CHAPTER 5

Dodgson's Rule and Young's Rule

Ioannis Caragiannis, Edith Hemaspaandra, and
Lane A. Hemaspaandra

5.1 Overview

Dodgson's and Young's election systems, dating from 1876 and 1977, are beautiful, historically resonant election systems. Surprisingly, both of these systems turn out to have highly intractable winner-determination problems: The winner problems of these systems are complete for parallel access to NP. This chapter discusses both the complexity of these winner-determination problems and approaches—through heuristic algorithms, fixed-parameter algorithms, and approximation algorithms—to circumventing that complexity.

5.2 Introduction, Election-System Definitions, and Results Overview

Charles Lutwidge Dodgson, better known under his pen name of Lewis Carroll, was a mathematics tutor at Oxford. In his 1876 pamphlet, "A Method of Taking Votes on More than Two Issues" (Dodgson, 1876), printed by the Clarendon Press, Oxford and headed "not yet published," he defined an election system that is compellingly beautiful in many ways, and yet that also turned out to be so subtle and complex, also in many ways, that it has in recent decades been much studied by computational social choice researchers.

Dodgson's election system is very simply defined. An election will consist of a finite number of voters, each voting by casting a linear order over (the same) finite set of candidates. (Recall that linear orders are inherently antisymmetric, i.e., are "tie-free.") A Condorcet winner (respectively, weak Condorcet winner) is a candidate a who, for each other candidate b, is preferred to b by strictly more than half (respectively, by at least half) of the voters. It is natural to want election systems to be Condorcet-consistent, that is, to have the property that if there is a Condorcet winner, he or she is the one and only winner under the election system. Dodgson's system is Condorcet-consistent. In fact, the system is defined based on each candidate's closeness to being a Condorcet

winner. Dodgson's view was that whichever candidate (or candidates if there is a tie for closest) was "closest" to being Condorcet winners should be the winner(s), and his system is a realization of that view.

In particular, the Dodgson score of a candidate, a, is the smallest number of sequential exchanges of adjacent candidates in preference orders such that after those exchanges a is a Condorcet winner. All candidates having the smallest Dodgson score among the candidates are the winner(s) in Dodgson's system.

For example, suppose our election is over the candidates a, b, and c, and there are two voters, one voting $a \succ b \succ c$ and the other also voting $a \succ b \succ c$. (Throughout this paper, when writing out a particular vote we use the strict linear order $\succ$ associated with the voter's linear order $\succsim$; as noted in Chapter 2, these are related by: $x \succ y \iff (x \succsim y \wedge \neg(y \succsim x))$.) The Dodgson score of c is four, since to make c a Condorcet winner we have to adjacently exchange c with b in the first voter, forming the vote $a \succ c \succ b$ and then, after that, we have to adjacently exchange c with a in the first vote, and then we need to do the same two exchanges in the second vote. The Dodgson scores of a and b are zero and two. In this example, there is one Dodgson winner, namely, a. However, if the votes had instead been $a \succ b \succ c$ and $b \succ a \succ c$, then the Dodgson scores of a, b, and c would be one, one, and four, and a and b would be the Dodgson winners.

The system just described is what Dodgson himself defined. (This chapter is designed to be self-contained. However, we mention that Chapter 2 provides to all interested readers an excellent treatment of the basics of voting theory, including such notions as Dodgson's election system, Condorcet winners, and so on.) However, some researchers have studied the following variant, sometimes still calling it Dodgson elections and not mentioning that it differs from Dodgson's notion. The election system WeakDodgson is defined exactly as above, except in terms of WeakDodgson scores, which are the number of sequential exchanges of adjacent candidates needed to make the given candidate become a weak Condorcet winner. The WeakDodgson scores of a, b, and c in the first example above are zero, one, and two, and in the second example above are zero, zero, and two. The WeakDodgson winners are the same as the Dodgson winners in the above examples. However, it is easy to construct examples where Dodgson and WeakDodgson produce different winner sets.

Dodgson's system is measuring each candidate's adjacent-exchange distance from being a Condorcet winner, and is electing the candidate(s) with the shortest such distance. Among the many beauties of Dodgson's system is that it is based on finding the minimum edit distance between the initial votes and a certain region in the space of all votes, under a certain basis of operations, in particular sequential adjacent-exchanges. The notion of edit distance is essential in a large number of fields, and is central in many area of algorithmics. Dodgson's use of this notion is a natural, lovely, and quite early example. The coverage of distance rationalizability in Chapter 8 will make clear that a distance-based framework can be used to capture and study a wide range of important voting systems.

H. Peyton Young (1977) defined his election system, now known as Young elections, in terms of a different type of distance. The Young score of a candidate, a, is defined to be the smallest number n such that there is a set of n voters such that a is a weak Condorcet winner when those n voters are removed from the election. All candidates having the

lowest Young score in a given election are its Young winner(s). The analogous system based on the number of deletions needed to make a given candidate a Condorcet winner will be called StrongYoung, and has also been studied, sometimes in papers still calling it Young elections and not mentioning that it differs from Young's notion. If a given candidate cannot be made a Condorcet winner by any number of deletions, we will say that its StrongYoung score is infinite. So, for example, in a zero-voter election, all candidates tie under the Young system, each with Young score zero, and all candidates tie under the StrongYoung system, each with StrongYoung score infinity. StrongYoung is clearly a Condorcet-consistent system.

Let us look at an election example and find its Young scores and its Young winner(s). Consider the election in which the candidates are a, b, c, and d, and the following six votes are cast:

1. $a \succ b \succ c \succ d$,
2. $a \succ b \succ c \succ d$,
3. $a \succ b \succ d \succ c$,
4. $c \succ a \succ d \succ b$,
5. $d \succ b \succ a \succ c$, and
6. $d \succ b \succ c \succ a$.

Candidate a—who actually is already a Condorcet winner—is certainly a weak Condorcet winner, and so has Young score zero. Candidate b is losing to a four to two among the six voters, and ties or beats each of c and d. Due to b's four to two loss against a, clearly the Young score of b is at least two, since deleting one vote closes the amount by which b trails a by at most one. If one deletes the votes numbered 1 and 2 above, b will tie with a two to two, but—horrors!—now loses to d one to three. So the fact that deleting 1 and 2 removes b's weakness with regard to a does not suffice to establish that the Young score of b is at most two. However, happily, it is easy to see that deleting the votes numbered 1 and 4 above indeed makes b become a weak Condorcet winner, and so b's Young score is at most two. Thus b's Young score is exactly two. It is also easy to see that d's Young score is exactly two, and the reader may wish to verify that as practice. c is a more interesting case than d is. Initially, c ties d three to three, loses to b five to one, and loses to a four to two. Due to the five to one loss to b, clearly c's Young score is at least four. However, c also trails a, and it is possible that removing some four votes that catch c up to b might not catch c up to a, or might even leave c losing to d. This observation, and the twist we ran into above related to computing b's Young score, are related to why computing Young scores turns out to be, as further mentioned later in this chapter, computationally difficult: The number of vote collections to be considered for potential deletion can be combinatorially explosive, and deleting a given vote can affect a given candidate in some helpful and some harmful ways at the same time. However, in this particular example, deleting votes 1, 2, 3, and 5 leaves c a weak Condorcet winner, and thus c's Young score is at most four. So c's Young score in fact is exactly four. Overall, candidate a is the one and only Young winner in this example—which of course follows immediately from the fact, mentioned near the start of this paragraph, that a is a Condorcet winner here.

As noted earlier, Dodgson in effect means StrongDodgson, but Young in effect means WeakYoung. This is simply due to the history of how these notions were defined by their creators. Throughout this chapter, we will use the terms Dodgson and StrongYoung for the versions based on Condorcet winners and will use the terms WeakDodgson and Young for the versions based on weak Condorcet winners.

Dodgson's and Young's election systems have followed paths of whipsaw twists and turns in terms of their computational properties, and this chapter is mostly focused on providing an overview of those paths, with a particular stress on Dodgson's system.

Briefly summarized, in the late 1800s Dodgson defined his system, and it was natural, compelling, and lovely—so much so that it was included in McLean and Urken's (1995) collection of the key papers in the multi-thousand-year history of social choice. However, as we will discuss in Section 5.3, in the late 1900s Bartholdi et al. (1989b) proved that the winner problem of this lovely system was NP-hard, and so under current standard assumptions in computer science is computationally intractable. Hemaspaandra et al. (1997a) then obtained a tight classification of the problem's computational complexity, and it became the first truly real-world-natural problem to be "complete" for the class of problems solvable through parallel access to NP—a very high level of complexity. That result was good news for complexity theory, as it populated that complexity class with a problem that clearly was highly nonartificial, since the election system had been defined in the 1800s, long before complexity theory even existed. However, the late 1900s results were grim news indeed regarding the computational difficulty of Dodgson elections.

