignment of its variables for which the value of the formula is 1. In this case, we say that the formula is *satisfied* by the assignment. It can be seen that formula (8.2) is satisfiable, for instance by the truth assignment $x_1 = 0$, $x_2 = 1$, $x_3 = 0$. But not all boolean formulae are satisfiable ($x \wedge \overline{x}$ being a trivial example). This poses the general problem:

Problem 8.7 Boolean Satisfiability (Sat)
Given*: a boolean formula f,*
Decide*: Is f satisfiable?*

Observe that SAT belongs to $\mathcal{NP}$: given appropriate values of the variables, it can be checked in polynomial time that the value of the formula is indeed 1. These values of the variables therefore constitute a succinct certificate. Cook (1971) and Levin (1973) proved, independently, that SAT is an example of an $\mathcal{NP}$-complete problem.

Theorem 8.8 THE COOK–LEVIN THEOREM
The problem SAT *is $\mathcal{NP}$-complete.* □

The proof of the Cook–Levin Theorem involves the notion of a Turing machine, and is beyond the scope of this book. A proof may be found in Garey and Johnson (1979) or Sipser (2005).

By applying Theorem 8.8, Karp (1972) showed that many combinatorial problems are $\mathcal{NP}$-complete. One of these is DIRECTED HAMILTON CYCLE. In order to explain the ideas underlying his approach, we need a few more definitions.

A variable x, or its negation $\overline{x}$, is a *literal* , and a disjunction or conjunction of literals is a *disjunctive* or *conjunctive clause*. Because the operations of disjunction and conjunction are associative, parentheses may be dispensed with within clauses. There is no ambiguity, for example, in the following formula, a conjunction of three disjunctive clauses, each consisting of three literals.

$$f := (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$$

Any conjunction of disjunctive clauses such as this one is referred to as a formula in *conjunctive normal form*. It can be shown that every boolean formula is equivalent, via a polynomial reduction, to one in conjunctive normal form (Exercise 8.3.1). Furthermore, as we explain below in the proof of Theorem 8.10, every boolean formula in conjunctive normal form is equivalent, again via a polynomial reduction, to one in conjunctive normal form with exactly three literals per clause. The decision problem for such boolean formulae is known as 3-SAT.

Problem 8.9 BOOLEAN 3-SATISFIABILITY (3-SAT)
GIVEN*: a boolean formula f in conjunctive normal form with three literals per clause,*
DECIDE*: Is f satisfiable?*

Theorem 8.10 *The problem* 3-SAT *is $\mathcal{NP}$-complete.*

Proof By the Cook–Levin Theorem (8.8), it suffices to prove that SAT $\preceq$ 3-SAT. Let f be a boolean formula in conjunctive normal form. We show how to construct, in polynomial time, a boolean formula f' in conjunctive normal form such that:

i) each clause in f' has three literals,
ii) f is satisfiable if and only if f' is satisfiable.

Such a formula f' may be obtained by the addition of new variables and clauses, as follows.

Suppose that some clause of f has just two literals, for instance the clause $(x_1 \vee x_2)$. In this case, we simply replace this clause by two clauses with three literals, $(x_1 \vee x_2 \vee x)$ and $(\overline{x} \vee x_1 \vee x_2)$, where x is a new variable. Clearly,

$$(x_1 \vee x_2) \equiv (x_1 \vee x_2 \vee x) \wedge (\overline{x} \vee x_1 \vee x_2)$$

Clauses with single literals may be dealt with in a similar manner (Exercise 8.3.2).

Now suppose that some clause $(x_1 \vee x_2 \vee \cdots \vee x_k)$ of f has k literals, where $k \geq 4$. In this case, we add $k-3$ new variables $y_1, y_2, \ldots, y_{k-3}$ and form the following $k-2$ clauses, each with three literals.

$$(x_1 \vee x_2 \vee y_1),\ (\overline{y}_1 \vee x_3 \vee y_2),\ (\overline{y}_2 \vee x_4 \vee y_3),\ \cdots\ (\overline{y}_{k-4} \vee x_{k-2} \vee y_{k-3}),\ (\overline{y}_{k-3} \vee x_{k-1} \vee x_k)$$

One may verify that $(x_1 \vee x_2 \vee \cdots \vee x_k)$ is equivalent to the conjunction of these $k-2$ clauses. We leave the details as an exercise (8.3.3). □

Theorem 8.10 may be used to establish the $\mathcal{NP}$-completeness of decision problems in graph theory such as Directed Hamilton Cycle by means of polynomial reductions.

As we have observed, in order to show that a decision problem Q in $\mathcal{NP}$ is $\mathcal{NP}$-complete, it suffices to find a polynomial reduction to Q of a known $\mathcal{NP}$-complete problem P. This is generally easier said than done. What is needed is to first decide on an appropriate $\mathcal{NP}$-complete problem P and then come up with a suitable polynomial reduction. In the case of graphs, the latter step is often achieved by means of a construction whereby certain special subgraphs, referred to as 'gadgets', are inserted into the instance of P so as to obtain an instance of Q with the required properties. An illustration of this technique is described in the inset overleaf, where we show how 3-Sat may be reduced to Directed Hamilton Cycle via an intermediate problem, Exact Cover.

Almost all of the decision problems that we come across in this book are known to belong either to the class $\mathcal{P}$ or to the class $\mathcal{NPC}$. One notable exception is the isomorphism problem:

Problem 8.11 Graph Isomorphism
Given: *two graphs G and H,*
Decide: *Are G and H isomorphic?*

The complexity status of this problem remains a mystery. Whilst the problem clearly belongs to $\mathcal{NP}$, whether it belongs to $\mathcal{P}$, to co-$\mathcal{NP}$, or to $\mathcal{NPC}$ is not known. Polynomial-time isomorphism-testing algorithms have been found for certain classes of graphs, including planar graphs (Hopcroft and Wong (1974)) and graphs of bounded degree (Luks (1982)), but these algorithms are not valid for all graphs. Graph Isomorphism might, conceivably, be a counterexample to Conjecture 8.3.

PROOF TECHNIQUE: POLYNOMIAL REDUCTION

We establish the NP-completeness of DIRECTED HAMILTON CYCLE by reducing 3-SAT to it via an intermediate problem, EXACT COVER, which we now describe.

Let $\mathcal{A}$ be a family of subsets of a finite set X. An *exact cover* of X by $\mathcal{A}$ is a partition of X, each member of which belongs to $\mathcal{A}$. For instance, if $X := \{x_1, x_2, x_3\}$ and $\mathcal{A} := \{\{x_1\}, \{x_1, x_2\}, \{x_2, x_3\}\}$, then $\{\{x_1\}, \{x_2, x_3\}\}$ is an exact cover of X by $\mathcal{A}$. This notion gives rise to the following decision problem.

Problem 8.12 EXACT COVER
GIVEN*: a set X and a family $\mathcal{A}$ of subsets of X,*
DECIDE*: Is there an exact cover of X by $\mathcal{A}$?*

We first describe a polynomial reduction of 3-SAT to EXACT COVER, and then a polynomial reduction of EXACT COVER to DIRECTED HAMILTON CYCLE. The chain of reductions:

$$\text{SAT} \preceq 3-\text{SAT} \preceq \text{EXACT COVER} \preceq \text{DIRECTED HAMILTON CYCLE}$$

will then imply that DIRECTED HAMILTON CYCLE is $\mathcal{NP}$-complete, by virtue of the Cook–Levin Theorem (8.8).

