Separations and Collapses in Computational Social Choice and Complexity Theory

by

Michael C. Chavrimootoo

Submitted in Partial Fulfillment of the

Requirements for the Degree

Doctor of Philosophy

Supervised by Professor Lane A. Hemaspaandra

Department of Computer Science
Arts, Sciences and Engineering
Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

2024

To my parents, who encouraged me to pursue my passions.

Table of Contents

Biographical Sketch

From a young age, Michael always found himself interested in mathematics, for which he exhibited a certain affinity. However, his interest and love for computers quickly overshadowed that, and his budding interest in elections only supplemented that distancing from mathematics. After moving from Mauritius to the University of Rochester (UR) for his undergraduate studies, he pursued courses in a variety of fields beyond computer science, namely, economics, law, mathematics, philosophy, political science, and psychology, among others.

At the end of his first four years at UR, Michael received a Bachelor of Science (B.S.) in Computer Science and a Bachelor of Arts (B.A.) in Political Science (with a focus on "Elections and Government"). During that time, he developed a keen interest in theoretical computer science and its applications in many fields, such as social choice theory, thereby reconnecting his passion for mathematics with his interest in computing and elections.

In the Fall of 2020, he joined the Computer Science Ph.D. program at the University of Rochester with Professor Lane A. Hemaspaandra as his advisor. During that time, he also received a Master of Science (M.S.) in Computer Science from the University of Rochester, and collaborated on over a dozen projects with 18 different coauthors. Additionally, he spent his summers teaching as an Adjunct Faculty at the Rochester Institute of Technology and as Adjunct Instructor at the University of Rochester, teaching nine courses on algorithms, computer models and limitations, and cryptography.

Acknowledgments

I would like to first thank my thesis committee and the thesis committee chair, namely Professors Edith Hemaspaandra, Lane A. Hemaspaandra, Anson Kahng, Saul Lubkin, Jörg Rothe, and Lenhart Schubert, for their feedback and insightful questions about the work described in this thesis.

I am especially indebted to Lane and Edith, who have been instrumental to my success in this program. I am grateful to Lane for introducing me to both complexity theory and computational social choice, for developing my interest in those topics, and for giving me the opportunity to study problems in those areas. I also am grateful to Lane for his mentorship and for his unfettered, insightful advice, among many other things. I also thank Edith for her support when I first started teaching at RIT, and her ever-so-insightful questions on my work that stumped me more often than not, and her advice and support while I was on the job market.

I thank Chen Ding for introducing me to the world of research and Daniel Štefankovič for starting my interest in theoretical computer science and in the study of algorithms. And I thank Piotr Faliszewski for his advice and insights.

To all my coauthors over the years: thank you! In particular, I would like to thank David E. Narváez for his many insightful discussions on generic oracles and proof assistants (among many other things), Henry B. Welles for his (infectiously) enthusiastic discussions about foundational questions regarding complexity, and

Benjamin Carleton for our many discussions that helped me better make sense of the problems we were tackling.

To my friends (especially Agustín Díaz Herrero, Brian C. Filipiak, Mervin Lim Pac Chong, and Mike A.D. Taylor) and family (especially my parents and my sisters): Thank you for your constant support and for always believing in me, even when I doubted myself. And I particularly want to thank my parents for teaching me to appreciate the beauty of both mathematics and the sciences from a young age, and for encouraging me to pursue my passion for computing, even when it did not make sense to them.

Finally, none of this would have been possible without the endless love, support, and companionship of my amazing partner Audrey F. Thompson, and without the joy that Buddy and Navi bring us.

Abstract

This thesis studies separations and collapses in new ways, in both computational social choice and computational complexity theory.

Since the seminal work of Bartholdi et al. (1989a,b, 1992) and Bartholdi and Orlin (1991) in computational social choice, the tools and techniques of theoretical computer science have been used to discover the complexities of a range of attacks on different election systems. That line of work has broadly assumed that the seemingly different attacks were in fact different. A key contribution of this thesis will be to explore, in the widely-studied electoral attack domain known as electoral control, the extent to which that assumption is correct. In particular, Hemaspaandra et al. (2020) showed that there were several electoral control problems that, although they had been studied separately for years, are in fact one and the same (when viewed as decision problems). This showed that researchers had for years been doing redundant work. We continue this line of work and uncover additional surprising relationships among electoral control types, thus discovering additional cases where duplicate effort has occurred, and helping the field avoid further duplicate work.

Building on this, for control types that collapse in the literature's standard notion of collapse—equality of their decision problems—this thesis studies whether the search complexities of the collapsing types also collapse (i.e., coincide). We find they do for all known collapsing types, and for all new collapsing types discovered in this thesis. We indeed go beyond that by showing that one often can efficiently

build search solutions for one type from the search solutions of the other type on the same input, and we provide a framework to study those search-complexity relationships.

In computational complexity theory, we study which pairs of ambiguity-bounded versions of NP stand (collapse) or fall (separate) together: one collapses to P exactly if the other does too. Prior to our work, the only known family of such connections was due to Watanabe (1988), and we present here new collections of such families.

Finally, in the area of the complexity of single-player games, we generalize a tool used to study reversible deterministic games to work for deterministic games with irreversible gravity. In some sense, this provides a "collapse" between the tools used to study two seemingly different categories of games. The particular too we study is the Nondeterministic Constraint Logic problem, which typically allows one to show PSPACE-hardness by constructing only two gadgets, but is unfortunately not directly applicable to the study of games with irreversible gravity. We devise a new method that when applied directly uses 32 gadgets, but we show that many of those gadgets are effectively equivalent (thus giving another type of "collapse") and that we can simulate all 32 gadgets by using only three gadgets. We apply this method to the study of the Hanano Puzzle and find that only two gadgets are needed to show the PSPACE-hardness of that game.

Looking at the bigger picture, this thesis looks in new ways at collapses and separations in computational social choice and computational complexity theory, making progress using a variety of tools ranging from proving theorems to providing new human and computer-discovered separations. Overall, we paint a clearer picture in computational social choice of what collapses and separations occur, and in computational complexity of what collapses stand or fall together.

Contributors and Funding Sources

Some of the content of this thesis is based on, and at times taken verbatim from, existing coauthored work (Carleton, Chavrimootoo, and Taliancich, 2021; Chavrimootoo and Welles, 2021; Carleton, Chavrimootoo, Hemaspaandra, Narváez, Taliancich, and Welles, 2022a,b, 2023a,b; Chavrimootoo, Ferland, Gibson, and Wilson, 2022; Chavrimootoo, Clingerman, and Luu, 2023a; Chavrimootoo, He, Kotler-Berkowitz, Liuson, and Nie, 2023b; Chavrimootoo, Le, Reidy, and Smith, 2023c) with my collaborators. Some parts of this thesis are also based on my single-authored papers (Chavrimootoo, 2022, 2023a,b).

Chapter 2 draws its definitions from Carleton, Chavrimootoo, Hemaspaandra, Narváez, Taliancich, and Welles (2022a, 2023b), Chavrimootoo (2023a), and Carleton, Chavrimootoo, Hemaspaandra, Narváez, Taliancich, and Welsh (2024).

As to the remaining chapters, Chapter 3 and Appendix A are based on content from Carleton, Chavrimootoo, Hemaspaandra, Narváez, Taliancich, and Welles (2022b, 2023b), Chapter 4 is based on content from Carleton, Chavrimootoo, Hemaspaandra, Narváez, Taliancich, and Welles (2022a), Chapter 5 is based on content from Carleton, Chavrimootoo, Hemaspaandra, Narváez, Taliancich, and Welsh (2024), Chapter 6 is based on content from Chavrimootoo (2022, 2023a), and Appendix B is based on Carleton, Chavrimootoo, and Taliancich (2021), Chavrimootoo and Welles (2021), Chavrimootoo, Ferland, Gibson, and Wilson (2022), Chavrimootoo, Clingerman, and Luu (2023a), Chavrimootoo, He, Kotler-

Berkowitz, Liuson, and Nie (2023b), and Chavrimootoo, Le, Reidy, and Smith (2023c).

This work was supported in part by NSF grant CCF-2006496.

List of Tables

List of Figures

1 Introduction

Much work in theoretical computer science (TCS) and in computational social choice (COMSOC) involves building models, proving properties of these models or of objects within these models, and proving relationships between models, among others.

A natural question one can ask oneself is whether there are examples of seemingly different models or objects being studied separately in the literature, and yet being the same if one inspects them more closely. The benefit of exploring this line of thought is self-evident: It makes little sense to waste resources (e.g., the time of researchers and reviewers) in studying the same objects and models multiple times. It is thus to our surprise that there are instances in the literature where some objects that were thought to differ and were thus treated separately for years actually turned out to be the exact same.

As an example, beyond the ones that this thesis deals with, we mention the following. The initial study of the complexity class IP, which we do not define here, was started by Babai (1985) and Goldwasser et al. (1985). However, it was eventually proven by Shamir (1992) that IP and PSPACE, an already well-known class at the time, in fact coincide. This was shocking for many reasons, in particular because the classes look—from their definitions—different, and yet are

not. This puts one of the major results preceding Shamir's—that the polynomial hierarchy PH is contained in IP (Lund et al., 1990)—into perspective; indeed, it was already known that PH $\subseteq$ PSPACE. Nonetheless, this seeming duplication of effort was *not* a waste of effort: This line of research involved using tools of theoretical computer science and mathematics to provide new insights.

This thesis aims at demonstrating that the tools of theoretical computer science and mathematics can be used to provide insights into a variety of fields, particularly in the study of the hardness of certain types of games, in the study of ambiguity-bounded computation, and more importantly, in the study of computational social choice.

Electoral control problems are well-studied in COMSOC. A common research problem in COMSOC is to study the decision complexity (e.g., in P or NP-hard) of an electoral control problem for a given election system (i.e., a rule that outputs a winner set given a set of candidates and a set of votes). When the problem is NP-hard, we say the election system is resistant to that control attack.[1] Resistance to control attacks is, naturally, a desirable property of an election system, and that motivated somewhat of a race in the literature to find a natural election system that was resistant to as many of the standard control attacks as possible. Yet, unbeknownst to the researchers at the time, several of the control problems under their consideration were actually exactly the same when viewed as decision problems. The first such cases were discovered by Hemaspaandra et al. (2020) and from then until our work, this direction remained largely unexplored.

Clarifying the relationships between problems and classes of problems is a fundamental task in complexity theory. (We provide a brief introduction to the topic in Section 2.1.) We use the tools and techniques of TCS and mathematics to develop our understanding of the problems we have at hand. So what else can

[1]For the sake of clarity, we are intentionally avoiding the issue of so-called "immunity" in this introduction.

we provably say "is the same" (i.e., collapse)? Or what else can we provably say "is not the same" (i.e., separate)?

The *goal of this thesis* is to present the results of projects in which we obtain answers to these questions, and to outline future research directions where we will continue to explore when certain objects are the same or differ.

In the context of electoral control problems, a decision problem asks if there is a successful move by the attacker that achieves their goal, such as making a given candidate win the election. We show that many electoral control problems are the same (i.e., collapse) when viewed as decision problems (Carleton et al., 2022b, 2023b). Since our model is the decision model, collapses are proved by proving sets to be equal, and separations are proved by showing that sets are not equal. We prove our collapse and "containment" results by using axiomatic methods at times and by giving direct arguments at other times. To prove our separations, we provide human-generated "separation witnesses" when possible. And in many other cases, we make use of computer programs (to automate the search for separation witnesses).

Of course, in the real world, the typical goal is to *find* a particular action within the set of allowed moves that achieves one's goal, which corresponds to the search model. We show that the collapses we or Hemaspaandra et al. (2020) prove in the decision model also hold in the search model (Carleton et al., 2022a, 2023a)! In effect, this means that in addition to collapsing control problems having the same decision complexity, their search complexities are tightly related. It is important to investigate this decision-to-search connection, as there are standard control attacks for which the decision complexities and the search complexities differ under reasonable assumptions (Hemaspaandra et al., 2020). We provide a framework to interrelate the search complexities of search problems. We also determine the search complexities of those problems (such as "polynomial-time computable" and "NP-hard") and provide a "bridge" theorem to link decision and

search complexities. The aforementioned bridge theorem leverages the notion of self-reducibility and helps us not duplicate work when proving search complexities.

In the context of ambiguity-bounded computation, we provably link the relationship between certain ambiguity-bounded versions of NP, e.g., UP and FewP. Prior to that, the only known linkage was that $\mathrm{P} = \mathrm{UP} \iff \mathrm{P} = \mathrm{UP}_{\mathcal{O}(1)}$ (Watanabe, 1988). We do so by proving new relationships of the form, for given ambiguity-bounded versions of NP $\mathcal{A}$ and $\mathcal{B}$, "$\mathrm{P} = \mathcal{A} \iff \mathrm{P} = \mathcal{B}$." We give one family of linkages by using padding and one family by using a new inductive approach. Finally, we show how to combine the two approaches to yield a family that contains the one obtained by our padding approach, but not the one obtained by our inductive approach.

In the study of the hardness of games, we establish that a well-known framework for studying the complexity of fully reversible, deterministic PSPACE-complete games can be used to study irreversible, deterministic PSPACE-complete games with gravity. Prior to our work (Chavrimootoo, 2022), all known approaches to study such games with gravity used ad hoc reductions that do not seem to generalize naturally. We show this connection by leveraging the notion of visibility representations, and introducing additional structure into the reductions. In some sense, we see that there is a type of "collapse" in the methods used to study these seemingly different classes of games. The new framework is applied to pinpoint the complexity of the Hanano Puzzle, a previously open issue. While the added structure seems to create a need for more gadgets, we show how a large number of seemingly different gadgets are effectively equivalent and interchangeable, which we view as a different type of collapse.

Finally, we include in the appendix our work on critiquing and verifying claims about separations and collapses in complexity theory, which involved 15 undergraduate students and resulted in six technical reports (Carleton et al., 2021; Chavrimootoo and Welles, 2021; Chavrimootoo et al., 2022, 2023a,b,c).

2 Preliminaries

This section introduces the concepts that this thesis considers, namely computational social choice—i.e., voting theory and electoral control—and complexity theory—i.e., the hardness of games and ambiguity-bounded computation.

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$, let $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$, let $\mathbb{R}$ denote the set of real numbers, let $\mathbb{R}^{\geq 1} = \{x \in \mathbb{R} \mid x \geq 1\}$, and let $\mathbb{R}^+ = \{x \in \mathbb{R} \mid x > 0\}$. As is standard, given two functions f and g from $\mathbb{N}$ to $\mathbb{R}^+$, we say that $f(n) = \mathcal{O}(g(n))$ exactly if there are positive natural numbers c and n_0 such that $(\forall n \geq n_0)[f(n) \leq cg(n)]$ (Sipser, 2013). For each $\mathcal{R} \subseteq \mathbb{R}$, we will say that a function $f : \mathbb{N} \to \mathcal{R}$ is monotonically nondecreasing exactly if $(\forall n_1, n_2 \in \mathbb{N})[n_1 \leq n_2 \implies f(n_1) \leq f(n_2)]$. This notion, for the case $\mathcal{R} = \mathbb{R}^{\geq 1}$, will be important in Chapter 5.

We first introduce general complexity theory, before introducing computational social choice and more specific topics in complexity.

2.1 Complexity Theory

This thesis assumes elementary familiarity with deterministic and nondeterministic Turing machines as discussed in such classic textbooks as Hopcroft and Ullman (1979) and Papadimitriou (1994).

Complexity theory is primarily concerned with the study of the "easiness"/"hardness" of problems and classes of problems. For example, given an arbitrary number $n \in \mathbb{N}$, how "easy" or how "hard" is it to answer the questions:

1. "Is n even?"
2. "Is n prime?"
3. "Is n the index of a Turing machine (TM) (under some standard enumeration of TMs) that halts when given its own encoding as an input?"

To even discuss what "easy" and "hard" mean, we must introduce the standard model in complexity, which is the decision model. Let us explain this by way of an example.

The yes/no question "Is an input $n \in \mathbb{N}$ an even number?" defines a decision problem. We can associate to that (decision) problem a set of inputs for which the answer to the question is "yes" (and we call those the Yes instances of the decision problem). (The standard terminology for such sets is "language," and in this thesis, we will often use the words "set," "problem," and "language" interchangeably when appropriate.) Thus the aforementioned decision problem is associated with the set EVEN $= \{n \in \mathbb{N} \mid n \text{ is an even number}\}$. Furthermore, we often abuse notation and use the name of the set—which is usually descriptive—to refer to the problem. Finally, a (complexity) class (of problems) is simply a set of languages.

When we say that a problem A is "easy" or "tractable" we mean that there is a polynomial-time algorithm (i.e., an algorithm whose runtime is bounded by a polynomial in the input size) that determines membership in A. The class of such easy problems is denoted by P, and the typical way of showing that a problem A is in P is to give a polynomial-time algorithm that determines membership in A.

Checking if a number is even is an easy task given in courses introducing programming, and checking if a number is prime is also easy (i.e., tractable) by

the primality breakthrough of Agrawal et al. (2004). The third question in our list is, however, "hard"—in fact, it is provably not in P.

Discussing "hardness" or "intractability" in complexity is not as simple as discussing easiness/tractability. We want a "hard" problem B to have the property that each of the (countably infinitely many) polynomial-time algorithms fails to determine membership in B, but, as it turns out, establishing such a property is not well-understood in complexity. While we can say with certainty that some problems are not in P, there is no known universal method to do this for every problem not in P. One way to address this issue is to leverage the power of polynomial-time many-one reductions.

Let FP denote the class of polynomial-time computable functions, i.e., for each $f \in \text{FP}$, there is a deterministic polynomial-time Turing machine that when given x as its input, halts with exactly $f(x)$ on its output tape. Set A is said to polynomial-time many-one reduce to set B (denoted $A \leq_m^p B$; for convenience, we will often just say "A many-one reduces to B") exactly if $(\exists f \in \text{FP})(\forall x)[x \in A \iff f(x) \in B]$. Given a complexity class $\mathcal{C}$ and set B, we say that B is $\mathcal{C}$-hard exactly if $(\forall C \in \mathcal{C})[C \leq_m^p B]$. If in addition to being $\mathcal{C}$-hard it holds that $B \in \mathcal{C}$, then B is said to be $\mathcal{C}$-complete. (We note in passing that it is possible, and not uncommon, to define "-hardness" and "-completeness" in terms of other forms of reductions, but in this thesis, those notions are defined in terms of polynomial-time many-one reductions.) Polynomial-time many-one reductions are transitive, and so in practice, to the show $\mathcal{C}$-completeness of a new set $B \in \mathcal{C}$, it is preferred to simply show that there is a known $\mathcal{C}$-complete set X such that $X \leq_m^p B$.

Polynomial-time many-one reductions are transitive and reflexive relations, and induce a partial ordering on the set of languages. Intuitively, if $A \leq_m^p B$, we understand that A is "no harder" than B. For a given class $\mathcal{C}$, a problem A being $\mathcal{C}$-hard is typically interpreted to mean that no problem in $\mathcal{C}$ is harder than A. And $\mathcal{C}$-complete problems are commonly interpreted to be the (equally) hardest

problems in $\mathcal{C}$.

The class NP refers to the set of languages accepted by nondeterministic polynomial-time Turing machines, and this class is central to one of the biggest open questions in complexity theory: the P versus NP problem. It is conjectured that P $\neq$ NP, and so we associate "hardness" with NP-hardness, i.e., in this thesis—as is common in computer science—we consider a problem A to be hard/intractable if it is NP-hard (recall that this means that for every $L \in$ NP, $L \leq_m^p A$). A classic example of an NP-hard problem is that of "boolean satisfiability": Given a boolean formula f, is there an assignment of values to its variables that make the formula evaluate to true? This problem is often denoted by SAT and is in fact NP-complete.

Though it is not explicitly relied upon in this thesis, based on the notions introduced so far, we can establish the following good-to-know fact.

Fact 2.1. *Suppose $A \leq_m^p B$.*

1. *If $B \in$ P, then $A \in$ P.*

2. *If A is* NP*-hard, then B is* NP*-hard.*

In complexity, we are also concerned with the notion of oracles and oracle machines. We can view an oracle as a language, i.e., a set. An oracle machine is a modified Turing machine that has the additional capability of querying an oracle A to determine the membership of a string in A (Sipser, 2013). As is standard, we denote the oracle machine M with oracle A by M^A. The common view is to treat oracles like black boxes, and we are not concerned with how membership in the oracle is determined. Sipser (2013) mentions that the term oracle is used to draw a connection to the "magic ability" of an oracle machine to query the oracle. We can also define complexity classes in terms of those oracles. For example P^A denotes the class of languages accepted by oracle machines with oracle A that run

in polynomial time relative to the oracle. The runtime of an oracle machine may be different when it is equipped with different oracles.

Other complexity theory concepts relevant to the thesis are introduced in Sections 2.4 and 2.5, and as needed in the relevant chapters.

2.2 Voting Theory

In this section, we introduce the formal framework of computational social choice.

An election is a pair (C, V), where C is a finite set of candidates and V is a finite collection of votes over those candidates in C. The collection of votes is often just called a vote set or a set of votes for simplicity, and it is implicitly understood to be a multiset. In this thesis, unless otherwise specified, we understand a vote to be a complete ordering of the set of candidates. For example, if $C = \{a, b, c\}$, then a possible set of votes is $\{a > b > c, a > c > b, a > b > c, b > c > a\}$. In Chapters 3 and 4, we will also consider a different type of vote, which we will discuss then.

An election system (sometimes also known as voting rule) is a function that maps an election (C, V) to a subset of C, i.e., the winner set. As is standard in COMSOC, we allow empty winner sets (which is not common in pure social choice theory). One reason to allow the empty winner set is to be consistent with the COMSOC literature, but we also feel it is more natural to do so as allowing the empty winner set is symmetric with allowing every candidate to win. Moreover, it is certainly possible in the real world to have an election where no candidate in elected. For example, under majority (where a candidate ranked first by more than half of the voters uniquely wins that election), if no candidate receives more than half of the votes, then no candidate is elected. This is in some sense what happened in the 2023 Speaker of the United States House of Representatives election. Fifteen majority elections were held as in each of the

first fourteen elections, no candidate had secured the majority of the votes. Each round, voters recast their votes, and by the fifteenth round, Kevin McCarthy was able to secure a majority of the votes and won the election.

Let us look at examples of an election system and the output of such a function on given inputs. A common and important election system is "plurality." In this system, each candidate receives one point for each vote in which they are ranked first. The winners are those candidates with maximal points.

Example 2.2. Let $C = \{a, b, c\}$ and let $V = \{a > b > c, a > b > c, a > b > c, b > a > c, b > a > c, c > b > a, c > b > a\}$. Based on the plurality election system, a receives three points, b receives two points, and c receives two points, thus making a the unique winner of the plurality election (C, V).

In the previous example, only one candidate wins. The next example depicts a plurality election with multiple winners.

Example 2.3. Let $C = \{a, b, c\}$ and let $V = \{a > b > c, a > b > c, a > b > c, b > a > c, b > a > c, b > a > c, c > b > a\}$. Based on the plurality election system, a receives three points, b receives three points, and c receives one point, thus making a and b the tied winners of the plurality election (C, V).

Election systems are known to satisfy properties that are often called axioms. For example, if an election system never produces multiple winners on each input, it is said to be resolute (equivalently, it is said to satisfy resoluteness). In this thesis, we will also look at classes of elections satisfying certain axioms rather than focus on concrete election systems.

2.3 Electoral Control

The computational study of certain control attacks was introduced by Bartholdi et al. (1992). Their work was focused on determining the complexity of those

control problems. They started the study of the notion known as "constructive control" in which the goal is to make a distinguished candidate win (perhaps uniquely or nonuniquely). Hemaspaandra et al. (2007) then introduced the analogous notion of "destructive control" in which the goal is to make a distinguished candidate not win. Again, "winning" can be defined as "uniquely winning" or "nonuniquely winning" (i.e., tied as winner).

The exact control actions are: adding voters, adding candidates, unlimited adding candidates, deleting voters, deleting candidates, partition of voters, run-off partition of candidates, and partition of candidates. Each of the control actions that involve partitioning (i.e., the last three mentioned above) involve running subelections wherein the winners (if any) of those subelections proceed to a final-round election. In their seminal work, Bartholdi et al. (1992) dismissed the importance of the tie-handling rule in these subelections, but Hemaspaandra et al. (2007) showed that the tie-handling rule matters. They defined two rules that have become standard in the literature: The ties promote (TP) rule allows every winner (even if tied) of a subelection to proceed to the final round, and the ties eliminate (TE) rule allows a winner of a subelection to proceed to the final round exactly if it is a unique winner of that subelection. They then used those two tie-handling rules to show that changing the tie-handling rule can make a particular control problem go from being in P to being NP-complete!

Definition 2.4 defines the 11 standard constructive control problems in the nonunique-winner model, and we will from that explain how the analogous definitions in the unique-winner model follow. We will also explain how to get the destructive control definitions from the constructive control ones. In our work, when given a collection of votes V over a set of candidates C, we will often consider elections of the form (C', V), where $C' \subseteq C$. As is standard, it is implicitly understood that in (C', V), V is a collection of votes restricted in the natural and obvious way to be over C'. The following definition is taken verbatim from

Carleton et al. (2022b).

Definition 2.4. *Let $\mathcal{E}$ be an election system.*

1. *In the* ***constructive control by adding candidates*** *problem for $\mathcal{E}$ (denoted by $\mathcal{E}$-CC-AC-NUW), we are given two disjoint sets of candidates C and A, V a collection of votes over $C \cup A$, a candidate $p \in C$, and a nonnegative integer k. We ask if there is a set $A' \subseteq A$ such that (i) $\|A'\| \leq k$ and (ii) p is a winner of $\mathcal{E}$ election $(C \cup A', V)$.*

2. *In the* ***constructive control by unlimited adding candidates*** *problem for $\mathcal{E}$ (denoted by $\mathcal{E}$-CC-UAC-NUW), we are given two disjoint sets of candidates C and A, V a collection of votes over $C \cup A$, and a candidate $p \in C$. We ask if there is a set $A' \subseteq A$ such that p is a winner of $\mathcal{E}$ election $(C \cup A', V)$.*[2]

3. *In the* ***constructive control by deleting candidates*** *problem for $\mathcal{E}$ (denoted by $\mathcal{E}$-CC-DC-NUW), we are given an election (C, V), a candidate $p \in C$, and a nonnegative integer k. We ask if there is a set $C' \subseteq C$ such that (i) $\|C'\| \leq k$, (ii) $p \notin C'$, and (iii) p is a winner of $\mathcal{E}$ election $(C - C', V)$.*

4. *In the* ***constructive control by adding voters*** *problem for $\mathcal{E}$ (denoted by $\mathcal{E}$-CC-AV-NUW), we are given a set of candidates C, two collections of votes, V and W, over C, a candidate $p \in C$, and a nonnegative integer k. We ask if there is a collection $W' \subseteq W$ such that (i) $\|W'\| \leq k$ and (ii) p is a winner of $\mathcal{E}$ election $(C, V \cup W')$.*

[2]This control type only differs from the former in the limit imposed by k. Historically, this version was defined first, but the abovementioned one was introduced to be symmetric with the control types that follow in this definition.

5. *In the* ***constructive control by deleting voters*** *problem for* $\mathcal{E}$ *(denoted by* $\mathcal{E}$-CC-DV-NUW*), we are given an election* (C, V)*, a candidate* $p \in C$*, and a nonnegative integer* k*. We ask if there is a collection* $V' \subseteq V$ *such that (i)* $\|V'\| \leq k$ *and (ii)* p *is a winner of* $\mathcal{E}$ *election* $(C, V - V')$*.*

6. *In the* ***constructive control by partition of voters*** *problem for* $\mathcal{E}$*, in the TP or TE tie-handling rule model (denoted by* $\mathcal{E}$-CC-PV-TP-NUW *or* $\mathcal{E}$-CC-PV-TE-NUW*, respectively), we are given an election* (C, V)*, and a candidate* $p \in C$*. We ask if there is a partition*[3] *of* V *into* V_1 *and* V_2 *such that* p *is a winner of the two-stage election where the winners of subelection* (C, V_1) *that survive the tie-handling rule compete (with respect to vote collection* V*) along with the winners of subelection* (C, V_2) *that survive the tie-handling rule. Each election (in both stages) is conducted using election system* $\mathcal{E}$*.*

7. *In the* ***constructive control by run-off partition of candidates*** *problem for* $\mathcal{E}$*, in the TP or TE tie-handling rule model (denoted by* $\mathcal{E}$-CC-RPC-TP-NUW *or* $\mathcal{E}$-CC-RPC-TE-NUW*, respectively), we are given an election* (C, V)*, and a candidate* $p \in C$*. We ask if there is a partition of* C *into* C_1 *and* C_2 *such that* p *is a winner of the two-stage election where the winners of subelection* (C_1, V) *that survive the tie-handling rule compete (with respect to vote collection* V*) against the winners of subelection* (C_2, V) *that survive the tie-handling rule. Each election (in both stages) is conducted using election system* $\mathcal{E}$*.*

8. *In the* ***constructive control by partition of candidates*** *problem for* $\mathcal{E}$*, in the TP or TE tie-handling rule model (denoted by* $\mathcal{E}$-CC-PC-TP-NUW

[3] A partition of a collection V is a pair of collections V_1 and V_2 such that $V_1 \cup V_2 = V$, where $\cup$ denotes multiset union. A partition of a set C is a pair of sets C_1 and C_2 such that $C_1 \cup C_2 = C$ and $C_1 \cap C_2 = \emptyset$, where $\cup$ and $\cap$ are standard set union and intersection.

or $\mathcal{E}$-CC-PC-TE-NUW*, respectively), we are given an election* (C, V)*, and a candidate* $p \in C$*. We ask if there is a partition of* C *into* C_1 *and* C_2 *such that* p *is a winner of the two-stage election where the winners of subelection* (C_1, V) *that survive the tie-handling rule compete (with respect to vote collection* V*) against all candidates in* C_2*. Each election (in both stages) is conducted using election system* $\mathcal{E}$*.*

This defines 11 electoral control problems—each of the last three definitions define two control problems, one for each tie-handling rule—also known as electoral control types. The remaining constructive control types are about the unique-winner model and are defined by changing "p is a winner" in each of the above definitions to "p is a unique winner," which leads to also changing the suffix of the control problem's abbreviation from "-NUW" to "-UW." The fact that these control problems are about constructive control is included in their abbreviations via the "CC" prefix. We now need to define the destructive variants (whose abbreviation prefixes will be "DC" rather than "CC"; note that "DC" can mean "destructive control" or "deleting candidates," but it is always clear from context what "DC" refers to). These are defined by simply changing the question to ask that "p is not a winner" or "p is not a unique winner," depending on the winner model under consideration.