Yet hardness results often are not the last word on a problem. Rather, they can serve as an invitation to researchers to find ways to sidestep the problem's hardness. That is exactly what happened in the case of Dodgson elections, in work done in the 2000s. It is known that, unless the polynomial hierarchy collapses, no heuristic algorithm for any NP-hard problem can have a subexponential error rate (see Hemaspaandra and Williams, 2012). So heuristic algorithms for the Dodgson election problem are limited in what they can hope to achieve. Nonetheless, it has been shown that there are quite good heuristic algorithms for the class of instances where the number of candidates is superquadratic in the number of voters. Section 5.4 presents such heuristic results. Section 5.5 discusses another approach to bypassing hardness results—parameterized algorithms. The results covered there show, for example, that the Dodgson winner problem is fixed-parameter tractable with respect to the number of candidates. That is, there is a uniform algorithm whose running time is the product of a polynomial in the instance's size and some (admittedly very large) function of the number of candidates. Finally, Section 5.6 studies a third approach to dealing with hardness, namely, approximation algorithms. That section presents results about approximating the Dodgson score and using approximation algorithms themselves as voting rules that achieve some social-choice properties that Dodgson's system lacks.

Young elections have been less extensively studied than Dodgson elections. But as this chapter will discuss, Young's system walked a twisty results road quite similar to the one Dodgson's system walked. Like Dodgson, Young is a natural election system; like Dodgson, long after Young's system was defined it was proven that even telling

who won is computationally intractable; and like Dodgson, for Young elections one can obtain fixed-parameter tractability results.

5.3 Winner-Problem Complexity

This section discusses the complexity of the winner problems of Dodgson, WeakDodgson, StrongYoung, and Young elections.

5.3.1 Basics and Background

To understand the complexity of the winner problems of Dodgson, WeakDodgson, StrongYoung, and Young elections, we will need to define and discuss an important level of the polynomial hierarchy that is far less well-known than the polynomial hierarchy's famous lowest levels, P and NP. This less well-known complexity class is the Θ_2^p level of the polynomial hierarchy, which captures the power of (polynomial-time) parallel access to NP.

Let us now define this class. We will assume that the reader is familiar with the definition of NP and has at least a general idea of what a Turing machine is. A set is in coNP exactly if its complement is in NP. We will not define the polynomial hierarchy in this chapter. However, we mention that it is widely believed that the polynomial hierarchy does not collapse. Thus any assumption that would imply that the polynomial hierarchy collapses is, in the eyes of modern computer science, viewed as highly unlikely to be true.

A Turing machine operating with parallel access to a set A is a standard Turing machine enhanced with an extra tape, called the query tape. On an arbitrary input x, the machine is allowed, after some computation, to write on the tape a sequence of binary strings (say $y_1, \ldots, y_k$), each separated by the special character #. The machine then can, at most once on each input, enter a special state, known as the query state, q_{ask_query}. After it does, the machine is by the definition of this model immediately (i.e., in one time step) placed into the state $q_{query\_answered}$, and the query tape's content is replaced with a k-bit vector containing the answers to the k questions "$y_1 \in A$?", ..., "$y_k \in A$?" After some additional computation the machine may halt and accept or halt and reject. Here, k need not be a constant; on different inputs, k might differ, and there might be no global bound on k.

For any string x, let $|x|$ denote the length of x, for example, $|01111| = 5$. A set B is said to belong to Θ_2^p exactly if there exists a Turing machine, M, and an NP set A, such that (i) there exists a polynomial p such that, for each input x, M operating with parallel access to A, on input x, halts and accepts or halts and rejects within time $p(|x|)$, and (ii) the set of all strings accepted by M operating with parallel access to A is B.

Informally put, Θ_2^p is capturing the power of what one can do with a machine that on input x can, in time polynomial in $|x|$, generate some list of queries to an NP set, and then, in light of the input and a magically delivered answer for each of those queries as to whether the queried string is in the NP set, can with at most polynomially long additional computation determine whether x is in the given set. Simply put, this

class is capturing the power of (polynomial-time) parallel access to NP. Although, as mentioned above, there is no a priori limit on the number of queries that can be asked in the (one) question string, the fact that the machine has only polynomial time to write the string ensures that there are at most polynomially many queries in the question string.

Θ_2^p is sometimes alternatively denoted $\mathrm{P}_{\|}^{\mathrm{NP}}$ or $\mathrm{P}_{tt}^{\mathrm{NP}}$; the $\|$ denotes parallel access and the tt stands for truth-table, which is the type of reduction on which the above definition of Θ_2^p is based. For those familiar with the polynomial hierarchy and its classes, the location of Θ_2^p within the polynomial hierarchy is $\Sigma_1^p \cup \Pi_1^p \subseteq \Theta_2^p \subseteq \Delta_2^p \subseteq \Sigma_2^p \cap \Pi_2^p$, or to state that without the jargon, $\mathrm{NP} \cup \mathrm{coNP} \subseteq \mathrm{P}_{\|}^{\mathrm{NP}} \subseteq \mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$. It is well-known that unless NP is a *strict* subset of Θ_2^p, the polynomial hierarchy collapses to NP.

Let us give a brief example showing membership in Θ_2^p. Consider the set of all (undirected, nonempty) graphs in which the largest clique in the graph has an odd number of nodes, that is, the problem odd-max-clique. This problem is clearly in Θ_2^p. Why? The standard clique problem is the set of all (G, ℓ), with G a graph and ℓ a natural number, such that there is a clique in G of size at least ℓ. So our machine to show that odd-max-clique is in Θ_2^p will, given a graph G having n nodes, write on the query tape the string $(G, 1)\#(G, 2)\#\cdots\#(G, n)$ and enter the state q_{ask_query}, and then from the state $q_{query\_answered}$ will look at the answer vector, which will be, for some j, j ones followed by $n - j$ zeros, and from the number of ones will easily be able to tell whether the largest clique is odd or even. For example, if the answer vector is 111000, we know the graph has cliques of size 1, 2, and 3, but not 4, 5, or 6, so the largest clique is of size 3, which is odd, so the machine in this case will enter an accepting state and halt.

Our oracle model allowed only a single question string, although that question string itself could be encoding polynomially many different simultaneous queries to the oracle. That is why this class is said to capture parallel access to NP. However, Θ_2^p is also known to exactly capture the set of languages accepted if, in our above polynomial-time model, one can query the oracle $\mathcal{O}(\log n)$ times, except now with each question string containing a single query rather than asking many queries combined. That is, informally, polynomial-time unbounded parallel access to NP has the same power as polynomial-time logarithmic-query sequential access to NP. Indeed, the class was first studied in the sequential version (Papadimitriou and Zachos, 1983), and only later was the connection to the parallel notion established (Hemachandra, 1989).

In complexity theory, reductions provide a tool to help classify complexity. We say a set B polynomial-time many-one reduces to a set D if there is a polynomial-time computable function f such that, for each x, $x \in B$ if and only if $f(x) \in D$. Informally, there is a simple to compute, membership-preserving mapping from B to D. It certainly follows that if D is easy to compute, then so is B.

For any complexity class $\mathcal{C}$, we say a set D is $\mathcal{C}$-hard if for every set $B \in \mathcal{C}$ it holds that B polynomial-time many-one reduces to D. Since $\mathrm{NP} \subseteq \Theta_2^p$, every Θ_2^p-hard problem is NP-hard. For any complexity class $\mathcal{C}$, we say a set D is $\mathcal{C}$-complete exactly if (a) $D \in \mathcal{C}$ and (b) D is $\mathcal{C}$-hard. The complete sets for a class are in some sense the quintessence of the class's power. They are members of the class, yet are so powerful that each other set in the class can be polynomial-time many-one reduced to them.

Θ_2^p itself has an interesting, unusual history. It is natural to worry, when one throws a party, whether anyone will come. In complexity theory, the analogous worry is that one will define a complexity class that seems intuitively natural, and yet the class will somehow not turn out to capture the complexity of important real-world problems.

The complexity of Dodgson elections helped Θ_2^p avoid being a dull party. In particular, by the mid-1990s, it was clear that Θ_2^p was important in complexity theory. For example, Kadin (1989) had proven that if NP has a "sparse Turing-complete set" then the polynomial hierarchy collapses to Θ_2^p; Θ_2^p was known to have a large number of different yet equivalent definitions (Wagner, 1990); and Θ_2^p was known to be closely connected to time-bounded Kolmogorov complexity theory (Hemachandra and Wechsung, 1991). Yet those were all results that would warm only the heart of a complexity theorist. Θ_2^p was known to have complete problems (see Wagner, 1987). But they were artificial or mathematical problems of a sort that might be interesting to theoretical computer scientists or logicians, yet that did not have the natural appeal of problems coming from compellingly important "real world" settings and challenges.

To this uneasily quiet party came the Dodgson winner problem, with party favors and noisemakers. The Dodgson winner problem turned out to be complete for Θ_2^p, and was unarguably natural, coming as it did from a question raised a hundred years earlier. And the party was soon humming, as many other problems, including such additional election-winner problems as StrongYoung elections and Kemeny elections, were shown to also be Θ_2^p-complete (Rothe et al., 2003; Hemaspaandra et al., 2005).