Theorem 8.13 3-SAT $\preceq$ EXACT COVER.

Proof Let f be an instance of 3-SAT, with clauses $f_1, \ldots, f_n$ and variables $x_1, \ldots, x_m$. The first step is to construct a graph G from f, by setting:

$$\begin{aligned} V(G) &:= \{x_i : 1 \le i \le m\} \cup \{\overline{x_i} : 1 \le i \le m\} \cup \{f_j : 1 \le j \le n\} \\ E(G) &:= \{x_i\overline{x_i} : 1 \le i \le m\} \cup \{x_i f_j : x_i \in f_j\} \cup \{\overline{x_i} f_j : \overline{x_i} \in f_j\} \end{aligned}$$

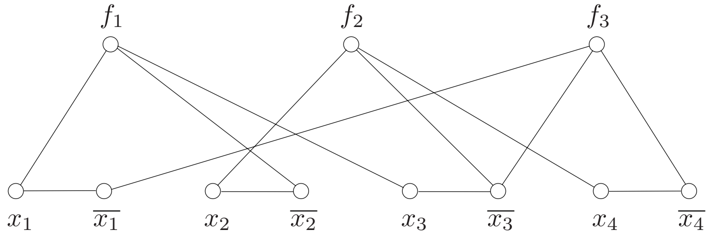
where the notation $x_i \in f_j$ ($\overline{x_i} \in f_j$) signifies that x_i ($\overline{x_i}$) is a literal of the clause f_j. The next step is to obtain an instance $(X, \mathcal{A})$ of EXACT COVER from this graph G. We do so by setting:

$$\begin{aligned} X &:= \{f_j : 1 \le j \le n\} \cup E(G) \\ \mathcal{A} &:= \{\partial(x_i) : 1 \le i \le m\} \cup \{\partial(\overline{x_i}) : 1 \le i \le m\} \\ &\quad \cup \{\{f_j\} \cup F_j : F_j \subset \partial(f_j), 1 \le j \le n\} \end{aligned}$$

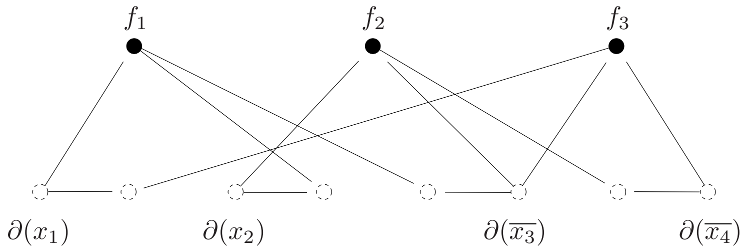
It can be verified that the formula f is satisfiable if and only if the set X has an exact cover by the family $\mathcal{A}$ (Exercise 8.3.4). □

Polynomial Reduction (continued)

For instance, if $f := (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$, the graph G obtained by this construction is:



In this example, the given formula is satisfied by the truth assignment $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$, and this truth assignment corresponds to the exact cover:



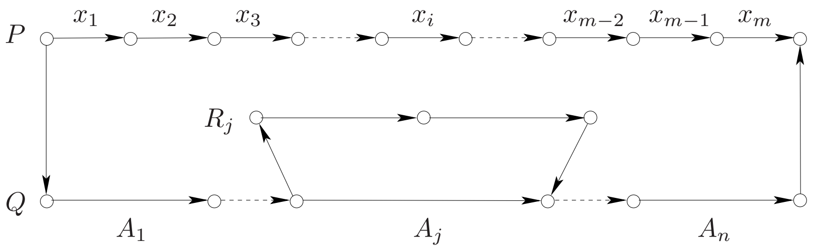
To round off the proof that Directed Hamilton Cycle is an $\mathcal{NP}$-complete problem, we describe a polynomial reduction of Exact Cover to Directed Hamilton Cycle.

Theorem 8.14 Exact Cover $\preceq$ Directed Hamilton Cycle.

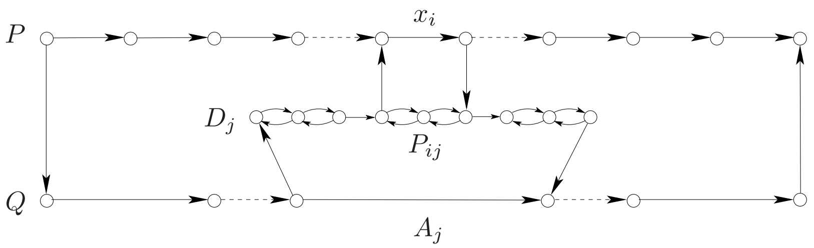
Proof Let $(X, \mathcal{A})$ be an instance of exact set cover, where $X := \{x_i : 1 \leq i \leq m\}$ and $\mathcal{A} := \{A_j : 1 \leq j \leq n\}$. We construct a directed graph G from $(X, \mathcal{A})$ as follows. Let P be a directed path whose arcs are labelled by the elements of X, Q a directed path whose arcs are labelled by the elements of $\mathcal{A}$ and, for $1 \leq j \leq n$, R_j a directed path whose vertices are labelled by the elements of A_j. The paths P, Q, and R_j, $1 \leq j \leq n$, are assumed to be pairwise disjoint. We add an arc from the initial vertex of P to the initial vertex of Q, and from the terminal vertex of Q to the terminal vertex of P.

Polynomial Reduction (continued)

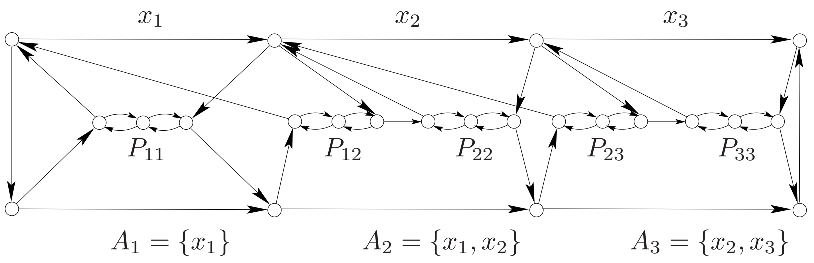
For $1 \le j \le n$, we also add an arc from the initial vertex of the arc A_j of Q to the initial vertex of R_j, and from the terminal vertex of R_j to the terminal vertex of A_j:



For $1 \le j \le n$, we now transform the directed path R_j into a digraph D_j by replacing each vertex x_i of R_j by a 'path' P_{ij} of length two whose edges are pairs of oppositely oriented arcs. Moreover, for every such 'path' P_{ij}, we add an arc from the initial vertex of P_{ij} to the initial vertex of the arc x_i of P, and one from the terminal vertex of x_i to the terminal vertex of P_{ij}:



We denote the resulting digraph by D. This construction, with $X := \{x_1, x_2, x_3\}$ and $\mathcal{A} := \{\{x_1\}, \{x_1, x_2\}, \{x_2, x_3\}\}$, is illustrated in the following figure.



