Chapter 8. Complexity of Algorithms

Note. In this chapter, we give several of the core ideas of the class Mathematical Modeling Using Graph Theory (MATH 5870). We classify problems according to their level of difficulty in terms the known algorithms to solve them. We define the classes $\mathcal{P}$, $\mathcal{NP}$, and co-$\mathcal{NP}$ (in Section 8.1). We consider polynomial-time algorithms and polynomial reductions which reduce given problems to those of known solution and the reduction is itself a polynomial time algorithm (in Section 8.2). We define the $\mathcal{NP}$-complete problems in Section 8.3 and mention the Traveling Salesman/Salesperson Problem (TSP) again. In Section 8.4 we consider approximations of inefficient algorithms with efficient ones. We consider greedy heuristics in Section 8.5, with attention paid to the Borůkvka-Kruskal Algorithm (Algorithm 8.23) for finding a minimum weight spanning tree. In Section 8.6, we briefly consider the numerical approaches of linear programming and integer programming.

Section 8.1. Computational Complexity

Note. In this section, we give several definitions needed in this chapter. We don't present any proofs, but state some problems and conjectures.

Definition. An *instance* of a problem is an application of the problem to one specific member of the family of graphs (or digraphs) to which the problem can apply. 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*.

Note. As examples of these definitions, an instance of the Minimum-Weight Spanning Tree Problem (problem 6.8 of Section 6.2) is the problem of finding an optimal tree in a particular given weighted connected graph. The Harní-Prim Algorithm (Algorithm 6.9) is an algorithm that accepts as input a weighted connected graph and returns as output an optimal (i.e., minimal weight) tree.

Note. In considering algorithms, we are interest in two things: (1) that the proposed algorithm actually works and produced the required output, and (2) the efficiency of the algorithm. We have seen, for example, that Algorithm 3.3 (Fleury's Algorithm of Section 3.3. Euler Tours) returns an Euler tour for a connected graph in Theorem 3.4, and that Algorithm 6.9 (the Janíl-Prim Algorithm of Section 6.2. Minimum-Weight Spanning Trees) returns an optimal tree in Theorem 6.10. In this chapter, we are interested in the efficiency of algorithms.

"Definition." The *computational complexity* (or simply *complexity*) of an algorithm, is the number of basic computational steps (such as arithmetical operations and comparisons) needed to complete the algorithm. This number depends on the size of the input, in our case the size of a graph G (usually related to the number of vertices and/or the number of edges, but possibly containing other information such as a weight on the edges). If the complexity is bounded above by a polynomial in the input size, the algorithm is called a *polynomial-time algorithm*. If the bounding polynomial is of degree 1, then the algorithm is *linear-time*, if the bounding polynomial is of degree 2 then the algorithm is *quadratic-time*, and so forth.

Note 8.1.A. If a polynomial time-algorithm is known for a problem, then the algorithm is "computationally feasible, even for large input graphs" (see page 174 of Bondy and Murty). In contrast, if an algorithm is not polynomial-time and instead is exponential in the size of the input, then it is not practical to run, except for "small" inputs. We denote the class of problems which are solvable by polynomial-time algorithms as $\mathcal{P}$.

Note. The tree-search algorithms discussed of Chapter 6, the Breadth-First Search (Algorithm 6.1), Depth-First Search (Algorithm 6.4), the Jarníl-Prim Algorithm (Algorithm 6.9), and Dijkstra's Algorithm (Algorithm 6.12), are polynomial-time algorithms. For example, in Breadth-First Search, each edge is examined for possible inclusion in the tree just twice: in Step 8 when one end of the edge is used to look for uncoloured neighbours, and a second time, also in Step 8, when the other end of the edge is considered for uncoloured neighbours. The Depth-First Search is similar (see its Step 10). So these two algorithms are linear in the number of edges m. However, the Max-Flow Min-Cut algorithm (Algorithm 7.9) is not in the class $\mathcal{P}$. In Exercise 8.1.1 it is to be shown, using a very elementary network, that the Max-Flow Min-Cut algorithm can perform an arbitrarily large number of iterations before returning a maximum flow.