As is standard in the literature, we view each control problem as a decision problem and use each control problem/type's abbreviation to denote the set of Yes instances to that decision problem.

Let us look at an example of control, by building on Example 2.2.

Example 2.5. Let $C = \{a, b, c\}$ and let $V = \{a > b > c, a > b > c, a > b > c, b > a > c, b > a > c, c > b > a, c > b > a\}$. It is easy to see that a receives three points, b receives two points, and c receives two points, thus making a the unique winner of the plurality election (C, V). However, we can

make b the unique winner by using partition of candidates. Consider the partition of candidates where $C_1 = \{a, b\}$ and $C_2 = \{c\}$. Let the tie-handling rule be TP, although in this example the TE rule also yields the same result. Then in election (C_1, V), a receives three points, while b receives four points and uniquely wins that subelection. Now the final round is the election $(C_2 \cup \{b\}, V) = (\{b, c\}, V)$, and in that election, b receives five points, while c only receives two points, making b the unique winner of the two-stage election. In our terminology, $(C, V, b) \in$ Plurality-CC-PC-TP-UW.

In addition to studying the complexity of control problems, it is also common to study the notion of immunity. Namely, in the unique-winner model, we say an election system is immune to a particular type of control if the given type of control can never change a candidate from not uniquely winning to uniquely winning (if the control type is constructive) (Bartholdi et al., 1992) or change a candidate from uniquely winning to not uniquely winning (if the control type is destructive) (Hemaspaandra et al., 2007). In the nonunique-winner model, we say an election system is immune to a particular type of control if the given type of control can never change a nonwinner to a winner (if the control type is constructive) or change a winner to a nonwinner (if the control type is destructive).

2.4 Ambiguity-Bounded Versions of NP

In our work related to ambiguity-bounded versions of NP, we will at times abuse notation, in cases where it is unambiguous to do so, by writing $f(n)$ to either denote the value that a function f maps n to, or to denote that function f itself. We assume in this thesis that $\log(\cdot) = \log_2(\cdot)$, unless noted otherwise.

The notion of ambiguity-bounded machines was introduced by Valiant (1976) when he introduced the class UP ("unambiguous polynomial time") as follows: A language L is in UP exactly if there exists a nondeterministic polynomial-time

Turing machine (NPTM) that on each input x has exactly one accepting path if $x \in L$ and has no accepting paths if $x \notin L$. It is known that $\mathrm{P} \subseteq \mathrm{UP} \subseteq \mathrm{NP}$, but it is not known if the inclusions are strict.

Starting with the work of Allender and Rubinstein (1988) that defined a polynomial-ambiguity sibling of UP, many other ambiguity-bounded versions have been defined and studied. UP is also sometimes called $\mathrm{UP}_{\leq 1}$ as the concept of bounded ambiguity can be generalized as follows. The notation we use is due to Lange and Rossmanith (1994).

We denote the number of accepting paths of an NPTM M on input x by $\#\mathrm{acc}_M(x)$. For a given function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$, a language L is in $\mathrm{UP}_{\leq f(n)}$ exactly if there exists an NPTM N such that for every $x \in L$, $1 \leq \#\mathrm{acc}_N(x) \leq f(|x|)$,[4] and for every $x \notin L$, $\#\mathrm{acc}_N(x) = 0$.

Given a class of functions $\mathcal{F}$, we also define $\mathrm{UP}_{\mathcal{F}} = \bigcup_{f \in \mathcal{F}} \mathrm{UP}_{\leq f(n)}$. This notation will be useful when discussing certain results in Chapter 5.

Among the existing ambiguity-bounded classes are (in some cases with slightly different notations in their original papers) the polynomial-ambiguity class $\mathrm{FewP} = \bigcup_{k \in \mathbb{N}^+} \mathrm{UP}_{\leq n^k + k}$ of Allender and Rubinstein (1988); the log-bounded ambiguity class $\mathrm{UP}_{\mathcal{O}(\log(n))}$ and also the classes $\mathrm{UP}_{\mathcal{O}(\sqrt{\log(n)})}$ and $\mathrm{UP}_{\mathcal{O}(\log(\log(n)))}$ of Hemaspaandra et al. (2022); and the k-bounded ambiguity classes $\mathrm{UP}_{\leq 2}$, $\mathrm{UP}_{\leq 3}$, ... of Beigel (1989), and their union, the constant-bounded ambiguity class $\mathrm{UP}_{\mathcal{O}(1)}$ of Hemaspaandra and Zimand (1993).[5] Note that $\mathrm{NP} = \mathrm{UP}_{2^{n^{\mathcal{O}(1)}}}$.

[4] Because $\#\mathrm{acc}_N$ always maps to a natural number, it is fine for us to use "$f(|x|)$" instead of "$\lfloor f(|x|) \rfloor$" (where $\lfloor \cdot \rfloor$ represents the floor function) as both options yield equivalent definitions for our purposes.

[5] There is a rather subtle issue at play in this sentence. Consider the class $\mathrm{UP}_{\leq \log(n)}$. Its definition automatically excludes strings of length 0 or 1, which is unnatural, and problematic as that would exclude a countably infinite number of sets in P from $\mathrm{UP}_{\leq \log(n)}$, so the statement $\mathrm{P} = \mathrm{UP}_{\leq \log(n)}$ is always false. To remedy this situation, we implicitly assume in this work that "$\log(\cdot)$" is a shorthand for "$\log(\max(2, \cdot))$."

Each of the above ambiguity-bounded versions of NP are further motivated by the fact that their equality to P characterizes the existence of complexity-theoretic one-way functions whose ambiguity (i.e., whose limits on the number of preimages of each string in the range) share the same bound.[6] The general correspondence is that $\mathrm{P} \neq \mathrm{UP}_{\leq f(n)} \iff$ there exists an $f(n)$-to-1 one-way function (Hemaspaandra et al., 2022).[7] Though Hemaspaandra et al. (2022) give the prior result in the general case, earlier case-specific results were given by Grollmann and Selman (1988).

For any claim regarding classes for which a generally agreed upon notion of relativization has been reached in the literature, we say that the claim holds *robustly* if it holds for each oracle A. For example, if we were to say that $\mathrm{P} \subseteq \mathrm{NP}$ holds robustly—and indeed it does—that would be asserting that for each oracle A it holds that $\mathrm{P}^A \subseteq \mathrm{NP}^A$. The vast majority of theorems, proofs, and proof techniques used to study complexity classes relativize. Thus, showing that a claim holds in some relativized world establishes that a wide range of proof techniques cannot prove that the claim fails in the "real" (i.e., unrelativized) world. However, there exist proof techniques and results, e.g., arithmetization and IP = PSPACE, that (in some cases controversially as to what the "right"/"fair" model of relativization is) seem not to relativize (Babai and Fortnow, 1991; Hartmanis et al., 1992; Lund et al., 1992; Shamir, 1992; Buhrman et al., 1998; Vereshchagin, 1994;

[6]Though it is not needed to understand the work in this thesis, we provide here briefly the definition of complexity-theoretic one-way functions as defined in Hemaspaandra and Ogihara (2002). For a function f, we let $\mathrm{range}(f)$ denote the range of f and $\mathrm{dom}(f)$ denote the domain of f. A (possibly nontotal) function f is *honest* if there is a polynomial q such that $(\forall y \in \mathrm{range}(f))(\exists x)[f(x) = y \wedge |x| \leq q(|y|)]$. Moreover, a (possibly nontotal) function f is *polynomial-time invertible* if there is a polynomial-time computable function g such that $(\forall y \in \mathrm{range}(f))[y \in \mathrm{dom}(g) \wedge g(y) \in \mathrm{dom}(f) \wedge f(g(y)) = y]$. A (possibly nontotal) function f is *one-way* if f is honest, polynomial-time computable, and *not* polynomial-time invertible.

[7]A function g is $f(n)$-to-one exactly if for each y, $\|\{x \mid g(x) = y\}\| \leq f(|y|)$. This notion too is not necessary to understand the work in this thesis.

Aaronson, 2020; Ji et al., 2020).

2.5 Hardness of Games

It is often noted that people in TCS thoroughly enjoy problem-solving and working on puzzle games. It is thus no surprise that a rich area of study in complexity theory is that of the hardness of games.

Hearn and Demaine's textbook (2009) provides a comprehensive overview of the complexity of games that had been studied at the time. They devised a framework—a family of Constrained Logic problems—to ease the task of devising many-one reductions to show the hardness (with respect to specific classes) of *reversible, deterministic games*.[8] In this thesis we are only concerned with the complexity class PSPACE—the class of languages that are accepted by Turing machines using only polynomial space (in the input size)—so we will focus on Nondeterministic Constraint Logic (NCL), which Hearn and Demaine (2009) used to show the PSPACE-hardness of many games (often significantly simplifying existing proofs in the literature). The definitions relating to NCL that follow are based on definitions of Hearn and Demaine (2009).

An NCL graph is a weighted, directed graph where each edge has weight one (aka red edge) or has weight two (aka blue edge). There is a global constraint that the sum of weights of incoming edges into each vertex must be at least two. This is known as the (minimum) inflow constraint.

There is one operation allowed on an NCL graph: We can flip the direction of an edge in an NCL graph provided that the flip does not violate the minimum inflow constraint.

[8]Though the framework was introduced in the context of games, it has found applications in other areas, such as in planning. We go into more details in Chapter 6.

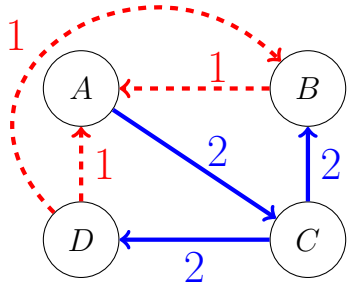


Figure 2.1: Example of an NCL graph.

Given an NCL graph $G = (V, E)$ and an edge $e \in E$, determining if there is a sequence of edge flips such that e is eventually flipped is PSPACE-complete. The problem turns out to remain PSPACE-complete even if

1. G is simple, i.e., has no self-loops, and no two vertices have more than one edge between them,

2. G is planar, i.e., can be drawn on a flat piece of paper without having any edges intersect (except at common endpoints), and

3. every vertex of G is either an AND vertex (i.e., has degree 3 and every edge connected to it is blue) or an OR vertex (i.e., has degree 3 with one connected edge being blue and the remaining two being red).

Such graphs are called planar AND/OR NCL graphs, but since these are the only types of NCL graphs considered in our work, we shall tacitly assume that every NCL graph considered henceforth is a planar AND/OR NCL graph. Figure 2.1 gives an example of an NCL graph (in this thesis, red edges are drawn using dotted lines while blue edges are drawn using solid lines).

We will be looking at the complexity of the Hanano Puzzle, which is a game of gravity, i.e., a game with a vertical dimension where objects that have no support

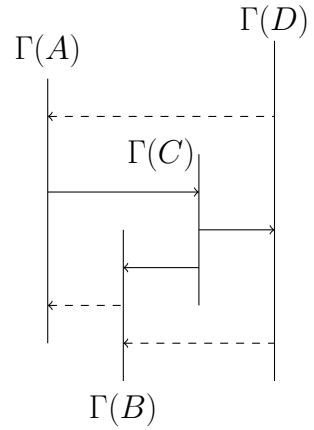


Figure 2.2: A (directed) visibility representation of Figure 2.1. For a vertex v, $\Gamma(v)$ denotes the vertical segment representing v. Solid (resp. dashed) edges from Figure 2.1 are drawn as solid (resp. dashed) horizontal lines.

fall until they encounter an object to support them. We'll say that the gravity is irreversible if the game has no provision for a fallen object to retrace its steps and return to its previous position. For example, the lack of a "jump" ability could satisfy this "irreversibility" notion. To abstract away the notion of "irreversible gravity," we will make use of the notion of visibility representations.

Definition 2.6 (Tamassia 2016)**.** *Given a graph $G = (V, E)$, a* visibility representation Γ *for G maps every vertex $v \in V$ to a vertical vertex segment $\Gamma(v)$ and every edge $(u, v) \in E$ to a horizontal edge segment $\Gamma(u, v)$ such that each horizontal edge segment $\Gamma(u, v)$ has its respective endpoints lying on the vertical vertex segments $\Gamma(u)$ and $\Gamma(v)$, and no other segment intersections or overlaps occur.*[9]

Figure 2.2 shows an example of a visibility representation of the graph in Figure 2.1.

[9]This definition deviates slightly from the standard one; in the standard definition, vertices are mapped to horizontal segments, and edges are mapped to vertical segments. For our purposes, both definitions are equivalent.

It is well-known how to compute the visibility representations of planar graphs in linear time, and those representations will be crucial for our (many-one) reduction.

Theorem 2.7 (Tamassia and Tollis 1986; Tamassia 2016)**.** *A graph admits a visibility representation if and only if it is planar. Furthermore, a visibility representation for a planar graph can be constructed in linear time.*[10]

[10] An earlier version (Chavrimootoo, 2022) of this work, unaware of the earlier results of Tamassia and Tollis (1986), proved a weaker version of this theorem: While our result established the "if" direction (the "only if" direction is trivial), our polynomial-time algorithm did not run in linear time.

3 Separating and Collapsing Electoral Control Types

No mathematician in the world would bother making these senseless distinctions: $2\,1/2$ *is a "mixed number" while* $5/2$ *is an "improper fraction." They're* equal*, for crying out loud. They are the exact same numbers, and have the exact same properties. – Lockhart (2009)*

3.1 Introduction

In a recent paper, Hemaspaandra et al. (2020) show that for every election system $\mathcal{E}$ where the votes are linear orders over the candidates it holds that (1) $\mathcal{E}$-DC-RPC-TE-UW = $\mathcal{E}$-DC-RPC-TE-NUW = $\mathcal{E}$-DC-PC-TE-NUW = $\mathcal{E}$-DC-PC-TE-UW, and (2) $\mathcal{E}$-DC-RPC-TP-NUW = $\mathcal{E}$-DC-PC-TP-NUW. Their proof builds on a characterization provided by Faliszewski et al. (2009). Our work finds that the proof of Hemaspaandra et al. (2020) does not actually rely on votes being linear orders, thus making their proofs hold for any election system $\mathcal{E}$.

Although researchers in the field initially thought that there were 44 distinct electoral control types (for each election system), there are 40. This was surprising as those problems had been treated differently for years by many prolific

researchers in computational social choice. In our work, we explore whether there are additional such equalities (i.e., collapses) or if all the other pairs of control types in fact do differ (i.e., separate). We prove that in the general case, there are no additional collapses, but if we bring our attention to concrete election systems such as veto and approval, then there are many more—fifteen to be precise—collapses hiding in plain sight. For each pair of control types for the concrete election systems plurality, veto, and approval, we determine the relationship between these types and classify them as either "equal" (i.e., collapsing), "$\subsetneq$," "$\supsetneq$," or "incomparable" (see Section 2.3 for precise definitions of each of these terms). Additionally, we provide some axiomatic sufficient conditions that will simplify future studies involving election systems satisfying these conditions. Table 3.1 provides a summary of our results.

Table 3.1: Summary of separations and collapses. This table is due to the technical report version of this work (Carleton et al., 2022b). Blue indicates results due to or inherited from Hemaspaandra et al. (2020). Red indicates results due to this work. The general-case line shows when collapses occur for all election systems over linear orders.

	Set Classification			Subclassification of Separations		
Election System	Separations	Collapses	Open	"$\subsetneq$"/"$\supsetneq$"	Incomparable	Open
General Case	$1 + 314 = 315^{\dagger}$	7	0	38	277	0
Plurality	315	7	0	38	277	0
Veto	314	$7 + 1 = 8$	0	58	256	0
Approval Voting	301	$7 + 14 = 21$	0	88	213	0

† Or $0 + 315$ for the pure social choice approach to candidate names (see Footnote 2 and the Related Works section of our technical report for more details).

Let us now define formally our notions of separations and collapses. Two control types are said to be compatible if their input types are the same. For example, $\mathcal{E}$-CC-RPC-TP-UW and $\mathcal{E}$-DC-PV-TE-NUW are compatible, but $\mathcal{E}$-CC-RPC-

TP-UW and $\mathcal{E}$-CC-AC-UW are not compatible. Fix an election system $\mathcal{E}$. We say that two compatible types $\mathcal{E}$-$\mathcal{T}_1$ and $\mathcal{E}$-$\mathcal{T}_2$ collapse exactly if $\mathcal{E}$-$\mathcal{T}_1 = \mathcal{E}$-$\mathcal{T}_2$. Otherwise, they separate. We further refine the type of separations into three cases. The first case (called "$\subsetneq$") holds exactly if $\mathcal{E}$-$\mathcal{T}_1 \subsetneq \mathcal{E}$-$\mathcal{T}_2$, the second case (called "$\supsetneq$") holds exactly if $\mathcal{E}$-$\mathcal{T}_1 \supsetneq \mathcal{E}$-$\mathcal{T}_2$, and finally, the third case (called incomparability) holds exactly if $\mathcal{E}$-$\mathcal{T}_1 \not\subseteq \mathcal{E}$-$\mathcal{T}_2$ and $\mathcal{E}$-$\mathcal{T}_1 \not\supseteq \mathcal{E}$-$\mathcal{T}_2$.

The first two abovementioned cases are what we also call "containments" and they establish a meaningful relationship: For two electoral control types $\mathcal{E}$-$\mathcal{T}_1$ and $\mathcal{E}$-$\mathcal{T}_2$, if $\mathcal{E}$-$\mathcal{T}_1 \subsetneq \mathcal{E}$-$\mathcal{T}_2$, it follows from standard set-theoretic notions that the existence of a successful control action under control type $\mathcal{E}$-$\mathcal{T}_1$ implies the existence of a successful control action under control type $\mathcal{E}$-$\mathcal{T}_2$. As we will see, we at times prove such relationships that are not what we would intuitively expect.

The notion of incomparability can itself be further refined, but it requires us to introduce a new notion; we can describe each type of separation and collapse in terms of a functional model as follows. Fix an election system $\mathcal{E}$ and a control type $\mathcal{T}$. A reduced input is an input to a control problem—a tuple—with the focus candidate removed. Then the function $f_{\mathcal{E}\text{-}\mathcal{T}}$ is function that, when given a reduced input I', outputs the set of candidates C' such that for each $c \in C'$, the tuple formed by merging I' and c in the obvious way belongs to $\mathcal{E}$-$\mathcal{T}$. It is easy to verify that for two compatible control types $\mathcal{E}$-$\mathcal{T}_1$ and $\mathcal{E}$-$\mathcal{T}_2$, $\mathcal{E}$-$\mathcal{T}_1 \subseteq \mathcal{E}$-$\mathcal{T}_2$ if and only if for each reduced input I', $f_{\mathcal{E}\text{-}\mathcal{T}_1}(I') \subseteq f_{\mathcal{E}\text{-}\mathcal{T}_2}(I')$.

We say that two compatible control types $\mathcal{E}$-$\mathcal{T}_1$ and $\mathcal{E}$-$\mathcal{T}_2$ are strongly incomparable exactly if there is a reduced input I such that $f_{\mathcal{E}\text{-}\mathcal{T}_1}(I) - f_{\mathcal{E}\text{-}\mathcal{T}_2}(I) \neq \emptyset$ and $f_{\mathcal{E}\text{-}\mathcal{T}_2}(I) - f_{\mathcal{E}\text{-}\mathcal{T}_1}(I) \neq \emptyset$. In many of our incomparability results we in fact establish strong incomparability.

The above definitions are focused on a fixed election system, but we can also consider a "general case." We consider a collapse to hold "in the general case" if it holds for every election system, even when the votes are not linear orders.

Naturally, a separation holds in the general case if it holds for some election system. Thus to show the separation of two types, it suffices to find one election system for which the two types differ. By looking at the general case, we are able to view a different form of incomparability: two control problems $\mathcal{T}_1$ and $\mathcal{T}_2$ are weakly incomparable in the general case if there is an election system $\mathcal{E}$ such that $\mathcal{E}$-$\mathcal{T}_1 \not\subseteq \mathcal{E}$-$\mathcal{T}_2$ and an election system $\mathcal{E}'$ such that $\mathcal{E}'$-$\mathcal{T}_1 \not\subseteq \mathcal{E}'$-$\mathcal{T}_2$.

We thus have three notions of incomparability: strong incomparability, which implies incomparability, which in turn implies weak incomparability (in the general case). We note however that in our work, we never deal with weak incomparability as we are always able to show either incomparability or strong incomparability.

Let us now look at the results established in this work.

3.2 Results in the General Case

Recall that the results in the general case refers to those results that hold for every election system (such as the collapses of Hemaspaandra et al. 2020).

We prove that there are no additional collapses in the general case, but show some previously unknown containment relationships.

Theorem 3.1. *Let $\mathcal{E}$ be an election system. For each $\mathcal{T} \in \{$DC-RPC-TP-UW, DC-PC-TP-UW$\}$, $\mathcal{E}$-$\mathcal{T} \subseteq \mathcal{E}$-DC-RPC-TE-NUW.*

Proof. Let $\mathcal{T} \in \{$DC-RPC-TP-UW, DC-PC-TP-UW$\}$. We will show $\mathcal{E}$-$\mathcal{T} \subseteq \mathcal{E}$-DC-RPC-TE-NUW. Suppose $(C, V, p) \in \mathcal{E}$-$\mathcal{T}$. Let (C_1, C_2) be a partition that witnesses (C, V, p)'s membership in $\mathcal{E}$-$\mathcal{T}$. It holds that either p participates in and loses in a subelection, or p participates in and does not win uniquely in the final round. If the former holds, then the partition (C_1, C_2) suffices as p will be eliminated in a subelection and thus will not proceed to the final round. If the

latter holds, then let D denote the set of candidates present during the final round. Clearly $p \in D$. Thus the partition $(D, C - D)$ witnesses (C, V, p)'s membership in $\mathcal{E}$-DC-RPC-TE-NUW since p either loses in the first subelection (D, V) or ties in that subelection and is eliminated by the tie-handling rule. □

For every pair of control types (in the general case), if they are not equal and if they are not in a $\supsetneq/\subsetneq$ relationship per Theorem 3.1, then we prove that they are incomparable, and we do so using separation witnesses that use the plurality election system.

3.3 Results about Plurality

In a plurality election, each vote is a linear order over the set of candidates, and the election system works as follows: Each candidate gets one point for each vote where they are ranked first and the set of winners is the set of candidates with maximal score (so we allow multiple winners).

We prove (via Table A.4) that there are no additional collapses or containments in plurality, and so, each strict containment/incomparability that we prove for plurality translates directly to a new result in the general case. Indeed, recall that in the general case, to show that for two types $\mathcal{T}_1$ and $\mathcal{T}_2$, it holds that $\mathcal{T}_1 \not\subseteq \mathcal{T}_2$, it suffices to find one election system where that relationship holds; for every pair of types for which we prove such a relationship in the general case, the witnessing election system is plurality, and so in some sense, we are "cutting down the work" by proving things about the general case by only studying one simple and natural election system.

3.4 Results about Veto

In a veto election, votes are linear orders of the candidates, and for each vote where a particular candidate is not ranked last, that candidate receives a point. A winner is a candidate with the highest number of points among all the candidates (naturally, there can be multiple winners). For example, if $C = \{a, b, c\}$ and $V = \{a > b > c, c > a > b\}$, then b and c each receive one point, and a receives two points and wins.

In veto elections, we prove one new collapse, and several strict containments. Our first collapse is the following.

Theorem 3.2. Veto-DC-PV-TE-UW = Veto-DC-PV-TE-NUW.

Proof. The $\supseteq$ relationship is immediate. The approach that follows resembles closely that of Maushagen and Rothe (2018). Let $(C, V, p) \in$ Veto-DC-PV-TE-UW. If there is a partition such that p is not a winner of the two-stage election, then $(C, V, p) \in$ Veto-DC-PV-TE-NUW. Otherwise, there is a partition and $c \in C$, $c \neq p$, such that p and c are both winners of the two-stage election. Suppose $\|C\| = 2$. Then p and c are both winners of the election (C, V). In this case, consider the partition $(V, \emptyset)$. Both candidates will tie and be eliminated, in both subelections. Suppose $\|C\| \geq 3$. Then there are two distinct candidates $d, e \in C - \{p\}$. Let V_1 denote the set of votes in which e is vetoed and let V_2 be the remaining votes. Now, consider the two-stage election with partition (V_1, V_2). Since e is never vetoed in V_2, p can at best tie with e in the subelection (C, V_2). Finally, d is never vetoed in V_1 (since all votes there veto e), so p cannot be a unique winner of (C, V_1). Thus p is eliminated in both first-round elections. □

By leveraging a key fact in the proof of Theorem 3.2, namely that when there are at least three candidates, destructive control by partition of voters using the TE rule is always possible, we establish the following result.

Theorem 3.3. *For each* $\mathcal{T} \in \{$DC-PV-TP-UW*,* DC-PV-TP-NUW$\}$*,* Veto-$\mathcal{T} \subsetneq$ Veto-DC-PV-TE-NUW*.*

Proof. Let $\mathcal{T} \in \{$DC-PV-TP-UW, DC-PV-TP-NUW$\}$. Suppose $(C, V, p) \in$ Veto-$\mathcal{T}$, and let (V_1, V_2) be a partition that witnesses this membership. Using the same technique as in the proof of Theorem 3.2, we can handle the case where $\|C\| \geq 3$, and thus we only need to consider the case where $\|C\| = 2$ in this proof. Let $C = \{c, p\}$. First, suppose that p is present in the final round. Then, since p is not a unique winner of the final round, all candidates in C are present in this round. Thus p is not a unique winner of the election (C, V), and so the partition $(V, \emptyset)$ witnesses (C, V, p)'s membership in Veto-DC-PV-TE-NUW. Now, suppose that p is absent from the final round. Then p is not a winner of (C, V_1) or (C, V_2), and so p receives strictly more vetoes than c in both V_1 and V_2. Thus p receives more vetoes than c in V, so again, p is not a unique winner of (C, V). A separation witness for the strict containment can be found in Table A.6. □

The previous result involves types that are about partitioning candidates, but we find that, surprisingly, we can show containments between control types that are about partitioning different elements of the election.

Theorem 3.4. *For each* $\mathcal{T} \in \{$DC-RPC-TE-NUW*,* DC-RPC-TP-UW*,* DC-RPC-TP-NUW*,* DC-PC-TP-UW$\}$*,* Veto-$\mathcal{T} \subsetneq$ Veto-DC-PV-TE-NUW*.*

Proof. Let $\mathcal{T} \in \{$DC-RPC-TE-NUW, DC-RPC-TP-UW, DC-RPC-TP-NUW, DC-PC-TP-UW$\}$. Suppose $(C, V, p) \in$ Veto-$\mathcal{T}$, and let (C_1, C_2) be a partition that witnesses this membership. As per the two preceding proofs (namely, of Theorems 3.2 and 3.3), we need only consider the case where $\|C\| = 2$. The case where p is present in the final round is handled as in Theorem 3.3, so suppose that p is absent from the final round. Without loss of generality, we may assume that $p \in C_1$. Then p is not a unique winner of (C_1, V), so $C_1 = C$. Thus p is not a

unique winner of (C, V), and the partition $(V, \emptyset)$ witnesses (C, V, p)'s membership in Veto-DC-PV-TE-NUW. A separation witness for the strict containment can be found in Table A.6. □

For every remaining pair of control types, we prove that they are incomparable.

3.5 Results about Approval

Unlike plurality and veto, approval uses a different type of votes. In an approval election (C, V), each vote is a $\|C\|$-bit vector, where each bit is associated with a candidate. A candidate is approved by a vote exactly if that candidate's bit is set to 1 in that vote. In an approval election, a candidate receives one point for each vote in which they are approved, and the set of winners is the set of candidates with maximal score.

Since we note in our paper (Carleton et al., 2023b) that the proof of Hemaspaandra et al. (2020) does not rely on vote types, their collapses apply to approval.

Corollary 3.5 (see Hemaspaandra et al. 2020)**.** *1. The following control types pairwise collapse:* Approval-DC-RPC-TE-UW*,* Approval-DC-RPC-TE-NUW*,* Approval-DC-PC-TE-NUW*, and* Approval-DC-PC-TE-UW*.*

2. Approval-DC-RPC-TP-NUW = Approval-DC-PC-TP-NUW*.*

We first discuss our results pertaining to axiomatic sufficient conditions. Let us first consider Property α, which states that p winning an election (C, V) implies that p remains a winner of every election (C', V) for which $p \in C' \subseteq C$ (Samuelson, 1938).[11] The "unique" version of this axiom (Property Unique-α) only differs in

[11]Readers familiar with the computational social choice literature will see this definition as being that of the Weak Axiom of Revealed Preferences (WARP). However, that is incorrect; the

that it requires p to be a unique winner, i.e., it states that p uniquely winning in an election (C, V) implies that p uniquely wins in each election (C', V) for which $p \in C' \subseteq C$. Hemaspaandra et al. (2007) prove interesting results relating to destructive control for those election systems that satisfy (Property) Unique-α, and so we draw on them to prove new relationships between electoral control types.

So far, all our nontrivial results have been about control by partition. We show that for election systems satisfying either Property α or satisfying Unique-α, we can relate types that are about deleting candidates and deleting voters. In the proof of Theorem 3.6, as in the proof of other theorems, we rely of the notion of immunity.

Theorem 3.6. *Let $\mathcal{E}$ be an election system that satisfies Unique-α. Then the following hold.*

1. $\mathcal{E}$-DC-DC-UW $\subseteq$ $\mathcal{E}$-DC-DV-UW.
2. $\mathcal{E}$-DC-DC-NUW $\subseteq$ $\mathcal{E}$-DC-DV-UW.