Polynomial Reduction (continued)

Observe, now, that the digraph D has a directed Hamilton cycle C if and only if the set X has an exact cover by the family of subsets $\mathcal{A}$. If C does not use the arc A_j, it is obliged to traverse D_j from its initial to its terminal vertex. Conversely, if C uses the arc A_j, it is obliged to include each one of the paths P_{ij} of D_j in its route from the terminal vertex of P to the initial vertex of P. Moreover, C traces exactly one of the paths P_{ij} $(x_i \in A_j)$ in travelling from the head of the arc x_i to its tail. The arcs A_j of Q which are included in C therefore form a partition of X. Conversely, to every partition of X there corresponds a directed Hamilton cycle of D.

Finally, the numbers of vertices and arcs of D are given by:

$$\begin{aligned} v(D) &= |X| + |\mathcal{A}| + 3\textstyle\sum_{j=1}^{n} |A_j| + 2 \\ a(D) &= |X| + 2|\mathcal{A}| + 7\textstyle\sum_{j=1}^{n} |A_j| + 2 \end{aligned}$$

Because both of these parameters are bounded above by linear functions of the size of the instance $(X, \mathcal{A})$, the above reduction is indeed polynomial. □

Corollary 8.15 *The problem* Directed Hamilton Cycle *is* $\mathcal{NP}$*-complete.* □

$\mathcal{NP}$-Hard Problems

We now turn to the computational complexity of optimization problems such as the Travelling Salesman Problem (TSP) (Problem 2.6). This problem contains Hamilton Cycle as a special case. To see this, associate with a given graph G the weighted complete graph on $V(G)$ in which the weight attached to an edge uv is zero if $uv \in E(G)$, and one otherwise. The resulting weighted complete graph has a Hamilton cycle of weight zero if and only if G has a Hamilton cycle. Thus, any algorithm for solving TSP will also solve Hamilton Cycle, and we may conclude that the former problem is at least as hard as the latter. Because Hamilton Cycle is $\mathcal{NP}$-complete (see Exercise 8.3.5), TSP is at least as hard as any problem in $\mathcal{NP}$. Such problems are called *$\mathcal{NP}$-hard* .

Another basic $\mathcal{NP}$-hard problem is:

Problem 8.16 Maximum Clique (Max Clique)
Given*: a graph G,*
Find*: a maximum clique in G.*

In order to solve this problem, one needs to know, for a given value of k, whether G has a k-clique. The largest such k is called the *clique number* of G, denoted $\omega(G)$.

If k is a fixed integer not depending on n, the existence of a k-clique can be decided in polynomial time, simply by means of an exhaustive search, because the number of k-subsets of V is bounded above by n^k. However, if k depends on n, this is no longer true. Indeed, the problem of deciding whether a graph G has a k-clique, where k depends on n, is $\mathcal{NP}$-complete (Exercise 8.3.9).

The complementary notion of a clique is a *stable set*, a set of vertices no two of which are adjacent. A stable set in a graph is *maximum* if the graph contains no larger stable set. The cardinality of a maximum stable set in a graph G is called the *stability number* of G, denoted $\alpha(G)$. Clearly, a subset S of V is a stable set in G if and only if S is a clique in $\overline{G}$, the complement of G. Consequently, the following problem is polynomially equivalent to MAX CLIQUE, and thus is $\mathcal{NP}$-hard also.

Problem 8.17 MAXIMUM STABLE SET (MAX STABLE SET)
GIVEN*: a graph G,*
FIND*: a maximum stable set in G.*

Exercises

$\star$**8.3.1** Let $f := f_1 \wedge f_2 \wedge \cdots \wedge f_k$ and $g := g_1 \wedge g_2 \wedge \cdots \wedge g_\ell$ be two boolean formulae in conjunctive normal form (where f_i, $1 \le i \le k$, and g_j, $1 \le j \le \ell$, are disjunctive clauses).

a) Show that:
   i) $f \wedge g$ is in conjunctive normal form,
   ii) $f \vee g$ is equivalent to a boolean formula in conjunctive normal form,
   iii) $\neg f$ is in disjunctive normal form, and is equivalent to a boolean formula in conjunctive normal form.
b) Deduce that every boolean formula is equivalent to a boolean formula in conjunctive normal form.

$\star$**8.3.2** Show that every clause consisting of just one literal is equivalent to a boolean formula in conjunctive normal form with exactly three literals per clause.

$\star$**8.3.3** Let $(x_1 \vee x_2 \vee \cdots \vee x_k)$ be a disjunctive clause with k literals, where $k \ge 4$, and let $y_1, y_2, \ldots, y_{k-2}$ be boolean variables. Show that:

$$\begin{aligned}(x_1 \vee x_2 \vee \cdots \vee x_k) \equiv {} & (x_1 \vee x_2 \vee y_1) \wedge (\overline{y}_1 \vee x_3 \vee y_2) \wedge (\overline{y}_2 \vee x_4 \vee y_3) \wedge \cdots \\ & \cdots \wedge (\overline{y}_{k-4} \vee x_{k-2} \vee y_{k-3}) \wedge (\overline{y}_{k-3} \vee x_{k-1} \vee x_k)\end{aligned}$$

$\star$**8.3.4** Let $f := f_1 \wedge f_2 \wedge \cdots \wedge f_n$ be an instance of 3-SAT, with variables $x_1, x_2, \ldots, x_m$. Form a graph G from f, and an instance $(X, \mathcal{A})$ of EXACT COVER from G, as described in the proof of Theorem 8.13.

a) Show that the formula f is satisfiable if and only if the set X has an exact cover by the family $\mathcal{A}$.

b) Show also that the pair $(X, \mathcal{A})$ can be constructed from f in polynomial time (in the parameters m and n).
c) Deduce that Exact Cover $\in \mathcal{NPC}$.
d) Explain why constructing a graph in the same way, but from an instance of Sat rather than 3-Sat, does not provide a polynomial reduction of Sat to Exact Cover.

⋆8.3.5

a) Describe a polynomial reduction of Directed Hamilton Cycle to Hamilton Cycle.
b) Deduce that Hamilton Cycle $\in \mathcal{NPC}$.

8.3.6 Let Hamilton Path denote the problem of deciding whether a given graph has a Hamilton path.

a) Describe a polynomial reduction of Hamilton Cycle to Hamilton Path.
b) Deduce that Hamilton Path $\in \mathcal{NPC}$.

8.3.7 Two problems P and Q are *polynomially equivalent*, written $P \equiv Q$, if $P \preceq Q$ and $Q \preceq P$.

a) Show that:

$$\text{Hamilton Path} \equiv \text{Hamilton Cycle} \equiv \text{Directed Hamilton Cycle}$$

b) Let Max Path denote the problem of finding the length of a longest path in a given graph. Show that Max Path $\equiv$ Hamilton Path.

8.3.8

a) Let k be a fixed positive integer. Describe a polynomial-time algorithm for deciding whether a given graph has a path of length k.
b) The length of a longest path in a graph G can be determined by checking, for each k, $1 \leq k \leq n$, whether G has a path of length k. Does your algorithm for the problem in part (a) lead to a polynomial-time algorithm for Max Path?