5.3.2 The Complexity of the Dodgson and Young Winner Problems

In 1989, Bartholdi et al. (1989b) proved that the Dodgson winner problem was NP-hard and left as an open issue whether it was NP-complete. In 1997, Hemaspaandra et al. (1997a) proved that the Dodgson winner problem was in fact Θ_2^p-complete. This implies that, unless NP = coNP, the problem is not NP-complete. Intuitively, the problem is too hard to be NP-complete.

It is natural to wonder why one should even bother to exactly classify a problem that is known to be NP-hard. After all, NP-hardness is already a powerful indicator of hardness. There are a number of answers to this question. The nerdy, technical answer that a complexity theorist might give is that improving a problem's complexity from NP-hardness to Θ_2^p-completeness tells us more about how unlikely the problem is to be solvable with certain other approaches to computation (see Hemaspaandra et al., 1997b, for a discussion of this). However, the truly compelling answer harks back to our earlier comment about complete sets capturing the core nature of their classes. By proving a set complete for a class, we learn much about the fundamental nature of the set—whether it is capturing, as NP-complete sets do, the power of polynomially bounded existential quantification connected to polynomial-time predicates, or whether it is capturing, as Θ_2^p-complete sets do, the power of parallel access to NP.

Formally, the Dodgson winner problem is a set, namely, the set of all triples (A, R, p)—where A is the set of candidates, R is the list of cast votes (each being a linear order over A), and $p \in A$—such that p is a winner of the given election, when conducted under Dodgson's election system. The following theorem pinpoints the complexity of the Dodgson winner problem.

Theorem 5.1 (Hemaspaandra et al., 1997a). *The Dodgson winner problem is* Θ_2^p*-complete.*

We do not have the space to give a proof of the above theorem. However, it will be important and interesting to sketch the philosophy behind and structure of the proof, as they are at first quite counterintuitive.

What is counterintuitive is that one proves the Dodgson winner problem to be Θ_2^p-hard through doing extensive work to prove that many properties of the Dodgson winner problem are computationally easy to handle. Those (three) easy properties regard trapping the potential scores of the winner to two adjacent values within the image of an NP-hardness reduction (we will refer back to this later as L1), creating in polynomial time a "double exposure" that merges two elections in a way that preserves key information from each (we will refer back to this later as L2), and providing (with some twists) a polynomial-time function that given a list of elections and a candidate of interest in each creates a single election such that the sum of the scores of each election's interesting candidate in its election is the score of a particular designated candidate in the single election. For concreteness, the last of those can be formally stated as the following "sum of the scores equals the score of the 'sum'" claim.

Lemma 5.2 (Hemaspaandra et al., 1997a). *There is a polynomial-time function, dodgsonsum, such that, for all* k *and for all* $(A_1, R_1, p_1), \ldots (A_k, R_k, p_k)$ *that are election triples (i.e.,* $p_i \in A_i$*, and the* R_i *are each a collection of linear orders over the candidates in* A_i*), each having an odd number of voters,* $dodgsonsum((A_1, R_1, p_1), \ldots, (A_k, R_k, p_k))$ *is an election triple* (A, R, p) *having an odd number of voters and it holds that the Dodgson score of* p *in the election* (A, R) *is exactly the sum over all* j *of the Dodgson score of* p_j *in election* (A_j, R_j)*.*

The natural question to ask is: Why on earth would one prove lots of things easy about Dodgson elections in order to prove that the Dodgson winner problem is extremely hard? The answer to this question is that, despite the "hardness" in its name, Θ_2^p-hardness is not just about hardness (and neither are other hardnesses, such as NP-hardness). Let us explain why, using NP-hardness for our example. Suppose for each string in $\{0, 1\}^*$ we independently flip an unbiased coin, and put the string in or out of a set A based on the outcome. With probability one, the obtained set A is so extraordinarily hard as to not even be computable. Yet under standard beliefs about NP (namely, that NP is not a subset of bounded probabilistic polynomial time), with probability one the set A we obtained is not NP-hard (see Ambos-Spies, 1986). Intuitively, the issue here is that NP-hardness is not just about hardness. To be NP-hard, a set indeed must have enough power to be usable to handle all NP sets. But that power must be so well-organized and accessible that polynomial-time many-one reductions can harness that power. Our random set A is simply chaos, and so provides no such organized power. Every time we prove something NP-hard, by a reduction, we are exploiting the organization of the set being mapped to. With NP, we usually do not think much about that. In contrast, Θ_2^p-hardness proofs are so demanding, and the amount of structure exploitation needed to establish Θ_2^p-hardness is so great, that this issue comes out from the shadows.

We need a lens to focus the structure provided by Lemma 5.2 and the two other "in polynomial time we can do many things regarding Dodgson elections" claims that we alluded to just before that result (though neither of those is stated formally in this chapter), and to use that structure to establish a Θ_2^p-hardness result for the Dodgson winner problem. For Θ_2^p-hardness proofs, the lens of choice is the following powerful technical lemma proven in the 1980s by the great German complexity theorist Klaus W. Wagner. χ_A denotes the characteristic function of A, that is, $\chi_A(y) = 1$ if $y \in A$ and $\chi_A(y) = 0$ if $y \notin A$.

Lemma 5.3 (Wagner, 1987). *Let A be any NP-complete set and let B be any set. Then B is Θ_2^p-hard if there is a polynomial-time function f such that, for all $k \geqslant 1$ and all $x_1, \ldots, x_{2k}$ satisfying $\chi_A(x_1) \geqslant \cdots \geqslant \chi_A(x_{2k})$, it holds that*

$$\|\{i \mid x_i \in A\}\| \equiv 1 \pmod 2 \iff f(x_1, \ldots, x_{2k}) \in B.$$

This can be used, for example, to show that odd-max-clique is Θ_2^p-hard (Wagner, 1987). Thus in light of our earlier example odd-max-clique in fact is Θ_2^p-complete.

Briefly put, the broad structure of the Θ_2^p-hardness proof for the Dodgson winner problem is as follows. The result we alluded to earlier as L1 basically seeks to show that the Dodgson winner problem is NP-hard through a reduction that achieves a number of additional properties. The original Bartholdi et al. (1989b) reduction showing NP-hardness for the Dodgson winner problem reduced from the exact cover by three-sets problem. However, that reduction does not have the properties needed to work in concert with Lemma 5.3. Nonetheless, L1 holds, because one can, by a reduction from a different NP-complete problem, three-dimensional matching, obtain the desired properties. Then using L1 and Lemma 5.2 together with Lemma 5.3, one can argue that the problem of telling whether candidate p_1's Dodgson score in an election (A_1, R_1) is less than or equal to candidate p_2's score in an election (A_2, R_2), with both $\|R_1\|$ and $\|R_2\|$ odd, is Θ_2^p-hard. Finally, using that result and the result we referred to earlier as L2 (a "merging" lemma), one can prove that Dodgson winner itself is Θ_2^p-hard. Thus rather extensive groundwork about the simplicity of many issues about Dodgson elections, used together with Wagner's Lemma, is what establishes Θ_2^p-hardness here.

Completeness for a class requires not just hardness for the class but also membership in the class. Yet we still have not argued that the Dodgson winner problem is in Θ_2^p. Happily, that is a very easy result to show. Given an election, a distinguished candidate p, and a natural number k, it clearly is an NP problem—called DodgsonScore in the literature—to determine whether the Dodgson score of p in that election is at most k. The way we see that this is in NP is that one can simply seek to guess a sequence of k sequential exchanges of adjacent candidates, making p a Condorcet winner. (In fact, this DodgsonScore is even NP-complete, as was established in the seminal paper of Bartholdi et al. (1989b).) Now, given an election instance, (A, R), and a candidate $p \in A$, we wish to determine within Θ_2^p whether p is a Dodgson winner. What we do is that we ask, in parallel, to the NP problem DodgsonScore every reasonable score question for every candidate. Note that even if a candidate p' is at the bottom of every vote, with $(\|A\| - 1)\|R\|$ sequential exchanges of adjacent candidates it can be moved to the top of every vote, at which point it easily is a Condorcet winner. So for each

candidate $p' \in A$, and each natural number i, $1 \leqslant i \leqslant (\|A\| - 1)\|R\|$, we ask whether the Dodgson score of p' is at most i. That is a single parallel round of $\|A\|(\|A\| - 1)\|R\|$ queries. From the answers, we immediately know the Dodgson score of each candidate, and so we can easily tell whether p is a Dodgson winner. Since this scheme meets the definition of Θ_2^p, we have established that the Dodgson winner problem is in Θ_2^p. In light of the already discussed Θ_2^p-hardness, we may conclude that the Dodgson winner problem is Θ_2^p-complete.