Proof. Fix any $\mathcal{E}$ that satisfies Property Unique-α.

---

seminal work on control (Bartholdi et al., 1992) introduced the notion captured by Property α in an ambiguous way that led the field to believe that they were giving the definition of WARP. This does not invalidate past results however, as they were all consistent with their definition of WARP. Rather, this merely indicates a naming that we believe—and as we've detailed in our technical reports (Carleton et al., 2022a,b)—should be revisited to be more consistent with the social choice literature's definitions of Property α and WARP. Relatedly, we now dub what was once known in computational social choice as "Unique-WARP" as "Property Unique-α" (and omitting the "Property" for conciseness when the context is clear). For a more thorough discussion on the relationship between WARP and Property α, both in computational social choice and in choice social theory, we refer the readers to our technical report "Search versus Search for Collapsing Electoral Control Types" (Carleton et al., 2022a).

1. The proof is analogous to that of Theorem 3.8, which we give later. Since $\mathcal{E}$ is immune to destructive control by deleting candidates in the unique-winner model, the inclusion is immediate as $\mathcal{E}$-DC-DC-UW consists of those inputs where the focus candidate is already not a unique winner.

2. It trivially holds that $\mathcal{E}$-DC-DC-NUW $\subseteq$ $\mathcal{E}$-DC-DC-UW. Thus the proof follows from the previous part of this theorem. □

Theorem 3.7. *Let $\mathcal{E}$ be an election system that satisfies Property α. Then $\mathcal{E}$-DC-DC-NUW $\subseteq$ $\mathcal{E}$-DC-DV-NUW.*

Proof. Fix any $\mathcal{E}$ that satisfies Property α. The proof is analogous to that of Theorem 3.6. Given an election (C, V) and $p \in C$, if p is a winner of $\mathcal{E}$ election (C, V), by Property α it follows that no deletion of candidates (other than p) can prevent p from being a winner. Thus $\mathcal{E}$-DC-DC-NUW consists of those inputs where the focus candidate is already not a winner. □

Beyond those, all the results we mention in text are about control by partition (of voters or candidates).

The next result extends the four-type collapse of Hemaspaandra et al. (2020) into a five-type collapse.

Theorem 3.8. *Let $\mathcal{E}$ be an election system that satisfies Unique-α. Then $\mathcal{E}$-DC-PC-TP-UW = $\mathcal{E}$-DC-PC-TE-UW.*

Proof. Fix any $\mathcal{E}$ that satisfies Property Unique-α. Let $\mathcal{E}$-$\mathcal{T}_1$ = $\mathcal{E}$-DC-PC-TP-UW and let $\mathcal{E}$-$\mathcal{T}_2$ = $\mathcal{E}$-DC-PC-TE-UW. Consider the two sets $A_\mathcal{E} = \{(C, V, p) \mid p \in C$ and p is a unique winner of $\mathcal{E}$ election $(C, V)\}$ and $B_\mathcal{E} = \{(C, V, p) \mid p \in C$ and p is not a unique winner of $\mathcal{E}$ election $(C, V)\}$. These sets form a partition of $Y = \{(C, V, p) \mid p \in C\}$, so of course $Y = A_\mathcal{E} \cup B_\mathcal{E}$. Clearly we also have that $\mathcal{E}$-$\mathcal{T}_1 \subseteq Y$ and that $\mathcal{E}$-$\mathcal{T}_2 \subseteq Y$. We will argue that $\mathcal{E}$-$\mathcal{T}_1 = B_\mathcal{E} = \mathcal{E}$-$\mathcal{T}_2$. Since $\mathcal{E}$,

like all systems satisfying Property Unique-α, is immune to both control types in the theorem statement, we have (recall that both of these types are destructive types) that $\mathcal{E}$-$\mathcal{T}_1 \cap A_{\mathcal{E}} = \emptyset$ and $\mathcal{E}$-$\mathcal{T}_2 \cap A_{\mathcal{E}} = \emptyset$, and thus it holds that $\mathcal{E}$-$\mathcal{T}_1 \subseteq B_{\mathcal{E}}$ and $\mathcal{E}$-$\mathcal{T}_2 \subseteq B_{\mathcal{E}}$. Fix $(C, V, p) \in B_{\mathcal{E}}$. Then the partition $(\emptyset, C)$ witnesses both $(C, V, p) \in \mathcal{E}$-$\mathcal{T}_1$ and $(C, V, p) \in \mathcal{E}$-$\mathcal{T}_2$, since in both cases the final round will simply be (C, V), and we know that since $(C, V, p) \in B_{\mathcal{E}}$, p will not be a unique winner of the final round. □

And we leverage Unique-α to show our first collapse that is about constructive control by showing that under any election system satisfying Unique-α, the only way to make a candidate a unique winner under those types of control is if the candidate is already a unique winner in the original election.

Theorem 3.9. *Let $\mathcal{E}$ be an election system that satisfies Unique-α. Then $\mathcal{E}$-CC-PC-TP-UW = $\mathcal{E}$-CC-RPC-TP-UW.*

Proof. We use an argument similar to that of Theorem 3.8. Fix any $\mathcal{E}$ that satisfies Property Unique-α. Let $\mathcal{E}$-$\mathcal{T}_1$ = $\mathcal{E}$-CC-PC-TP-UW and let $\mathcal{E}$-$\mathcal{T}_2$ = $\mathcal{E}$-CC-RPC-TP-UW. Consider the two sets $A_{\mathcal{E}} = \{(C, V, p) \mid p \in C$ and p is a unique winner of $\mathcal{E}$ election $(C, V)\}$ and $B_{\mathcal{E}} = \{(C, V, p) \mid p \in C$ and p is not a unique winner of $\mathcal{E}$ election $(C, V)\}$. These sets form a partition of $Y = \{(C, V, p) \mid p \in C\}$, so of course $Y = A_{\mathcal{E}} \cup B_{\mathcal{E}}$. Clearly we also have that $\mathcal{E}$-$\mathcal{T}_1 \subseteq Y$ and that $\mathcal{E}$-$\mathcal{T}_2 \subseteq Y$. We will show that $\mathcal{E}$-$\mathcal{T}_1 = A_{\mathcal{E}} = \mathcal{E}$-$\mathcal{T}_2$. Hemaspaandra et al. (2007) show that any election system that satisfies Property Unique-α is, under the TP tie-handling rule in the unique-winner model, immune to constructive control by both run-off partition of candidates and partition of candidates. Thus $\mathcal{E}$-$\mathcal{T}_1 \cap B_{\mathcal{E}} = \emptyset$ and $\mathcal{E}$-$\mathcal{T}_2 \cap B_{\mathcal{E}} = \emptyset$, and it holds that $\mathcal{E}$-$\mathcal{T}_1 \subseteq A_{\mathcal{E}}$ and $\mathcal{E}$-$\mathcal{T}_2 \subseteq A_{\mathcal{E}}$. Fix $(C, V, p) \in A_{\mathcal{E}}$. Then the partition $(\emptyset, C)$ witnesses that $(C, V, p) \in \mathcal{E}$-$\mathcal{T}_1$ as the final round will simply be (C, V), and we know that since $(C, V, p) \in A_{\mathcal{E}}$, p will be the unique winner of the final round. Additionally, the partition $(\emptyset, C)$ also witnesses that

$(C, V, p) \in \mathcal{E}\text{-}\mathcal{T}_2$ as no one will proceed from subelection $(\emptyset, V)$, and only p will proceed from subelection (C, V) (since $(C, V, p) \in A_{\mathcal{E}}$), and p must be the unique winner of $(\{p\}, V)$, the final round (since $\mathcal{E}$ satisfies Property Unique-α and p is the unique winner of $\mathcal{E}$ election (C, V)). □

As a corollary to the methods used in the proof of the above theorem, we get the following result.

Corollary 3.10. *Let $\mathcal{E}$ be an election system that satisfies Unique-α. Then, for each $\mathcal{T} \in$ {CC-PC-TE-UW, CC-PC-TE-NUW, CC-RPC-TE-UW, CC-RPC-TE-NUW, CC-PV-TE-UW, CC-PV-TE-NUW, CC-PV-TP-UW, CC-PV-TP-NUW}, it holds that $\mathcal{E}$-CC-PC-TP-UW $\subseteq \mathcal{E}\text{-}\mathcal{T}$ (equivalently, $\mathcal{E}$-CC-RPC-TP-UW $\subseteq \mathcal{E}\text{-}\mathcal{T}$).*

Proof. Fix any $\mathcal{E}$ that satisfies Property Unique-α, fix a $\mathcal{T} \in$ {CC-PC-TE-UW, CC-PC-TE-NUW, CC-RPC-TE-UW, CC-RPC-TE-NUW, CC-PV-TE-UW, CC-PV-TE-NUW, CC-PV-TP-UW, CC-PV-TP-NUW}, and fix $(C, V, p) \in \mathcal{E}$-CC-PC-TP-UW. Since $\mathcal{E}$-CC-PC-TP-UW consists of only those inputs where p is already a unique winner of the $\mathcal{E}$ election (C, V), the trivial partition (i.e., $(\emptyset, C)$ if $\mathcal{T}$ is about partitioning candidates and $(\emptyset, V)$ if $\mathcal{T}$ is about partitioning votes) is a witness that $(C, V, p) \in \mathcal{E}\text{-}\mathcal{T}$ as p will win at least one subelection in which it participates (if $\mathcal{T}$ is about partitioning votes, then p participates in two subelections) and wins uniquely (because $\mathcal{E}$ satisfies Property Unique-α). The "equivalently" follows from Theorem 3.9. □

Of course, since approval clearly satisfies both Property α and Unique-α, all the abovementioned results that are about these two axioms also hold for approval. In the case where we only prove a containment relationship, we note that the containment is strict under approval, and we include the separation witnesses in Table A.8.

Corollary 3.11. *Under approval, the following control-type relationships hold.*

1. DC-PC-TP-UW = DC-PC-TE-UW = DC-RPC-TE-UW = DC-RPC-TE-NUW = DC-PC-TE-NUW.
2. DC-RPC-TP-NUW = DC-PC-TP-NUW.
3. DC-DC-UW $\subsetneq$ DC-DV-UW.
4. DC-DC-NUW $\subsetneq$ DC-DV-NUW.
5. DC-DC-NUW $\subsetneq$ DC-DV-UW.
6. CC-PC-TP-UW = CC-RPC-TP-UW.
7. *For each* $\mathcal{T} \in$ {CC-PC-TE-UW, CC-PC-TE-NUW, CC-RPC-TE-UW, CC-RPC-TE-NUW, CC-PV-TE-UW, CC-PV-TE-NUW, CC-PV-TP-UW, CC-PV-TP-NUW}, *it holds that* CC-PC-TP-UW $\subsetneq \mathcal{T}$.

We now focus on approval-specific arguments. First, we extend the five-type collapse above to a six-type collapse.

Theorem 3.12. Approval-DC-RPC-TP-UW = Approval-DC-PC-TP-UW.

Proof. We will show that both sets coincide with the set $B = \{(C, V, p) \mid p \in C$ and p is not a unique winner of approval election $(C, V)\}$.

It follows from the fact that approval is immune with respect to DC-RPC-TP-UW and DC-PC-TP-UW (Hemaspaandra et al., 2007) that Approval-DC-RPC-TP-UW and Approval-DC-PC-TP-UW are both subsets of B. Fix $(C, V, p) \in B$. Then the partition $(\emptyset, C)$ witnesses that $(C, V, p) \in$ Approval-DC-PC-TP-UW as the final stage will be approval election (C, V). Similarly, the partition $(\emptyset, C)$ also witnesses that $(C, V, p) \in$ Approval-DC-RPC-TP-UW. This follows from the fact that p is not a unique winner of the approval election (C, V) and so there must

exist a different candidate d who is a winner of election (C, V) and so whose score is at least as large as p's score. No one will proceed from subelection $(\emptyset, V)$ and d will proceed from subelection (C, V) (along with those candidates, if any, who tie with d) and be a winner of the final round. Thus even if p proceeds to the final round, it will not be a unique winner. □

To help us prove the next collapse, we will again appeal to the notion of immunity, but this time, we must prove a new relationship that holds in the nonunique-winner model (see Hemaspaandra et al. 2007 for the analogous result in the unique-winner model).

Theorem 3.13. *Approval is immune to constructive control by partition of candidates and run-off partition of candidates under the* TP *tie-handling rule in the nonunique-winner model.*

Proof. We first show that a winner (possibly tied) of the two-stage approval election induced by partition of candidates under the TP tie-handling rule must be a winner without any control action (i.e., in the input's election). Fix an election (C, V) and let $p \in C$ be a candidate such that $(C, V, p) \in$ Approval-CC-PC-TP-NUW. Thus it holds that p's score is at least as high as the score of the other candidates present in the final round. Since no candidate is eliminated by the tie-handling rule, each candidate that is eliminated in the first round has score strictly less than some candidate that is present in the final round. It follows that p's score is at least as high as the score of every other candidate in C. Thus p is a winner of the approval election (C, V).

Essentially the same argument line can be used to show that for each $(C, V, p) \in$ Approval-CC-RPC-TP-NUW, it holds that p is a winner of approval election (C, V). □

We can now prove the following result.

Theorem 3.14. Approval-CC-PC-TP-NUW = Approval-CC-RPC-TP-NUW.

Proof. We will show that Approval-CC-PC-TP-NUW and Approval-CC-RPC-TP-NUW are comprised of exactly those inputs where the focus candidate is already a nonunique winner, following the approach in the proofs of Theorems 3.8 and 3.9.

Consider the following two disjoint sets: $A = \{(C, V, p) \mid p \in C$ and p is a winner of approval election $(C, V)\}$ and $B = \{(C, V, p) \mid p \in C$ and p is not a winner of approval election $(C, V)\}$. It's clear that Approval-CC-PC-TP-NUW $\subseteq (A \cup B)$ and Approval-CC-RPC-TP-NUW $\subseteq (A \cup B)$. By Theorem 3.13, it holds that approval is immune to constructive control by partition of candidates and run-off partition of candidates under the TP tie-handling rule in the nonunique-winner model. Thus Approval-CC-PC-TP-NUW $\cap B = \emptyset$ and Approval-CC-RPC-TP-NUW $\cap B = \emptyset$, and it holds that Approval-CC-PC-TP-NUW $\subseteq A$ and Approval-CC-RPC-TP-NUW $\subseteq A$. Fix $(C, V, p) \in A$. Then the partition $(\emptyset, C)$ witnesses that both $(C, V, p) \in$ Approval-CC-PC-TP-NUW and $(C, V, p) \in$ Approval-CC-RPC-TP-NUW as no candidate will have score higher than p's score (because p is a winner of approval election (C, V)), and since the tie-handling rule does not eliminate candidates, p will always proceed from the subelection it participates in to the final round. Thus no candidate can defeat p in the final round, making p a winner of the two-stage election. □

The above theorem is proved by showing that for both types of control, the only way to succeed is if the distinguished candidate is already a winner (possibly tied) in the original election. This allows us to prove the following corollary.

Corollary 3.15. *For each* $\mathcal{T} \in$ {CC-PC-TE-NUW, CC-RPC-TE-NUW, CC-PV-TP-NUW}*, it holds that* Approval-CC-PC-TP-NUW $\subsetneq$ Approval-$\mathcal{T}$.

Proof. Fix $\mathcal{T} \in$ {CC-PC-TE-NUW, CC-RPC-TE-NUW, CC-PV-TP-NUW} and fix $(C, V, p) \in$ Approval-CC-PC-TP-NUW. By the proof of Theorem 3.14, we

know that Approval-CC-PC-TP-NUW consists of only those inputs where p is already a nonunique winner of the approval election (C, V). In the first case, where $\mathcal{T}$ is about partitioning candidates, then the partition $(\{p\}, C - \{p\})$ is a witness that $(C, V, p) \in$ Approval-$\mathcal{T}$ as p will uniquely win the subelection it participates in and proceed to the final round. Regardless of which candidates proceed to the final round from the other subelection, they will not have score greater than p's (or they would have defeated p in the (C, V) election) and so p is a winner of the two-stage election. In the second case, where $\mathcal{T}$ is about partitioning votes (i.e., $\mathcal{T} =$ CC-PV-TP-NUW), then the partition $(\emptyset, V)$ suffices as everyone proceeds to the final round (because all the candidates tie in the subelection $(C, \emptyset)$ and the tie-handling rule does not eliminate candidates) and since p is already a winner of the approval election (C, V), they are a winner of the final round. The corresponding separation witness can be found in Table A.8. □

We now use direct arguments to complete our remaining proofs.

A similar relationship to the one below also holds under veto, but we use a different approach to prove this collapse.

Theorem 3.16. Approval-DC-PV-TE-UW = Approval-DC-PV-TE-NUW.

Proof. The $\supseteq$ relationship is immediate.

$\subseteq$: Let $(C, V, p) \in$ Approval-DC-PV-TE-UW and let (V_1, V_2) be a vote partition that witnesses this membership. As the tie-handling rule is the TE rule, it must hold (since if p uniquely won both subelections it necessarily would uniquely win the second-round election) that either p is eliminated in both subelections (either by tieing or by losing) or p uniquely wins one subelection, some other candidate d uniquely wins the other subelection, and p does not uniquely win the final round (thus d's score must be at least as large as p's score when using vote set V). This also tells us that $\|C\| \geq 2$. In the first case, the partition (V_1, V_2) witnesses $(C, V, p) \in$ DC-PV-TE-NUW since p will not proceed to the final round. In the

second case, using the partition $(V, \emptyset)$ suffices for the following reasons. This is because in the subelection (C, V), since d's score is at least as large as that of p, either d uniquely wins the subelection or both p and d tie and are eliminated by the tie-handling rule, and in the subelection $(C, \emptyset)$, every candidate ties and so all candidates are eliminated by the tie-handling rule (since $\|C\| \geq 2$). Thus p is eliminated in both subelections, does not proceed to the final round, and so is not a final-round winner. □

The following collapse, which is about constructive control, is also proven using direct arguments. The actual proof of Theorem 3.17 is more involved than necessary, but it allows us to directly establish the result in Corollary 3.18.

Theorem 3.17. Approval-CC-PC-TE-NUW = Approval-CC-RPC-TE-NUW.

Proof. We structure this proof so as to yield not just this theorem but also the related result that we state as Corollary 3.18.

$\subseteq$: Let $(C, V, p) \in$ Approval-CC-PC-TE-NUW and let (C_1, C_2) be a candidate partition that witnesses this membership. Consider the case where p uniquely wins the final round. If $p \in C_1$, then p uniquely wins (C_1, V) and in the final round defeats all candidates in C_2. If $p \in C_2$, then in the final round p defeats the candidate (if any) that survives the TE tie-handling rule regarding the subelection (C_1, V) as well as all the candidates in $C_2 - \{p\}$. Regardless of which case holds, the partition (C_1, C_2) will witness $(C, V, p) \in$ Approval-CC-RPC-TE-NUW since p will uniquely win its subelection and then will defeat any candidate that moves forward from the other subelection. If p does not uniquely win the final round, then there is at least one other candidate that ties with p in the final round. We consider two cases. If $p \in C_1$, then p must uniquely win in (C_1, V) and as (since (C_1, C_2) witnesses $(C, V, p) \in$ Approval-CC-PC-TE-NUW) no candidate in C_2 can have a score greater than p's, the partition (C_1, C_2) suffices to witness $(C, V, p) \in$ Approval-CC-RPC-TE-NUW. If $p \in C_2$, then under partition (C_1, C_2)

p could first-round tie with a candidate and be eliminated (under run-off partition of candidates due to the TE rule). However, let T denote the (possibly empty) set of candidates (other than p) that tie with p in (C_2, V). Then the partition $(C_1 \cup T, C_2 - T)$ witnesses $(C, V, p) \in$ Approval-CC-RPC-TE-NUW, since p will uniquely win $(C_2 - T, V)$ and will either tie or defeat the winner (if any) of $(C_1 \cup T, V)$.

$\supseteq$: Let $(C, V, p) \in$ Approval-CC-RPC-TE-NUW and let (C_1, C_2) be a candidate partition that witnesses this membership. Without loss of generality, assume that $p \in C_1$. Thus it holds that p uniquely wins (C_1, V). If p uniquely wins the final round, then p also defeats the candidate (if any) that moves forward from (C_2, V). Thus the partition (C_2, C_1) will also witness $(C, V, p) \in$ Approval-CC-PC-TE-NUW (since p has strictly more approvals than any candidate other than itself). If p does not uniquely win the final round, then there is another candidate d, who is the unique winner of (C_2, V) and ties with p in the final round. Again the partition (C_2, C_1) suffices to witness $(C, V, p) \in$ Approval-CC-PC-TE-NUW since d proceeds to the final round and both p and d win there due to their numbers of approvals. □

Corollary 3.18. Approval-CC-PC-TE-UW = Approval-CC-RPC-TE-UW.

Proof. This is an immediate corollary to the above proof of Theorem 3.17, as the proof was intentionally structured to ensure that if the witness of one type made p a unique winner of the final round, then the constructed-above (sometimes different) partition for the other type also made p a unique winner of the final round under that other type of control. □

The following containment is not too surprising, and involves two types that are both about partitioning voters.

Theorem 3.19. Approval-DC-PV-TP-UW $\subsetneq$ Approval-DC-PV-TE-NUW.

Proof. The proof is similar in flavor to that of Theorem 3.16. Let $(C, V, p) \in$ Approval-DC-PV-TP-UW and let (V_1, V_2) be a vote partition that witnesses this membership. There are two cases to consider. Either p uniquely wins in exactly one subelection (it certainly cannot uniquely win in both, else p would be the unique final-round winner), or p is not a unique winner in either subelection. In the latter case, the partition (V_1, V_2) witnesses that $(C, V, p) \in$ Approval-DC-PV-TE-NUW as p never survives the TE tie-handling rule (because it is not a unique winner in either subelection). In the former case, the partition $(V, \emptyset)$ witnesses that $(C, V, p) \in$ Approval-DC-PV-TE-NUW. Why does this hold? In that case, there must exist a candidate $d \neq p$ that in the Approval-DC-PV-TP-UW setting under partition (V_1, V_2) proceeds to the final stage and in the second round has a score at least as high as that of p. So in the Approval-DC-PV-TE-NUW setting's first round under the partition $(V, \emptyset)$, in the subelection (C, V) candidate p can at best tie d, and thus certainly cannot move forward under the TE tie-handling rule. Additionally, no one moves forward from subelection $(C, \emptyset)$ (everyone ties and, since in the current case we know there are at least two candidates, everyone is eliminated from that subelection). This shows the containment. The separation witness for the strict containment can be found in Table A.8. □

However, we find the following two results are more surprising as they are about partitioning different elements of the election, and so one would, intuitively, expect an incomparability result here.

Theorem 3.20. Approval-DC-RPC-TE-NUW $\subsetneq$ Approval-DC-PV-TP-UW.

Proof. Let $(C, V, p) \in$ Approval-DC-RPC-TE-NUW and let (C_1, C_2) be a candidate partition that witnesses this membership. Then, with respect to that partition, p is eliminated either in its subelection (either by tieing or by losing) or in the final round. In both cases, this must happen because there is another candidate d such that p's score is at most d's score (using vote set V). By using

the partition $(\emptyset, V)$, we know that every candidate will proceed to the final round (everyone ties in $(C, \emptyset)$ and proceeds to the final round) and in that final round, either d will tie p or d will defeat p. In either case, p is not a unique winner. This shows the containment. The separation witness for the strict containment can be found in Table A.8. □

As a corollary to the above two theorems, we obtain the easy-to-see result below.

Corollary 3.21. Approval-DC-RPC-TE-NUW $\subsetneq$ Approval-DC-PV-TE-NUW.

3.6 Conclusion and Open Directions

Our work thus establishes which pairs of electoral control types collapse (1) under every election system, (2) under plurality, (3) under veto, and (4) under approval. Additionally, for the remaining pairs that do not collapse, we prove whether they are incomparable.

We also give new axiomatic sufficient conditions that imply collapses. An interesting open direction would be to further explore this axiomatic sufficient approach. Being able to fully characterize collapses (in the sense of this work) seems to require a deeper understanding of control and election systems, and how they relate to axioms.

It would also be interesting to study the separations and collapses that hold for other popular and important election systems, and we have work in progress in that direction. Finally, we mention that just because control types are equal as sets in the decision model does not mean that they are "equivalent" in the search model. The early work of Borodin and Demers (1976) gives a concrete case where search and decision "rip apart" under reasonable assumptions. We explore that direction in Chapter 4.

4 Search versus Search for Collapsing Electoral Control Types

A [relationship] is like a pointillist painting. In order to see it in its entirety, you have to take a step back. – Byakuya Kuchiki (Kubo, 2015)

4.1 Introduction

Chapter 3 covers collapsing control types and other definitions that are not in Chapter 2 but are relevant to the current section. In this chapter, we only consider those control problems that are about partitioning (voters or candidates), as our results on collapsing control types only involve partition-based types.

Our study of collapsing control types, although interesting, is restricted to the so-called decision model, i.e., we study electoral control problems as decision problems, which is the norm in both theoretical computer science and computational social choice. However, in the real world, people are more concerned with understanding the search versions of problems rather than their decision counterparts. More precisely, the decision model only tells us whether a successful control action

exists, but in practice, it is far more desirable to know what the control action is. It is often implicitly assumed that results in the decision model have a natural translation to the search model, making a problem either "easy" in both models or "hard" in both models. Unfortunately, the gap from decision to search is not a negligible one as there are problems that are known to be "easy" in the decision model, but "hard" in the search model (under reasonable assumptions).

To our knowledge, Borodin and Demers (1976) were the first to show that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then there exists a set of boolean formulas that is in P and yet no polynomial-time algorithm can compute a satisfying assignment of an arbitrary formula from that set. Informally, this means that there is language in P having a search variant that is not polynomial-time computable. The assumption is certainly reasonable; the decision complexity of integer factorization is known to be in $\mathrm{NP} \cap \mathrm{coNP}$ and so as to not violate our expectation of modern cryptosystems being secure, we also expect—or at the very least hope—that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$. We briefly mention below some other papers that leverage the above result of Borodin and Demers (1976). Bellare and Goldwasser (1994) studied search in the context of cryptography, which is a natural place to see the assumption that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$. Hemaspaandra and Narváez (2017) studied the complexity of finding so-called nontrivial backbones in boolean formulas. And finally, Hemaspaandra et al. (2020), whose work we centrally rely on in the study of separations and collapses of electoral control types, show that there are election systems for which certain electoral attack problems have polynomial-time decision complexity, and yet their search counterparts cannot have polynomial-time complexity.

The decision model lends itself naturally to comparing decision problems as one can simply use basic set-theoretic notions, as we do. When it comes to search problems however, the way to study their relationships is not as obvious. Megiddo and Papadimitriou (1991) had originally given a framework to interrelate the complexity of search problems, but their general method introduced a loophole that

could be exploited to "interreduce" (under some notion specific to their framework) even SAT and the empty set! The flaw was subtle, and so we redefined some of their notions, while also correcting the error in their paper.

When dealing with these search variants of electoral control problems, we take a slight departure from our standard notation. In this chapter, we allow a control type's name to also include the election system. For example, we will say "let $\mathcal{T}$ = $\mathcal{E}$-CC-PC-TE-UW" rather than have the "$\mathcal{E}$" exist outside the "$\mathcal{T}$". In doing so, we now lose the ability to identify when two control types are about the same election system, and so we introduce some additional terminology. To avoid confusion with our definitions from Chapter 3, we present the definitions that are unique to this chapter within the chapter itself.

If two control type $\mathcal{T}_1$ and $\mathcal{T}_2$ are compatible (i.e., their input fields are the same), then we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ collapse if $\mathcal{T}_1 = \mathcal{T}_2$ (as decision problems). If the two control types are about the same election system $\mathcal{E}$, and they are compatible, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are $\mathcal{E}$-matched. Finally, if $\mathcal{T}_1$ and $\mathcal{T}_2$ are about the same election system $\mathcal{E}$, and they collapse, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are a collapsing pair.

4.2 Search Notions for Control Types: Reducibility and Complexity

In this section, we discuss several notions about search problems in the context of electoral control, but those notions naturally generalize to other settings.

For a given election system $\mathcal{E}$, the winner problem is given as

$$W_{\mathcal{E}} = \{(C, V, p) \mid p \in C \text{ and } p \text{ is a winner of the } \mathcal{E} \text{ election } (C, V)\}.$$

Let us now define what we mean by search variants/problems of control problems. Let $\mathcal{E}$ be an election system. Each control problem about $\mathcal{E}$ has an associated language, i.e., the set of inputs on which the given control action succeeds. Naturally

associated with each of those languages is an "$\mathrm{NP}^{W_\mathcal{E}}$ search problem": On a valid input, guess a control action under the given control type and output it if it is a successful control action under the given control type.

We will first discuss the notions relating the reducibilities of search problems as they are used in this work, and we will then discuss the complexity of search problems.

4.2.1 Search Reducibilities

We will discuss below a framework to interrelate the complexity of search problems. Namely, we will leverage the framework of Megiddo and Papadimitriou (1991), except that we will further restrict the way problems can interrelate, allow our reductions to have access to an oracle so as to have results that hold even for election systems whose winner problems are not polynomial-time computable, and we will fix a subtle loophole in the framework of Megiddo and Papadimitriou (1991).

Consistent with the work of Megiddo and Papadimitriou (1991), let Σ be a finite alphabet with at least two characters, and let $R \subseteq \Sigma^* \times \Sigma^*$ be a relation that is decidable in polynomial time relative to oracle A and polynomially-balanced, i.e., there is a polynomial p such that $(x, y) \in R$ implies $|y| \leq p(|x|)$. Without the "polynomially-balanced" requirement solutions could be exponentially long and thus not polynomial-time computable, regardless of the oracle. The relation R describes a search problem Π_R where the task is to, given $x \in \Sigma^*$, find a $y \in \Sigma^*$ such that $(x, y) \in R$. If no such y exists, then output "no," and that relation is decidable in P^A. The class of such search problems is called FNP^A; this departs from the notion of Megiddo and Papadimitriou (1991), who are only concerned with the class FNP—that is, the case where $A = \emptyset$—but remains consistent with their nomenclature.