⋆8.3.9

a) Let $f = f_1 \wedge f_2 \wedge \cdots \wedge f_k$ be an instance of 3-Sat (where the f_i, $1 \leq i \leq k$, are disjunctive clauses, each containing three literals). Construct a k-partite graph G on $7k$ vertices (seven vertices in each part) such that f is satisfiable if and only if G has a k-clique.
b) Deduce that 3-Sat and Max Clique are polynomially equivalent.

8.3.10 Let k be a positive integer. The following problem is a generalization of 3-Sat.

Problem 8.18 Boolean k-Satisfiability *(k-Sat)*
Given*: a boolean formula f in conjunctive normal form with k literals per clause,*
Decide*: Is f satisfiable?*

Show that:

a) 2-Sat $\in \mathcal{P}$,
b) k-Sat $\in \mathcal{NPC}$ for $k \geq 3$.

8.4 Approximation Algorithms

For $\mathcal{NP}$-hard optimization problems of practical interest, such as the Travelling Salesman Problem, the best that one can reasonably expect of a polynomial-time algorithm is that it should always return a feasible solution which is not too far from optimality.

Given a real number $t \geq 1$, a *t-approximation algorithm* for a minimization problem is an algorithm that accepts any instance of the problem as input and returns a feasible solution whose value is no more than t times the optimal value; the smaller the value of t, the better the approximation. Naturally, the running time of the algorithm is an equally important factor. We give two examples.

Problem 8.19 Maximum Cut (Max Cut)
Given*: a weighted graph (G, w),*
Find*: a maximum-weight spanning bipartite subgraph F of G.*

This problem admits a polynomial-time 2-approximation algorithm, based on the ideas for the unweighted case presented in Chapter 2 (Exercise 2.2.2). We leave the details as an exercise (8.4.1).

A somewhat less simple approximation algorithm was obtained by Rosenkrantz et al. (1974), who considered the special case of the Travelling Salesman Problem in which the weights satisfy the *triangle inequality*:

$$w(xy) + w(yz) \geq w(xz), \quad \text{for any three vertices } x, y, z. \tag{8.3}$$

Problem 8.20 Metric Travelling Salesman Problem (Metric TSP)
Given*: a weighted complete graph G whose weights satisfy inequality (8.3),*
Find*: a minimum-weight Hamilton cycle C of G.*

Theorem 8.21 Metric TSP *admits a polynomial-time 2-approximation algorithm.*

Proof Applying the Jarnìk–Prim Algorithm (6.9), we first find a minimum-weight spanning tree T of G. Suppose that C is a minimum-weight Hamilton cycle of G. By deleting any edge of C, we obtain a Hamilton path P of G. Because P is a spanning tree of G and T is a spanning tree of minimum weight,

$$w(T) \leq w(P) \leq w(C)$$

We now duplicate each edge of T, thereby obtaining a connected even graph H with $V(H) = V(G)$ and $w(H) = 2w(T)$. Note that this graph H is not even a subgraph of G, let alone a Hamilton cycle. The idea is to transform H into a Hamilton cycle of G, and to do so without increasing its weight. More precisely, we construct a sequence $H_0, H_1, \ldots, H_{n-2}$ of connected even graphs, each with vertex set $V(G)$, such that $H_0 = H$, H_{n-2} is a Hamilton cycle of G, and $w(H_{i+1}) \leq w(H_i)$, $0 \leq i \leq n-3$. We do so by reducing the number of edges, one at a time, as follows.

Let C_i be an Euler tour of H_i, where $i < n-2$. The graph H_i has $2(n-2)-i > n$ edges, and thus has a vertex v of degree at least four. Let xe_1ve_2y be a segment of the tour C_i; it will follow by induction that $x \neq y$. We replace the edges e_1 and e_2 of C_i by a new edge e of weight $w(xy)$ linking x and y, thereby bypassing v and modifying C_i to an Euler tour C_{i+1} of $H_{i+1} := (H_i \setminus \{e_1, e_2\}) + e$. By the triangle inequality (8.3),

$$w(H_{i+1}) = w(H_i) - w(e_1) - w(e_2) + w(e) \leq w(H_i)$$

The final graph H_{n-2}, being a connected even graph on n vertices and n edges, is a Hamilton cycle of G. Furthermore,

$$w(H_{n-2}) \leq w(H_0) = 2w(T) \leq 2w(C) \qquad \square$$

The relevance of minimum-weight spanning trees to the Travelling Salesman Problem was first observed by Kruskal (1956). A $\frac{3}{2}$-approximation algorithm for Metric TSP was found by Christofides (1976). This algorithm makes use of a polynomial-time algorithm for weighted matchings (discussed in Chapter 16; see Exercise 16.4.24). For other approaches to the Travelling Salesman Problem, see Jünger et al. (1995).

The situation with respect to the general Travelling Salesman Problem, in which the weights are not subject to the triangle inequality, is dramatically different: for any integer $t \geq 2$, there cannot exist a polynomial-time t-approximation algorithm for solving TSP unless $\mathcal{P} = \mathcal{NP}$ (Exercise 8.4.4). The book by Vazirani (2001) treats the topic of approximation algorithms in general. For the state of the art regarding computational aspects of TSP, we refer the reader to Applegate et al. (2007).

Exercises

$\star$**8.4.1** Describe a polynomial-time 2-approximation algorithm for Max Cut (Problem 8.19).

8.4.2 Euclidean TSP
The *Euclidean Travelling Salesman Problem* is the special case of Metric TSP in which the vertices of the graph are points in the plane, the edges are straight-line

segments linking these points, and the weight of an edge is its length. Show that, in any such graph, the minimum-weight Hamilton cycles are crossing-free (that is, no two of their edges cross).

8.4.3 Show that Metric TSP is $\mathcal{NP}$-hard.

$\star$**8.4.4**

a) Let G be a simple graph with $n \geq 3$, and let t be a positive integer. Consider the weighted complete graph (K, w), where $K := G \cup \overline{G}$, in which $w(e) := 1$ if $e \in E(G)$ and $w(e) := (t-1)n + 2$ if $e \in E(\overline{G})$. Show that:
   i) (K, w) has a Hamilton cycle of weight n if and only if G has a Hamilton cycle,
   ii) any Hamilton cycle of (K, w) of weight greater than n has weight at least $tn + 1$.

b) Deduce that, unless $\mathcal{P} = \mathcal{NP}$, there cannot exist a polynomial-time t-approximation algorithm for solving TSP.

8.5 Greedy Heuristics

A *heuristic* is a computational procedure, generally based on some simple rule, which intuition tells one should usually yield a good approximate solution to the problem at hand.

One particularly simple and natural class of heuristics is the class of greedy heuristics. Informally, a *greedy heuristic* is a procedure which selects the best current option at each stage, without regard to future consequences. As can be imagined, such an approach rarely leads to an optimal solution in each instance. However, there are cases in which the greedy approach does indeed work. In such cases, we call the procedure a *greedy algorithm.* The following is a prototypical example of such an algorithm.