Is this Θ_2^p-completeness result just a trick of the particular model of Dodgson elections, or does it hold even for natural variants? Research has shown that Θ_2^p-completeness holds even for many natural variants of the Dodgson winner problem. The WeakDodgson winner problem is Θ_2^p-complete (Brandt et al., 2010a, 2010b), asking whether p is the one and only Dodgson winner (i.e., is a so-called unique winner) is Θ_2^p-complete (Hemaspaandra et al., 2009), and asking whether p is the one and only WeakDodgson winner is Θ_2^p-complete (Brandt et al., 2010b). Even comparing two Dodgson scores in the same election is Θ_2^p-complete (Hemaspaandra et al., 1997a).

Still, there are limits to how much one can vary the problem and remain hard. For example, if one considers elections in which the electorate has so-called single-peaked preferences (Black, 1948)—an extremely important notion in political science—the complexity of the winner problem for Dodgson and WeakDodgson elections falls to polynomial time (Brandt et al., 2010a).

Although we have so far been discussing the Dodgson winner problem, the key results mentioned above also hold for the Young winner problem. In 2003, the complexity of the StrongYoung winner problem was pinpointed by Rothe et al. (2003) as being Θ_2^p-complete. Θ_2^p-completeness also holds for the Young winner problem (Brandt et al., 2010a, 2010b):

Theorem 5.4. *The Young winner problem is Θ_2^p-complete.*

Θ_2^p-completeness also holds for case of asking whether p is the one and only StrongYoung winner (Hemaspaandra et al., 2009), and for the case of asking whether p is the one and only Young winner (Brandt et al., 2010b).

Similarly to the Dodgson case, if one considers elections in which the electorate has single-peaked preferences, the complexity of the winner problem for Young and StrongYoung elections falls to polynomial time (Brandt et al., 2010a).

Dodgson and Young are not the only election systems whose winner problem turns out to be hard. For example, the lovely election system known as Kemeny elections (Kemeny, 1959; Kemeny and Snell, 1960) (see Chapters 2 and 4 for more on Kemeny elections) also has a Θ_2^p-complete winner (and unique winner) problem (Hemaspaandra et al., 2005, 2009).

Although an understandable first reaction to a Θ_2^p-hardness result might be despair and resignation, it surely is better to be positive and make the best of the situation. For example, we mentioned above that for a restricted-domain setting, called single-peaked electorates, the complexity of the Dodgson winner problem vanishes. In the coming sections, we will look at three other approaches to living with intractability results: heuristic algorithms, parameterized algorithms, and approximation algorithms.

5.4 Heuristic Algorithms

Suppose we are faced with a problem for which getting an efficient deterministic algorithm that is correct on all inputs seems unlikely, for example due to the problem being NP-hard or Θ_2^p-hard. A natural next step is to seek an algorithm that is correct a very high portion of the time.

There are severe complexity-theoretic barriers to even that goal. As mentioned earlier, it is known that, unless the polynomial hierarchy collapses, no NP-hard problem (and thus no Θ_2^p-hard problem) has (deterministic) heuristic algorithms whose asymptotic error rate is subexponential (see Hemaspaandra and Williams, 2012). Still, even a heuristic algorithm whose asymptotic error rate is not subexponential can be valuable. In fact, despite the above result, heuristic algorithms are a valuable tool when faced with complex problems.

Even better than heuristic algorithms that often are correct would be heuristic algorithms that often are self-knowingly correct. A heuristic algorithm for a total function f is said to be self-knowingly correct if, on each input x, the function (*i*) outputs a claim as to the value of $f(x)$, and also outputs either "definitely" or "maybe," and (*ii*) whenever the function outputs (y, "definitely") it holds that $f(x) = y$. When the second output component is "maybe," the first output component might, or might not, equal $f(x)$. Of course, the goal is to build self-knowingly correct algorithms that very often have "definitely" as their second output component.

Since we will now often be speaking of drawing random elections, for the rest of this section we assume that in m-candidate elections the candidate names are always $1, \ldots, m$, and so in drawing a random election all that is at issue will be the votes. So in Theorem 5.5 below, the "election" will refer just to R, and both the function and the `GreedyWinner` algorithm we will discuss in the next paragraph will take R and p as their input.

Consider the function that on input (R, p)—where R is a list of votes (each a linear order) that for some j are all over the candidates $1, 2, \ldots, j$ and $p \in \{1, \ldots, j\}$—equals Yes if p is a Dodgson winner of that election and equals No otherwise, that is, the function in effect computes the characteristic function of the Dodgson winner problem (j is not fixed; it may differ on different inputs). It turns out that there is a frequently self-knowingly correct polynomial-time heuristic algorithm, called `GreedyWinner`, for the function just described, and so, in effect, for the Dodgson winner problem.

Theorem 5.5 (Homan and Hemaspaandra, 2009).

1. *For each (election, candidate) pair it holds that if* `GreedyWinner` *outputs "definitely" as its second output component, then its first output component correctly answers the question, "Is the input candidate a Dodgson winner of the input election?"*
2. *For each $m \in \{1, 2, 3, \ldots\}$ and $n \in \{1, 2, 3, \ldots\}$, the probability that an election E selected uniformly at random from all elections having m candidates and n votes (i.e., all $(m!)^n$ elections having m candidates and n votes have the same likelihood of being selected) has the property that there exists at least one candidate p' such that* `GreedyWinner` *on input (E, p') outputs "maybe" as its second output component is less than $2(m^2 - m)e^{\frac{-n}{8m^2}}$.*

What this says is that the portion of m-candidate, n-voter elections that `GreedyWinner` is self-knowingly correct on is at least $1 - 2(m^2 - m)e^{\frac{-n}{8m^2}}$. For example, if one looks at the asymptotics as m goes to infinity, and with n being some superquadratic polynomial of m, for example, $n = m^{2.00001}$, the error rate will go to zero exponentially fast.

How does the `GreedyWinner` algorithm work? It is almost alarmingly simple. In fact, the reason one can get such an explicit bound, rather than just being able to draw plots from experiments as so many papers do, is in large part due to the algorithm's simplicity. The simplicity of the algorithm makes it possible, in this case, to well-analyze its performance. That is why, in the `GreedyScore` algorithm below, we tie our hands by making at most one exchange per vote; it suffices to get the desired result and it simplifies the analysis.

`GreedyWinner` is built on top of a heuristic algorithm called `GreedyScore`, which given an election and a candidate p', seeks to, in a frequently self-knowingly correct way, compute the Dodgson score of p' in the election. What `GreedyScore` does is simply this: It goes through the votes one at a time, and in each it looks at what one candidate (if any), c, is immediately preferred (i.e., adjacently preferred) to p' in that vote, and if at that moment p' is not yet strictly beating c in terms of how many voters prefer one to the other, then `GreedyScore` exchanges p' and c in that vote. If at the end of this process, p' is a Condorcet winner, then the algorithm outputs as its first component the number of exchanges it made, and outputs as its second component "definitely." It does so because an obvious lower bound for the number of adjacent exchanges needed to make p' a Condorcet winner is the sum, over all candidates, of how many voters must change from preferring c to p' to instead preferring p' to c. But if we made p' become a Condorcet winner only by exchanging it with things that were initially upside-adjacent to it in the given vote, and we only did such exchanges if p' was at the moment of exchange still behind the candidate it was being exchanged with in their head-on-head contest, then our algorithm clearly uses no more exchanges than that lower bound. And so our algorithm has truly found the Dodgson score of p'.

Intuitively, if the number of voters is sufficiently large relative to the number of candidates, then it is highly likely that the above `GreedyScore` procedure will self-knowingly succeed, that is, that we can make up all the deficits that p' has simply by exchanging it with rivals that are immediately adjacent to it. (After all, for any two candidates c and d, it is easy to see that for $1/m$ of the possible vote choices c will adjacently beat d within that vote. So the expected value of the number of votes in which c will adjacently beat d is n/m.) The claim about frequent success can be made more precise. In particular, if one fixes a candidate, say candidate 1, and draws uniformly at random an m-candidate, n-voter election, the probability that `GreedyScore`'s second component is "maybe" is less than $2(m-1)e^{\frac{-n}{8m^2}}$.

The `GreedyWinner` algorithm, on input (R, p), is simply to first run `GreedyScore` on (R, p). If "maybe" is the second component, we ourselves output "maybe" as our second component (and the first component does not matter). Otherwise, we run the `GreedyScore` algorithm on (R, p') for each candidate $p' \neq p$. If each of those runs results in a score that is not less than what we computed for p and each has a second component "definitely," then we output (as to whether p is a Dodgson winner) Yes (in fact, in this case, our algorithm self-knowingly correctly

knows all the Dodgson scores and thus the complete set of Dodgson winners, and p is one of them), with second component "definitely," and otherwise if all second components were "definitely" we output No (in this case, our algorithm now self-knowingly correctly knows all the Dodgson scores and thus the complete set of Dodgson winners, and p is not one of them), with second component "definitely," and otherwise we output second component "maybe" (and the first component does not matter). By probability arguments (using the union theorem and a variant of Chernoff's Theorem), we can formally establish the claim made in the previous paragraph about `GreedyScore`, and Theorem 5.5's claim about the (in)frequency with which the self-knowingly correct algorithm `GreedyWinner` outputs "maybe."