Going back to our control types, the "x" refers to the input to the control type. In the case of partition-based control types, the inputs are of the form (C, V, p), and y is a partition of either the candidate set or the vote set, depending on the nature of the control type under consideration.

Let $\mathcal{E}$ be an election system and let $\mathcal{T}$ be a partition-based control type that is about $\mathcal{E}$. The relation $R_{\mathcal{T}}$ is the natural $\mathrm{P}^{W_{\mathcal{E}}}$ polynomially-balanced relation for the search problem of $\mathcal{T}$. The resulting search problem, in the terminology of Megiddo and Papadimitriou (1991) is denoted by $\Pi_{R_{\mathcal{T}}}$, but we often omit the "R" and just write $\Pi_{\mathcal{T}}$ for simplicity. It is easy to see that $\Pi_{\mathcal{T}} \in \mathrm{FNP}^{W_{\mathcal{E}}}$, e.g., $\Pi_{\mathcal{E}\text{-DC-PC-TP-NUW}} \in \mathrm{FNP}^{W_{\mathcal{E}}}$, and since approval's winner problem is in P, the search problem $\Pi_{\text{Approval-DC-PC-TP-NUW}}$ is in FNP.

While we showed that collapsing control types are equal in the decision model, and thus have the same decision complexity, we wish to show that they are also "equal" under some notion of equivalence in the search model. We leverage a type of reduction defined by Megiddo and Papadimitriou (1991) to give a framework for interrelating the search complexities of collapsing electoral control types: Given a solution from one problem on a given input, we will construct a solution for the other problem on the same input, provided that the two problems are collapsing electoral control types that are about the same election system.

From the work of Megiddo and Papadimitriou (1991), a reduction from problem Π_R to problem Π_S is a pair of polynomial-time computable functions f and g such that for any $x \in \Sigma^*$, it holds that $(x, g(y)) \in R \iff (f(x), y) \in S$. Essentially, what this attempts to say is that given an input x to problem Π_R, we can map to $f(x)$ in polynomial time, and the solution to problem Π_S on input $f(x)$ when fed to g (which is polynomial-time computable) is the solution to problem Π_R on input x. So under that interpretation, the reduction certainly relates the complexity of problem Π_R to that of problem Π_S. But, this definition does not say what it should be saying. Indeed—even if we patch the missing quantifier

on y—consider the case where some x has a solution relative to R, $f(x)$ is just a string that has no solution relative to S, and g never maps to a solution of x relative to R. By making both sides of the " $\Longleftrightarrow$ " false, we are able to satisfy the definition, but that does not give us any meaningful way of interrelating the search problems as no "solution transfer" has occurred. So we patch this flaw in our own definitions, which also have other distinctions from the above definition.

First, we allow our reductions to have access to the winner problem of the election system under consideration, as we wish to draw meaningful relationships between the search complexity of control types even when the winner problem of the election system is not in P. Second, we restrict the reduction further by requiring that f be the identity function. This is because collapsing types are equal sets. So we are dealing with the same inputs, but with different witness schemes, and since our goal is to draw connections between those witnesses, it is most natural to restrict f to be the identity function.

We note that our definitions do not explicitly mention the "Π" formalism, but that formalism is exactly what is at the core of our definitions.

Definition 4.1. *1. For an election system $\mathcal{E}$ and $\mathcal{E}$-matched, collapsing*[12] *control types $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that "$\mathcal{T}_1$ is polynomially search-reducible to $\mathcal{T}_2$ with respect to $\mathcal{E}$" (denoted by $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$) if there is a reduction that runs in polynomial time relative to $W_{\mathcal{E}}$, and on each input (I, S) where I is an input to $\mathcal{T}_1$ and S is a solution for I with respect to $\mathcal{T}_2$, outputs a solution S' for I with respect to $\mathcal{T}_1$.*

2. For an election system $\mathcal{E}$ and $\mathcal{E}$-matched, collapsing control types $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that "$\mathcal{T}_1$ is polynomially search-equivalent to $\mathcal{T}_2$ with respect to $\mathcal{E}$" (denoted by $\mathcal{T}_1 \equiv^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$) if $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$ and $\mathcal{T}_2 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_1$.

[12]The reason we include "collapsing" in the definition is that if it is omitted, then one would trivially satisfy the notion on a given I whenever I was not in $\mathcal{T}_2$. That is, our notion is focused on pairs of types that collapse—where each input is either in both or not in both.

We note two things. The first is the intuition behind the notation. Though the reduction is from $\mathcal{T}_1$ to $\mathcal{T}_2$, the reduction expects a solution to $\mathcal{T}_2$ on the given input. This is meant to be interpreted as $\mathcal{T}_2$ being "so powerful" that a solution to it on a given input allows us to construct a solution to $\mathcal{T}_1$ given the same input. Second, we note that in the above definition, we allow our reduction to have access to the winner problem of the election, since the complexity of the latter could be not polynomial-time computable, thus not making our methods very general; indeed, for those control types that collapse regardless of the election system, translating the decision-collapse into search-collapses involves being able to evaluate the winner problem on up to three inputs (see the proof of Theorem 4.6 for an example). We now state an analogous definition wherein an oracle to the election system's winner problem is not provided.

Definition 4.2. *1. For an election system $\mathcal{E}$ and $\mathcal{E}$-matched, collapsing control types $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that "$\mathcal{T}_1$ is polynomially search-reducible to $\mathcal{T}_2$" (denoted by $\mathcal{T}_1 \leq^p_{\text{search}} \mathcal{T}_2$) if there is a reduction that runs in polynomial time and on each input (I, S), where I is an input to $\mathcal{T}_1$ and S is a solution for I with respect to $\mathcal{T}_2$, outputs a solution S' for I with respect to $\mathcal{T}_1$.*

2. For an election system $\mathcal{E}$ and $\mathcal{E}$-matched, collapsing control types $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that "$\mathcal{T}_1$ is polynomially search-equivalent to $\mathcal{T}_2$" (denoted by $\mathcal{T}_1 \equiv^p_{\text{search}} \mathcal{T}_2$) if $\mathcal{T}_1 \leq^p_{\text{search}} \mathcal{T}_2$ and $\mathcal{T}_2 \leq^p_{\text{search}} \mathcal{T}_1$.

We note that the notion closest to our Definitions 4.1 and 4.2 is that of Levin reductions. Definitions and discussions of Levin reductions are provided in Piterman and Fisman's notes from a course by Oded Goldreich (1998). Other easily accessible sources are (Arora and Barak, 2009; Henry (455), 2013). However, we note that the notion is not comparable with ours since it requires that solutions be transferred both forward and backward, whereas our reductions only transfer solutions in one direction. Furthermore, in our Definition 4.1, we make the winner

problem of the election available to the reduction machine, which does not happen in Levin reductions. We must however mention that in their 2008 textbook, Goldreich (2008) modified their definition of Levin reductions. In that case, our notions are very similar, but they differ slightly as we focus on collapsing electoral types, and so our problem-to-problem reduction is the identity function, and we sometimes make the winner problem available via an oracle.

It is easy to see that when an election system's winner problem is polynomial-time computable, i.e., in FP, the reduction machine can simply skip the oracle queries and determine the outcome of the election in polynomial time. In such a case, the reduction machine is still a polynomial-time machine. We thus have the following proposition, which is clearly true, and so we do not provide a proof for it.

Proposition 4.3. *Let $\mathcal{E}$ be an election system such that $W_{\mathcal{E}} \in \mathrm{P}$, and let $\mathcal{T}_1$ and $\mathcal{T}_2$ be $\mathcal{E}$-matched control types.*

1. *If $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$, then $\mathcal{T}_1 \leq^{p}_{\text{search}} \mathcal{T}_2$.*
2. *If $\mathcal{T}_1 \equiv^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$, then $\mathcal{T}_1 \equiv^{p}_{\text{search}} \mathcal{T}_2$.*

These above-defined notions give us a clear way to relate the complexity of search problems,[13] but they do not provide us with a nice notion of "hardness"

[13] Definitions 4.1 part 1 and 4.2 part 1 do not require that if S is not a solution to $\mathcal{T}_2$ then the reduction declares that fact. Rather, the definitions are simply about efficiently obtaining a solution to $\mathcal{T}_1$ given a solution to $\mathcal{T}_2$. However, we mention that if one changed Definition 4.1 part 1 to require detection of nonsolution-hood, the set of pairs $(\mathcal{T}_1, \mathcal{T}_2)$ for which the reduction held would not change at all, since with the $W_{\mathcal{E}}$ oracle one can check whether S is a solution to $\mathcal{T}_2$. Although Definition 4.2 part 1 is not in general guaranteed to be unchanged if it is altered to require detection of the case where S is not a solution to $\mathcal{T}_2$, it clearly does remain unchanged by that alteration whenever $W_{\mathcal{E}} \in \mathrm{P}$. Most of the cases to which we apply Definition 4.2 indeed satisfy $W_{\mathcal{E}} \in \mathrm{P}$; in particular, plurality, veto, and approval voting each satisfy $W_{\mathcal{E}} \in \mathrm{P}$.

to indicate that a polynomial-time algorithm might not exist for specific search problems. We provide this notion next, and we do so by leveraging the theory of multivalued "functions" of Book et al. (1984).

4.2.2 Search Complexities

We do not detail the theory of Book et al. (1984), but we draw only on the notion of multivalued functions in the sense below. The problem Π_R is related to a multivalued function $\widehat{\Pi}_R$ that on an input x, is undefined—or maps to some special symbol $\perp$—if x has no solution relative to Π_R and otherwise, maps to the set of all such solutions. The crucial notion we rely on is that of a "single-valued refinement (of a multivalued function $\widehat{\Pi}_R$)," which is any function that when given an input x

1. is undefined, i.e., outputs a special symbol such as $\perp$ if x has no solution relative to R (and so there is no y such that $(x, y) \in R$), and

2. maps to exactly one solution of x relative to R if x has at least one solution relative to R (and so the function maps to some y such that $(x, y) \in R$).

This notion gives us the tools we want to define what we mean by having a search problem be "easy" or "hard." Naturally, a search problem $\mathcal{T}$ is easy if it has at least one refinement (of its associated multivalued function) that is polynomial-time computable (i.e., in FP). We say that the search problem for $\mathcal{T}$ is NP-hard if for each of its refinements f, it holds that $\mathrm{NP} \subseteq \mathrm{P}^f$. We also call the search problem for $\mathcal{T}$ NP-easy if it has a refinement that can be computed in polynomial time given an oracle to SAT, i.e., the refinement is in $\mathrm{FP}^{\mathrm{NP}}$. If the search problem for $\mathcal{T}$ is both NP-easy and NP-hard, then we say that it is SAT-equivalent. In some sense, this provides an analogue to the notion of NP-completeness in the decision model as NP-hardness captures the fact that the problem is so hard that

it can be used to solve any NP problem, and NP-easiness captures the fact that the problem is easy enough that we can solve it using access to SAT.

Together with the notion of reductions we introduce, we prove the following useful propositions.

Proposition 4.4. *Let A and B be two polynomially search-equivalent problems. If A is* SAT*-equivalent, then B is* SAT*-equivalent.*

Proof. Fix two polynomially search-equivalent search problems A and B, with A being SAT-equivalent. We need to show that, per our definitions, B is both NP-hard and NP-easy.

NP-hard: Let g be an arbitrary refinement of $\widehat{\Pi}_B$. We need to show that $\mathrm{NP} \subseteq \mathrm{P}^g$. It suffices to show that $\mathrm{SAT} \in \mathrm{P}^g$.

Let r be a function that witnesses $A \leq^p_{\text{search}} B$. It follows that the function h that on arbitrary input x maps to $\perp$ if $g(x)$ maps to $\perp$, and that otherwise maps to $r(x, g(x))$, is a refinement of $\widehat{\Pi}_A$.

Since A is NP-hard, we have that $\mathrm{SAT} \in \mathrm{P}^h$. Let T denote the polynomial-time algorithm (with oracle h) that decides SAT. Here is the P^g algorithm T' for SAT. Our polynomial-time algorithm T' (with oracle g) simulates T, and each time T asks a question, y, to its oracle, T' asks the same question to its oracle g, and if the answer is $\perp$ then T' acts as if the answer T got is $\perp$, and otherwise T' acts as if the answer T got is $h(y)$. So the outcome of T' with oracle g is precisely the same as that of T with oracle h, and so we have proven that $\mathrm{SAT} \in \mathrm{P}^g$.

NP-easy: It suffices to show that $\widehat{\Pi}_B$ has a refinement in $\mathrm{FP}^{\mathrm{SAT}}$. Let s denote a function witnessing $B \leq^p_{\text{search}} A$. Since A is SAT-equivalent, there is a refinement t of $\widehat{\Pi}_A$ such that $t \in \mathrm{FP}^{\mathrm{SAT}}$. But it also holds, for each x, that $t(x)$ is a solution for x under A if and only if (t is not $\perp$ and) $s(x, t(x))$ is a solution for x with respect to B. Thus the function, h, that on input x is $\perp$ if $t(x)$ is $\perp$ and otherwise is $s(x, t(x))$ is a refinement of $\widehat{\Pi}_B$. But since $t \in \mathrm{FP}^{\mathrm{SAT}}$, clearly so is h. $\square$

Proposition 4.5. *Let A and B be two polynomially search-equivalent problems. If A is polynomial-time computable, then B is polynomial-time computable.*

Proof. Fix two polynomially search-equivalent search problems A and B, with A being polynomial-time computable. We need to show that B is polynomial-time computable, by providing a refinement of $\widehat{\Pi}_B$ in FP.

Let f be a polynomial-time computable refinement of $\widehat{\Pi}_A$ and let g be a function witnessing $B \leq^p_{\text{search}} A$. The function h that on input x is $\perp$ if $f(x)$ is $\perp$ and otherwise is $g(x, f(x))$ is clearly a polynomial-time computable refinement of $\widehat{\Pi}_B$. □

In this work, we ask whether the decision collapses hold in the search model, under the notion of polynomial search-equivalence (with respect to the winner problem of the election system). We will show that there is a tight relationship between the search complexities of all known collapsing electoral control types.

4.3 Search Equivalences for the Collapses of Hemaspaandra et al. (2020)

Our first set of results establishes that each of the decision collapses of Hemaspaandra, Hemaspaandra, and Menton (Hemaspaandra et al., 2020) holds in the search model.

We first show the result for the two-type collapse.

Theorem 4.6. *For each election system $\mathcal{E}$, $\mathcal{E}$-DC-RPC-TP-NUW $\equiv^{p,\mathcal{E}}_{\text{search}}$ $\mathcal{E}$-DC-PC-TP-NUW.*

Proof. Let $\mathcal{T}_1 = \mathcal{E}$-DC-PC-TP-NUW and $\mathcal{T}_2 = \mathcal{E}$-DC-RPC-TP-NUW.

$\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$: We give a polynomial-time algorithm performing the reduction, given access to oracle $W_{\mathcal{E}}$. On input (I, S), where $I = (C, V, p)$ and $p \in C$, do the following. If the syntax is bad, halt. Otherwise, check that S is a solution to I under $\mathcal{T}_2$. This may involve up to three calls to the oracle. If S is not a good solution, then simply halt as the definition does not require anything in this case (although note also the comments in Footnote 13). Since S is a solution and our model is NUW, p must have participated and lost either in one of the two first-round elections or (in fact, exclusive-or) in the final-round election. In that election, let C' be the candidate set (note that $p \in C'$ and p is not a winner of (C', V)). Then output as the successful $\mathcal{T}_1$ solution $C_1 = C'$ and $C_2 = C - C'$. In $\mathcal{T}_1$ on input I, p will be eliminated in the (C', V) first-round election.[14]

$\mathcal{T}_2 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_1$: We give the algorithm. On input (I, S), where $I = (C, V, p)$ and $S = (C_1, C_2)$, check for bad syntax and by using the $W_{\mathcal{E}}$ oracle up to two times as per the nature of $\mathcal{T}_1$, check that S is a solution to I under $\mathcal{T}_1$. If the syntax is bad or S is not a solution under $\mathcal{T}_1$, then simply halt. Otherwise, since S is a solution under $\mathcal{T}_1$, p was a participant in and eliminated either in the (C_1, V) contest or the final-round contest (involving the winners of (C_1, V) and the candidates in C_2). In the former case, output (C_1, C_2), and in the latter case let D be the set final-round participants and output $(D, C - D)$. This is a solution for $\mathcal{T}_2$. □

Showing the search equivalence of the elements of the four-type collapse of Hemaspaandra et al. (2020) would require quite a number of reductions, so we prove the following useful proposition to help reduce the amount of work.

[14]This direction of the proof illustrates the importance of giving the reduction access to the oracle for $W_{\mathcal{E}}$. Suppose we tried to claim that this direction held without any oracle use by, if S is (C_1, C_2), just outputting $S' = (C_1, C_2)$. But then if p participated in and lost in the (C_2, V) first-round $\mathcal{T}_2$ case, S' might not be a solution with respect to $\mathcal{T}_1$. Can we fix that by, if $p \in C_2$, just outputting (C_2, C_1)? No. Maybe in that case under $\mathcal{T}_2$ candidate p lost in (C_2, V), or maybe it lost in $(\text{Winners}_{\mathcal{E}}(C_2, V) \cup \text{Winners}_{\mathcal{E}}(C_1, V), V)$. But if it was the latter, in $\mathcal{T}_1$ our second-round election is $(\text{Winners}_{\mathcal{E}}(C_2, V) \cup C_1, V)$ and it is possible that p wins in that.

Proposition 4.7. *If $\mathcal{E}$ is an election system, $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ are pairwise $\mathcal{E}$-matched control types, $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$, and $\mathcal{T}_2 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_3$, then $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_3$. That is, for $\mathcal{E}$-matched types $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$, $\leq^{p,\mathcal{E}}_{\text{search}}$ is transitive.*

Proof. Let $\mathcal{E}$ be an election system and let $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ be pairwise $\mathcal{E}$-matched control types such that $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$ via f and $\mathcal{T}_2 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_3$ via g (with f and g both running in polynomial time given oracle $W_{\mathcal{E}}$). We will show that $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_3$. On input (I, S), where $I = (C, V, p)$, if S is a solution for $\mathcal{T}_3$ on input I, then on input $((C, V, p), S)$ g outputs a solution S' for $\mathcal{T}_2$ on input I. Additionally, if S' is a solution for $\mathcal{T}_2$ on input I, then on input $((C, V, p), S')$ f outputs a solution S'' for $\mathcal{T}_1$ on input I. Thus if S is a solution for $\mathcal{T}_3$ on input I, then by applying g and then f in the manner just described we obtain, running in polynomial time with oracle $W_{\mathcal{E}}$, a solution S'' for $\mathcal{T}_1$ on input I. Thus $\mathcal{T}_1 \leq^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_3$. □

We are now ready to prove that the second type of decision collapse of Hemaspaandra et al. (2020) also holds in the search model.

Theorem 4.8. *For every election system $\mathcal{E}$, and each pair $(\mathcal{T}_1, \mathcal{T}_2)$ among the four collapsing control types $\mathcal{E}$-*DC-RPC-TE-NUW*, $\mathcal{E}$-*DC-PC-TE-NUW*, $\mathcal{E}$-*DC-RPC-TE-UW*, and $\mathcal{E}$-*DC-PC-TE-UW*, we have that $\mathcal{T}_1 \equiv^{p,\mathcal{E}}_{\text{search}} \mathcal{T}_2$.*

Proof. Let $\mathcal{E}$ be an election system. We will make a closed cycle of $\leq^{p,\mathcal{E}}_{\text{search}}$ reductions involving these four types. In light of Proposition 4.7, this suffices to establish the theorem. In each part, as per the reduction definition, we will assume our input is (I, S), with $I = (C, V, p)$. We define the following shorthand to make some statements easier to state: For an election (C, V), let UniqueWinnerIfAny$_{\mathcal{E}}(C, V)$ denote a set containing exactly the unique winner of the $\mathcal{E}$ election (C, V) if one exists, or no element otherwise.

$\mathcal{E}$-DC-RPC-TE-UW $\leq^{p,\mathcal{E}}_{\text{search}}$ $\mathcal{E}$-DC-RPC-TE-NUW: A solution (C_1, C_2) for I under $\mathcal{E}$-DC-RPC-TE-NUW is always a solution for I under $\mathcal{E}$-DC-RPC-TE-UW,

since UW is stricter in the final round. If p was eliminated under $\mathcal{E}$-DC-RPC-TE-NUW, then p is also eliminated under $\mathcal{E}$-DC-RPC-TE-UW. So our reduction here can simply output the purported solution it is given. (Even if that is not a correct solution to the right-hand side, the reduction's action is legal, since if given a nonsolution as input all the reduction has to do, under the definition of this reduction type, is not run for too long.)

$\mathcal{E}$-DC-RPC-TE-NUW $\leq^{p,\mathcal{E}}_{\text{search}}$ $\mathcal{E}$-DC-PC-TE-NUW: Say we are for $I = (C, V, p)$ given a purported solution $S = (C_1, C_2)$ to $\mathcal{E}$-DC-PC-TE-NUW. If S is not a successful solution, immediately reject. Otherwise, either (a) $p \in C_1$ but p is not a unique winner of (C_1, V), or (b) $p \in \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup C_2$ yet p is not a winner of $(\text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup C_2, V)$. Using our oracle, determine which of (a) or (b) holds (exactly one must hold if S was a solution). If (a) holds, output (C_1, C_2). This is then a successful solution on I to $\mathcal{E}$-DC-RPC-TE-NUW. If (b) holds, then set $D = \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup C_2$ and output $(D, C - D)$ and this is a successful solution of I to $\mathcal{E}$-DC-RPC-TE-NUW, since we know that p is not a winner of (D, V), so it certainly is not a unique winner of (D, V), and so in our $\mathcal{E}$-DC-RPC-TE-NUW first round p participates and is eliminated.

$\mathcal{E}$-DC-PC-TE-NUW $\leq^{p,\mathcal{E}}_{\text{search}}$ $\mathcal{E}$-DC-PC-TE-UW: Say we are given for $I = (C, V, p)$ a purported solution $S = (C_1, C_2)$ for I to $\mathcal{E}$-DC-PC-TE-UW. If S is not a solution, immediately reject. Otherwise, we know that (a) $p \in C_1$ and p does not uniquely win in (C_1, V), exclusive-or (b) $p \in \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup C_2$ yet p is not a unique winner of $(\text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup C_2, V)$. If (a) holds, output (C_1, C_2). This is a successful solution of I for $\mathcal{E}$-DC-PC-TE-NUW as p is eliminated in the first round. If (b) holds, set $D = \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup C_2$ and output $(D, C - D)$. This is a successful solution of I for $\mathcal{E}$-DC-PC-TE-NUW as p will be eliminated in the first round.

$\mathcal{E}$-DC-PC-TE-UW $\leq^{p,\mathcal{E}}_{\text{search}}$ $\mathcal{E}$-DC-RPC-TE-UW: Say we are given $I = (C, V, p)$

and a purported solution $S = (C_1, C_2)$ for $\mathcal{E}$-DC-RPC-TE-UW for I. If S is not a solution, immediately reject. Otherwise, since C_1 and C_2 are symmetric in RPC, w.l.o.g. assume $p \in C_1$ (otherwise, if $p \in C_2$, rename C_1 and C_2 so that $p \in C_1$). So exactly one of (a) and (b) holds, where (a) and (b) are: (a) $p \in C_1$ and p is not a unique winner of (C_1, V), and (b) $p \in \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_2, V)$ and p is not a unique winner of $(\text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_2, V), V)$. If (a) holds, output (C_1, C_2) and that is a successful solution for I of $\mathcal{E}$-DC-PC-TE-UW as p is eliminated in the first round. If (b) holds, set $D = \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_1, V) \cup \text{UniqueWinnerIfAny}_{\mathcal{E}}(C_2, V)$, and output $(D, C - D)$ and that is a successful solution for I of $\mathcal{E}$-DC-PC-TE-UW as p is eliminated in the first round. □

This is indeed surprising. We know that there are election systems where search and decision separate, but what these two theorems show is that for the "general" (i.e., holding for every election system) collapses of Hemaspaandra et al. (2020), the related search complexities are intimately linked.

And as a result, we can establish the following search equivalences about plurality, veto, and approval, which follow from Proposition 4.3, and from Theorems 4.6 and 4.8.

Corollary 4.9. *For each* $\mathcal{E} \in \{\text{Plurality}, \text{Veto}, \text{Approval}\}$ *the following hold.*

1. $\mathcal{E}$-DC-RPC-TP-NUW $\equiv^{p}_{\text{search}}$ $\mathcal{E}$-DC-PC-TP-NUW.
2. *For each pair* $(\mathcal{T}_1, \mathcal{T}_2)$ *among the four collapsing types* $\mathcal{E}$-DC-RPC-TE-NUW, $\mathcal{E}$-DC-PC-TE-NUW, $\mathcal{E}$-DC-RPC-TE-UW, *and* $\mathcal{E}$-DC-PC-TE-UW, *it holds that* $\mathcal{T}_1 \equiv^{p}_{\text{search}} \mathcal{T}_2$.

4.4 Search Equivalences for the New Collapses of Chapter 3

Unlike the proofs of collapses of Hemaspaandra et al. (2020), the proofs of collapses that we provide in Chapter 3 also implicitly establish search equivalence.

Corollary 4.10 (to the collapse results of Chapter 3)**.**

1. Veto-DC-PV-TE-UW $\equiv^{p}_{\text{search}}$ Veto-DC-PV-TE-NUW.
2. *For each election system $\mathcal{E}$ that satisfies Unique-α, it holds that* $\mathcal{E}$-DC-PC-TP-UW $\equiv^{p}_{\text{search}}$ $\mathcal{E}$-DC-PC-TE-UW.[15]
3. *For each election system $\mathcal{E}$ that satisfies Unique-α, it holds that* $\mathcal{E}$-CC-PC-TP-UW $\equiv^{p}_{\text{search}}$ $\mathcal{E}$-CC-RPC-TP-UW.
4. Approval-DC-RPC-TP-UW $\equiv^{p}_{\text{search}}$ Approval-DC-PC-TP-UW.
5. Approval-CC-PC-TP-NUW $\equiv^{p}_{\text{search}}$ Approval-CC-RPC-TP-NUW.
6. Approval-DC-PV-TE-UW $\equiv^{p}_{\text{search}}$ Approval-DC-PV-TE-NUW.
7. Approval-CC-PC-TE-NUW $\equiv^{p}_{\text{search}}$ Approval-CC-RPC-TE-NUW.
8. Approval-CC-PC-TE-UW $\equiv^{p}_{\text{search}}$ Approval-CC-RPC-TE-UW.

Proof. For all parts except 2 and 3 we have $W_{\mathcal{E}} \in \mathrm{P}$ so we can check if the S of the input is a valid solution, and so in those parts we below assume that input always is a solution for $I = (C, V, p)$ of the problem on the right-hand side of the reduction. For parts 2 and 3 we cannot and do not make that assumption.

[15] One might expect here and in part 3 of this corollary the weaker conclusion "$\equiv^{p,\mathcal{E}}_{\text{search}}$." But in both these parts we mean and prove "$\equiv^{p}_{\text{search}}$."

1. Veto-DC-PV-TE-UW $\leq^p_{\text{search}}$ Veto-DC-PV-TE-NUW: A solution (C_1, C_2) for I under Veto-DC-PV-TE-NUW is always a solution for I under Veto-DC-PV-TE-UW, so we just output (C_1, C_2).

   Veto-DC-PV-TE-NUW $\leq^p_{\text{search}}$ Veto-DC-PV-TE-UW: The proof of Theorem 3.2 shows how to construct, given $I = (C, V, p)$ and a solution $S = (C_1, C_2)$ for Veto-DC-PV-TE-UW, a solution for I to Veto-DC-PV-TE-NUW, and we note that this construction can easily be done in polynomial time.

2. Let $\mathcal{E}$ be an election system that satisfies Unique-α. The proof of Theorem 3.8 shows that $\mathcal{E}$-DC-PC-TP-UW = $\mathcal{E}$-DC-PC-TE-UW = $B_{\mathcal{E}} = \{(C, V, p) \mid p \in C$ and p is not a unique winner of the $\mathcal{E}$ election $(C, V)\}$. They also show that for each $I \in B_{\mathcal{E}}$, $(\emptyset, C)$ is a solution to I for both $\mathcal{E}$-DC-PC-TP-UW and $\mathcal{E}$-DC-PC-TE-UW.

   $\mathcal{E}$-DC-PC-TP-UW $\leq^p_{\text{search}}$ $\mathcal{E}$-DC-PC-TE-UW: Let our input be (I, S). If S is a solution to I for $\mathcal{E}$-DC-PC-TE-UW, then $I \in B_{\mathcal{E}}$. So output $(\emptyset, C)$. (Note that there is no guarantee on the output if S is not a solution of I for $\mathcal{E}$-DC-PC-TE-UW and that is fine since our reduction type does not require us to make any such guarantee. This fact implicitly holds throughout the rest of this proof and so we do not mention it again.)

   $\mathcal{E}$-DC-PC-TE-UW $\leq^p_{\text{search}}$ $\mathcal{E}$-DC-PC-TP-UW: Let our input be (I, S). If S is a solution to I for $\mathcal{E}$-DC-PC-TP-UW, then $I \in B_{\mathcal{E}}$. So output $(\emptyset, C)$.

3. Let $\mathcal{E}$ be an election system that satisfies Unique-α. The proof of Theorem 3.9 shows that $\mathcal{E}$-CC-PC-TP-UW = $\mathcal{E}$-CC-RPC-TP-UW = $A_{\mathcal{E}} = \{(C, V, p) \mid p \in C$ and p is the unique winner of the $\mathcal{E}$ election $(C, V)\}$. They also show that for each $I \in A_{\mathcal{E}}$, $(\emptyset, C)$ is a solution to I for both $\mathcal{E}$-CC-PC-TP-UW and $\mathcal{E}$-CC-RPC-TP-UW.