The Borůvka–Kruskal Algorithm

The Jarník–Prim algorithm for the Minimum-Weight Spanning Tree Problem, described in Section 6.2, starts with the root and determines a nested sequence of trees, terminating with a minimum-weight spanning tree. Another algorithm for this problem, due to Borůvka (1926a,b) and, independently, Kruskal (1956), starts with the empty spanning subgraph and finds a nested sequence of forests, terminating with an optimal tree. This sequence is constructed by adding edges, one at a time, in such a way that the edge added at each stage is one of minimum weight, subject to the condition that the resulting subgraph is still a forest.

Algorithm 8.22 The Borůvka–Kruskal Algorithm

```
Input: a weighted connected graph $G = (G, w)$
Output: an optimal tree $T = (V, F)$ of $G$, and its weight $w(F)$
1: set $F := \emptyset$, $w(F) := 0$ ($F$ denotes the edge set of the current forest)
2: while there is an edge $e \in E \setminus F$ such that $F \cup \{e\}$ is the edge set of a forest do
3:     choose such an edge $e$ of minimum weight
4:     replace $F$ by $F \cup \{e\}$ and $w(F)$ by $w(F) + w(e)$
5: end while
6: return $((V, F), w(F))$
```

Because the graph G is assumed to be connected, the forest (V, F) returned by the Borůvka–Kruskal Algorithm is a spanning tree of G. We call it a *Borůvka–Kruskal tree.* The construction of such a tree in the electric grid graph of Section 6.2 is illustrated in Figure 8.3. As before, the edges are numbered according to the order in which they are added. Observe that this tree is identical to the one returned by the Jarník–Prim Algorithm (even though its edges are selected in a different order). This is because all the edge weights in the electric grid graph happen to be distinct (see Exercise 6.2.1).

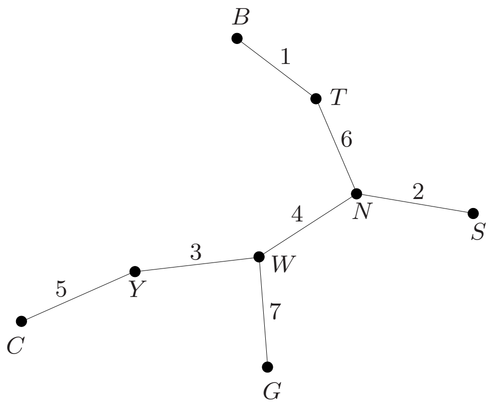


Fig. 8.3. An optimal tree returned by the Borůvka–Kruskal Algorithm

In order to implement the Borůvka–Kruskal Algorithm efficiently, one needs to be able to check easily whether a candidate edge links vertices in different components of the forest. This can be achieved by colouring vertices in the same component by the same colour and vertices in different components by distinct colours. It then suffices to check that the ends of the edge have different colours. Once the edge has been added to the forest, all the vertices in one of the two merged components are recoloured with the colour of the other component. We leave the details as an exercise (Exercise 8.5.1).

The following theorem shows that the Borůvka–Kruskal Algorithm runs correctly. Its proof resembles that of Theorem 6.10, and we leave it to the reader (Exercise 8.5.2).

Theorem 8.23 *Every Borůvka–Kruskal tree is an optimal tree.* □

The problem of finding a maximum-weight spanning tree of a connected graph can be solved by the same approach; at each stage, instead of picking an edge of minimum weight subject to the condition that the resulting subgraph remains a forest, we pick one of maximum weight subject to the same condition (see Exercise 8.5.3). The origins of the Borůvka–Kruskal Algorithm are recounted in Nešetřil et al. (2001) and Kruskal (1997).

Independence Systems

One can define a natural family of greedy heuristics which includes the Borůvka–Kruskal Algorithm in the framework of set systems.

A set system $(V, \mathcal{F})$ is called an *independence system* on V if $\mathcal{F}$ is nonempty and, for any member F of $\mathcal{F}$, all subsets of F also belong to $\mathcal{F}$. The members of $\mathcal{F}$ are then referred to as *independent sets* and their maximal elements as *bases.* (The independent sets of a matroid, defined in Section 4.4, form an independence system.)

Many independence systems can be defined on graphs. For example, if $G = (V, E)$ is a connected graph, we may define an independence system on V by taking as independent sets the cliques of G (including the empty set). Likewise, we may define an independence system on E by taking the edge sets of the forests of G as independent sets; the bases of this independence system are the edge sets of spanning trees. (This latter independence system is the cycle matroid of the graph, defined in Section 4.4.)

Consider, now, an arbitrary independence system $(V, \mathcal{F})$. Suppose that, with each element x of V, there is an associated nonnegative weight $w(x)$, and that we wish to find an independent set of maximum weight, where the *weight* of a set is defined to be the sum of the weights of its elements. A naive approach to this problem would be to proceed as follows.

Heuristic 8.24 Greedy Heuristic (for independence systems)

Input: an independence system $(V, \mathcal{F})$ with weight function $w : V \to \mathbb{R}^+$
Output: a maximal independent set F of $(V, \mathcal{F})$, and its weight $w(F)$

```
1: set $F := \emptyset$, $w(F) := 0$
2: **while** there is an element $x \in V \setminus F$ such that $F \cup \{x\}$ is independent **do**
3:     choose such an element $x$ of minimum weight $w(x)$
4:     replace $F$ by $F \cup \{x\}$ and $w(F)$ by $w(F) + w(x)$
5: **end while**
6: return $(F, w(X))$
```

As we have seen, the GREEDY HEURISTIC always returns an optimal solution when the independence system consists of the edge sets of the forests of a graph, no matter what the edge weights. (More generally, as Rado (1957) observed, the GREEDY HEURISTIC performs optimally whenever $\mathcal{F}$ is the family of independent sets of a matroid, regardless of the weight function w.) By contrast, when the independent sets are the cliques of a graph, the GREEDY HEURISTIC rarely returns an optimal solution, even if all the weights are 1, because most graphs have maximal cliques which are not maximum cliques. (If the GREEDY HEURISTIC unfailingly returns a maximum weight independent set, no matter what the weight function of the independence system, then the system must, it turns out, be a matroid (see, for example, Oxley (1992)).)

We remark that greedy heuristics are by no means limited to the framework of independence systems. For example, if one is looking for a longest x-path in a graph, an obvious greedy heuristic is to start with the trivial x-path consisting of just the vertex x and iteratively extend the current x-path by any available edge. This amounts to applying depth-first search from x, stopping when one is forced to backtrack. The path so found will certainly be a maximal x-path, but not necessarily a longest x-path. Even so, this simple-minded greedy heuristic proves to be effective when combined with other ideas, as we show in Chapter 18.

Exercises

$\star$**8.5.1** Refine the Borůvka–Kruskal Algorithm in such a way that, at each stage, vertices in the same component of the forest F are assigned the same colour and vertices in different components are assigned distinct colours.

$\star$**8.5.2** Prove Theorem 8.23.

$\star$**8.5.3** Show that the problem of finding a maximum-weight spanning tree of a connected graph can be solved by choosing, at each stage, an edge of maximum weight subject to the condition that the resulting subgraph is still a forest.