Note. Bondy and Murty comment (see page 175): "...our analysis of these algorithms is admittedly cursory, and leaves out many pertinent details...." They give as a more formal, rigorous source Christos Papadimitriou's *Computational Com-*

plexity, Addison Wesley Publishing (1994). A reading of the contents of this book reveals that it is indeed deep! Its Part I is on algorithms and it covers Turing machines and computability. Part II is on logic and covers Boolean logic, first-order logic, and undecidability. Part III is the most relevant part to our current conversation, and covers complexity classes, reduction, $\mathcal{NP}$-complete problems, and co-$\mathcal{NP}$ problems (topics we discuss below). Parts IV and V give more details on complexity and consider the class $\mathcal{P}$ and classes beyond $\mathcal{NP}$.

Note. In addition to the class $\mathcal{P}$ of problems which are solvable by polynomial-time algorithms, there many basic problems for which it is unknown whether polynomial-time algorithms may possibly exist or not. One such problem is determining whether two graphs are isomorphic or not (we will state the as Problem 8.11 in hrefhttps://faculty.etsu.edu/gardnerr/5340/notes-Bondy-Murty-GT/Bondy-Murty-GT-6-1.pdfSection 8.3. $\mathcal{NP}$-Complete Problems). This leads to a class of problems for which it is not known whether a polynomial-time algorithm may exist or not. This class of problems is denoted $\mathcal{NP}$, which stands for *nondeterministic polynomial-time*. We give additional explanation of this below, but will with some informality. Bondy and Murty reference D. B. Shmoys and É. Tardos's "Computational Complexity," which appears as Chapter 29, pages 1599–1647, in *Handbook of Combinatorics, Volume 2*, edited by R. L. Graham, M. Grötschel, and L. Lovász, Elsevier Science B.V. (1995). This dense but rigorous work contains very formal definitions of the classes $\mathcal{P}$, $\mathcal{NP}$, and $\mathcal{NP}$-complete. We now consider the more informal approach of Bondy and Murty.

"Definition." A *decision problem* is a question whose answer is either 'yes' or 'no.' A decision problem belongs to *class* $\mathcal{P}$ if there is a polynomial-time algorithm that solves any instance of the problem.

Note. Bondy and Murty next use the undefined term "certificate." We appeal to the Wikipedia page on "certificate" for some insight on this idea. The next definition is a paraphrasing of the Wikipedia information.

"Definition." A *certificate* is a string that certifies the answer to a computation. We say "string" because it is represented as a sequence of 0's and 1's in the theoretical setting. A certificate is a "solution path" which is used to check whether a problem gives the answer 'yes' or 'no.'

"Definition." A decision problem 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 called a *succinct certificate*. A decision problem belongs to the *class co-*$\mathcal{NP}$ if, given any instance of the problem whose answer is 'no,' there is a certificate validating this fact which can be checked in polynomial time.

Note. In other words, a decision problem is in class co-$\mathcal{NP}$ if there is a succinct certificate that confirms that the decision problem has answer 'no.' The reason this class is called "co-$\mathcal{NP}$" is because it deals with the complement of a $\mathcal{NP}$ problem (that is, a problem involving a negation so that the 'yes' involved in an $\mathcal{NP}$ problem is replaced with a 'no' in a co-$\mathcal{NP}$ problem).

Note. Since the class $\mathcal{P}$ consists of decision problems for which there is a polynomial-time algorithm that solves any instance of the problem (where both decisions 'yes' and 'no' are covered), then we have $\mathcal{P} \subseteq \mathcal{NP}$ and $\mathcal{P} \subseteq$ co-$\mathcal{NP}$. That is, $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$.

Note. Let G be an arbitrary graph and consider the decision problem of if G is bipartite. To illustrate the (vague) idea of a certificate, we have that a bipartite graph with bipartition (X, Y) is a succinct certificate (in which case the decision problem yields 'yes') since it suffices to check that each edge of the graph has one end in X and one end in Y. That is, the decision problem is in the class $\mathcal{NP}$ Also, by Theorem 4.7, a non bipartite graph contains an odd cycle, so any such cycle is a succinct certificate (in which case the decision problem yields 'no'). So the decision problem is also in the class co-$\mathcal{NP}$. We have then that the decision problem is in $\mathcal{NP} \cap$ co-$\mathcal{NP}$. This alone does not imply that the decision problem is in $\mathcal{P}$, but it is to be shown in Exercise 6.1.3 that this is in fact the case (so concerns over checking instances of the problem in polynomial time are allayed).