This section has been speaking about Dodgson elections, and is based on the results of Homan and Hemaspaandra (2009). Independent work of McCabe-Dansted et al. (2008) studies essentially these issues for the case of WeakDodgson elections, and using the same general approach obtains related results for that case; see the discussion in Section 5.7.

5.5 The Parameterized Lens

Another approach that aims to cope with the inherent computational difficulty of Dodgson's and Young's election systems is the design of fixed-parameter tractable algorithms. The algorithmic challenge is the following: Is there an algorithm that computes the Dodgson (or StrongYoung) score, whose running time is polynomial for each fixed value of some important parameter of the problem, such as the number of candidates? The question falls within the research agenda of the area known as parameterized computational complexity (Downey and Fellows, 1999; Niedermeier, 2006). In general, that area's goal is to identify whether the computational explosion occurring in algorithms for NP-hard problems can be attributed solely to a certain parameter of the problem. In applications where that parameter typically takes on only small values, an algorithm with a running time that depends superpolynomially on only that parameter might be hoped to be of practical use.

In our case, attractive parameters include the number, m, of candidates; the number, n, of votes; and the number, k, of editing operations. For the Dodgson score, k denotes the number of sequential exchanges of adjacent candidates in the votes, while for the StrongYoung score, k denotes the number of votes deleted from the electorate. As a simple, initial example, fixed-parameter tractability with respect to the parameter n is clear in StrongYoung elections. Namely, one can conduct an exhaustive search over the 2^n different subsets of votes of the original profile and find (if one exists) a subset of maximum size in which the desired candidate is the Condorcet winner. The number of votes in this subset is the StrongYoung score of the preferred candidate p; if no such subset exists, we will output ∞ as p's StrongYoung score. So it is clear that the StrongYoung score is fixed-parameter tractable with respect to the number of voters.

The Dodgson score and the StrongYoung score are fixed-parameter tractable for the parameter m. This follows by a seminal result of Lenstra, Jr. (1983) that implies that a problem is fixed-parameter tractable when it can be solved by an integer linear program (ILP) in which the number of variables is upper-bounded by a function solely

depending on the parameter. In particular, the seminal work of Bartholdi et al. (1989b) handles Dodgson score by integer linear programming in a way that, as has often been noted, tacitly establishes that the Dodgson score is fixed-parameter tractable with respect to the number of candidates. ILPs can also be used to compute the Young and the StrongYoung scores (see Young, 1977).

Furthermore, the Dodgson score has been proved to be fixed-parameter tractable for the parameter k using dynamic programming, a standard tool for designing fixed-parameter tractable algorithms. The key idea is to solve the problem by solving subproblems and combining overlapping solutions in order to compute the overall solution. Dynamic programming avoids multiple computation of the same (sub)solution by storing it in a so-called dynamic-programming table and by accessing its value from the table when needed.

We will now present the main ideas behind the way Betzler et al. (2010) have, using dynamic programming, upper-bounded the parameterized complexity of checking whether a candidate's Dodgson score is at most a given value. Let us be given a profile R with n votes (each specified as a linear order). We will now explore how to efficiently compute the Dodgson score of a particular candidate in that profile, say candidate a. We will denote by deficit(a, y, R) the deficit of candidate a with respect to candidate y in profile R, that is, the (minimum) number of voters who have to change their preference so that a beats y in their pairwise election. For example, if there are ten voters and eight initially prefer y to a, so a loses to y eight to two, deficit$(a, y, R) = 4$ since with the right four changed votes a will squeak past y to win by six to four. P will denote the set of candidates with respect to whom a has a positive deficit under our profile R.

The idea is to build a table whose entries store information about how candidate a can be pushed upward in the votes so that the deficit with respect to each candidate of P is eventually decreased to 0. This requires storing intermediate information concerning subsets of votes and partial decreases of the deficit in the table entries. The table for this has $n + 1$ rows. Row i will contain information about the first i votes of the profile. Each column of the table will be labeled by a vector d, and that vector will have an entry for each candidate of P, with $d(y)$ being an integer between 0 and deficit(a, y, R). Entry $T(i, d)$ of the table stores the minimum number of total upward pushes of a in the first i votes of R that will suffice to decrease a's initial deficit with respect to each $y \in P$ by at least $d(y)$. (By a "push," we mean a single exchange of adjacent candidates in a preference order.) We place ∞ in the table's $T(i, d)$ entry if even pushing a to the top of the first i votes is not enough to achieve the improvements demanded by the vector d. Using $\widetilde{d}$ to denote the vector with $\widetilde{d}(y) = \text{deficit}(a, y, R)$ for each candidate y of P, it is clear that the entry $T(n, \widetilde{d})$ will contain the Dodgson score.

The entries of the table are initialized to be $T(0, d) = 0$ if $d = (0, 0, \ldots, 0)$ and $T(0, d) = \infty$ otherwise. The entries of the ith row (doing this first for the $i = 1$ row, then the $i = 2$ row, and so on) can then be computed from the information stored in the entries of the $(i - 1)$st row. Before presenting the formal definition of this computation, let us give a small example. Let us focus on the first i votes of a profile R, for which we want to compute the Dodgson score of candidate a. Furthermore, let us suppose that the ith vote is $d \succ b \succ c \succ a$. Let us as our example seek to complete the least costly way to promote a (i.e., the minimum number of exchanges) in the first i votes in such a way (if any exists using pushes among just those votes) as to decrease the deficit of a

Table 5.1. *Profile and table example for computing the Dodgson score*

(a) A profile.

1	2	3
d	d	c
b	a	d
c	b	a
a	c	b

(b) The dynamic-programming table, T, for computing the Dodgson score of candidate a.

	(0, 0)	(0, 1)	(0, 2)	(1, 0)	(1, 1)	(1, 2)
0	0	∞	∞	∞	∞	∞
1	0	3	∞	1	3	∞
2	0	1	4	1	2	4
3	0	1	2	1	2	3

Note: The table at the right is used to compute the Dodgson score of candidate a in the profile at the left. In both the profile here and in Table 5.2, our tabular vote displays are arranged "top down," for example, the leftmost column of this profile indicates that the vote of the first voter is $d \succ b \succ c \succ a$. The "1 2 3" on the top row of profiles, both here and in Table 5.2, indicates the voters, for example, the column headed by a "3" is about voter 3. In the profile given in part (a), the deficits of a with respect to the candidates b, c, and d are 0, 1, and 2. So $P = \{c, d\}$ and each column label refers to a's deficits against c and d. The Dodgson score is the value, 3, that is computed for the entry $T(3, (1, 2))$, and it is achieved by pushing a one position upward in the second vote and two positions upward in the third vote.

with respect to candidates c and d by one and two, respectively. This can be computed by considering several different alternatives. One possibility is to use the least costly way to decrease the deficit of a with respect to d in the first $i - 1$ votes by one and then push a three positions upward in the ith vote to cut by an additional one the deficits with respect to each of c and d. Another possibility is to use the least costly way to decrease the deficit of a with respect to d by two in the first $i - 1$ votes and push a one position upward in the ith vote, to shrink by one its deficit with respect to c. A third possibility is to just use the least costly way to decrease the deficits by 1 and 2 in the first $i - 1$ votes and leave the ith vote unaltered. The entry of the table corresponding to the ith row and the column corresponding to the deficit decrease vector (1, 2) will store the best among all the possibilities, including those mentioned above. This example shows how an entry in row i can be relatively easily computed if we already have in hand all the entries of row $i - 1$.

We are now ready to formally present the computation of entry $T(i, d)$ based on the entries in row $i - 1$. We use $L_i^j(d)$ for the set of all vectors of decreases of deficits such that if those decreases are satisfied over the first $i - 1$ votes of R then that will ensure that the decreases specified in d are satisfied over the first i voters of R when candidate a is pushed j positions upward in the ith vote. We use h^i to denote the number of candidates that voter i prefers to a. Then $T(i, d)$ will be assigned the value stated by the right-hand side below:

$$T(i, d) = \min_{0 \leqslant j \leqslant h^i} \min_{d' \in L_i^j(d)} \{T(i - 1, d') + j\}.$$

A completed table for an example with three votes and four candidates is provided as Table 5.1.

Using the approach sketched above and additional technical arguments, Betzler et al. (2010) prove that testing whether the Dodgson score of a given candidate is at most k is fixed-parameter tractable with respect to the parameter k.

It is important to mention that negative statements are also known. For example, the Dodgson score problem is not fixed-parameter tractable with respect to parameter n (the number of votes) unless a complexity-theoretic statement known as the

exponential-time hypothesis is false. This follows from the fact that the problem is W[1]-hard (Fellows et al., 2010); W[1]-hardness is a central hardness notion in parameterized complexity. Young elections are also intractable with respect to the score parameter, k. In particular, Betzler et al. (2010) prove that the StrongYoung score problem is complete for the parameterized complexity class W[2].