8.5.4 Give an example of a weighted independence system, all of whose bases have the same number of elements, but for which the GREEDY HEURISTIC (8.24) fails to return an optimal solution.

8.5.5 Consider the following set of real vectors.

$$V := \{(1,0,0,1),(1,1,1,1),(1,0,1,2),(0,1,0,1),(0,2,-1,1),(1,-1,0,0)\}$$

Find a linearly independent subset of V whose total number of zeros is maximum by applying the GREEDY HEURISTIC (8.24).

8.6 Linear and Integer Programming

A *linear program* (*LP*) is a problem of maximizing or minimizing a linear function of real variables that are subject to linear equality or inequality constraints. By means of simple substitutions, such as replacing an equation by two inequalities, any LP may be transformed into one of the following form.

$$\text{maximize } \mathbf{cx} \quad \text{subject to} \quad \mathbf{Ax} \le \mathbf{b} \quad \text{and} \quad \mathbf{x} \ge \mathbf{0} \tag{8.4}$$

where $\mathbf{A} = (a_{ij})$ is an $m \times n$ matrix, $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ a $1 \times n$ row vector, and $\mathbf{b} = (b_1, b_2, \ldots, b_m)$ an $m \times 1$ column vector. The m inequalities $\sum_{j=1}^{n} a_{ij}x_j \le b_i$, $1 \le i \le m$, and the n nonnegativity conditions $x_j \ge 0$, $1 \le j \le n$, are known as the *constraints* of the problem. The function $\mathbf{cx}$ to be maximized is called the *objective function*.

A column vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ in $\mathbb{R}^n$ is a *feasible solution* to (8.4) if it satisfies all $m + n$ constraints, and a feasible solution at which the objective function $\mathbf{cx}$ attains its maximum is an *optimal solution*. This maximum is the *optimal value* of the LP.

Associated with every LP, there is another LP, called its dual. The *dual* of the LP (8.4) is the LP:

$$\text{minimize } \mathbf{yb} \quad \text{subject to} \quad \mathbf{yA} \ge \mathbf{c} \quad \text{and} \quad \mathbf{y} \ge \mathbf{0} \tag{8.5}$$

With reference to this dual LP, the original LP (8.4) is called the *primal* LP.

Not every LP has a feasible solution. Moreover, even if it does have one, it need not have an optimal solution: the objective function might be unbounded over the set of feasible solutions, and thus not achieve a maximum (or minimum). Such an LP is said to be *unbounded*.

The following proposition implies that if both the primal and the dual have feasible solutions, neither is unbounded.

Proposition 8.25 Weak Duality Theorem
Let $\mathbf{x}$ *be a feasible solution to (8.4) and* $\mathbf{y}$ *a feasible solution to its dual (8.5). Then*

$$\mathbf{cx} \le \mathbf{yb} \tag{8.6}$$

Proof Because $\mathbf{c} \le \mathbf{yA}$ and $\mathbf{x} \ge \mathbf{0}$, we have $\mathbf{cx} \le \mathbf{yAx}$. Likewise $\mathbf{yAx} \le \mathbf{yb}$. Inequality (8.6) follows. □

Corollary 8.26 *Let* $\mathbf{x}$ *be a feasible solution to (8.4) and* $\mathbf{y}$ *a feasible solution to its dual (8.5). Suppose that* $\mathbf{cx} = \mathbf{yb}$. *Then* $\mathbf{x}$ *is an optimal solution to (8.4) and* $\mathbf{y}$ *is an optimal solution to (8.5).* □

The significance of this corollary is that if equality holds in (8.6), the primal solution $\mathbf{x}$ serves as a succinct certificate for the optimality of the dual solution $\mathbf{y}$, and *vice versa*. A remarkable and fundamental theorem due to von Neumann (1928) guarantees that one can always certify optimality in this manner.

Theorem 8.27 Duality Theorem
If an LP has an optimal solution, then its dual also has an optimal solution, and the optimal values of these two LPs are equal. □

A wide variety of graph-theoretical problems may be formulated as LPs, albeit with additional *integrality constraints*, requiring that the variables take on only integer values. In some cases, these additional constraints may be ignored without affecting the essential nature of the problem, because the LP under consideration can be shown to always have optimal solutions that are integral. In such cases, the duals generally have natural interpretations in terms of graphs, and interesting results may be obtained by applying the Duality Theorem. Such results are referred to as *min–max theorems.* As a simple example, let us consider the problem of finding a maximum stable set in a graph.

It clearly suffices to consider graphs without isolated vertices. Let G be such a graph. With any stable set S of G, we may associate its $(0,1)$-valued incidence vector $\mathbf{x} := (x_v : v \in V)$, where $x_v := 1$ if $v \in S$, and $x_v := 0$ otherwise. Because no stable set can include more than one of the two ends of any edge, every such vector $\mathbf{x}$ satisfies the constraint $x_u + x_v \leq 1$, for each $uv \in E$. Thus, the problem Max Stable Set is equivalent to the following LP (where $\mathbf{M}$ is the incidence matrix of G).

$$\text{maximize } \mathbf{1x} \ \text{ subject to } \ \mathbf{M}^t\mathbf{x} \leq \mathbf{1} \ \text{ and } \ \mathbf{x} \geq \mathbf{0} \tag{8.7}$$

together with the requirement that $\mathbf{x}$ be integral. The dual of (8.7) is the following LP, in which there is a variable y_e for each edge e of G.

$$\text{minimize } \mathbf{y1} \ \text{ subject to } \ \mathbf{yM}^t \geq \mathbf{1} \ \text{ and } \ \mathbf{y} \geq \mathbf{0} \tag{8.8}$$

Consider an integer-valued feasible solution $\mathbf{y}$ to this dual LP. The support of $\mathbf{y}$ is a set of edges of G that together meet every vertex of G. Such a set of edges is called an *edge covering* of G. The number of edges in a minimum edge covering of a graph G without isolated vertices is denoted by $\beta'(G)$.

Conversely, the incidence vector of any edge covering of G is a feasible solution to (8.8). Thus the optimal value of (8.8) is a lower bound on $\beta'(G)$. Likewise, the optimal value of (8.7) is an upper bound on $\alpha(G)$. By the Weak Duality Theorem, it follows that, for any graph G without isolated vertices, $\alpha(G) \leq \beta'(G)$. In general, these two quantities are not equal (consider, for example, K_3). They are, however, always equal for bipartite graphs (see inset).

A linear program in which the variables are constrained to take on only integer values is called an *integer linear program* (*ILP*). Any ILP may be transformed to one of the following form.

$$\text{maximize } \mathbf{cx} \ \text{ subject to } \ \mathbf{Ax} \leq \mathbf{b}, \ \ \mathbf{x} \geq \mathbf{0}, \ \text{ and } \ \mathbf{x} \in \mathbb{Z} \tag{8.9}$$

As already mentioned, Max Stable Set can be formulated as an ILP. Because Max Stable Set is $\mathcal{NP}$-hard, so is ILP. On the other hand, there do exist polynomial-time algorithms for solving linear programs, so LP is in $\mathcal{P}$.