$\mathcal{E}$-CC-PC-TP-UW $\leq^p_{\text{search}}$ $\mathcal{E}$-CC-RPC-TP-UW: Let our input be (I, S). If S is a solution to I for $\mathcal{E}$-CC-RPC-TP-UW, then $I \in A_{\mathcal{E}}$. So output $(\emptyset, C)$.

$\mathcal{E}$-CC-RPC-TP-UW $\leq^p_{\text{search}}$ $\mathcal{E}$-CC-PC-TP-UW: Let our input be (I, S). If S is a solution to I for $\mathcal{E}$-CC-PC-TP-UW, then $I \in A_{\mathcal{E}}$. So output $(\emptyset, C)$.

4. The proof of Theorem 3.12 shows that Approval-DC-PC-TP-UW = Approval-DC-RPC-TP-UW = $B = \{(C, V, p) \mid p \in C$ and p is not a unique winner of the approval election $(C, V)\}$. They also show that for each $I \in B$, $(\emptyset, C)$ is a solution to I for both Approval-DC-PC-TP-UW and Approval-DC-RPC-TP-UW.

   Approval-DC-PC-TP-UW $\leq^p_{\text{search}}$ Approval-DC-RPC-TP-UW: Let our input be (I, S). Since S is a solution to I for Approval-DC-RPC-TP-UW, $I \in B$. So output $(\emptyset, C)$.

   Approval-DC-RPC-TP-UW $\leq^p_{\text{search}}$ Approval-DC-PC-TP-UW: Let our input be (I, S). Since S is a solution to I for Approval-DC-PC-TP-UW, $I \in B$. So output $(\emptyset, C)$.

5. The proof of Theorem 3.14 shows that Approval-CC-PC-TP-NUW = Approval-CC-RPC-TP-NUW = $A = \{(C, V, p) \mid p \in C$ and p is a winner of the approval election $(C, V)\}$. They also show that for each $I \in A$, $(\emptyset, C)$ is a solution to I for both Approval-CC-PC-TP-NUW and Approval-CC-RPC-TP-NUW.

   Approval-CC-PC-TP-NUW $\leq^p_{\text{search}}$ Approval-CC-RPC-TP-NUW: Let our input be (I, S). Since S is a solution to I for Approval-CC-RPC-TP-NUW, $I \in A$. So output $(\emptyset, C)$.

   Approval-CC-RPC-TP-NUW $\leq^p_{\text{search}}$ Approval-CC-PC-TP-NUW: Let our input be (I, S). Since S is a solution to I for Approval-CC-PC-TP-NUW, $I \in S$. So output $(\emptyset, C)$.

6. Approval-DC-PV-TE-UW $\leq^p_{\text{search}}$ Approval-DC-PV-TE-NUW: On input $I = (C, V, p)$ and S, since a solution for I to Approval-DC-PV-TE-NUW is a solution for I to Approval-DC-PV-TE-UW, output S.

   Approval-DC-PV-TE-NUW $\leq^p_{\text{search}}$ Approval-DC-PV-TE-UW: The proof of Theorem 3.16 shows how to construct, given $I = (C, V, p)$ and a solution S for Approval-DC-PV-TE-UW, a solution to I for Approval-DC-PV-TE-NUW, and we note that the construction can be done in polynomial time.

7. Approval-CC-PC-TE-NUW $\leq^p_{\text{search}}$ Approval-CC-RPC-TE-NUW: The proof of Theorem 3.17 shows how to construct, given $I = (C, V, p)$ and a solution S for Approval-CC-RPC-TE-NUW, a solution to I for Approval-CC-PC-TE-NUW, and we note that the construction can be done in polynomial time.

   Approval-CC-RPC-TE-NUW $\leq^p_{\text{search}}$ Approval-CC-PC-TE-NUW: The proof of Theorem 3.17 shows how to construct, given $I = (C, V, p)$ and a solution S for Approval-CC-PC-TE-NUW, a solution to I for Approval-CC-RPC-TE-NUW, and we note that the construction can be done in polynomial time.

8. Approval-CC-PC-TE-UW $\leq^p_{\text{search}}$ Approval-CC-RPC-TE-UW: The proof of Corollary 3.18 shows how to construct, given $I = (C, V, p)$ and a solution S for Approval-CC-RPC-TE-UW, a solution to I for Approval-CC-PC-TE-UW, and we note that the construction can be done in polynomial time.

   Approval-CC-RPC-TE-UW $\leq^p_{\text{search}}$ Approval-CC-PC-TE-UW: The proof of Corollary 3.18 shows how to construct, given $I = (C, V, p)$ and solution S for Approval-CC-PC-TE-UW, a solution to I for Approval-CC-RPC-TE-UW, and we note that the construction can be done in polynomial time. □

It is well-known that approval satisfies Unique-α, and so the following corollary holds.

Corollary 4.11. *1. For each pair* $(\mathcal{T}_1, \mathcal{T}_2)$ *among the six collapsing types* Approval-DC-RPC-TE-NUW*,* Approval-DC-RPC-TE-UW*,* Approval-DC-PC-TE-NUW*,* Approval-DC-PC-TE-UW*,* Approval-DC-RPC-TP-UW*, and* Approval-DC-PC-TP-UW*, it holds that* $\mathcal{T}_1 \equiv^{p}_{\text{search}} \mathcal{T}_2$*.*

2. Approval-CC-PC-TP-UW $\equiv^{p}_{\text{search}}$ Approval-CC-RPC-TP-UW*.*

Proof. This corollary is easy to see, but we include the proof for completeness.

1. This follows directly from Corollaries 4.9 and 4.10 (Parts 2 and 4 of the latter) since approval satisfies Unique-α.

2. Similarly, this follows directly from Corollary 4.10 since approval satisfies Unique-α. □

4.5 Concrete Search Complexities of Collapsing Electoral Control Types

In this section, we prove that each control type involved in a search-equivalence proven in this thesis is either polynomial-time computable or SAT-equivalent. Table 4.1 provides a comprehensive summary of our results and indicates which are polynomial-time computable (denoted "polynomial") in the table or are SAT-equivalent. Propositions 4.4 and 4.5 are not listed in the References column, since it is clear when they are being drawn on.

Our strategy is, for each known equivalence class of (decision-)collapsing electoral control types for plurality, voting, and approval, to determine the search complexity of at least one element. We will then leverage the search equivalences (see Propositions 4.4 and 4.5) to determine the search complexity of the remaining elements in the equivalence classes.

Collapsing Control Types	Search Complexities	References
Plurality-DC-RPC-TP-NUW, Plurality-DC-PC-TP-NUW	SAT-equivalent	Theorem 4.6 Corollary 4.19
Plurality-DC-RPC-TE-NUW, Plurality-DC-RPC-TE-UW, Plurality-DC-PC-TE-NUW, Plurality-DC-PC-TE-UW	SAT-equivalent	Theorem 4.8 Corollary 4.19
Veto-DC-PV-TE-NUW, Veto-DC-PV-TE-UW	polynomial	Corollary 4.10 Theorem 4.13
Veto-DC-RPC-TP-NUW, Veto-DC-PC-TP-NUW	SAT-equivalent	Theorem 4.6 Corollary 4.19
Veto-DC-RPC-TE-NUW, Veto-DC-RPC-TE-UW, Veto-DC-PC-TE-NUW, Veto-DC-PC-TE-UW	SAT-equivalent	Theorem 4.8 Corollary 4.19
Approval-DC-RPC-TP-NUW, Approval-DC-PC-TP-NUW	polynomial	Theorems 4.6 and 4.12
Approval-DC-RPC-TP-UW, Approval-DC-PC-TP-UW, Approval-DC-RPC-TE-NUW, Approval-DC-RPC-TE-UW, Approval-DC-PC-TE-NUW, Approval-DC-PC-TE-UW	polynomial	Corollary 4.10 Theorem 4.12
Approval-DC-PV-TE-UW, Approval-DC-PV-TE-NUW	polynomial	Corollary 4.10 Theorem 4.13
Approval-CC-RPC-TP-UW, Approval-CC-PC-TP-UW	polynomial	Corollary 4.10 Theorem 4.12
Approval-CC-RPC-TP-NUW, Approval-CC-PC-TP-NUW	polynomial	Corollary 4.10 Theorem 4.12
Approval-CC-RPC-TE-UW, Approval-CC-PC-TE-UW	polynomial	Corollary 4.10 Theorem 4.13
Approval-CC-RPC-TE-NUW, Approval-CC-PC-TE-NUW	polynomial	Corollary 4.10 Theorem 4.13

Table 4.1: For each collection of (decision-problem) collapsing partition control types for plurality, veto, and approval elections we have shown that their search problems are also of the same complexity.

We prove our results about polynomial-time computability, before proving SAT-equivalences.

The first theorem is only about new algorithms (i.e., not in the literature) that we provide for control with respect to approval.

Theorem 4.12. *The search problem for each of the following control problems is polynomial-time computable:*

1. Approval-DC-PC-TE-UW*,*
2. Approval-DC-PC-TP-NUW*,*
3. Approval-CC-PC-TP-UW*, and*
4. Approval-CC-PC-TP-NUW*.*

Proof. 1. Consider the case of Approval-DC-PC-TE-UW. Approval is known to be immune to DC-PC-TE-UW (Hemaspaandra et al., 2007). So on each input (C, V, p), we have: If p is a unique winner of election (C, V) under approval voting, then $p \notin$ Approval-DC-PC-TE-UW (i.e., there exists no candidate partition under which p is not a final-round unique winner in the PC-TE two-stage election process under approval voting). Our polynomial-time search-problem algorithm thus is the following: On input (C, V, p), in polynomial time determine whether p is a unique winner under approval of election (C, V). If it is, output $\perp$ (indicating there is no partition that will prevent p from being the unique winner in the PC-TE two-stage election process under approval, regarding (C, V)). Otherwise, output as our solution the partition $(\emptyset, C)$, since this will make the final-round election be (C, V), and we know in this "otherwise" case that p does not uniquely win there.

2. It is not hard to see that approval is also immune to DC-PC-TP-NUW: Let p be a winner of an election (C, V). Let k denote the number of ballots (votes) that approve of p in V. It follows that no candidate is approved by more than k votes (else p would not be a winner). Since the TP handling

rule is used, p can never be eliminated from a subelection as no candidate is approved by more votes than p. Thus p always proceeds to the final round, and is a winner. So the Approval-DC-PC-TP-NUW case follows by the above proof with each instance of the word "unique" removed and each TE and UW respectively changed to TP and NUW.

3. Since approval is immune to CC-PC-TP-UW (Hemaspaandra et al., 2007), this case is analogous to case 1, of course by asking whether p is *not* a unique winner of (C, V), and proceeding in the obvious way, again using $(\emptyset, C)$ as our output partition in those cases that do not output $\perp$.

4. Since approval is immune to CC-PC-TP-NUW (see Chapter 3), this case is analogous to case 2, of course by asking whether p is *not* a winner of (C, V), and proceeding in the obvious way, again using $(\emptyset, C)$ as our output partition in those cases that do not output $\perp$. □

This next theorem about approval and veto follows primarily from existing work in the literature, and in one case, we provide a modification to an existing algorithm to establish our result.

Theorem 4.13. *The search problem for each of the following control problems is polynomial-time computable:*

1. Veto-DC-PV-TE-UW.

2. Approval-DC-PV-TE-UW.

3. Approval-CC-RPC-TE-UW.

4. Approval-CC-RPC-TE-NUW.

Proof. 1. Maushagen and Rothe (2018) show that Veto-DC-PV-TE-UW $\in$ P and their algorithm detects whether a solution exists and explicitly constructs a solution in polynomial time when a solution exists.

2. Hemaspaandra et al. (2007) show that Approval-DC-PV-TE-UW $\in$ P and their construction detects whether a solution exists and explicitly constructs a solution in polynomial time when a solution exists.

3. Hemaspaandra et al. (2007) show that Approval-CC-RPC-TE-UW $\in$ P and their construction detects whether a solution exists and explicitly constructs a solution in polynomial time when a solution exists.

4. Our algorithm, which is a modified version of one of Hemaspaandra et al. (2007), proceeds as follows: On input (C, V, p), for each $a \in C$, let y_a be the number of votes in V that approve a and let $Y = \max\{y_a \mid a \in C\}$. If $y_p \neq Y$ and $\|\{a \in C \mid y_a = Y\}\| = 1$, then output $\perp$. Otherwise, output $(\{p\}, C-\{p\})$. Why does $(\{p\}, C-\{p\})$ work? Note that in this "otherwise" case, we have that either (a) $y_p = Y$ or (b) $\|\{a \in C \mid y_a = Y\}\| \geq 2$. If (a) holds, then the partition $(\{p\}, C - \{p\})$ makes p a winner in the final round, since p will uniquely win and move forward from its subelection, and no candidate has more approvals than p so even if a candidate moves forward from the other subelection it cannot prevent p from being a final-round winner. If (a) fails and (b) holds, then the partition $(\{p\}, C - \{p\})$ will make p a winner (indeed, a unique winner) in the final round, since in the other subelection there will be at least two candidates that are approved by Y votes, so they tie as winners of that subelection and are eliminated; thus no candidates will move forward from that first-round election.

   The algorithm just given clearly runs in polynomial time. □

We now move on to proving our results about SAT-equivalence. Showing SAT-equivalence for an individual problem is quite tedious. Since the theme behind our work on collapsing control types involved seeing where/if duplicate work happens, we aimed at having a meta-theorem to help us show SAT-equivalence by leveraging known decision-complexity results, thereby avoiding duplicate work.

Theorem 4.14. *Given an election system $\mathcal{E}$ satisfying $W_{\mathcal{E}} \in \mathrm{P}$, if $\mathcal{T}$ is one of our partition-based control types involving $\mathcal{E}$ and (the decision problem) $\mathcal{T}$ is* NP*-complete, then the search problem for $\mathcal{T}$ is* SAT*-equivalent.*

Proof. Assume $\mathcal{T}$ is one of our partition-based control types, that $\mathcal{E}$ is the election system of $\mathcal{T}$, that $W_{\mathcal{E}} \in \mathrm{P}$, and that (the decision problem) $\mathcal{T}$ is NP-complete.

NP-hard: Let f be any refinement of $\widehat{\Pi}_{\mathcal{T}}$. Since $\mathcal{T}$ is NP-complete, it suffices to show $\mathcal{T} \in \mathrm{P}^f$ to show that $\mathrm{NP} \subseteq \mathrm{P}^f$. Our polynomial-time algorithm to decide $\mathcal{T}$ with function oracle f proceeds as follows: If x is not a triple of the form (C, V, p) where C is a set of candidates, V is a vote collection over C of the vote-type of $\mathcal{E}$, and $p \in C$, then reject. Otherwise, query the function oracle with x and verify, in polynomial time (using the fact that $W_{\mathcal{E}} \in \mathrm{P}$), whether the oracle response is a solution to the control problem. If it is, accept, and otherwise reject.

NP-easy: It is easy to see that there is a refinement of $\widehat{\Pi}_{\mathcal{T}}$, h, such that $h \in \mathrm{FP}^{\mathrm{NP}}$. Let us focus on $\mathcal{T}$ being a voter partition type. The candidate cases are exactly analogous except we build the candidate rather than the voter partition. On input (C, V, p), h makes sure that the input is of the form (C, V, p), that $p \in C$, and that the votes in V are of the type appropriate for $\mathcal{E}$; if not output $\perp$. Otherwise, use a single call to SAT to determine if $(C, V, p) \in \mathcal{T}$. If not, output $\perp$. Otherwise, by an easy binary search, with an NP oracle, we will construct a set V_1 such that $(V_1, V - V_1)$ is a solution to $\mathcal{T}$ for (C, V, p). In fact, if we naturally encode partitions into binary strings, we can binary search, with any NP-complete set such as SAT as our oracle, to find the lexicographically smallest encoding of a $V_1 \subseteq V$ such that $(V_1, V - V_1)$ is a solution to (C, V, p) with respect to $\mathcal{T}$, and then we will output $(V_1, V - V_1)$. (The "helper" NP set for the binary search is simply $\{(C, V, p, \mathcal{C}) \mid (\exists \mathcal{C}')[\mathcal{C}' \geq_{\mathrm{lex}} \mathcal{C}$ and $\mathcal{C}'$ encodes a V_1 such that $(V_1, V - V_1)$ is a solution to (C, V, p) with respect to $\mathcal{T}]\}$. Since this is an NP set, questions to it can be polynomial-time transformed into questions to SAT.) $\square$

Equipped with this powerful tool, we can now prove our SAT-equivalence results. We first establish a new NP-completeness result that is new to the literature, i.e., that Plurality-DC-PC-TP-NUW is NP-complete.

An earlier paper by Hemaspaandra et al. (2007) established that Plurality-DC-PC-TP-UW is NP-complete, but for our study we need the analogous in the NUW model, and so we prove the following theorem by building on the proof that Plurality-DC-PC-TP-UW is NP-complete.

Theorem 4.15. *As a decision problem,* Plurality-DC-PC-TP-NUW *(and equivalently,* Plurality-DC-RPC-TP-NUW*) is* NP*-complete.*

Proof. The analogous result in the unique-winner model was established by Hemaspaandra et al. (2007). Our proof, which we include for completeness, closely follows their proof. We use the same construction (i.e., reduction), but our correctness argument involves small yet important modifications.

Membership in NP is immediately clear. We now prove NP-hardness. We in particular provide a reduction from the Hitting Set problem, a known NP-complete problem (Garey and Johnson, 1979).

The Hitting Set problem is defined as follows. Given a set $B = \{b_1, b_2, \ldots, b_m\}$, a family $S = \{S_1, S_2, \ldots, S_n\}$ of subsets of B, and a positive integer k, does S have a hitting set of size at most k? (That is, is there a set $B' \subseteq B$ with $\|B'\| \leq k$ such that, for each i, $S_i \cap B' \neq \emptyset$?)

We now state that construction that Hemaspaandra et al. (2007) used for their NP-hardness reduction for Plurality-DC-PC-TP-UW, since we will use the same construction as our P-hardness reduction for Plurality-DC-PC-TP-NUW.

Construction 4.16 (Hemaspaandra et al. 2007)**.** *Given a triple (B, S, k), where $B = \{b_1, b_2, \ldots, b_m\}$, $S = \{S_1, S_2, \ldots, S_n\}$ is a family of subsets of B, and $k \leq m$ is a positive integer, construct the following election:*

1. *The candidate set is* $C = B \cup \{c, w\}$.

2. *The vote set* V *is defined as:*

   (a) *There are* $2(m-k)+2n(k+1)+4$ *votes of the form* $c > w > \cdots$, *where the "*$\cdots$*" means that the remaining candidates are in some arbitrary order.*

   (b) *There are* $2n(k+1)+5$ *votes of the form* $w > c > \cdots$.

   (c) *For each* $i \in \{1, \ldots, n\}$, *there are* $2(k+1)$ *votes of the form* $S_i > c > \cdots$, *where "*S_i*" denotes the elements of* S_i *in some arbitrary order.*

   (d) *For each* $j \in \{1, \ldots, m\}$, *there are two votes of the form* $b_j > w > \cdots$.

3. *The distinguished candidate is* c.

We now state two claims that will be used to prove that the reduction works in our case.

Claim 4.17 (Hemaspaandra et al. 2007)**.** *If* B' *is a hitting set of* S *of size* k, *then* w *is the unique winner of the plurality election* $(B' \cup \{c, w\}, V)$.

Claim 4.18. *Let* $D \subseteq B \cup \{w\}$. *If* c *is not a winner of plurality election* $(D \cup \{c\}, V)$, *then there exists a set* $B' \subseteq B$ *such that*

1. $D = B' \cup \{w\}$,

2. w *is the unique winner of plurality election* $(B' \cup \{c, w\}, V)$, *and*

3. B' *is a hitting set of* S *of size less than or equal to* k.

Proof. Fix $D \subseteq B \cup \{w\}$ such that c is not a winner of plurality election $(D \cup \{c\}, V)$. We will show that the above three properties hold, by using a modified version of the argument used by Hemaspaandra et al. (2007).

For a given election, we will let $score(d)$ denote the number of votes that rank candidate d first in that election.

First, notice that for each $b \in D \cap B$, $score(b) < score(c)$ in $(D \cup \{c\}, V)$. Since c is not a winner of that election, it must hold that w is the unique winner of that election, and thus $score(w) > score(c)$.

Let $B' \subseteq B$ be such that $D = B' \cup \{w\}$. Then $D \cup \{c\} = B' \cup \{c, w\}$. Thus it follows that w is the unique winner of the election $(B' \cup \{c, w\})$, proving the first two properties.

Finally, observe that in $(B' \cup \{c, w\}, V)$, it holds that

1. $score(w) = 2n(k+1) + 5 + 2(m - \|B'\|)$, and
2. $score(c) = 2(m-k) + 2n(k+1) + 4 + 2(k+1)\ell$,

where ℓ is the number of sets in S that have an empty intersection with B', i.e., are not "hit by B'." Since w is the unique winner of the election, it follows that

$$\begin{aligned} score(c) &< score(w), \\ 2(m-k) + 2(k+1)\ell &< 1 + 2(m - \|B'\|), \text{ and} \\ (k+1)\ell + \|B'\| - k &< 1/2. \end{aligned}$$

Since ℓ is a nonnegative integer, the only value that it can have here is 0, and so it follows that B' is a hitting set of S of size at most k. □

Now to conclude the proof of Theorem 4.15, we will leverage the two claims above to show that the following two items are equivalent:

1. There is a set $B' \subseteq B$ of size at most k that is a hitting set of S.
2. There is a partition of C such that c can be prevented from being a winner of the two-stage plurality election conducted under the PC-TP-NUW model.

Let B' be a hitting set of S of size k. Let $C_1 = B' \cup \{c, w\}$ and $C_2 = C - C_1$. By Claim 4.17 it holds that w is the unique winner of the subelection (C_1, V), and thus c does not proceed to the final round and is not a winner.

Suppose there is a partition of C such that c is not a winner of the corresponding two-stage plurality election. It must hold that c is eliminated either in the first-round election or in the final-round election. Then it holds that there is a set $D \subseteq B \cup \{w\}$ such that c is not a winner of plurality election $(D \cup \{c\}, V)$. It directly follows from Claim 4.18 that S has a hitting set of size at most k. This concludes the proof that Plurality-DC-PC-TP-NUW is NP-complete.

The "equivalently" in the theorem's statement follows from the fact that for every election system $\mathcal{E}$, $\mathcal{E}$-DC-RPC-TP-NUW = $\mathcal{E}$-DC-PC-TP-NUW (Hemaspaandra et al., 2020). □

For each class of pairwise-decision-collapsing electoral control types of interest, we now know the decision complexity of at least one element, thus allowing us to establish the concrete search complexities of the remaining problems by leveraging our Theorem 4.14.

Corollary 4.19. *The search-problem versions of the following are* SAT*-equivalent.*

1. Plurality-DC-RPC-TP-NUW
2. Plurality-DC-RPC-TE-UW
3. Veto-DC-RPC-TP-NUW
4. Veto-DC-RPC-TE-NUW

Proof. Each of these four is NP-complete, the first by Theorem 4.15, the second is from Hemaspaandra et al. (2007), and the remaining two are from Maushagen and Rothe (2018). The SAT-equivalence follows from those four NP-completenesses, by Theorem 4.14. □

4.6 Conclusion and Open Directions

Our work introduces a notion of equivalence between collapsing electoral control types and establishes that the collapses of Hemaspaandra et al. (2020) and our collapses listed in Chapter 3 hold in the search model. In doing so, we establish a new NP-completeness result in the decision model, prove a powerful result that shows that every NP-complete (decision) partition-control problem is SAT-equivalent in the search model, and we prove interesting results about transference of hardness using our notion of equivalence.

An interesting open direction would be to prove dichotomy theorems (if they exist) in this setting. However, even in the decision setting for unweighted control those are not known to exist, though they are known to exist for other forms/settings of electoral attacks (see Hemaspaandra et al. 2014, Hemaspaandra and Schnoor 2016, and Faliszewski et al. 2015). Another direction would be to find a dichotomy theorem that leverages decision-complexity results (in the sense of Theorem 4.14).

5 Linked Fates: Expanding the Range of Ambiguity-Based Class Pairs Known to Stand or Fall Together

The optimist proclaims that we live in the best of all possible worlds; and the pessimist fears this is true. – Cabell (1927)

5.1 Introduction

The class $\mathrm{UP}_{\leq 1}$, more commonly known as UP, which was introduced by Valiant (1976), is often said to capture "unambiguous nondeterministic computation." Since its introduction, it has been involved in many results that are not necessarily about core complexity theory. In particular, the class UP has been at the center of many results in the study of one-way functions. We briefly discuss that relation in this section and refer interested readers to Watanabe (1988), Beigel (1989), and Hemaspaandra and Zimand (1993), or for a more self-contained overview of the relationship between ambiguity-bounded versions of NP and one-way functions, see also Hemaspaandra and Ogihara (2002, Chapter 2). From a complexity perspec-

tive, we are primarily concerned with the relationship established by Watanabe that for each $k \in \mathbb{N}^+$, $\mathrm{P} = \mathrm{UP}_{\leq 1} \iff \mathrm{P} = \mathrm{UP}_{\leq k}$.

In this chapter, we are concerned with which pairs of functions yield ambiguity-bounded versions of NP with "linked fates," i.e., for which functions f_1 and f_2 such that $(\forall n \in \mathbb{N})[f_1(n) \leq f_2(n)]$ does it hold that $\mathrm{P} = \mathrm{UP}_{\leq f_1(n)} \iff \mathrm{P} = \mathrm{UP}_{\leq f_2(n)}$.

Fact 5.1. *If f_1 and f_2 are functions from $\mathbb{N}$ to $\mathbb{R}^{\geq 1}$ such that $(\forall n \in \mathbb{N})[f_1(n) \leq f_2(n)]$, then $\mathrm{P} = \mathrm{UP}_{\leq f_1(n)} \implies \mathrm{P} = \mathrm{UP}_{\leq f_2(n)}$.*

The above fact follows directly from the definitions of $\mathrm{UP}_{\leq f_1(n)}$ and $\mathrm{UP}_{\leq f_2(n)}$.

Watanabe's (1988) result settles the case where each function is bounded by a constant, i.e., for each function, there exists a constant k such that the function never maps to a number greater than k. However, over the last 36 years, there has been no success in strengthening the work of Watanabe beyond the constant case, and no other linked fates have been established.

We define two new types of linked fates. As a first result, we show that for any function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$ that is monotonically nondecreasing and $k \in \mathbb{N}^+$, $\mathrm{UP}_{\leq f(n)}$ and $\mathrm{UP}_{\leq f(n^k)}$ have linked fates, i.e., $\mathrm{P} = \mathrm{UP}_{\leq f(n)} \iff \mathrm{P} = \mathrm{UP}_{\leq f(n^k)}$. In some sense, a polynomial increase in the argument size is not enough to separate from P. The result itself relies on a simple, yet powerful technique: padding.

Perhaps more surprisingly, we even show that our new type of linked fate and Watanabe's approach can be merged, and we can get the best advantages of each approach simultaneously. More specifically, for each $k \in \mathbb{N}^+$ and each monotonically nondecreasing $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$, we show that $\mathrm{P} = \mathrm{UP}_{\leq f(n)} \iff \mathrm{P} = \mathrm{UP}_{\leq f(n^k)+\mathcal{O}(1)}$.

In fact, to establish the proof of the result in the prior paragraph, we establish that for each function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$ and $j \in \mathbb{N}$, "$\mathrm{P} = \mathrm{UP}_{\leq f(n)} \iff \mathrm{P} = \mathrm{UP}_{\leq f(n)+j}$" holds even without the assumption that f is monotonically nondecreasing (we give and prove this result as Theorem 5.5). This, in some sense,

provides a second type of linked fate since it is not implied by the result stated in the previous paragraph (because there is no assumption that f is monotonically nondecreasing in Theorem 5.5). However, this type of linked fate is probably less interesting that the one described in the previous paragraph.

5.2 Main Results

In this section, we will discuss two differing approaches to showing that pairs of classes have linked fates. The first approach is one implicitly achieved by Watanabe, who showed that unambiguous one-way functions exist if and only if constant-ambiguity one-way functions exist (Watanabe, 1988). (We here are speaking of complexity-theoretic one-way functions, not cryptographic one-way functions.) Viewed through the lens of the relationships between one-way functions and ambiguity-bounded classes (see Grollmann and Selman 1988; Hemaspaandra and Zimand 1993; Hemaspaandra and Ogihara 2002), that equivalently establishes the following.

Theorem 5.2 (Watanabe 1988)**.** *For each $k \in \mathbb{N}^+$, it holds that* $\mathrm{P} = \mathrm{UP}_{\leq 1} \iff \mathrm{P} = \mathrm{UP}_{\leq k}$.

The treatment in Hemaspaandra and Ogihara (2002) proves that implication directly within the language of ambiguity-bounded classes, rather than indirectly through one-way functions. Theorem 5.2 shows that for every two constants $k_1 \in \mathbb{N}^+$ and $k_2 \in \mathbb{N}^+$, $k_1 \leq k_2$, it holds that the fates of $\mathrm{UP}_{\leq k_1}$ and $\mathrm{UP}_{\leq k_2}$ are linked. Or, taking the extreme case, it says that the fates of $\mathrm{UP}_{\leq 1}$ and $\mathrm{UP}_{\mathcal{O}(1)}$ are linked: $\mathrm{P} = \mathrm{UP}_{\leq 1} \iff \mathrm{P} = \mathrm{UP}_{\mathcal{O}(1)}$.

The above result, Theorem 5.2 of Watanabe, is the only result we know of in the literature that implies a linked-fates situation for polynomial-time, ambiguity-bounded nondeterminism. However, we now give completely different families of

such linked-fates cases. We do so via the power of padding—a technique that has been central in complexity theory in a remarkably varied range of settings, such as the elegant construction of universal complete sets for many complexity classes (Hartmanis, 1978), the study of whether all NP-complete sets are isomorphic to SAT (Berman and Hartmanis, 1977), and the connection between polynomial-time and exponential-time complexity classes (Book, 1974; Hartmanis et al., 1985)—and by appealing to a new inductive approach that subsumes the proof of Theorem 5.2 given by Hemaspaandra and Ogihara (2002).