5.6 Approximation Algorithms

We now focus specifically on Dodgson elections. Since Dodgson scores are hard to compute exactly in general, an alternative approach is to view the Dodgson score computation as a combinatorial optimization problem and exploit the rich and beautiful theory of approximation algorithms (e.g., see Vazirani, 2001) in order to approximate the Dodgson score. Briefly, the challenge is to obtain efficient (i.e., polynomial-time) algorithms that return scores that are provably close to the Dodgson score. Furthermore, such an approximation algorithm can be used as an alternative voting rule to Dodgson's rule under some circumstances. We discuss these issues below.

We consider algorithms that receive as input a candidate p from an m-candidate set A and an n-voter election profile R over A, and return a score for p. We denote the score returned by an algorithm Y when applied on such an input by $\mathrm{sc}_Y(p, R)$. Also, $\mathrm{sc}_D(p, R)$ will denote the Dodgson score. An algorithm Y is said to be a Dodgson approximation if $\mathrm{sc}_Y(x, R) \geqslant \mathrm{sc}_D(x, R)$ for every candidate $x \in A$ and every profile R. Also, Y is said to have an approximation ratio of $\rho \geqslant 1$ if $\mathrm{sc}_Y(x, R) \leqslant \rho \cdot \mathrm{sc}_D(x, R)$, for every candidate x and every profile R over A.

Let us give a trivial example. Again, denote by $\mathrm{deficit}(x, y, R)$ the deficit of candidate x with respect to candidate y in profile R, that is, the minimum number of voters who have to change their preference so that x beats y in their pairwise election. Consider the algorithm Y that, given a candidate x and a preference profile R, returns a score of $\mathrm{sc}_Y(x, R) = (m-1) \cdot \sum_{y \in A-\{x\}} \mathrm{deficit}(x, y, R)$. It is easy to show that this algorithm is a Dodgson approximation and, furthermore, has approximation ratio at most $m-1$. In particular, it is possible to make x beat y in a pairwise election by pushing x to the top of the preferences of $\mathrm{deficit}(x, y, R)$ voters, and clearly this requires at most $(m-1) \cdot \mathrm{deficit}(x, y, R)$ sequential exchanges of adjacent candidates. By summing over all $y \in A - \{x\}$, we obtain an upper bound of $\mathrm{sc}_Y(x, R)$ on the Dodgson score of x. On the other hand, given $x \in A$, for every $y \in A - \{x\}$ we require $\mathrm{deficit}(x, y, R)$ sequential adjacent-exchanges that push x above y in the preferences of some voter in order for x to beat y in a pairwise election. Moreover, these sequential adjacent-exchanges do not decrease the deficit with respect to any other candidate. Therefore, $\sum_{y \in A-\{x\}} \mathrm{deficit}(x, y, R) \leqslant \mathrm{sc}_D(x, R)$, and by multiplying by $m-1$ we get that $\mathrm{sc}_Y(x, R) \leqslant (m-1) \cdot \mathrm{sc}_D(x, R)$.

5.6.1 Achieving Logarithmic Approximation Ratios

In this section we present two Dodgson approximations with approximation ratios logarithmic in the number of candidates. One is a combinatorial, greedy algorithm and the other is an algorithm based on linear programming.

Table 5.2. *An example of the execution of Section 5.6.1's greedy algorithm (to compute the score of candidate p) on an election with 3 votes and 11 candidates*

(a) Initial profile.			(b) After step 1.			(c) After step 2.			(d) After step 3.		
1	2	3	1	2	3	1	2	3	1	2	3
b	b	c	b	b	c	b	p	c	b	p	c
d_1	d_4	b	d_1	d_4	b	d_1	b	b	d_1	b	p
d_2	d_5	d_6	d_2	d_5	d_6	d_2	d_4	d_6	d_2	d_4	b
c	p	d_7	p	p	d_7	p	d_5	d_7	p	d_5	d_6
d_3	c	d_8	c	c	d_8	c	c	d_8	c	c	d_7
p	d_1	p	d_3	d_1	p	d_3	d_1	p	d_3	d_1	d_8
d_4	d_2	d_1	d_4	d_2	d_1	d_4	d_2	d_1	d_4	d_2	d_1
d_5	d_3	d_2	d_5	d_3	d_2	d_5	d_3	d_2	d_5	d_3	d_2
d_6	d_6	d_3	d_6	d_6	d_3	d_6	d_6	d_3	d_6	d_6	d_3
d_7	d_7	d_4	d_7	d_7	d_4	d_7	d_7	d_4	d_7	d_7	d_4
d_8	d_8	d_5	d_8	d_8	d_5	d_8	d_8	d_5	d_8	d_8	d_5

We present the greedy algorithm first. This is a far more numerically driven greedy algorithm than the ones mentioned in Section 5.4. Given a profile R and a special candidate $p \in A$, those candidates $a \in A - \{p\}$ with deficit$(p, a, R) > 0$ are said to be alive. Candidates that are not alive, that is, those with deficit$(p, a, R) = 0$, are said to be dead. In each step, the algorithm selects an optimally cost-effective push (i.e., a least cost-ineffective push) of candidate p in the preference of some voter. The cost-ineffectiveness of pushing p in the preference of a voter i is defined as the ratio between the total number of positions p is moved upward in the preference of i compared with the original profile R, and the number of currently live candidates relative to which p gains as a result of this push. Note that the optimally cost-effective push (i.e., the push with the lowest cost-ineffectiveness) at each step may not be unique; in this case, tie-breaking has to be used in order to select one of the optimally cost-effective pushes.

After selecting an optimally cost-effective push, the algorithm decreases the deficit of p by one for each live candidate a relative to which p gains by that push. Candidates with respect to whom p has zero deficit become dead. The algorithm terminates when no live candidates remain; its output is the total number of positions that candidate p is pushed upward in the preferences of all voters.

An example of the execution of the algorithm is depicted in Table 5.2. In the initial profile R of this example, candidate p has deficits deficit$(p, b, R) = 2$, deficit$(p, c, R) = 1$, and deficit$(p, d_i, R) = 0$ for $1 \leqslant i \leqslant 8$. So candidates b and c are alive and candidates $d_1, \ldots, d_8$ are dead. At the first step of the algorithm, there are several different ways of pushing candidate p upward in order to gain relative to one or both of the live candidates b and c. Among them, the one with the smallest cost-ineffectiveness is to push p upward in the first vote. In this way, p moves two positions upward and gains relative to the live candidate c for a cost-ineffectiveness of 2. Any other push of p in the initial profile has cost-ineffectiveness at least 2.5 since p has to be pushed at least three positions upward in order to gain relative to one live candidate and at least five positions upward in order to gain relative to both b and c. After step 1, candidate c is dead. Then, in step 2, there are three ways to push candidate p upward so that it gains relative to the live candidate b: either pushing it to the top of

the first vote (this has cost-ineffectiveness 5 because p would have moved five positions in total compared to the initial first vote), or pushing it to the top of the second vote (with cost-ineffectiveness 3), or pushing it four positions upward in the third vote (with cost-ineffectiveness 4). The algorithm picks the second option. Then, in step 3, the algorithm can either push candidate p to the top of the first vote or push it four positions upward in the third vote. The former has a cost-ineffectiveness of 5 (recall that cost-ineffectiveness is defined using the total number of positions p would move compared to its position at the initial profile), while the latter has a cost-ineffectiveness of 4 and is the push the algorithm picks. After step 3, all candidates are dead and the algorithm terminates by returning the total number of positions p is pushed upward, that is, 9.

Since the algorithm terminates when all candidates in $A - \{p\}$ are dead, it is clear that p becomes a Condorcet winner. The analysis of this greedy algorithm uses a linear programming relaxation of the Dodgson score. Given the profile R with a set of voters N and a set of m candidates A, denote by r^i the rank of candidate p in the preference of voter i. For every voter $i \in N$, denote by $\mathcal{S}^i$ the subcollection that consists of the sets S^i_k for $k = 1, \ldots, r^i - 1$, where the set S^i_k contains the live candidates that appear in positions $r^i - k$ to $r^i - 1$ in the preference of voter i. We denote by $\mathcal{S}$ the (multiset) union of the subcollections $\mathcal{S}^i$ for $i \in N$. The problem of computing the Dodgson score of candidate p on profile R is equivalent to selecting sets from $\mathcal{S}$ of minimum total size so that at most one set is selected among the ones in $\mathcal{S}^i$ for each voter i and each candidate $a \in A - \{p\}$ appears in at least deficit(p, a, R) selected sets. This can be expressed by an integer linear program using a binary variable $x(S)$ to denote whether the set $S \in \mathcal{S}$ has been selected. We present the relaxation of this LP below, where the integrality constraint for the variables has been relaxed to fractional values between 0 and 1:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{i \in N} \sum_{k=1}^{r^i - 1} k \cdot x(S^i_k) \\
\text{subject to} \quad & \forall a \in A - \{p\}, \sum_{i \in N} \sum_{S \in \mathcal{S}^i : a \in S} x(S) \geqslant \text{deficit}(p, a, R) \\
& \forall i \in N, \sum_{S \in \mathcal{S}^i} x(S) \leqslant 1 \\
& \forall S \in \mathcal{S}, \ 0 \leqslant x(S) \leqslant 1.
\end{aligned}$$

Clearly, the Dodgson score of candidate p is an upper bound on the optimal objective value of this LP.