Proof Technique: Total Unimodularity

Recall that a matrix $\mathbf{A}$ is *totally unimodular* if the determinant of each of its square submatrices is equal to 0, +1, or −1. The following theorem provides a sufficient condition for an LP to have an integer-valued optimal solution.

Theorem 8.28 *Suppose that* $\mathbf{A}$ *is a totally unimodular matrix and that* $\mathbf{b}$ *is an integer vector. If (8.4) has an optimal solution, then it has an integer optimal solution.*

Proof The set of points in $\mathbb{R}^n$ at which any single constraint holds with equality is a hyperplane in $\mathbb{R}^n$. Thus each constraint is satisfied by the points of a closed half-space of $\mathbb{R}^n$, and the set of feasible solutions is the intersection of all these half-spaces, a convex polyhedron P.

Because the objective function is linear, its level sets are hyperplanes. Thus, if the maximum value of $\mathbf{cx}$ over P is z^*, the hyperplane $\mathbf{cx} = z^*$ is a supporting hyperplane of P. Hence $\mathbf{cx} = z^*$ contains an extreme point (a corner) of P. It follows that the objective function attains its maximum at one of the extreme points of P.

Every extreme point of P is at the intersection of n or more hyperplanes determined by the constraints. It is thus a solution to a subsystem of $\mathbf{Ax} = \mathbf{b}$. Using the hypotheses of the theorem and applying Cramér's rule, we now conclude that each extreme point of P is an integer vector, and hence that (8.4) has an integer optimal solution. □

Because the incidence matrix of a bipartite graph is totally unimodular (Exercise 4.2.4), as a consequence of the above theorem, we have:

Theorem 8.29 *Let G be a bipartite graph with incidence matrix* $\mathbf{M}$. *Then the LPs*

$$\text{maximize } \mathbf{1x} \quad \text{subject to} \quad \mathbf{M}^t\mathbf{x} \leq \mathbf{1} \quad \text{and} \quad \mathbf{x} \geq \mathbf{0} \tag{8.10}$$

$$\text{minimize } \mathbf{y1} \quad \text{subject to} \quad \mathbf{yM}^t \geq \mathbf{1} \quad \text{and} \quad \mathbf{y} \geq \mathbf{0} \tag{8.11}$$

both have $(0,1)$*-valued optimal solutions.* □

This theorem, in conjunction with the Duality Theorem, now implies the following min–max equality, due independently to D. König and R. Rado (see Schrijver (2003)).

Theorem 8.30 The König–Rado Theorem
In any bipartite graph without isolated vertices, the number of vertices in a maximum stable set is equal to the number of edges in a minimum edge covering. □

Total Unimodularity (continued)

The König–Rado Theorem (8.30) implies that the problem of deciding whether a bipartite graph has a stable set of cardinality k is in co-$\mathcal{NP}$; when the answer is 'no', an edge cover of size less than k provides a succinct certificate of this fact. In fact, as shown in Chapter 16, there is a polynomial algorithm for finding a maximum stable set in any bipartite graph.

A second application of this proof technique is given below, and another is presented in Section 19.3.

One approach to the problem of determining the value of a graph-theoretic parameter such as α is to express the problem as an ILP of the form (8.9) and then solve its *LP relaxation*, that is, the LP (8.4) obtained by dropping the integrality constraint $\mathbf{x} \in \mathbb{Z}$. If the optimal solution found happens to be integral, as in Theorem 8.29, it will also be an optimal solution to the ILP, and thus determine the exact value of the parameter. In any event, the value returned by the LP will be an upper bound on the value of the parameter. This upper bound is referred to as the *fractional version* of the parameter. For example, the LP (8.10) returns the *fractional stability number*, denoted α^*.

The fractional stability number of a graph may be computed in polynomial time. However, in general, α can be very much smaller than α^*. For example, $\alpha(K_n) = 1$, whereas $\alpha^*(K_n) = n/2$ for $n \geq 2$. Taking into cognisance the fact that no stable set of a graph can include more than one vertex of any clique of the graph, one may obtain a LP associated with Max Stable Set with tighter constraints than (8.7) (see Exercise 8.6.3).

Matchings and Coverings in Bipartite Graphs

We now describe a second application of total unimodularity, to matchings in bipartite graphs. A *matching* in a graph is a set of pairwise nonadjacent links. With any matching M of a graph G, we may associate its $(0, 1)$-valued incidence vector. Since no matching has more than one edge incident with any vertex, every such vector $\mathbf{x}$ satisfies the constraint $\sum\{x_e : e \in \partial(v)\} \leq 1$, for all $v \in V$. Thus the problem of finding a largest matching in a graph is equivalent to the following ILP.

$$\text{maximize } \mathbf{1x} \text{ subject to } \mathbf{Mx} \leq \mathbf{1} \text{ and } \mathbf{x} \geq \mathbf{0} \tag{8.12}$$

(where $\mathbf{M}$ is the incidence matrix of G), together with the requirement that $\mathbf{x}$ be integral. The dual of (8.12) is the following LP.

$$\text{minimize } \mathbf{y1} \text{ subject to } \mathbf{yM} \geq \mathbf{1} \text{ and } \mathbf{y} \geq \mathbf{0} \tag{8.13}$$

Because the incidence matrix of a bipartite graph is totally unimodular (Exercise 4.2.4), as a consequence of the above theorem, we now have:

Theorem 8.31 *When G is bipartite, (8.12) and (8.13) have* $(0,1)$*-valued optimal solutions.* □

If $\mathbf{x}$ is a $(0,1)$-valued feasible solution to (8.12), then no two edges of the set $M := \{e \in E : x_e = 1\}$ have an end in common; that is, M is a matching of G. Analogously, if $\mathbf{y}$ is a $(0,1)$-valued feasible solution to (8.13), then each edge of G has at least one end in the set $K := \{v \in V : y_v = 1\}$; such a set is called a *covering* of G. These two observations, together with the Duality Theorem, now imply the following fundamental min–max theorem, due independently to König (1931) and Egerváry (1931).

Theorem 8.32 The König–Egerváry Theorem
In any bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimum covering. □

Just as the König–Rado Theorem (8.30) shows that the problem of deciding whether a bipartite graph $G[X, Y]$ has a stable set of k vertices is in co-$\mathcal{NP}$, the König–Egerváry Theorem shows that the problem of deciding whether such a graph has a matching of k edges is in co-$\mathcal{NP}$. When the answer is 'no', a covering of cardinality less than k provides a succinct certificate of this fact. The König–Egerváry Theorem can also be derived from the arc version of Menger's Theorem (7.16) (see Exercise 8.6.7). The maximum number of edges in a matching of a graph G is called the *matching number* of G and denoted $\alpha'(G)$.

If G is nonbipartite, (8.12) may have optimal solutions that are not integral. For example, when G is a triangle, it can be seen that $(1/2, 1/2, 1/2)$ is an optimal solution. However, Edmonds (1965b) showed that one may introduce additional constraints that are satisfied by all the incidence vectors of matchings in a graph so that the resulting linear program has integer optimal solutions (see Exercise 8.6.8). This was the basis for his solution to the optimal matching problem. Matchings are discussed in detail in Chapter 16.