We will use the power of padding to reduce how much nondeterminism is needed relative to the input length, via increasing the input length.

Theorem 5.3. *For each $k \in \mathbb{N}^+$ and each function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$, it holds that* $\mathrm{UP}_{\leq f(n^k)} \leq^p_m \mathrm{UP}_{\leq f(n)}$.

Proof. Fix k and f to satisfy the requirements of the theorem statement. Let L belong to $\mathrm{UP}_{\leq f(n^k)}$. Let a be a character in L's alphabet.

We claim that $L \leq^p_m \{x \cdot a^{|x|^k - |x|} \mid x \in L\}$, where $\cdot$ denotes concatenation, and a^j denotes the string consisting of j many "a"s. In particular, the polynomial-time many-one reduction simply is the mapping from x to $x \cdot a^{|x|^k - |x|}$.

However, $\{x \cdot a^{|x|^k - |x|} \mid x \in L\} \in \mathrm{UP}_{\leq f(n)}$. In particular, on an arbitrary input y, the $\mathrm{UP}_{\leq f(n)}$ machine will parse the input into x and $a^{|x|^k - |x|}$, or if the input cannot be parsed as that, will reject as the input is clearly not in $\{x \cdot a^{|x|^k - |x|} \mid x \in L\}$. Since $n^k - n$ is monotonically nondecreasing, it is impossible that a given input string can have two different x values (say, at different lengths) that can be parsed as being its x value. Our $\mathrm{UP}_{\leq f(n)}$ machine then simulates the $\mathrm{UP}_{\leq f(n^k)}$ machine for L, running on input x. Since $|x \cdot a^{|x|^k - |x|}| = |x|^k$, the $f(|x|^k)$ ambiguity bound that the $\mathrm{UP}_{\leq f(n^k)}$ machine for L will have in its simulated run ensures that our machine will itself satisfy an $f(n)$ bound relative to its input size, since its input size itself is $|x|^k$. □

Theorem 5.4. *For each $k \in \mathbb{N}^+$ and each monotonically nondecreasing function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$, it holds that* $\mathrm{P} = \mathrm{UP}_{\leq f(n)} \iff \mathrm{P} = \mathrm{UP}_{\leq f(n^k)}$.

Proof. Fix k and f to satisfy the theorem's requirements. The left-to-right direction follows from Theorem 5.3 and the fact that P is closed downwards under polynomial-time many-one reductions. Because f is monotonically nondecreasing, it follows that $(\forall n)[f(n) \leq f(n^k)]$, and so the right-to-left direction holds by Fact 5.1. □

This theorem provides the first result on "linked fates" with respect to ambiguity-bounded versions of NP since the work of Watanabe (1988). For more common functions such as $n + 1$ and 2^n, this padding argument yields even more dramatic collapses (see Corollary 5.9), because, informally, increasing the input size effectively permits us to have more accepting paths. However, when dealing with slow-growing functions (e.g., $\log \log \log(n)$), padding does not seem to buy us much. In our next theorem, we show how to get an additional constant number of accepting paths "for free."

Theorem 5.5. *For each function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$ and for each $j \in \mathbb{N}$, it holds that* $\mathrm{P} = \mathrm{UP}_{\leq f(n)} \iff \mathrm{P} = \mathrm{UP}_{\leq f(n)+j}$.

Proof. By Fact 5.1, the right-to-left direction holds. We now prove the left-to-right direction.

Fix a function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$. We prove that for each $j \in \mathbb{N}$, $\mathrm{P} = \mathrm{UP}_{\leq f(n)} \implies \mathrm{P} = \mathrm{UP}_{\leq f(n)+j}$ by induction on j.

The base case (when $j = 0$) is trivial. The inductive hypothesis we prove is that for each $j \in \mathbb{N}$, $\mathrm{P} = \mathrm{UP}_{\leq f(n)+j} \implies \mathrm{P} = \mathrm{UP}_{\leq f(n)+j+1}$.

Fix a natural number j, suppose $\mathrm{P} = \mathrm{UP}_{\leq f(n)+j}$, pick an arbitrary $L \in \mathrm{UP}_{\leq f(n)+j+1}$, and let M be an NPTM that witnesses that membership (i.e., $(\forall x)[(x \in L \implies 1 \leq \#\mathrm{acc}_M(x) \leq f(|x|)+j+1) \wedge (x \notin L \implies \#\mathrm{acc}_M(x) = 0)]$).

Informally, the intuition for the approach we employ is as follows. We wish to "poison" exactly one accepting path of M for each input $x \in L$ with $\#\text{acc}_M(x) > 1$—that is, we wish to have exactly one accepting path of M on x become a rejecting path whenever M has more than one accepting path on input x. In doing so, we would be decreasing M's number of accepting paths on each input with multiple accepting paths (without changing the language it accepts), thus bounding the number of accepting paths by $f(|x|) + j$ for each $x \in L$. This is however not a realistic goal as doing so seems to require the NPTM to know when it has more than one accepting path. But we can draw insight from this scenario. Consider now this informal description of the machine M' whose behavior is the same as that of M, except that on each input $x \in L$, it has exactly one of its accepting path(s) "poisoned" (and we do not specify the "how" as that is not relevant in our informal description). The language of M' is

$$R = \{x \in L \mid \#\text{acc}_M(x) > 1\},$$

which is similar to the one used to prove Theorem 5.2 by Hemaspaandra and Ogihara (2002). Our biggest task is to prove that $R \in \text{P}$ under the current assumptions, which will be crucial to us in proving that $L \in \text{P}$. Unfortunately, we cannot use the same approach used by Hemaspaandra and Ogihara (2002); in that approach, an NPTM only needs to guess a constant number of distinct paths, but here, the number of paths may be too large for an NPTM to guess. So we use a different approach to prove that $R \in \text{P}$, and then use that fact to conclude that $L \in \text{P}$.

To prove that $R \in \text{P}$, we first define the auxiliary language

$$S = \{(x, p) \mid x \in L \text{ and } p \text{ is an accepting path of } M \text{ on } x \text{ and } M \text{ has an accepting path on input } x \text{ that is lexicographically smaller than } p\}$$

and prove that $S \in \text{P}$.[16]

[16]In this proof, we tacitly assume without loss of generality that we have access to a

Lemma 5.6. $S \in \mathrm{P}$.

Proof. Let N_S be an NPTM that does the following on input z. If z is not a pair, reject. Otherwise, let (x, p) be the pair represented by z. If p is an accepting path of M on x, then guess a path p' of M on x that is lexicographically smaller than p (if no such path exists, then reject). If p' is an accepting path, then we accept on the current path. If p' is not an accepting path, then we reject on the current path.

We first argue correctness. If N_S accepts z, it must be that $z = (x, p)$ for some $x \in L$ and some accepting path p of M on x, and that M on input x has an accepting path p' that is lexicographically smaller than p, i.e., $z \in S$. If $z \in S$, then z is a pair, let us call it (x, p). The machine N_S verifies that $x \in L$ using p, and nondeterministically guesses a path p' of M on x that is lexicographically smaller than p. Since $(x, p) \in S$, there is such a path p' that is accepting, and so N_S accepts.

To conclude, it is clear that N_S runs in nondeterministic polynomial time and that for any $(x, p) \in S$, the number of accepting paths is $\#\mathrm{acc}_M(x) - 1 \leq f(|x|) + j \leq f(|z|) + j$, as the (lexicographically) smallest accepting path of M on x is "poisoned." So $S \in \mathrm{UP}_{\leq f(n)+j} = \mathrm{P}$. □

We now use Lemma 5.6 to prove that $R \in \mathrm{P}$.

Lemma 5.7. $R \in \mathrm{P}$.

Proof. Let N_R be an NPTM that does the following on input x. Guess a path p of M on x. If p is not an accepting path, reject. If $(x, p) \in S$, then accept. Otherwise, reject.

---

polynomial-time computable total order over the accepting paths of Turing machines. Indeed, one can encode an accepting path as a finite string over a finite alphabet (with at least two elements) using only polynomial amount of space. And the standard lexicographical ordering over those strings is a total order that satisfies our assumption.

Because $S \in \mathrm{P}$, the membership test of (x, p) in S can be computed in polynomial time. So N_R clearly runs in nondeterministic polynomial time. Also, for each input $x \in L$, there are only up to $f(|x|) + j$ accepting paths, and N_R accepts if and only if M has more than one accepting path on x, so it follows that $L(N_R) = R \in \mathrm{UP}_{\leq f(n)+j} = \mathrm{P}$. □

Now to conclude, consider the NPTM N_L that does the following on input x. If $x \in R$, accept. Otherwise, immersively simulate M on input x.

Since $R \in \mathrm{P}$, computing the membership of x in R can be computed in polynomial time. So N_L clearly runs in nondeterministic polynomial time, accepts L, and has at most one accepting path on any input. So, $L \in \mathrm{UP} = \mathrm{P}$. □

Naturally, we can combine Theorems 5.4 and 5.5 and prove the theorem below.

Theorem 5.8. *For each monotonically nondecreasing function $f : \mathbb{N} \to \mathbb{R}^{\geq 1}$ and each $k \in \mathbb{N}^+$, $\mathrm{P} = \mathrm{UP}_{\leq f(n)} \iff \mathrm{P} = \mathrm{UP}_{f(n^k)+\mathcal{O}(1)}$.*

Proof. Fix f and k to satisfy the requirements of the theorem statement.

For the right-to-left direction, for every function $j(n)$ that is $\mathcal{O}(1)$, $j(n)$ is positive by the definition of big-Oh (see Sipser 2013, Chapter 7) and by the monotonicity of f, $(\forall n \in \mathbb{N})[f(n^k) + j(n) \geq f(n)]$. So the right-to-left direction holds by Fact 5.1.

We now prove the left-to-right direction. Assume that $\mathrm{P} = \mathrm{UP}_{\leq f(n)}$. By the definition of big-Oh, $\mathrm{UP}_{f(n^k)+\mathcal{O}(1)} = \bigcup_{j \in \mathbb{N}} \mathrm{UP}_{\leq f(n^k)+j}$. Fix $j \in \mathbb{N}$. It suffices to show that $\mathrm{P} = \mathrm{UP}_{\leq f(n^k)+j}$. (One may be concerned here that the case where $j = 0$ is problematic as the ambiguity bound may be 0, but since f maps to $\mathbb{R}^{\geq 1}$ that can never happen.) It follows from our assumption that $\mathrm{P} = \mathrm{UP}_{\leq f(n)}$ and Theorem 5.5 that $\mathrm{P} = \mathrm{UP}_{\leq f(n)+j}$. Moreover, $f(n) + j$ is clearly a monotonically nondecreasing function from $\mathbb{N}$ to $\mathbb{R}^{\geq 1}$, so by Theorem 5.4, it follows that $\mathrm{P} = \mathrm{UP}_{\leq f(n^k)+j}$. □

As a corollary to Theorem 5.8, we obtain the following concrete examples of linked fates.

Corollary 5.9.

1. $\mathrm{P} = \mathrm{UP}_{\leq \log\log(n)} \iff \mathrm{P} = \mathrm{UP}_{\mathcal{O}(1)+\log\log(n)}$.
2. $\mathrm{P} = \mathrm{UP}_{\leq \log(n)} \iff \mathrm{P} = \mathrm{UP}_{\mathcal{O}(\log(n))}$.
3. *For each* $k \in \mathbb{N}^+$, $\mathrm{P} = \mathrm{UP}_{\leq n^k+k} \iff \mathrm{P} = \mathrm{FewP}$.
4. *For each* $j \in \mathbb{N}^+$, $\mathrm{P} = \mathrm{UP}_{\leq j+n^{\log(n)}} \iff \mathrm{P} = \mathrm{UP}_{\mathcal{O}(1)+n^{\mathcal{O}(\log(n))}}$.
5. *For each* $k \in \mathbb{N}^+$, $\mathrm{P} = \mathrm{UP}_{\leq 2^{n^k}} \iff \mathrm{P} = \mathrm{NP}$.

Theorem 5.4 shows that polynomially-bounded increases in the argument of the ambiguity limit yield linked-fate pairs of classes, and the corollaries just given provide a number of concrete examples. In a similar fashion, Theorem 5.5 shows that an addition of a constant amount of ambiguity also yields linked-fate pairs of classes.

5.3 Related Work

It is important not to conflate our issue of (un)linked fates of two classes with the different issue of whether the two classes can be separated in some relativized world. A clear example of this is the constant-ambiguity case mentioned in Section 5.2. Watanabe (1988) proved (in light of the connection between ambiguity-bounded classes and one-way functions; see Grollmann and Selman 1988; Hemaspaandra and Zimand 1993; Hemaspaandra and Ogihara 2002) that if $\mathrm{P} = \mathrm{UP}_{\leq 1}$ then $\mathrm{P} = \mathrm{UP}_{\leq 2}$. And that implication holds in the real world and every relativized world. Nonetheless, there are oracles A for which $\mathrm{UP}^A_{\leq 1} \neq \mathrm{UP}^A_{\leq 2}$ holds, and indeed that separation holds with probability one relative to a random oracle

(Beigel, 1989); however, again by Watanabe's relativizable result, relative to that oracle A, it certainly cannot hold that $\mathrm{P}^A = \mathrm{UP}^A_{\leq 1}$.

Historically, the first person to show a pair of classes whose ambiguity levels were distant enough that in some relativized world their fates were not linked was Rackoff, who in the early 1980s showed that there is an oracle B relative to which (and recall that $\mathrm{NP} = \mathrm{UP}_{2^{n^{\mathcal{O}(1)}}}$) $\mathrm{P}^B = \mathrm{UP}^B \neq \mathrm{NP}^B$ (Rackoff, 1982). Viewed through the lens of robustness, it implies that any improvement of Watanabe's work where UP and NP have linked fates cannot hold robustly. Rackoff's result was substantially tightened in a paper by Fortnow and Rogers (2002) (itself building on techniques of Blum and Impagliazzo 1987 and Hartmanis and Hemachandra 1991) which, although it is not explicitly stated in the paper, builds an oracle A such that $\mathrm{P}^A = \mathrm{UP}^A \neq \mathrm{FewP}^A$. In fact, Fortnow and Rogers effectively show that relative to a "sufficiently generic size-bounded generic oracle" (Fenner et al., 2003) A there is a $k \in \mathbb{N}^+$ such that $\mathrm{P}^A = \mathrm{UP}^A \neq \mathrm{UP}^A_{\leq n+k}$, and our Theorem 5.8 implies that $\mathrm{UP}_{\leq n+k}$ and FewP have linked fates, not just in the real world, but also in every relativized world.

5.4 Conclusions and Open Problems

In this thesis, we provide new families of functions with linked fates, which to our knowledge had not been done since the seminal work of Watanabe (1988). We expect, in in-progress work (Carleton et al., 2024), to provide proofs showing that Watanabe's and this chapter's results are optimal with respect to results that hold robustly (i.e., in all relativized worlds).

We of course would like to provide structural collapses, e.g., of the polynomial hierarchy PH, that follow from our assumptions. However, this is an area that has proved difficult, as even certain dramatic assumptions, such as P = UP or even UP = NP, are not known to imply the collapse of the polynomial hierarchy.

Indeed, regarding the P = UP assumption, it is known by Blum and Impagliazzo (1987) (see also Sheu and Long 1996, p. 425) that there is a relativized world in which $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP} = \mathrm{UP}$ yet the polynomial hierarchy is infinite. However, there is no known relativized world where UP = NP and the polynomial hierarchy is infinite (Hemaspaandra et al., 1995; Fortnow, 2021). And it is open whether there is an oracle world in which UP is in the high hierarchy and PH is infinite—in fact, Sheu and Long (1996, p. 425) explicitly mentioned this open problem in 1996, and, to the best of our knowledge, it remains open even now, decades later.

We have seen that Watanabe (1988) showed how to give a constant addition (and we strengthened that result), and we showed that for each monotonically nondecreasing function f we can change $f(n)$ to $f(n^k)$. (And we even showed that the two methods can be combined.) Aside from these two methods that we found, it would be interesting to see if there are other techniques that can create linked fates and if they can be combined with our two methods.

6 Defying Gravity and Gadget Numerosity: The Complexity of the Hanano Puzzle

I defy gravity. – Marilyn Monroe (Time, 1954)

6.1 Introduction

Many games have been studied through the lens of complexity theory, namely as decision problems. Single-player games are typically NP-complete, and two-player games tend to be PSPACE-complete (Hearn and Demaine, 2009). It is of course possible to have a one-player game be PSPACE-complete, and that indeed does tend to happen if the number of moves in an instance of the game is not bounded by some polynomial in the size of the instance or if the instance's layout is dynamic (Hearn and Demaine, 2009). We notice that one-player games also seem to become hard when they are subject to "irreversible gravity," even though one may expect otherwise as each move seems to be further constrained.

The work on the complexity of games has long focused on what we understand to be "fully reversible games," meaning games for which every action a can be followed by an action b that effectively undoes the effects of action a (without the

use of an "undo" button). In this chapter, we see how our work (Chavrimootoo, 2023a) provides a notion of equivalence (i.e., a notion of collapse) between fully reversible games and games with irreversible gravity. We do so by adding structure to our graphs (through visibility representations, which we define in Chapter 2), and this structure seems to crowd the space of gadgets needed to carry out our reduction, namely the method seems to require 32 different gadgets. We prove however that three suffice, and *collapse* all the other ones, i.e., we prove them to be equivalent under some meaningful notion, thereby exhibiting another notion of collapse.

In this work, we consider a one-player game with irreversible gravity that hints that it could be PSPACE-complete, and we show that its decision complexity is indeed such. The game in question is the Hanano Puzzle, which was developed by video game creator Qrostar (2011). A few years ago, Liu and Yang showed that a decision version of the game is NP-hard, and left open the question of determining if it is also NP-complete (Liu and Yang, 2019). We prove the PSPACE-hardness by giving an indirect reduction from Nondeterministic Constraint Logic (NCL), which is a known PSPACE-complete problem that tends to be useful in proving the PSPACE-hardness of games with sliding blocks (Hearn and Demaine, 2009), by simulating edge flips on the NCL graph using the movement of blocks along a grid. A major challenge of the reduction is circumventing the unwanted effects of gravity that limit of reversibility the game is (e.g., because there is no "jump" feature). We introduce a method grounded in graph theory. Reductions from NCL typically only require two gadgets. However, taken at face value, our method drastically increases the number of gadgets needed to 32, but we prove interesting results about those gadgets and are able to give our reduction using only three gadgets in general (but if we restrict our attention to the Hanano Puzzle, then two gadgets suffice)! These results are independent of the Hanano Puzzle, and so we believe they have applications to other games with properties similar to those

of the Hanano Puzzle.

6.2 Related Work

The literature on the complexity of games is rich and covers a variety of games. For general earlier results, we refer readers to Appendix A of Hearn and Demaine's book (2009), which contains an extensive survey of games whose complexities were known at their time of writing. Some more recent results include showing that connected multiagent path finding is PSPACE-hard (Calviac et al., 2023) (see also earlier work on showing the PSPACE-completeness of multiagent connected path planning by Tateo et al. 2018), motion planning with robots is PSPACE-complete (Demaine et al., 2018), motion planning through doors is PSPACE-complete (Ani et al., 2020), motion planning with multiple robots in certain settings is PSPACE-hard (Brocken et al., 2020), 1×1 Rush Hour is PSPACE-complete (Brunner et al., 2020), Push-1F is PSPACE-complete (Ani et al., 2022)—a 20-year-old open problem, as it was proven in 2002 that for all $k \geq 2$, Push-kF is PSPACE-complete (Hearn et al., 2002)—and Wordle is NP-hard (Subercaseaux and Lokshtanov, 2022).

The introduction of NCL (Hearn and Demaine, 2005) helped simplify the process of showing that many games with sliding blocks are PSPACE-complete by limiting the number of gadgets to simulate to two. The work on motion planning through doors (Ani et al., 2020) provides a framework to show the PSPACE-hardness of certain problems by simulating *one* gadget. However, that paper's contribution does not solve the major problem faced by classifying the Hanano Puzzle: circumventing certain effects of gravity. There are games with gravity that were studied prior to the introduction of NCL. For example, Friedman (2001) proved Cubic to be NP-hard using a similar construction to that of Liu and

Yang (2019).[17] Clickomania is another such game. It is a one-player game with a bounded number of moves, and it is in fact NP-complete (Biedl et al., 2002). Solving a level of Super Mario Brothers (SMB), which is another game with gravity, has also been proven to be PSPACE-complete (Demaine et al., 2016). However, the framework used in that proof does not rely on NCL, since SMB is not a game that involves pulling blocks. Another famous game with gravity is Tetris. While the "offline" version is NP-complete (Breukelaar et al., 2004), in the general case, it is NP-hard (Asif et al., 2020). On the other hand, Jelly-no-Puzzle, also by Qrostar, is known to be NP-hard is the general case (Yang, 2018). Our work uses NCL to study a game with sliding blocks and irreversible gravity, and extends this line of work by providing a framework to study such games using only three gadgets in general, and by having only two gadgets when focusing on the Hanano Puzzle.

6.3 The Hanano Puzzle

The Hanano Puzzle comprises different levels. A level of the game is an $n \times m$ grid (with $n, m > 0$) that contains only the following components: immovable gray blocks, movable gray blocks, (movable) colored blocks, colored flowers, and empty spaces. Each colored block/flower can be red, blue, or yellow. Each flower is immovable and is affixed to some block. If that block is movable, then whenever it moves, the affixed flower moves with the block (see Figure 6.1d). Gray blocks can be of arbitrary shape and size, while all other components are 1×1 objects. In our gadgets, we try to the best of our ability to minimize the number of sides of each movable gray block. A block can slide (see Figure 6.1a) left or right, one step at a time. For a slide to occur, the space that the block will occupy after the

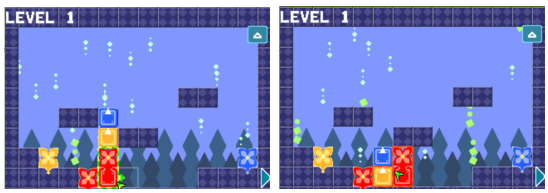
[17] That paper actually claims to show the NP-completeness of Cubic. However, there is no apparent proof or observation in the text of a matching upper bound.

slide must either be empty or be occupied by part of the block that is sliding. Two adjacent blocks of width one can also be swapped in one step (see Figure 6.1b) the positions of the two blocks can be swapped without moving any other component of the grid.

(a) Example of a slide (Qrostar, 2011).

(b) Example of a swap (Qrostar, 2011).

(c) Example of a blooming flower (Qrostar, 2011).



(d) Example of effects of gravity (Qrostar, 2011).

Figure 6.1: Screenshots of the Hanano Puzzle (reproduced with permission from Qrostar 2022).

Figure 6.1 shows screenshots of the game that show sample game moves. Note that the checkered cells are what we call "immovable gray blocks." Movable gray blocks are not depicted in these figures. Because this is a game with gravity, after the player makes a move, every movable block that is not directly supported will fall (see Figure 6.1d). This can be viewed as happening in a single step. Each colored block contains an arrow, pointing either up, down, left, or right. If a colored block touches (by sharing a side; touching corners have no effect) a flower of the same color, a flower will bloom from the side of the colored block indicated by the arrow (see Figure 6.1c), and the flower will stay affixed/attached to that block. We will sometimes say that the block has bloomed when this happens. If the blooming side is in contact with a block, the blooming flower attempts to

"force" its way out by pushing against the surface in contact with the blooming side. This may result in that block in contact with the blooming side to be shifted, or in the blooming block to be shifted. If no shift is possible, then the flower does not bloom. A block can only bloom once and that action cannot be undone. Additionally, if the new flower is in contact with a different block of the same color, chain bloomings can occur within the same step. To solve (complete) a level, one must make every colored block bloom. Formally, we determine the complexity of HANANO = $\{H \mid H$ is a solvable level of the Hanano Puzzle$\}$.

6.4 Overview of the Approach

The common way of showing PSPACE-hardness by reducing from NCL is to give a gadget—that is, an instance of the game—that simulates an AND vertex and giving a second gadget that simulates the OR vertex. The gadgets typically simulate edge flips in an NCL graph by using the movement of blocks between the gadgets. However, in our case, such a direct reduction will not work; due to gravity, many moves are not reversible, and so such a reduction is not guaranteed to work. Liu and Yang (2019) did not encounter this problem as their reduction from CIRCUIT-SAT (a known NP-complete problem) leveraged the fact that bits only "move" in one direction in a boolean circuit.

We will adopt the same approach of designing gadgets that simulate AND/OR NCL vertices. For each edge in the original graph, we will have a blue block in the gadget representing the vertex to which the edge is incident. Thus the location of those blue blocks will represent the orientation of edges in the original graph. We will ensure that these blocks can move between the correct gadgets while being subject to the inflow constraints of NCL. It is however evident that this design faces a clear challenge; not every edge will be horizontal, and so gravity prevents the blocks from moving in certain directions (thus making our HANANO instances

not as reversible as the original NCL instances). Thankfully, the planarity of the NCL graphs allows us to leverage a known graph-theoretic technique: visibility representations.

Our reduction will thus first compute the visibility representation of the given graph, thus giving us the guarantee that all edges are horizontal. Hence, those "edge flips" in the game of interest (here, the Hanano Puzzle) are fully reversible, to the extent that is required to be compatible with NCL's reversibility.

6.5 Gadgets and Schemas

We know that each vertex in the NCL graph is connected to exactly three edges. So this tells us that each gadget must have exactly three entry points, and each of these entry points can lie on either the left side or the right side of the gadget. It is easier to consider that on each side, there is a bottom entry point, a middle entry point, and a top entry point, and that based on the specific gadget, three of the six entry points will be "blocked off." We introduce some notation to describe the different gadgets that arise from this combinatorial issue. Each gadget is assigned a label of the form $x_1x_2x_3|y_1y_2y_3$, where for each $i \in \{1,2,3\}$, $\{x_i, y_i\} \in \{\{R,\cdot\},\{B,\cdot\}\}$, and the list $[x_1, x_2, x_3, y_1, y_2, y_3]$ contains either exactly three Bs, or exactly one B and two Rs. Thus $\cdot\cdot R|RB\cdot$ is the name given to the gadget with a bottom entry point on the left for a blue block representing a red edge, a top entry point on the right for a blue block representing a red edge, and a middle entry point on the right for a blue block representing a blue edge. By a simple counting argument, we can see that this creates an explosion in the number of gadgets as we need to implement eight OR gadgets and 24 AND gadgets (thus 32 gadgets in total) However, we will show how to construct all our gadgets from just three gadgets! Moreover, we note that when our attention is restricted to the Hanano Puzzle, we only need to construct *two* gadgets.

All our movable blocks and flowers will be blue. To help identify the blocks and flowers of the gadgets in proofs, we give those items an ID. For example, the label "B2" indicates the second blue block in the gadget, whereas the label "BF1" indicates the first blue flower. (We still need to specify the color to distinguish from gray blocks.) It's important to note that *all our blocks bloom upwards.* Next to each flower will be a boldfaced white line to indicate where the flower is attached. Additionally, our gadgets will contain some grid lines to help the reader better gauge the distances.

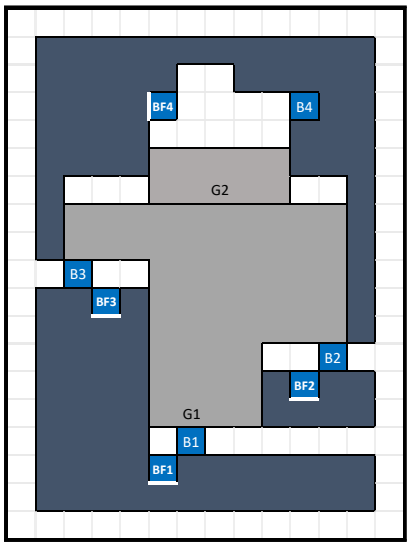
Let us first look at how to construct the OR gadgets. Figure 6.2a shows a gadget for $B \cdot\cdot| \cdot BB$.

Theorem 6.1. *The gadget in Figure 6.2a respects the* OR *constraints.*

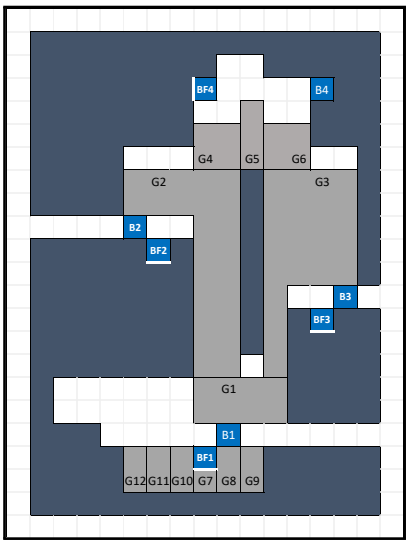
Proof. First notice that each movable gray block has very limited movement. G2 can only move up by one "unit," and G1 can either move up or move down by one unit. Thus for any blue block Bx, the only flower that it can reach in that gadget is BFx. Now, the only way for B4 to bloom is if B4 is in contact with BF4, and it must be on BF4's right side (the only other exposed side of BF4 is the bottom side, but if B4 is directly under BF4 it will not have enough room to bloom). Thus B4 can bloom if and only if G2 moves up by one unit. This happens if and only if G1 moves up by one unit, which happens if and only if one of B1, B2, or B3 blooms. Finally, notice that if B1, B2, and B3 all leave the gadget, then G1 and G2 both drop by one unit with no possibility of returning to their original configuration, thus making it impossible to bloom B4.