Note. We now consider the following problem.

Problem 8.1. Hamilton Cycle.

Given: *a graph G.*

Decide: *Does G have a Hamilton cycle?*

If the decision problem has answer 'yes' then a Hamilton cycle would serve as a succinct certificate. So this problem is in the class $\mathcal{NP}$. But if the graph does not

have a Hamilton cycle, we do not know what to use as a succinct certificate! This does not mean that the problem is not in co-$\mathcal{NP}$; in fact, it is unknown whether the problem is in co$\mathcal{NP}$ or not. This is discussed more in Chapter 18. Hamilton Cycles.

Note. We have stated that $\mathcal{P} \subseteq \mathcal{NP}$. A natural question is whether these classes are in fact equal (or not). It was conjectured by Stephen Cook (December 14, 1939–present) in 1971 that these classes are different. His conjecture appears in "The complexity of theorem proving procedures." *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151-158 (1971). The conjecture was independently made by the Russian Leonid Levin (November 2, 1948–present) in "Universal Enumeration Problems" [in Russian],*Problems of Information Transmission*, **9**(3), 115-116 (1973). You can view Cook's paper on the ACM Digitial Library webpage, and Levin's paper in Russian is online on the Problems of Information Transmission webpage. Bondy and Murty, untraditionally, also include Jack R. Edmonds (April 5, 1934–present) in their name of the conjecture:

Conjecture 8.2. The Cooks-Edmonds-Levin Conjecture.

$$\mathcal{P} \neq \mathcal{NP}.$$

Note. Jack Edmonds in "Minimum Partition of a Matroid into Independent Subsets," *Journal of Research of the Natural Bureau of Standards—B. Mathematics and Mathematical Physics*, **69B**(1 and 2), 67–72 (1965) (available online at the NIST Technical Series Publications webpage) proposed the following conjecture

(notice that this predates the work of Cook and Levin):

Conjecture 8.3. Edmonds Conjecture.

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

In fact, Jack Edmonds has a rock with the conjecture engraved on it in his yard in Ontario:

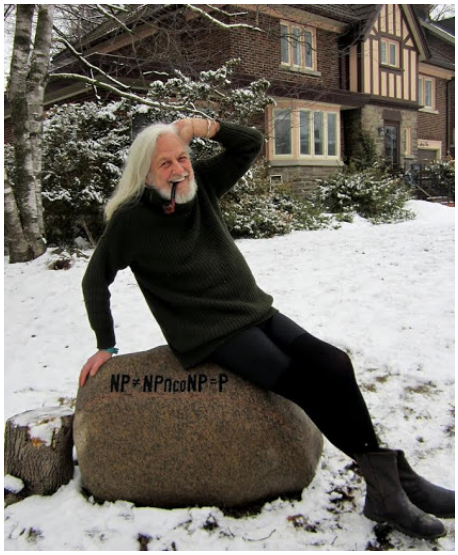

Image from Wikipedia webpage on Jack Edmonds

One of Jack Edmonds' big contributions to graph theory is his paper "Paths, Trees, and Flowers, " *Canadian Journal of Mathematics*, **17**, 449–467 (1965), available online on Javier Bernal's webpage (on the National Institute of Standards and Technology (NIST) U.S. Department of Commerce server). The webpages in the last two notes were accessed on 5/27/2022.

Note. We finish this section by mentioning two classes offered by the ETSU Department of Computing. Both are graduate-only classes and cover material mentioned in this section:

CSCI 5610. Formal Languages and Computational Complexity. (Prerequisites: MATH 2710, CSCI 2210 or consent of the instructor.) Problem-solving is a fundamental aspect of computer science. This course teaches students how to reduce a computational problem to its simplest form and analyze the problem to determine its inherent computational complexity. Topics include formal languages and automata theory, Turing machines, computational complexity, and the theory of NP-completeness. When Offered: Variable.

CSCI 5620. Analysis Of Algorithms. (Prerequisites: Differential and integral calculus, discrete structures, data structures.) This course covers basic techniques for analyzing algorithmic complexity. It describes the design and analysis of selected algorithms for solving important problems that arise often in applications of computer science, including sorting, selection, graph theory problems (e.g., shortest path, graph traversals), string matching, dynamic programming problems, NP-complete problems. When Offered: Fall, alternate years [odd falls].

Revised: 6/25/2022