Using the fact that the incidence matrix of a directed graph is totally unimodular (Exercise 1.5.7a), Menger's Theorem (7.16) may be derived from the Duality Theorem. Further examples of min–max theorems are presented in Chapters 16 and 19. For additional information on these and other applications of linear programming, see Chvátal (1983), Lovász and Plummer (1986), and Schrijver (2003).

Exercises

8.6.1 Complementary Slackness
Let $\mathbf{x}$ and $\mathbf{y}$ be feasible solutions to the LP (8.4) and its dual (8.5), respectively. Show that these solutions are optimal if and only if:

$$\sum_{j=1}^{n} a_{ij}x_j < b_i \Rightarrow y_i = 0, \quad 1 \le i \le m, \quad \text{and}$$
$$\sum_{i=1}^{m} a_{ij}y_i > c_j \Rightarrow x_j = 0, \quad 1 \le j \le n$$

(The above conditions are known as the *complementary slackness conditions* for optimality. They play an important role in the solution of optimization problems involving weighted graphs.)

$\star$**8.6.2** Clique Covering
A *clique covering* of a graph is a set of cliques whose union is the entire vertex set of the graph.

a) Show that the stability number of a graph is bounded above by the minimum number of cliques in a clique covering.
b) Give an example of a graph in which these two quantities are unequal.

8.6.3 Let $\mathcal{K}$ denote the set of all cliques of a graph G and let $\mathbf{K}$ denote the incidence matrix of the hypergraph $(V, \mathcal{K})$. Consider the LP:

$$\text{maximize } \mathbf{1x} \ \text{ subject to } \ \mathbf{K}^t\mathbf{x} \leq \mathbf{1} \ \text{ and } \ \mathbf{x} \geq \mathbf{0} \tag{8.14}$$

and its dual:

$$\text{minimize } \mathbf{y1} \ \text{ subject to } \ \mathbf{yK}^t \geq \mathbf{1} \ \text{ and } \ \mathbf{y} \geq \mathbf{0} \tag{8.15}$$

Show that:

a) an integer-valued vector $\mathbf{x}$ in $\mathbb{R}^V$ is a feasible solution to (8.14) if and only if it is the incidence vector of a stable set of G,
b) a $(0,1)$-valued vector $\mathbf{y}$ in $\mathbb{R}^{\mathcal{K}}$ is a feasible solution to (8.15) if and only if it is the incidence vector of a clique covering.

8.6.4 Show that the set of feasible solutions to (8.14) is a subset of the set of feasible solutions to (8.7), with equality when G is triangle-free.

8.6.5 Let $\alpha^{**}(G)$ denote the optimal value of (8.14).

a) Show that, for any graph G, $\alpha \leq \alpha^{**} \leq \alpha^*$.
b) Give examples of graphs for which these inequalities are strict.

8.6.6 Let G be a simple graph with $n \geq 3$, and let $\mathbf{x} := (x_e : e \in E) \in \mathbb{R}^E$. Consider the following system of linear inequalities.

$$\begin{aligned}
\sum_{e \in \partial(X)} x_e &\geq 2 \ \text{ if } \ \emptyset \subset X \subset V \\
\sum_{e \in \partial(v)} x_e &= 2 \ \text{ for all } \ v \in V \\
x_e &\leq 1 \ \text{ for all } \ e \in E \\
x_e &\geq 0 \ \text{ for all } \ e \in E
\end{aligned}$$

a) Show that the integer-valued feasible solutions to this system are precisely the incidence vectors of the Hamilton cycles of G.

b) Let $\mathbf{c} \in \mathbb{R}^E$ be a weight function on G. Deduce from (a) that an optimal solution to the TSP for this weighted graph is provided by an optimal solution to the ILP that consists of maximizing the objective function $\mathbf{cx}$ subject to the above constraints, together with the integrality constraint $\mathbf{x} \in \mathbb{Z}$.
(Grötschel et al. (1988) have given a polynomial-time algorithm for solving the LP relaxation of this ILP.)

$\star$**8.6.7**

a) Transform the problem of finding a maximum matching in a bipartite graph $G[X,Y]$ into the problem of finding a maximum collection of arc-disjoint directed (x,y)-paths in a related digraph $D(x,y)$.
b) Deduce the König–Egerváry Theorem from Menger's Theorem (7.16).

8.6.8 Show that every integer feasible solution $\mathbf{x}$ to the LP (8.12) satisfies the inequality

$$\sum_{e \in E(X)} x_e \leq \frac{1}{2}(|X|-1)$$

for any odd subset X of V of cardinality three or more.
(Edmonds (1965b) showed that, by adding these inequalities to the set of constraints in (8.12), one obtains an LP every optimal solution of which is $(0,1)$-valued.)

$\star$**8.6.9** Farkas' Lemma
The following two LPs are duals of each other.

$$\begin{array}{llll} \text{maximize} & \mathbf{0x} & \text{subject to} & \mathbf{Ax} = \mathbf{0} \quad \text{and} \quad \mathbf{x} \geq \mathbf{b} \\ \text{minimize} & -\mathbf{zb} & \text{subject to} & \mathbf{yA} - \mathbf{z} = \mathbf{0} \quad \text{and} \quad \mathbf{z} \geq \mathbf{0} \end{array}$$

Farkas' Lemma (see Section 20.1) says that exactly one of the two linear systems:

$$\mathbf{Ax} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{b} \quad \text{and} \quad \mathbf{yA} \geq \mathbf{0}, \quad \mathbf{yAb} > 0$$

has a solution. Deduce Farkas' Lemma from the Duality Theorem (8.27).

$\star$**8.6.10** The following two LPs are duals of each other.

$$\begin{array}{llll} \text{minimize} & \mathbf{y0} & \text{subject to} & \mathbf{yA} \geq \mathbf{b} \\ \text{maximize} & \mathbf{bx} & \text{subject to} & \mathbf{Ax} = \mathbf{0} \quad \text{and} \quad \mathbf{x} \geq \mathbf{0} \end{array}$$

A variant of Farkas' Lemma says that exactly one of the two linear systems:

$$\mathbf{yA} \geq \mathbf{b} \quad \text{and} \quad \mathbf{Ax} = \mathbf{0}, \quad \mathbf{x} \geq \mathbf{0}, \quad \mathbf{bx} > 0$$

has a solution. Deduce this variant of Farkas' Lemma from the Duality Theorem (8.27).

8.6.11 Prove the inequality $\alpha(G) \leq \beta'(G)$ directly, without appealing to the Weak Duality Theorem (8.25).

8.7 Related Reading

ISOMORPHISM-COMPLETENESS

As mentioned earlier, the complexity status of GRAPH ISOMORPHISM is unknown. There is strong theoretical evidence to support the belief that the problem is not $\mathcal{NP}$-complete (see, for example, Babai (1995)), and its rather unique status has led to the notion of isomorphism-completeness: a problem is said to be *isomorphism-complete* if it is polynomially equivalent to GRAPH ISOMORPHISM. The Legitimate Deck Problem, mentioned in Section 2.8, is one such problem (see Harary et al. (1982) and Mansfield (1982)). The problem of finding the orbits of a graph is 'isomorphism-hard'. For these and other examples, we refer the reader to Babai (1995).