We conclude by noting that we could have merged G1 and G2 into a single block, but opted not to as we sought to minimize the number of sides on each movable gray block. □

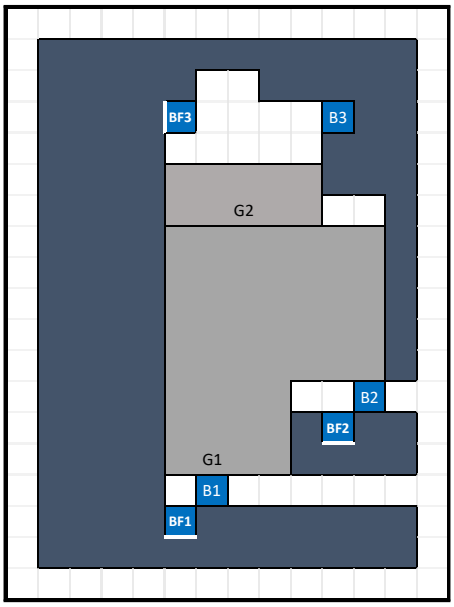
Now, for convenience we define a constrained blue edge terminator gadget, which will allow us to fix the direction of a particular blue edge without connecting



(a) $B\cdot\cdot|\cdot BB$ gadget.



(b) $R\cdot\cdot|\cdot RB$ gadget.



(c) Red bend gadget.

Figure 6.2: Our three gadgets: an OR gadget, an AND gadget, and a red bend gadget.

it to any other nodes. We state the following proposition in a general form, i.e., its proof will not depend on the Hanano Puzzle's properties.

Proposition 6.2. *The constrained blue edge terminator gadget can be constructed using any gadget that respects the* OR *constraints.*

Proof. We want the constrained blue edge terminator to be a gadget that, when attached to a tunnel that represents a blue edge, will force the block that represents the edge's orientation to be inside itself (i.e., inside the constrained blue edge terminator) so as to not violate the inflow constraints.

Fix a gadget that satisfies the same constraints as an NCL OR vertex. There must exist a configuration of the gadget that corresponds to the NCL OR with one (blue) edge pointing into the vertex and the remaining edges (also blue) pointing out of the vertex. Now, block off the tunnels that correspond to the two edges pointing of the vertex. This configuration of the gadget correctly constraints the represented blue edge's orientation. □

A natural question to ask is whether $B\cdot\cdot|\cdot BB$ is special, or whether this result can be achieved using any of the other OR gadgets, and we answer in the positive that indeed, any of the eight OR gadgets suffices. We first note that it suffices to consider the gadgets for $B\cdot\cdot|\cdot BB$, $\cdot B\cdot|B\cdot B$, $\cdot\cdot B|BB\cdot$, and $\cdots|BBB$, as the remaining ones can be obtained via vertical symmetry. In an abuse of notation, we will sometimes use the shorthand for a gadget to refer to the gadget that is obtained from it via vertical symmetry in our schemas. Edges that are connected to the constrained blue edge terminator have a direction assigned and point to a $\oslash$ to indicate the termination. The remaining edges have no direction, indicating that they can be assigned in any way that satisfies the minimum inflow constraints. Though the proof is omitted, Figure 6.3 provides the necessary constructions.

The following theorem is proven by giving a closed loop of constructions. In the following cases, we assume we have a blue edge terminator, since it is in some sense a "freebie." Gadget (2) is implemented using only instances of gadget (1). Gadget (3) is implemented using only instances of gadget (2). Gadget (4) is imple-

mented using only instances of gadget (3). And finally, gadget (1) is implemented using only instances of gadget (4). It thus suffices to construct one OR gadget. The proof of Lemma 6.6 follows a similar strategy.

Lemma 6.3. *For each gadget in the following list, the remaining gadgets in that same list can be constructed from that initial gadget: 1.* $B \cdot \cdot | \cdot BB$*, 2.* $\cdot B \cdot | B \cdot B$*, 3.* $\cdot \cdot B | BB \cdot$*, and 4.* $\cdot \cdot \cdot | BBB$*.*

Proof. We prove this lemma by showing how to construct 2 from 1, how to construct 3 from 2, how to construct 4 from 3, and finally how to construct 1 from 4. In each case, we will have a gadget that satisfies the same constraints as an NCL OR vertex, so we tacitly appeal to Proposition 6.2 to, for "free," have a constrained blue edge terminator. We construct in Figure 6.3 schematic diagrams to aid in our proof.

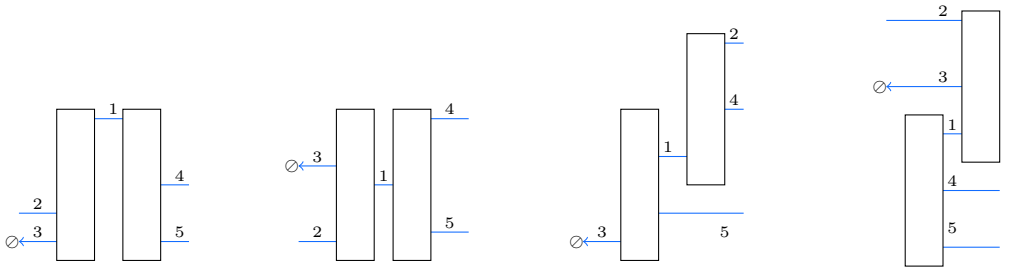
From 1 to 2. Figure 6.3a depicts the construction. The edges of the $\cdot B \cdot | B \cdot B$ gadget are 2, 4, and 5. If those edges all point out (i.e., 2 points left and the other two point right), then the minimum inflow constraint is violated as 3 cannot be flipped and 1 can only point into one the two gadgets. Thus one of 2, 4, or 5 must always be pointed inwards, and this gadget satisfies the same constraints and as NCL OR vertex.

From 2 to 3. Figure 6.3b depicts the construction. The edges of the $\cdot B \cdot | B \cdot B$ gadget are 2, 4, and 5. If those edges all point out (i.e., 2 points left and the other two point right), then the minimum inflow constraint is violated as 3 cannot be flipped and 1 can only point into one the two gadgets. Thus one of 2, 4, or 5 must always be pointed inwards, and this gadget satisfies the same constraints and as NCL OR vertex.

From 3 to 4. Figure 6.3c depicts the construction. The edges of the $\cdot B \cdot | B \cdot B$ gadget are 2, 4, and 5. If those edges all point out (i.e., they all point right), then the minimum inflow constraint is violated as 3 cannot be flipped and 1 can only

point into one the two gadgets. Thus one of 2, 4, or 5 must always be pointed inwards, and this gadget satisfies the same constraints and as NCL OR vertex.

From 4 to 1. Figure 6.3d depicts the construction. The edges of the $\cdot B \cdot | B \cdot B$ gadget are 2, 4, and 5. If those edges all point out (i.e., 2 points left and the other two point right), then the minimum inflow constraint is violated as 3 cannot be flipped and 1 can only point into one the two gadgets. Thus one of 2, 4, or 5 must always be pointed inwards, and this gadget satisfies the same constraints and as NCL OR vertex. □



(a) $\cdot B \cdot | B \cdot B$. (b) $\cdot\cdot B | BB \cdot$. (c) $\cdots | BBB$. (d) $B \cdot\cdot | \cdot BB$.

Figure 6.3: Schemas showing the "equivalence" of the OR gadgets.

We now present one gadget that respects the AND constraints.

Theorem 6.4. *The gadget in Figure 6.2b respects the* AND *constraints.*

Proof. Let us first describe the gadget before arguing its correctness.

G4 and G6 can only move up by one "unit," and G2 and G3 can each either move up or move down by one unit. Thus each colored block, can only reach one flower. If both B2 and B3 exit the gadget, then B1 must remain to support G1 (which in turn supports G2 and G3), as otherwise, G2 and G3 will drop by one unit and G4 and G6 will never be able to move up. Similarly, if B1 is to exit, all the gray blocks must remain supported. This is only possible if G1 is stowed to the left and B2 and B3 remain in the gadget to, respectively, support

G3 and G2. Additionally, the area underneath B1 is made up of multiple movable gray blocks for a simple reason: B1 must be able to move horizontally without blooming (either to exit the gadget or two carry G1 to "stow" it). By this setting, we can move the location of BF1 (by swapping G7 with an adjacent movable block of width one; BF1 is affixed to G7 and the two will move as if they were one 2×1 block) to make sure it is always on the left of B1.

Now, the only way for B4 to bloom is if B4 is in contact with BF4, and it must be on BF4's right side (the only other exposed side of BF4 is the bottom side, but if B4 is directly under BF4 it will not have enough room to bloom). Thus B4 can bloom if and only if G4 and G6 move up by one unit. This can happen exactly if both G2 and G3 move up by one unit. There are two ways this can happen: Either both B2 and B3 bloom, or B1 blooms (thus pushing G1 up by one unit). Thus B4 blooms if and only if either B2 and B3 bloom in the gadget, or B1 blooms in the gadgets. □

Through Theorem 6.1 and Lemma 6.3, we have handled the OR gadgets. However, the AND gadgets are a harder case to handle, but we can construct all of them using two gadgets: a gadget that satisfies the AND constraints, and a newly-defined "red bend" gadget. Lemma 6.6 gives an analogous result to Lemma 6.3, and Figure 6.4 shows the necessary constructions. We now define an important property of the red bend gadget that is used in the proof of Lemma 6.6. Intuitively, the results means that the inflow constraint on red bend gadgets is one (or two under the "blue bend" interpretation).

Proposition 6.5. *The red bend gadget is solvable if and only if at least one red block remains in the gadget throughout the game play.*

Proof. The design of the gadget is simply a restricted/modified version of that in Figure 6.2a, so we omit the description of the gadget.

$\Longrightarrow$: Suppose the gadget is solvable. Then B3 must come in contact with BF3. This is only possible if both G1 and G2 move up by one unit. For this to happen, either B1 or B2 must bloom while supporting G1.

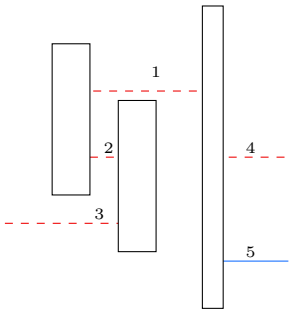
$\Longleftarrow$: Suppose that either B1 or B2 blooms while supporting G1. In both cases, G1 moves up by one unit, and pushes G2 up by one unit, allowing B3 to come in contact with BF3 to bloom. □

Earlier, we mentioned that when our attention is focus on HANANO, we only need two gadgets. This is indeed possible because all the blocks that we use are of the same color, and so we can define the red bend gadget to be a restricted version of the OR gadget. Consider the gadget in Figure 6.2a. If we place a constrained blue edge terminator at the tunnel for B3, then the resulting gadget is essentially the red bend gadget.

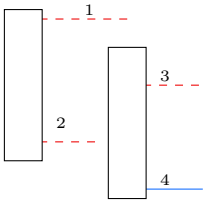
Lemma 6.6. *For each gadget in the following list, the remaining gadgets in that same list can be constructed from that initial gadget, the red bend gadget, and any* OR *gadget: 1.* $R\cdot\cdot|\cdot RB$*, 2.* $\cdot R\cdot|R\cdot B$*, 3.* $\cdots|RRB$*, 4.* $\cdot\cdot B|RR\cdot$*, 5.* $\cdots|BRR$*, 6.* $\cdot\cdot R|BR\cdot$*, 7.* $B\cdot\cdot|\cdot RR$*, 8.* $\cdot R\cdot|B\cdot R$*, 9.* $\cdots|RBR$*, 10.* $R\cdot\cdot|\cdot BR$*, 11.* $\cdot B\cdot|R\cdot R$*, and 12.* $\cdot\cdot R|RB\cdot$*.*

Proof. We represent our red bend gadgets using a vertex with exactly two red edges on the same side of the gadget. The structure of this proof resembles that of Lemma 6.3.

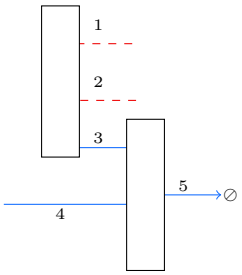
From 1 to 2. Figure 6.4a depicts the construction. If edge 5 points right, then edge 1 points right and edge 4 points left. Thus edge two must point left, leaving edge 3 to point right. If edge 5 points left, then edge 4 is free to point in either direction. In that case, we can fix edge 1 to point left and edge 2 to point right, thus leaving edge 3 to point in any direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.



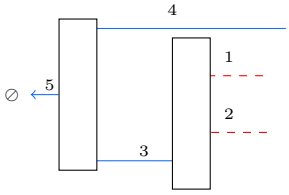
(a) $\cdot R \cdot | R \cdot B.$



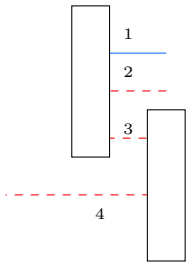
(b) $\cdots | RRB.$



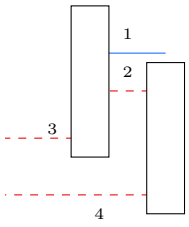
(c) $\cdot\cdot B | RR\cdot.$



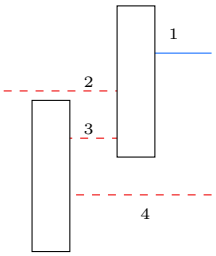
(d) $\cdots | BRR.$



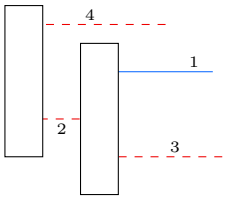
(e) $\cdot\cdot R | BR\cdot.$



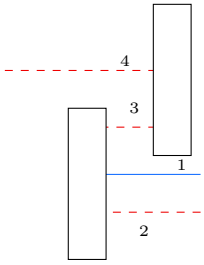
(f) $B \cdot\cdot | \cdot RR.$



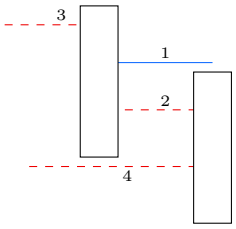
(g) $\cdot R \cdot | B \cdot R.$



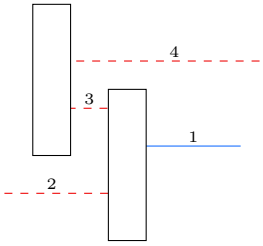
(h) $\cdots | RBR.$



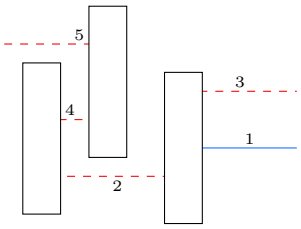
(i) $R \cdot\cdot | \cdot BR.$



(j) $\cdot B \cdot | R \cdot R.$



(k) $\cdot\cdot R | RB\cdot.$



(l) $R \cdot\cdot | \cdot RB.$

Figure 6.4: Schemas showing the "equivalence" of the AND gadgets.

In the rest of this proof, whenever we use a number x, we implicitly mean "edge x."

From 2 to 3. Figure 6.4b depicts the construction. If 4 points right, then 2 must point right and 3 must point left. Thus 1 must point left. If 4 points left, we can fix 2 to point left, leaving 1 and 3 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 3 to 4. Figure 6.4c depicts the construction. If 4 points left, then 3 must point right, forcing both 1 and 2 to point left. If 4 points right, we can fix 3 to point left, thus leaving 1 and 2 free to point in any direction. Thus, this gadget satisfies the same constraints as an NCL AND vertex.

From 4 to 5. Figure 6.4d depicts the construction. If 4 points right, then 3 must point left, forcing both 1 and 2 to point left. If 4 points left, we can fix 3 to point right, thus leaving 1 and 2 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 5 to 6. Figure 6.4e depicts the construction. If 1 points right, then 2 and 3 must point left, and so 4 must point left. If 1 points left, then we can fix 3 to point right, leaving 2 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 6 to 7. Figure 6.4f depicts the construction. If 1 points right, then 2 must point left and 3 must point right, and so 4 must point right. If 1 points left, then we can fix 2 to point right, leaving 3 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 7 to 8. Figure 6.4g depicts the construction. If 1 points right, then 2 and 3 must point right too, and so 4 must point left. If 1 points left, then we can fix 3 to point left, leaving 2 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 8 to 9. Figure 6.4h depicts the construction. If 1 points right, then 2

must point right and 3 must point left, and so 4 must point left. If 1 points left, then we can fix 2 to point left, leaving 3 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 9 to 10. Figure 6.4i depicts the construction. If 1 points right, then 2 and 3 must point left, and so 4 must point right. If 1 points left, then we can fix 3 to point right, leaving 2 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 10 to 11. Figure 6.4j depicts the construction. If 1 points right, then 2 must point left and 3 must point right, and so 4 must point right. If 1 points left, then we can fix 2 to point right, leaving 3 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 11 to 12. Figure 6.4k depicts the construction. If 1 points right, then 2 and 3 must point right, and so 4 must point left. If 1 points left, then we can fix 3 to point left, leaving 2 and 4 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex.

From 12 to 1. Figure 6.4l depicts the construction. If 1 points right, then 2 must point right and 3 must point left, and so 4 must point left, and 5 must point right. If 1 points left, then we can fix 2 to point left and fix 4 to point right, leaving 3 and 5 free to point in either direction. Therefore, this gadget satisfies the same constraints as an NCL AND vertex. □

6.6 Main Result

It is easy to see that HANANO $\in$ PSPACE. Indeed, a nondeterministic machine can on a given level of the Hanano Puzzle simply nondeterministically guess the next move to make and accept if and if only the level is solved. The machine will only use polynomial space (in fact, linear space).

Using the defined gadgets and the visibility representation, we can guide a reduction from NCL to HANANO, thus yielding the following theorem.

Theorem 6.7. HANANO *is* PSPACE*-complete even if (1) all flowers and colored blocks have the same color, and (2) colored blocks can only bloom upwards.*

Proof. Since HANANO $\in$ PSPACE, it suffices to show that NCL $\leq_m^p$ HANANO. Consider the function—which will clearly be polynomial-time computable—that we describe in the next paragraph. We assume without loss of generality that the input is an NCL graph and a valid target edge, as we can easily detect in polynomial time if is not and map to a fixed element that is not in HANANO.

Construct in polynomial time a visibility representation for the input NCL graph and construct a game grid based on the visibility representation, replacing each vertex of the graph by a suitable gadget, and replacing edges with the appropriate tunnels. The game grid will be polynomially larger than the visibility representation since the gadgets have constant-bounded size. We must ensure that the game is only solvable when the target edge $e = (u, v)$ is flipped. Let b denote the blue block representing edge e. If the flower that blooms b is attached to an immovable gray block, replace that flower with an immovable gray block. Otherwise, the flower that blooms b must be attached to the top of a 1×1 movable gray block. Replace that gray block (along with the attached flower) with a 2×1 movable gray block. There is now no flower in the gadget for v that can bloom b, so to bloom, b must move to the gadget for u. If the game is solvable, then b must bloom, and so there will exist a sequence of block movements corresponding to edge flips, so the edge e can be flipped in G. If there is a sequence of edge flips that eventually flips edge e in G, there is sequence of block movements that respect the inflow constraints and eventually see b move from the gadget representing v to the gadget representing u. Thus the colored block b (and all the other ones in the game) can bloom, and the game is solvable. Finally, note that all the colored

blocks in our gadgets have their arrows pointing up, and that we only use blue blocks/flowers. □

Figure 6.5 shows a sketch of a reduction from NCL to HANANO by using as a working example the graph in Figure 2.1. We already have a visibility representation from Figure 2.2. Let the target edge to be flipped be (C, B), i.e., edge 4.

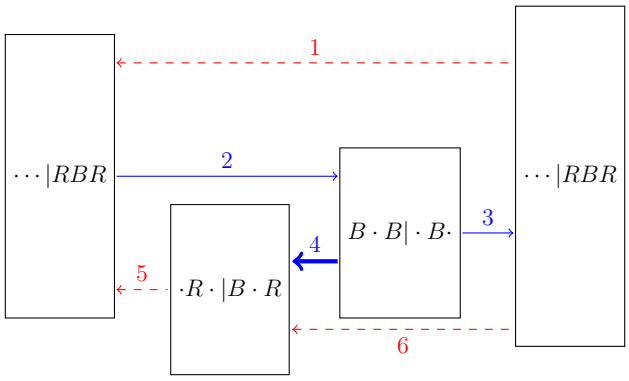


Figure 6.5: Sketch of how the gadgets map onto the planar grid.

The horizontal lines now represent tunnels, the $\cdot R \cdot |B \cdot R$ gadget has one less blue flower, and empty spaces can be filled with immovable gray blocks. The target edge has been boldfaced.

6.7 Conclusion and Open Problems

Our result establishes that HANANO cannot be NP-complete unless NP = PSPACE, thus closing Liu and Yang's open problem.

The use of visibility representations in the context of NCL graphs is an important contribution of this work. Even more important are Proposition 6.2, and Lemmas 6.3 and 6.6 as those help make this new method even more attractive by reducing the number of gadgets needed and by being independent of HANANO.

Furthermore, we believe this new result about NCL should help researchers working on multiagent path planning/finding show the PSPACE-hardness of real-world problems they face that involve gravity.

As an open direction, it would be interesting to find out if we could further reduce the number of gadgets to two in the general case, or even down to one! Another interesting problem would be to show that our gadgets can be implemented with movable gray blocks that have only four sides (i.e., rectangles); we are seeing some levels of success along that latter problem in in-progress research.

Another rather natural future direction would be to apply this method to other games. We suspect that it can be used to study the complexity of Jelly-no-Puzzle, another game by Qrostar (Yang, 2018).

Bibliography

Aaronson, S. 2020. MIP* = RE. Shtetl-Optimized (the blog of Scott Aaronson), `www.scottaaronson.com/blog/?p=4512`.

Agrawal, M., N. Kayal, and N. Saxena. 2004. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793.

Allender, E. and R. Rubinstein. 1988. P-printable sets. *SIAM Journal on Computing*, 17(6):1193–1202.

Ani, J., J. Bosboom, E. Demaine, Y. Diomidov, D. Hendrickson, and J. Lynch. 2020. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *Proceedings of the 10th International Conference on Fun with Algorithms*, volume 157, pages 3:1–3:23.

Ani, J., L. Chung, E. Demaine, Y. Diomidov, D. Hendrickson, and J. Lynch. 2022. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude. In *Proceedings of the 11th International Conference on Fun with Algorithms*, volume 226, pages 3:1–3:30.

Arora, S. and B. Barak. 2009. *Computational Complexity: A Modern Approach*. Cambridge University Press.

Asif, S., M. Coulombe, E. Demaine, M. Demaine, A. Hesterberg, J. Lynch, and M. Singhal. 2020. Tetris is NP-hard even with $O(1)$ rows or columns. *Journal of Information Processing*, 28:942–958.

Babai, L. 1985. Trading group theory for randomness. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 421–429. ACM Press.

Babai, L. and L. Fortnow. 1991. Arithmetization: A new method in structural complexity theory. *Computational Complexity*, 1(1):41–66.

Bartholdi, J., III and J. Orlin. 1991. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354.

Bartholdi, J., III, C. Tovey, and M. Trick. 1989a. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241.

Bartholdi, J., III, C. Tovey, and M. Trick. 1989b. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165.

Bartholdi, J., III, C. Tovey, and M. Trick. 1992. How hard is it to control an election? *Mathematical and Computer Modeling*, 16(8–9):27–40.

Baumeister, D., G. Erdélyi, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. 2010. Computational aspects of approval voting. In J. Laslier and M. Sanver, editors, *Handbook on Approval Voting*, pages 199–251. Springer.

Beigel, R. 1989. On the relativized power of additional accepting paths. In *Proceedings of the 4th Structure in Complexity Theory Conference*, pages 216–224. IEEE Computer Society Press.

Bellare, M. and S. Goldwasser. 1994. The complexity of decision versus search. *SIAM Journal on Computing*, 23(1):97–119.

Ben-Sasson, E. and A. Wigderson. 2001. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169.

Berman, L. and J. Hartmanis. 1977. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322.

Biedl, T., E. Demaine, M. Demaine, R. Fleischer, L. Jacobson, and J. I. Munro. 2002. The complexity of Clickomania. In R. J. Nowakowski, editor, *More Games of No Chance*, pages 389–404. Cambridge University Press, Cambridge, England.

Blum, Manuel and Russel Impagliazzo. 1987. Generic oracles and oracle classes. In *Proceedings of the 28th IEEE Symposium on Foundations of Computer Science*, pages 118–126. IEEE Computer Society Press.

Book, R. 1974. Tally languages and complexity classes. *Information and Control*, 26(2):186–193.

Book, R., T. Long, and A. Selman. 1984. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13(3):461–487.

Borodin, A. and A. Demers. 1976. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR 76-284, Department of Computer Science, Cornell University, Ithaca, NY.

Breukelaar, R., E. Demaine, S. Hohenberger, H. Hoogeboom, W. Kosters, and D. Liben-Nowell. 2004. Tetris is hard, even to approximate. *International Journal of Computational Geometry & Applications*, 14(1–2):41–68.

Brocken, T., G. van der Heijden, I. Kostitsyna, L. Lo-Wong, and R. Surtel. 2020. Multi-robot motion planning of k-colored discs is PSPACE-hard. In *Proceedings of the 10th International Conference on Fun with Algorithms*, volume 157, pages 15:1–15:16.

Brunner, J., L. Chung, E. Demaine, D. Hendrickson, A. Hesterberg, A. Suhl, and A. Zeff. 2020. 1×1 Rush Hour with fixed blocks is PSPACE-complete. In *Proceedings of the 10th International Conference on Fun with Algorithms*, volume 157, pages 7:1–7:14.

Buhrman, H., L. Fortnow, and T. Thierauf. 1998. Nonrelativizing separations. In *Proceedings of the 13th Annual IEEE Conference on Computational Complexity*, pages 8–12. IEEE Computer Society Press.

Cabell, J. 1927. *The Silver Stallion*, volume 3. Robert M. McBride & Company.

Calviac, I., O. Sankur, and F. Schwarzentruber. 2023. Improved complexity results and an efficient solution for connected multi-agent path finding. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, pages 896–904. International Foundation for Autonomous Agents and Multiagent Systems.

Carleton, B., M. Chavrimootoo, L. Hemaspaandra, D. Narváez, C. Taliancich, and H. Welles. 2022a. Search versus search for collapsing electoral control types. Technical Report arXiv:2207.03049 [cs.GT], Computing Research Repository, arXiv.org/corr/. Revised, February 2024.

Carleton, B., M. Chavrimootoo, L. Hemaspaandra, D. Narváez, C. Taliancich, and H. Welles. 2022b. Separating and collapsing electoral control types. Technical Report arXiv:2207.00710 [cs.MA], Computing Research Repository, arXiv.org/corr/. Revised, February 2023.

Carleton, B., M. Chavrimootoo, L. Hemaspaandra, D. Narváez, C. Taliancich, and H. Welles. 2023a. Search versus search for collapsing electoral control types. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, pages 2682–2684. International Foundation for Autonomous Agents and Multiagent Systems.

Carleton, B., M. Chavrimootoo, L. Hemaspaandra, D. Narváez, C. Taliancich, and H. Welles. 2023b. Separating and collapsing electoral control types. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, pages 1743–1751. International Foundation for Autonomous Agents and Multiagent Systems.

Carleton, B., M. Chavrimootoo, L. Hemaspaandra, D. Narváez, C. Taliancich, and M. Welsh. 2024. Linked fates: How small of an ambiguity increase can make the difference between equaling and separating from P? In preparation.

Carleton, B., M. Chavrimootoo, and C. Taliancich. 2021. A critique of Keum-Bae Cho's proof that P $\subsetneq$ NP. Technical Report arXiv:2104.01736 [cs.CC], Computing Research Repository, arXiv.org/corr/.

Chavrimootoo, M. 2022. Defying gravity: The complexity of the Hanano Puzzle. Technical Report arXiv:2205.03400 [cs.CC], Computing Research Repository, arXiv.org/corr/. Revised, April 2023.

Chavrimootoo, M. 2023a. Defying gravity and gadget numerosity: The complexity of the Hanano Puzzle. In *Proceedings of the 25th International Conference on Descriptional Complexity of Formal Systems*, pages 36–50.

Chavrimootoo, M. 2023b. Separations and collapses in computational social choice. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, pages 3026–3028. International Foundation for Autonomous Agents and Multiagent Systems.

Chavrimootoo, M., I. Clingerman, and Q. Luu. 2023a. A critique of Sopin's "PH = PSPACE". Technical Report arXiv::2301.03487 [cs.CC], Computing Research Repository, arXiv.org/corr/.

Chavrimootoo, M., E. Ferland, E. Gibson, and A. Wilson. 2022. A closer look at some recent proof compression-related claims. Technical Report

arXiv:2212.12150 [cs.CC], Computing Research Repository, arXiv.org/corr/. Submitted to *Studia Logica* in January, 2023.

Chavrimootoo, M., Y. He, M. Kotler-Berkowitz, H. Liuson, and Z. Nie. 2023b. Evaluating the claims of "SAT requires exhaustive search". Technical Report arXiv:2312.02071 [cs.CC], Computing Research Repository, arXiv.org/corr/.

Chavrimootoo, M., T. Le, M. Reidy, and E. Smith. 2023c. On Czerwinski's "P $\neq$ NP relative to a P-complete oracle". Technical Report arXiv:2312.04395 [cs.CC], Computing Research Repository, arXiv.org/corr/.

Chavrimootoo, M. and H. Welles. 2021. A critique of Kumar's "Necessary and sufficient condition for satisfiability of a boolean formula in CNF and its implications on P versus NP problem". Technical Report arXiv:2112.06062 [cs.CC], Computing Research Repository, arXiv.org/corr/.

Cho, Keum-Bae. 2018. Indistinguishable binomial decision tree of 3-SAT: Proof of class P is a proper subset of class NP. Technical Report arXiv:1801.09673 [cs.CC], Computing Research Repository, arXiv.org/corr/. Version 1.

Czerwinski, R. 2023. P $\neq$ NP relative to a P-complete oracle. Technical Report arXiv:2305.02226 [cs.CC], Computing Research Repository, arXiv.org/corr/.

Demaine, E., I. Grosof, J. Lynch, and M. Rudoy. 2018. Complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms*, volume 100, pages 18:1–18:21.

Demaine, E., G. Viglietta, and A. Williams. 2016. Super Mario Bros. is harder/easier than we thought. In *Proceedings of the 9th International Conference on Fun with Algorithms*, pages 13:1–13:14.

Faliszewski, P., E. Hemaspaandra, and L. Hemaspaandra. 2015. Weighted electoral control. *Journal of Artificial Intelligence Research*, 52:507–542.