The analysis uses a technique that is known as dual fitting and is similar to the analysis of a greedy algorithm for the related constrained set multicover problem; see Rajagopalan and Vazirani (1999) and Vazirani (2001, pp. 112–116). The idea is to use the decisions taken by the algorithm and construct a feasible solution for the dual (maximization) LP that has value at most the score returned by the algorithm divided by H_{m-1}, where $H_k = 1 + \frac{1}{2} + \cdots + \frac{1}{k}$ denotes the kth harmonic number. By a simple duality argument, this implies that the score returned by the algorithm is at most H_{m-1} times the optimal objective value of the above LP and, consequently, at most H_{m-1} times the Dodgson score of p.

This suggests a different algorithm, in particular, an LP-based algorithm for approximating the Dodgson score of a candidate p without explicitly providing a way of pushing p upward in the preferences of some voters in a way making p become the Condorcet winner. This algorithm just uses the LP relaxation above, computes its optimal objective value, and returns this value multiplied by H_{m-1} as a score for candidate p. Then the approximation ratio of H_{m-1} is obvious. The algorithm is also a Dodgson approximation, since the score returned by this section's greedy algorithm (which is an upper bound for the Dodgson score of p) is not higher than the score returned by the LP-based algorithm. The following statement summarizes our discussion.

Theorem 5.6 (Caragiannis et al., 2012b). *This section's greedy algorithm and LP-based algorithm are Dodgson approximations, each with approximation ratio* H_{m-1}.

5.6.2 Approximation Algorithms as Alternative Voting Rules?

A Dodgson approximation naturally induces a voting rule by electing the candidate(s) with minimum score. Arguably, such a voting rule maintains some echo of the basic philosophy behind Dodgson's election system—more strongly so if it is a very good approximation. But can it really be used as a voting rule? Trying to support a yes answer to this question requires us to discuss an issue that we have not yet touched on. One can argue that for a voting rule to be attractive, it should not only be easy to compute, but also, ideally, should have certain properties that are considered desirable from a social-choice point of view. Several such properties are not satisfied by Dodgson's rule, and this is the main reason why the rule has been criticized in the social-choice literature, see, for example, Brandt (2009a) and the references therein.

We will see that Dodgson approximations, in return for their core disadvantage of merely being an approximation to Dodgson's rule, can satisfy desirable social-choice properties, even while also providing polynomial-time algorithms. Before going on to the three social-choice properties we will discuss, it is important to make clear just how greatly Dodgson approximations can distort Dodgson's rule, especially since we commented above that Dodgson approximations in some way echo the flavor and philosophy of Dodgson. The best Dodgson approximation we consider in this section has an approximation ratio of 2. Consider a three-candidate election for which the actual Dodgson scores of the candidates are 10, 11, and 12. A Dodgson approximation having ratio 2 could give for these candidates, respectively, scores of 18, 16, and 14. That is, the ordering of even an excellent Dodgson approximation can be a complete inversion of the actual Dodgson ordering, and the worst Dodgson loser can be named the unique winner. Clearly, the fact that even a 1.000001 approximation-ratio algorithm can completely invert the entire ranking of the candidates is a troubling (but not far from unavoidable—see the discussion at the end of Section 5.6.3) feature of using approximations as voting rules.

In the following, when we say that a Dodgson approximation satisfies a social-choice property we are referring to the voting rule induced by the algorithm. As a warm up, observe that the voting rule induced by any Dodgson approximation (regardless of its approximation ratio) is Condorcet-consistent, basically because anything times zero is zero. So every Dodgson approximation, regardless of how bad its ratio is, must

assign score of 0 to any Condorcet winner. But since Dodgson approximations never underestimate scores, any candidate who is not a Condorcet winner will be assigned a score of at least 1. So any Condorcet winner will be the one and only winner under any Dodgson approximation. (Thank goodness Dodgson did not add a one in the definition of his scores. That would destroy the above claim, which is deeply dependent on the special nature of zero.) Of course, Dodgson's system itself also is Condorcet-consistent.

We will now move on to discuss two other socially desirable properties: monotonicity and homogeneity. We will see that these properties can be achieved by good Dodgson approximations that run in polynomial time.

A voting rule is said to be monotonic if a winning candidate always remains winning after it is pushed upward in the preferences of some of the voters. Dodgson's rule is known to be monotonic when there are at most three candidates and to be nonmonotonic for each number of candidates greater than or equal to four (Fishburn, 1982, p. 132). The intuition for the latter is that if a voter ranks x directly above y and y above z, exchanging x and y may not help y if it already beats x, but may help z defeat x. The two approximation algorithms presented in Section 5.6.1 are also nonmonotonic.

In contrast, the Dodgson approximation that returns $(m-1)\cdot\sum_{y\in A-\{x\}}$ deficit(x, y, R) as the score of candidate x is monotonic as a voting rule. Indeed, consider a preference profile R and a winning candidate x. Pushing x upward in the preferences of some of the voters can neither increase its score (since its deficit with respect to each other candidate does not increase) nor decrease the score of any other candidate $y \in A - \{x\}$ (since the deficit of y with respect to each candidate in $A - \{x, y\}$ remains unchanged and its deficit with respect to x does not decrease). So we already have a monotonic Dodgson approximation with approximation ratio $m - 1$. In the following we present much stronger results.

A natural "monotonization" of Dodgson's voting rule yields a monotonic Dodgson approximation with approximation ratio of 2. The main idea is to define the winning set of candidates for a given profile first and then assign the same score to the candidates in the winning set and a higher score to the nonwinning candidates. Roughly speaking, the winning set is defined so that it contains the Dodgson winners for the given profile as well as the Dodgson winners of other profiles that are necessary so that monotonicity is satisfied. More formally put, we say that an n-vote election profile R' is a y-improvement of profile R for some candidate $y \in A$ if R' is obtained by starting from R and pushing y upward in the preferences of zero or more voters.

Monotonization proceeds as follows. Let M denote the new voting rule we are constructing. Denote by $W(R)$ the set of winners of M (or the winning set) for profile R; we will soon specify which candidates belong to $W(R)$. Let $\Delta = \max_{y\in W(R)} \mathrm{sc}_D(y, R)$. The voting rule M assigns a score of $\mathrm{sc}_M(y, R) = \Delta$ to each candidate $y \in W(R)$ and a score of

$$\mathrm{sc}_M(y, R) = \max\{\Delta + 1, \mathrm{sc}_D(y, R)\}$$

to each candidate $y \notin W(R)$. All that remains is to define the winning set $W(R)$. This is done as follows: For each profile R^* and each Dodgson winner y^* of R^*, include y^* in the winning set $W(R')$ of each profile R' that is a y^*-improvement of R^*.

Theorem 5.7 (Caragiannis et al., 2014b). *M is a monotonic Dodgson approximation with an approximation ratio of 2.*

That M is monotonic and is a Dodgson approximation follow immediately from the definitions of the winning set $W(R)$ and the scores returned by M. The proof of the approximation ratio bound is based on the following technical property: Pushing a candidate y upward does not increase his or her Dodgson score and does not decrease the Dodgson score of any other candidate by a factor larger than 2. The upper bound provided by Theorem 5.7 is the best possible: No monotonic Dodgson approximation can have an approximation ratio smaller than 2. This negative statement does not use any complexity-theoretic assumptions and actually holds for exponential-time algorithms as well. Actually, monotonization (in the rather naive approach described above) yields an exponential-time algorithm.

So from the computational point of view, the above algorithm is not at all satisfactory. Fortunately, a polynomial-time implementation of monotonization is possible, although it involves an unavoidable (see Section 5.6.3) logarithmic loss in the approximation ratio. There are two main obstacles that one has to overcome in order to implement monotonization in polynomial time. First, as discussed in Section 5.3, computing the Dodgson score and deciding whether a given candidate is a Dodgson winner are computationally hard problems. This obstacle can be overcome using the score returned by the polynomial-time LP-based Dodgson approximation that we presented in Section 5.6.1 instead of using the Dodgson score itself. Even in this case, given a profile R, we still need to be able to detect when a candidate y is the winner according to the LP-based voting rule in some profile R' of which the current one is a y-improvement; if this is the case, y has to be included in the winning set $W(R)$ of profile R. This means that exponentially many profiles may have to be checked in order to determine the winning set of the current profile. This obstacle is overcome by Caragiannis et al. (2014b) using the notion of pessimistic estimators. These are quantities defined in terms of the current profile only and are used to identify the winning set in polynomial time. The next statement follows using these two high-level ideas and additional technical arguments.

Theorem 5.8 (Caragiannis et al., 2014b). *There exists a monotonic polynomial-time Dodgson approximation with an approximation ratio of $2H_{m-1}$.*

Let us now turn to homogeneity. A voting rule is said to be homogeneous if, for every integer $k \geqslant 2$, its outcome does not change when replacing each vote in the preference profile with k identical copies of the vote. Fishburn (1977) observed that Dodgson's rule is not homogeneous. The intuition behind this is that if candidates x and y are tied in a pairwise election the deficit of x with respect to y does not increase by duplicating the profile, but if x strictly loses to y in a pairwise election then the deficit scales with the number of copies.

Tideman (2006, pp. 199–201) presents the following simplified version of Dodgson's rule and proves that it is both homogeneous and monotonic. A Condorcet winner—if one exists—in an election profile R is the sole winner according to Tideman's rule.

Otherwise, the rule assigns a score of

$$\mathrm{sc}_{\mathrm{Td}}(x, R) = \sum_{y \in A - \{x\}} \max\{0, n - (2 \cdot \|\{i \in N : x \succ_i y\}\|)\}$$

to each candidate x, and the candidate(s) with the minimum score win. In the above equation, the notation $x \succ_i y$ indicates that voter i prefers candidate x to candidate y. Unfortunately, this score definition does not provide a Dodgson approximation. For example, a candidate who is tied with some candidates and beats the rest has a score of 0, yet 0 is lower than its Dodgson score. However, we in fact can give a different scoring framework, Td′, that is a Dodgson approximation and that will elect exactly the same winners as does Tideman's simplified variant of Dodgson's rule (and thus will be monotonic and homogeneous). Td′ is defined as follows. If a candidate x is a Condorcet winner, then it has score $\mathrm{sc}_{\mathrm{Td}'}(x, R) = 0$. Otherwise, Td′ "scales" the score of x as follows:

$$\mathrm{sc}_{\mathrm{Td}'}(x, R) = m \cdot \mathrm{sc}_{\mathrm{Td}}(x, R) + m(1 + \log m).$$

Clearly, $\mathrm{sc}_{\mathrm{Td}'}(x, R)$ can be computed in time polynomial in n and m.

Theorem 5.9 (Caragiannis et al., 2014b). *Td′ is a monotonic, homogeneous, polynomial-time Dodgson approximation with an approximation ratio of $\mathcal{O}(m \log m)$.*

This approximation ratio is the best possible; a matching $\Omega(m \log m)$ lower bound holds for any algorithm that is homogeneous (Caragiannis et al., 2014b).

5.6.3 Hardness of Approximation

The best polynomial-time Dodgson approximations presented in Section 5.6.1 achieve—keeping in mind that $H_m = \ln n + \Theta(1)$—asymptotic approximation ratios of $\mathcal{O}(\log m)$. Under standard assumptions about NP, all polynomial-time Dodgson approximations have approximation ratios that are $\Omega(\log m)$, so the above-mentioned approximations from the previous section have ratios that are optimal within a constant, and in fact that constant can be kept down to 2. This claim is implicit in McCabe-Dansted (2006). Later, Caragiannis et al. (2012b) explicitly obtained and stated the following result, using a reduction from minimum set cover and well-known inapproximability thresholds of Feige (1998) and Raz and Safra (1997).

Theorem 5.10 (Caragiannis et al., 2012b). *There exists a constant $\beta > 0$ such that it is NP-hard to approximate the Dodgson score of a given candidate in an election with m candidates to within a factor of $\beta \ln m$. Furthermore, for any $\epsilon > 0$, there is no polynomial-time $\left(\frac{1}{2} - \epsilon\right) \ln m$-approximation for the Dodgson score of a given candidate unless all problems in NP have algorithms running in time $k^{\mathcal{O}(\log \log k)}$, where k is the input size.*

One might wonder why our particular notion of approximation has been used. For example, a natural alternative approach would be to approximate some notion of Dodgson ranking. Unfortunately, the following statement shows that this is an impossible goal: Efficient approximation algorithms for Dodgson ranking are unlikely to exist. For the purpose of the theorem below, a Dodgson ranking of an election

instance is an ordering of the candidates such that if $i < j$ then the ith candidate in the ordering has Dodgson score no greater than the jth candidate in the ordering.

Theorem 5.11 (Caragiannis et al., 2012b). *Given a profile with m candidates and a special candidate p, it is NP-hard to decide whether p is a Dodgson winner or has rank at least $m - 6\sqrt{m}$ in any Dodgson ranking.*

5.7 Bibliography and Further Reading

Dodgson's election system first appeared in Dodgson's 1876 pamphlet (Dodgson, 1876). The computational complexity of the winner problem for Dodgson's system was shown NP-hard in the seminal paper of Bartholdi et al. (1989b), and was shown Θ_2^p-complete by Hemaspaandra et al. (1997a), see also Brandt et al. (2010b, p. 54). Young's election system was defined by him in 1977 (Young, 1977), and the complexity of StrongYoung was pinpointed as being Θ_2^p-complete by Rothe et al. (2003). See Brandt (2009a) and the references therein for perspectives on why Dodgson proposed his system and discussions of Dodgson's system in terms of not satisfying certain properties.

A number of other papers discuss the complexity of Dodgson and Young elections or variants of those elections (Hemaspaandra et al., 2009; Brandt et al., 2010a, 2010b, 2015b). Readers interested in the complexity of these election systems may be interested in the work showing that Kemeny's election system (Kemeny, 1959; Kemeny and Snell, 1960)—see also Chapter 4—has a Θ_2^p-complete winner problem (Hemaspaandra et al., 2005) and a Θ_2^p-complete unique winner problem (Hemaspaandra et al., 2009). Complexity has also been broadly used as a tool with which to block attacks on elections, such as manipulation (Bartholdi et al., 1989a), bribery (Faliszewski et al., 2009b), and control (Bartholdi et al., 1992); see Chapters 6 and 7, and see also the surveys by Faliszewski et al. (2009d, 2010).

Θ_2^p, in its "logarithmic number of sequential queries to NP" definition, was first studied in the early 1980s, by Papadimitriou and Zachos (1983). Hemachandra (1989) showed that that definition yields the same class of sets as the unbounded-parallel definition. Θ_2^p-completeness can also apply to a range of problems quite different from the election problems discussed in this chapter. For example, determining when greedy algorithms well-approximate maximum independent sets is known to be Θ_2^p-complete (Hemaspaandra and Rothe, 1998). The most important tool for proving Θ_2^p-completeness is Lemma 5.3, due to Wagner (1987). Readers more generally interested in complexity will find an excellent, accessible introduction in the textbook of Bovet and Crescenzi (1993), and a more advanced and technique-based tour is provided by Hemaspaandra and Ogihara (2002).

The material presented in our heuristics section (Section 5.4) is based on the work of Homan and Hemaspaandra (2009) about using greedy heuristics for Dodgson elections. The independent work of McCabe-Dansted et al. (2008) studies the use of greedy heuristics for WeakDodgson elections. The two papers are based on the same central insight and obtain related results. However, there are some nontrivial differences between the two papers and their claims; these differences are discussed in detail in Section 1 of Homan and Hemaspaandra (2009).

Readers interested in the theory of parameterized computational complexity can find a systematic treatment in textbooks such as the ones by Downey and Fellows (1999) and Niedermeier (2006). Betzler et al. (2012) survey the progress in that field in relation to voting and cover both winner determination and other problems, for several voting rules.

The first approximation algorithms for voting rules (e.g., Kemeny) are implicit in the papers of Ailon et al. (2005), Coppersmith et al. (2006), and Kenyon-Mathieu and Schudy (2007). The material presented in Section 5.6 is from Caragiannis et al. (2012b, 2014b). Several interesting results have not been covered. For example, as an alternative to Tideman's simplified Dodgson rule, the maximin voting rule yields a Dodgson approximation with approximation ratio m^2 (Faliszewski et al., 2011b). Caragiannis et al. (2014b) discuss additional social-choice properties that are more difficult than monotonicity to achieve by good Dodgson approximations. Finally, observe that Section 5.6 does not contain any results related to Young's rule. Unfortunately, such good (polynomial-time) approximations are unlikely to exist. For example, unless $\mathsf{P} = \mathsf{NP}$, the StrongYoung score is not approximable within any factor by polynomial-time algorithms (Caragiannis et al., 2012b).

Acknowledgments

We are grateful to Markus Brill, Jörg Rothe, and the editors for helpful suggestions on an earlier version. Any remaining errors are the sole responsibility of the authors. We appreciatively acknowledge the support of grants NSF-CCF-0915792, NSF-CCF-1101452, and NSF-CCF-1101479.