Faliszewski, P., E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. 2009. Llull and Copeland voting computationally resist bribery and constructive control. *Journal of Artificial Intelligence Research*, 35:275–341.

Fenner, S., L. Fortnow, S. Kurtz, and L. Li. 2003. An oracle builder's toolkit. *Information and Computation*, 182(2):95–136.

Fortnow, L. 2021. Worlds to die harder for: Open oracle questions for the 21st century. *SIGACT News*, 52(3):26–36.

Fortnow, L. and J. Rogers. 2002. Separability and one-way functions. *Computational Complexity*, 11(3–4):137–157.

Friedman, E. 2001. Cubic is NP-complete. In *34th Annual Floria MAA Section Meeting*.

Garey, M. and D. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Goldreich, O. 2008. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press.

Goldwasser, S., S. Micali, and C. Rackoff. 1985. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 291–304. ACM Press.

Gordeev, L. and E. Haeusler. 2019. Proof compression and NP versus PSPACE. *Studia Logica*, 107(1):53–83.

Gordeev, L. and E. Haeusler. 2020. Proof compression and NP versus PSPACE II. *Bulletin of the Section of Logic*, 49(3):213–230.

Gordeev, L. and E. Haeusler. 2022. Proof compression and NP versus PSPACE II: Addendum. *Bulletin of the Section of Logic*, 51(2):197–205.

Grollmann, J. and A. Selman. 1988. Complexity measures for public-key cryptosystems. *SIAM Journal on Computing*, 17(2):309–335.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science*, 39:297–308.

Hartmanis, J. 1978. *Feasible Computations and Provable Complexity Properties*. CBMS-NSF Regional Conference Series in Applied Mathematics #30. SIAM.

Hartmanis, J., R. Chang, S. Chari, D. Ranjan, and P. Rohatgi. 1992. Relativization: A revisionistic retrospective. *Bulletin of the EATCS*, 47:144–153.

Hartmanis, J. and L. Hemachandra. 1991. One-way functions and the nonisomorphism of NP-complete sets. *Theoretical Computer Science*, 81(1):155–163.

Hartmanis, J., N. Immerman, and V. Sewelson. 1985. Sparse sets in NP−P: EXPTIME versus NEXPTIME. *Information and Control*, 65(2–3):159–181.

Hearn, R. and E. Demaine. 2005. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96.

Hearn, R. and E. Demaine. 2009. *Games, Puzzles, and Computation*. CRC Press.

Hearn, R., E. Demaine, and M. Hoffmann. 2002. Push-2-F is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, pages 31–35.

Hemaspaandra, E., L. Hemaspaandra, and C. Menton. 2020. Search versus decision for election manipulation problems. *ACM Transactions on Computation*, 12:1–42.

Hemaspaandra, E., L. Hemaspaandra, and J. Rothe. 2007. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285.

Hemaspaandra, E., L. Hemaspaandra, and H. Schnoor. 2014. A control dichotomy for pure scoring rules. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 712–720. AAAI Press.

Hemaspaandra, E. and H. Schnoor. 2016. Dichotomy for pure scoring rules under manipulative electoral actions. In *Proceedings of the 22nd European Conference on Artificial Intelligence*, pages 1071–1079. IOS Press.

Hemaspaandra, L., M. Juvekar, A. Nadjimzadah, and P. Phillips. 2022. Gaps, ambiguity, and establishing complexity-class containments via iterative constant-setting. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science*, pages 57:1–57:15. Leibniz International Proceedings in Informatics (LIPIcs) #241.

Hemaspaandra, L. and D. Narváez. 2017. The opacity of backbones. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 3900–3906. AAAI Press.

Hemaspaandra, L. and M. Ogihara. 2002. *The Complexity Theory Companion*. Springer-Verlag.

Hemaspaandra, L., A. Ramachandran, and M. Zimand. 1995. Worlds to die for. *SIGACT News*, 26(4):5–15.

Hemaspaandra, L. and M. Zimand. 1993. Strong forms of balanced immunity. Technical Report TR-480, Department of Computer Science, University of Rochester, Rochester, NY. Revised, May 1994.

Henry (455). 2013. Levin reduction. `https://planetmath.org/levinreduction`.

Hopcroft, J. and J. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Ji, Z., A. Natarajan, T. Vidick, J. Wright, and H. Yuen. 2020. MIP* = RE. Technical Report arXiv:2001.04383 [quant-ph], Computing Research Repository, arXiv.org/corr/.

Kubo, T. 2015. *Bleach Volume 63: Hear, Fear, Here*, chapter 569: The White Haze, pages 153–170. Viz Media.

Lange, K.-J. and P. Rossmanith. 1994. Unambiguous polynomial hierarchies and exponential size. In *Proceedings of the 9th Structure in Complexity Theory Conference*, pages 106–115. IEEE Computer Society Press.

Liu, Z. and C. Yang. 2019. Hanano Puzzle is NP-hard. *Information Processing Letters*, 145:6–10.

Lockhart, P. 2009. *A Mathematician's Lament: How School Cheats Us Out of Our Most Fascinating and Imaginative Art Form*. Bellevue Literary Press.

Lund, C., L. Fortnow, H. Karloff, and N. Nisan. 1990. Algebraic methods for interactive proof systems. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, pages 2–10. IEEE Computer Society Press.

Lund, C., L. Fortnow, H. Karloff, and N. Nisan. 1992. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868.

Manoj, K. 2021. Necessary and sufficient condition for satisfiability of a boolean formula in CNF and its implications on P versus NP problem. Technical Report arXiv:2101.05597v3 [cs.CC], Computing Research Repository, arXiv.org/corr/. Revised, May 2021.

Maushagen, C. and J. Rothe. 2018. Complexity of control by partitioning veto elections and of control by adding candidates to plurality elections. *Annals of Mathematics and Artificial Intelligence*, 82(4):219–244.

Megiddo, N. and C. Papadimitriou. 1991. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324.

Papadimitriou, C. 1994. *Computational Complexity*. Addison-Wesley.

Piterman, N. and D. Fisman. 1998. Introduction to complexity theory Lecture 2: NP-completeness and self reducibility. Lecture Notes for a course given by Oded Goldreich, `https://www.wisdom.weizmann.ac.il/~oded/PS/CC/l2.ps`, notes taken by N. Piterman and D. Fisman.

Qrostar. 2011. Hanano Puzzle. `https://qrostar.skr.jp/en/hanano/`.

Qrostar. 2022. Personal communication.

Rackoff, C. 1982. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29(1):261–268.

Samuelson, P. 1938. A note on the pure theory of consumer's behaviour. *Economica*, 5(17):61–71.

Shamir, A. 1992. IP = PSPACE. *Journal of the ACM*, 39(4):869–877.

Sheu, M. and T. Long. 1996. UP and the low and high hierarchies: A relativized separation. *Mathematical Systems Theory*, 29(5):423–450.

Sipser, M. 2013. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition.

Sopin, V. 2014. PH = PSPACE. Technical Report arXiv:1411.0628v20 [cs.CC], Computing Research Repository, arXiv.org/corr/. Revised, November 2022.

Subercaseaux, B. and D. Lokshtanov. 2022. Wordle is NP-hard. In *Proceedings of the 11th International Conference on Fun with Algorithms*, volume 226, pages 19:1–19:8.

Tamassia, R. 2016. *Handbook of Graph Drawing and Visualization*. Chapman and Hall/CRC.

Tamassia, R. and I. Tollis. 1986. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1:321–341.

Tateo, D., J. Banfi, A. Riva, F. Amigoni, and A. Bonarini. 2018. Multiagent connected path planning: PSPACE-completeness and how to deal with it. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 4735–4742. AAAI Press.

Time. 1954. Cinema: To Aristophanes & back. *Time*, LXVII(20). No Author Listed.

Valiant, L. 1976. The relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23.

Vereshchagin, N. 1994. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Russian Academy of Sciences–Izvestiya–Mathematics*, 42(2):261–298.

Walsh, T. 2000. SAT v CSP. In *Proceedings of the 6th International Conference of Principles and Practice of Constraint Programming*, pages 441–456. Springer-Verlag Lecture Notes in Computer Science #1894.

Watanabe, O. 1988. On hardness of one-way functions. *Information Processing Letters*, 27(3):151–157.

Xu, K. and G. Zhou. 2023. SAT requires exhaustive search. Technical Report arXiv:2302.09512 [cs.CC], Computing Research Repository, arXiv.org/corr/. Revised September 21, 2023.

Yang, C. 2018. On the Complexity of Jelly-no-Puzzle. In *Japanese Conference on Discrete and Computational Geometry, Graphs, and Games*, pages 165–174. Springer.

A Tables Relating to Separating and Collapsing Electoral Control Types

This appendix serves two purposes. For one, it provides a list of all the separation witnesses (and which results they apply to), and also, it provides a tabulation of our results.

A.1 Compatible Control Types

Table A.1: The 44 types of control and a description of which components are part of the input for each one. The input type thus partitions the control types into five equivalence classes as to compatibility of inputs.

Control Type	Candidates	Votes	Focus Candidate	Spoiler Candidates	Spoiler Votes	Limit (Natural Number)
CC-PV-TE-UW, CC-PV-TE-NUW, CC-PV-TP-UW, CC-PV-TP-NUW, CC-PC-TE-UW, CC-PC-TE-NUW, CC-PC-TP-UW, CC-PC-TP-NUW, CC-RPC-TE-UW, CC-RPC-TE-NUW, CC-RPC-TP-UW, CC-RPC-TP-NUW, DC-PV-TE-UW, DC-PV-TE-NUW, DC-PV-TP-UW, DC-PV-TP-NUW,	Yes	Yes	Yes	No	No	No

Control Type	Candidates	Votes	Focus Candidate	Spoiler Candidates	Spoiler Votes	Limit (Natural Number)
DC-PC-TE-UW, DC-PC-TE-NUW, DC-PC-TP-UW, DC-PC-TP-NUW, DC-RPC-TE-UW, DC-RPC-TE-NUW, DC-RPC-TP-UW, DC-RPC-TP-NUW						
CC-AC-UW, CC-AC-NUW, DC-AC-UW, DC-AC-NUW	Yes	Yes	Yes	Yes	No	Yes
CC-DC-UW, CC-DC-NUW, CC-DV-UW, CC-DV-NUW, DC-DC-UW, DC-DC-NUW, DC-DV-UW, DC-DV-NUW	Yes	Yes	Yes	No	No	Yes
CC-AV-UW, CC-AV-NUW, DC-AV-UW, DC-AV-NUW	Yes	Yes	Yes	No	Yes	Yes
CC-UAC-UW, CC-UAC-NUW, DC-UAC-UW, DC-UAC-NUW	Yes	Yes	Yes	Yes	No	No

A.2 Tables

We use Tables A.3, A.5, and A.7 to list the separation witnesses used with respect to each election system (respectively, plurality, veto, and approval). Each witness appears on a different row. Each table has six columns: the ID column assigns a unique identifier to the separation witness presented in a given row, the columns C and V refer to the original candidate set and vote set of the election, the column S refers to the spoiler set of candidates, the column U refers to the spoiler set of votes, and the column k refers to the limit that the control type may impose (e.g., adding/delete candidates/voters). For each row, if a column is not relevant to the usage of that separation witness (for example, a separation witness of Plurality-CC-UAC-UW and Plurality-CC-UAC-NUW need not have k be specified), then the related entry is set to "-" to denote that it is not relevant.

Using those unique identifiers, we give our results within Tables A.4, A.6, and A.8. We do not explicitly state when we are appealing to the trivial containments that arise from the UW/NUW variants of the same control type.

Furthermore, to make it easier to read the table and identify equivalences, we color-code the table. The key for the color-coding is in Table A.2. Each equivalence class formed by collapsing control types is assigned a unique color. (As an aside, we note that we can certainly view control types with no assigned color, i.e., with a white background, as each forming an equivalence class of size one, but we omit that discussion as it is not particularly useful to consider here and in the next paragraph.)

Since we are dealing with equivalence classes, we find it useful to designate a canonical element for each equivalence class so as to save on effort. Indeed, doing so allowed us to focus on those canonical elements when working through our separation/collapse proofs instead of duplicating effort and studying multiple elements of the same equivalence class separately. We thus denote the canonical element of each equivalence class by boldfacing its name, making it easy to identify.

Finally, let us note that the font size of Tables A.4, A.6, and A.8 has been reduced to make the tables fit within the page in a way that retains readability.

Table A.2: Equivalence classes and their respective colors. Each boldfaced entry indicates the canonical element of its equivalence class.

Class	Color
Plurality-DC-RPC-TE-NUW, Plurality-DC-RPC-TE-UW, Plurality-DC-PC-TE-UW, Plurality-DC-PC-TE-NUW	
Plurality-DC-RPC-TP-NUW, Plurality-DC-PC-TP-NUW	
Veto-DC-RPC-TE-NUW, Veto-DC-RPC-TE-UW, Veto-DC-PC-TE-UW, Veto-DC-PC-TE-NUW	
Veto-DC-RPC-TP-NUW, Veto-DC-PC-TP-NUW	
Veto-DC-PV-TE-NUW, Veto-DC-PV-TE-UW	
Approval-DC-PV-TE-NUW, Approval-DC-PV-TE-UW	

Class	Color
Approval-DC-RPC-TE-NUW, Approval-DC-RPC-TE-UW, Approval-DC-PC-TE-UW, Approval-DC-PC-TE-NUW, Approval-DC-RPC-TP-UW, Approval-DC-PC-TP-UW	
Approval-DC-RPC-TP-NUW, Approval-DC-PC-TP-NUW	
Approval-CC-RPC-TP-UW, Approval-CC-PC-TP-UW	
Approval-CC-RPC-TP-NUW, Approval-CC-PC-TP-NUW	
Approval-CC-RPC-TE-UW, Approval-CC-PC-TE-UW	
Approval-CC-RPC-TE-NUW, Approval-CC-PC-TE-NUW	

A.2.1 Plurality Tables

Table A.3: List of separation witnesses in plurality. We note the computer-generated entries with a "†" superscript.

ID	C	S	V	U	k
Plur.1	$\{a,b,c\}$	-	$\{a>b>c, b>a>c, c>a>b\}$	-	-
Plur.2	$\{a,b\}$	-	$\{a>b, b>a\}$	-	-
Plur.3	$\{a,b\}$	-	$\{a>b\}$	-	-
Plur.4†	$\{a,b,c,d\}$	-	$\{b>c>d>a, d>a>c>b, b>c>d>a, a>c>b>d, a>b>d>c, d>a>b>c, c>d>b>a, d>a>c>b, a>c>b>d, d>c>b>a, b>c>d>a, a>b>d>c, d>b>c>a, a>d>c>b, b>c>d>a, c>a>b>d, b>a>d>c, a>c>d>b\}$	-	-
Plur.5	$\{a,b,c\}$	-	$\{a>b>c, a>b>c, a>c>b, a>c>b, b>a>c, b>a>c, b>a>c, c>a>b, c>a>b, c>a>b\}$	-	-
Plur.6	$\{a,b,c\}$	-	$\{b>c>a, c>b>a\}$	-	-
Plur.7	$\{a,b,c,d\}$	-	$\{a>b>c>d, c>d>a>b, d>b>a>c\}$	-	-
Plur.8	$\{a,b,c\}$	-	$\{a>b>c, a>b>c, b>a>c, b>a>c, c>b>a\}$	-	-
Plur.9	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, a>b>c>d, b>a>c>d, c>b>a>d, d>b>a>c\}$	-	-
Plur.10	$\{a,b,c\}$	-	$\{a>b>c, a>b>c, b>a>c, c>a>b\}$	-	-
Plur.11	$\{a,b,c,d,e\}$	-	$\{c>b>a>d>e, c>d>e>a>b, a>d>b>c>e, c>d>b>e>a, c>b>e>d>a, d>e>b>c>a, d>b>e>c>a, a>b>d>e>c, e>c>b>d>a, c>a>b>d>e, b>e>a>c>d, a>d>b>e>c, d>a>c>e>b, a>b>c>e>d, c>d>e>b>a, e>d>c>a>b, e>d>a>b>c\}$	-	-
Plur.12	$\{a,b,c\}$	-	$\{a>b>c, a>c>b, b>a>c, c>a>b\}$	-	-
Plur.13	$\{a,b,c\}$	-	$\{a>b>c, a>c>b, b>c>a, c>b>a\}$	-	-

ID	C	S	V	U	k
Plur.14	$\{a,b,c\}$	-	$\{a>b>c, a>c>b, b>c>a, b>c>a, b>c>a, c>b>a, c>b>a, c>b>a\}$	-	-
Plur.15	$\{a,b,c\}$	-	$\{a>b>c, a>c>b, b>a>c, b>c>a, c>b>a\}$	-	-
Plur.16	$\{a,b,c\}$	-	$\{a>b>c, a>c>b, b>a>c, b>c>a, c>a>b\}$	-	-
Plur.17	$\{a,b,c\}$	-	$\{a>b>c, a>c>b, b>a>c, b>c>a, c>a>b, c>b>a\}$	-	-
Plur.18	$\{a,b,c\}$	-	$\{a>b>c, b>c>a, b>c>a, c>b>a, c>b>a, c>b>a\}$	-	-
Plur.19	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, a>b>c>d, b>a>c>d, c>b>a>d, d>b>a>c\}$	-	-
Plur.20	$\{a,b,c,d\}$	-	$\{a>c>b>d, b>a>c>d, b>a>c>d, c>b>a>d, d>c>b>a\}$	-	-
Plur.21	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, b>c>a>d, b>c>a>d, c>d>b>a\}$	-	-
Plur.22	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, a>b>c>d, b>c>d>a, c>b>d>a, d>a>c>b, d>b>c>a, d>b>c>a\}$	-	-
Plur.23	$\{a,b,c\}$	-	$\{b>a>c, b>a>c, b>a>c, b>a>c, b>a>c, b>a>c, a>b>c, a>b>c, a>b>c, a>b>c, a>b>c, a>b>c, c>b>a, c>b>a, c>b>a\}$	-	-
Plur.24	$\{a,b,c,d\}$	-	$\{b>a>d>c, b>a>d>c, b>a>d>c, b>a>d>c, b>a>d>c, c>a>b>d, c>a>b>d, c>a>b>d, c>a>b>d, d>a>b>c, d>a>b>c\}$	-	-
Plur.25†	$\{a,b,c,d,e,f\}$	-	$\{d>e>b>f>c>a, b>f>c>a>e>d, b>e>c>a>d>f, f>e>a>b>d>c, b>a>e>d>f>c, a>c>d>e>b>f, c>e>f>b>a>d\}$	-	-
Plur.26†	$\{a,b,c,d,e,f,g\}$	-	$\{c>d>g>f>b>e>a, a>f>b>c>d>g>e, g>c>a>d>e>b>f, a>g>f>d>e>b>c, e>g>a>d>b>c>f, d>f>e>a>g>c>b, f>a>d>g>e>c>b, b>g>a>c>f>d>e, a>c>g>b>f>d>e\}$	-	-
Plur.27	$\{a,b\}$	$\{c\}$	$\{c>b>a\}$	-	1
Plur.28	$\{a,b\}$	-	$\{a>b\}$	$\{a>b\}$	1
Plur.29	$\{a,b\}$	$\{c\}$	$\{a>c>b\}$	-	1
Plur.30	$\{a,b\}$	-	$\{a>b, a>b\}$	-	0
Plur.31	$\{a,b,c,d\}$	-	$\{b>c>d>a, b>c>d>a, a>b>c>d\}$	-	2
Plur.32	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, a>b>c>d, a>b>c>d, a>b>c>d, a>b>c>d, b>c>d>a, b>c>d>a, b>c>d>a, c>b>d>a, c>b>d>a, c>b>d>a, d>b>c>a, d>b>c>a, d>b>c>a\}$	-	2
Plur.33	$\{a\}$	$\emptyset$	$\{a\}$	-	0
Plur.34	$\{a\}$	-	$\{a\}$	-	0
Plur.35	$\{a\}$	-	$\{a\}$	$\emptyset$	0
Plur.36	$\{a\}$	$\emptyset$	$\{a\}$	-	-
Plur.37	$\{a,b\}$	$\emptyset$	$\{a>b, b>a\}$	-	0
Plur.38	$\{a,b\}$	-	$\{a>b, b>a\}$	-	0
Plur.39	$\{a,b\}$	-	$\{a>b, b>a\}$	$\emptyset$	0
Plur.40	$\{a,b\}$	$\emptyset$	$\{a>b, b>a\}$	-	-
Plur.41	$\{a,b\}$	-	$\{b>a, b>a\}$	-	1
Plur.42	$\{a,b\}$	-	$\{a>b, a>b, b>a\}$	-	1
Plur.43	$\{a,b\}$	-	$\{a>b, b>a\}$	-	1
Plur.44	$\{a,b,c\}$	-	$\{a>b>c, b>c>a, b>c>a, c>b>a, c>b>a\}$	-	-
Plur.45	$\{a,b,c\}$	-	$\{a>b>c, b>c>a, c>b>a\}$	-	-
Plur.46	$\{a,b,c\}$	-	$\{a>b>c, b>c>a, b>c>a, b>c>a, c>b>a, c>b>a, c>b>a\}$	-	-
Plur.47†	$\{a,b,c,d\}$	-	$\{c>b>a>d, d>c>a>b, b>a>d>c, c>b>d>a, a>b>c>d, d>b>c>a, a>d>b>c\}$	-	-

ID	C	S	V	U	k
Plur.48†	$\{a, b, c, d, e\}$	-	$\{a > c > b > d > e, d > c > b > a > e, c > d > b > e > a, e > d > b > a > c, a > d > b > e > c, b > e > d > a > c, a > d > e > b > c, e > d > a > b > c, c > a > e > d > b, b > e > d > a > c, d > c > e > b > a\}$	-	-
Plur.49†	$\{a, b, c, d\}$	-	$\{c > a > b > d, b > a > c > d, c > b > a > d, b > d > c > a, d > a > b > c, c > b > d > a, a > d > b > c, a > b > d > c, c > d > a > b\}$	-	-
Plur.50†	$\{a, b, c, d, e\}$	-	$\{a > d > e > b > c, e > c > b > a > d, c > b > a > e > d, e > a > d > b > c, b > d > a > e > c, e > a > b > d > c, b > c > e > a > d, d > c > b > a > e, d > c > b > a > e\}$	-	-

Table A.4: Table of separations and collapses (here denoted by EQ) in plurality voting. The Classification column partitions each separation into one of the three cases, $\subsetneq$, $\supsetneq$, and INCOMP. (For cases of INCOMP for which we happen to have established that the separation holds with strong incomparability, we have noted that with a "$*$" superscript.)

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TE-UW	CC-PV-TE-NUW	$\subsetneq$	Plur.2
CC-PV-TE-UW	CC-PV-TP-UW	INCOMP	Plur.1 + Plur.50
CC-PV-TE-UW	CC-PV-TP-NUW	INCOMP	Plur.2 + Plur.46
CC-PV-TE-UW	CC-RPC-TE-UW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-UW	CC-RPC-TE-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-UW	CC-RPC-TP-UW	INCOMP	Plur.1 + Plur.24
CC-PV-TE-UW	CC-RPC-TP-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-UW	CC-PC-TE-UW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-UW	CC-PC-TE-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-UW	CC-PC-TP-UW	INCOMP	Plur.1 + Plur.24
CC-PV-TE-UW	CC-PC-TP-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-UW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PV-TE-UW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PV-TE-UW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PV-TE-UW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PV-TE-UW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PV-TE-NUW	CC-PV-TP-UW	INCOMP	Plur.2 + Plur.50
CC-PV-TE-NUW	CC-PV-TP-NUW	INCOMP	Plur.14 + Plur.50
CC-PV-TE-NUW	CC-RPC-TE-UW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-NUW	CC-RPC-TE-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-NUW	CC-RPC-TP-UW	INCOMP	Plur.2 + Plur.24
CC-PV-TE-NUW	CC-RPC-TP-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-NUW	CC-PC-TE-UW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-NUW	CC-PC-TE-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-NUW	CC-PC-TP-UW	INCOMP	Plur.2 + Plur.24
CC-PV-TE-NUW	CC-PC-TP-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TE-NUW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-PV-TP-UW	INCOMP*	Plur.3

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TE-NUW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PV-TE-NUW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PV-TE-NUW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PV-TE-NUW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PV-TP-UW	CC-PV-TP-NUW	$\subsetneq$	Plur.21
CC-PV-TP-UW	CC-RPC-TE-UW	INCOMP	Plur.20 + Plur.23
CC-PV-TP-UW	CC-RPC-TE-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TP-UW	CC-RPC-TP-UW	INCOMP	Plur.1 + Plur.23
CC-PV-TP-UW	CC-RPC-TP-NUW	INCOMP	Plur.2 + Plur.23
CC-PV-TP-UW	CC-PC-TE-UW	INCOMP	Plur.18 + Plur.23
CC-PV-TP-UW	CC-PC-TE-NUW	INCOMP	Plur.2 + Plur.23
CC-PV-TP-UW	CC-PC-TP-UW	INCOMP	Plur.1 + Plur.23
CC-PV-TP-UW	CC-PC-TP-NUW	INCOMP	Plur.2 + Plur.23
CC-PV-TP-UW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PV-TP-UW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PV-TP-UW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PV-TP-UW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PV-TP-UW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PV-TP-NUW	CC-RPC-TE-UW	INCOMP	Plur.2 + Plur.24
CC-PV-TP-NUW	CC-RPC-TE-NUW	INCOMP	Plur.18 + Plur.23
CC-PV-TP-NUW	CC-RPC-TP-UW	INCOMP	Plur.2 + Plur.24
CC-PV-TP-NUW	CC-RPC-TP-NUW	INCOMP	Plur.15 + Plur.24
CC-PV-TP-NUW	CC-PC-TE-UW	INCOMP	Plur.2 + Plur.24
CC-PV-TP-NUW	CC-PC-TE-NUW	INCOMP	Plur.18 + Plur.23
CC-PV-TP-NUW	CC-PC-TP-UW	INCOMP	Plur.2 + Plur.24
CC-PV-TP-NUW	CC-PC-TP-NUW	INCOMP	Plur.20 + Plur.23
CC-PV-TP-NUW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PV-TP-NUW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TP-NUW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PV-TP-NUW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PV-TP-NUW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-RPC-TE-UW	CC-RPC-TE-NUW	$\subsetneq$	Plur.2
CC-RPC-TE-UW	CC-RPC-TP-UW	INCOMP*	Plur.22
CC-RPC-TE-UW	CC-RPC-TP-NUW	INCOMP	Plur.2 + Plur.44
CC-RPC-TE-UW	CC-PC-TE-UW	INCOMP	Plur.22 + Plur.49
CC-RPC-TE-UW	CC-PC-TE-NUW	INCOMP	Plur.2 + Plur.49
CC-RPC-TE-UW	CC-PC-TP-UW	INCOMP	Plur.18 + Plur.47
CC-RPC-TE-UW	CC-PC-TP-NUW	INCOMP	Plur.2 + Plur.44
CC-RPC-TE-UW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-RPC-TE-UW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-RPC-TE-UW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-RPC-TE-UW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-RPC-TE-NUW	CC-RPC-TP-UW	INCOMP	Plur.2 + Plur.4
CC-RPC-TE-NUW	CC-RPC-TP-NUW	INCOMP	Plur.18 + Plur.45
CC-RPC-TE-NUW	CC-PC-TE-UW	INCOMP	Plur.2 + Plur.47
CC-RPC-TE-NUW	CC-PC-TE-NUW	INCOMP	Plur.15 + Plur.49
CC-RPC-TE-NUW	CC-PC-TP-UW	INCOMP	Plur.2 + Plur.47
CC-RPC-TE-NUW	CC-PC-TP-NUW	INCOMP	Plur.6 + Plur.45
CC-RPC-TE-NUW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-RPC-TE-NUW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-RPC-TE-NUW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-RPC-TE-NUW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-RPC-TP-UW	CC-RPC-TP-NUW	$\subsetneq$	Plur.2

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-RPC-TP-UW	CC-PC-TE-UW	INCOMP	Plur.18 + Plur.47
CC-RPC-TP-UW	CC-PC-TE-NUW	INCOMP	Plur.2 + Plur.49
CC-RPC-TP-UW	CC-PC-TP-UW	INCOMP	Plur.7 + Plur.47
CC-RPC-TP-UW	CC-PC-TP-NUW	INCOMP	Plur.2 + Plur.49
CC-RPC-TP-UW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-RPC-TP-UW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-RPC-TP-UW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-RPC-TP-UW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-RPC-TP-NUW	CC-PC-TE-UW	INCOMP	Plur.2 + Plur.9
CC-RPC-TP-NUW	CC-PC-TE-NUW	INCOMP	Plur.6 + Plur.49
CC-RPC-TP-NUW	CC-PC-TP-UW	INCOMP	Plur.2 + Plur.47
CC-RPC-TP-NUW	CC-PC-TP-NUW	INCOMP	Plur.8 + Plur.49
CC-RPC-TP-NUW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-RPC-TP-NUW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-RPC-TP-NUW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-RPC-TP-NUW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PC-TE-UW	CC-PC-TE-NUW	$\subsetneq$	Plur.2
CC-PC-TE-UW	CC-PC-TP-UW	INCOMP	Plur.17 + Plur.48
CC-PC-TE-UW	CC-PC-TP-NUW	INCOMP	Plur.2 + Plur.44
CC-PC-TE-UW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PC-TE-UW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PC-TE-UW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PC-TE-UW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PC-TE-UW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PC-TE-UW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PC-TE-UW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PC-TE-UW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PC-TE-UW	DC-PC-TE-UW	INCOMP*	Plur.3

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PC-TE-UW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PC-TE-UW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PC-TE-UW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PC-TE-NUW	CC-PC-TP-UW	INCOMP	Plur.2 + Plur.48
CC-PC-TE-NUW	CC-PC-TP-NUW	INCOMP	Plur.6 + Plur.48
CC-PC-TE-NUW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PC-TE-NUW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PC-TE-NUW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PC-TE-NUW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PC-TP-UW	CC-PC-TP-NUW	$\subsetneq$	Plur.2
CC-PC-TP-UW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PC-TP-UW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PC-TP-UW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PC-TP-UW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PC-TP-UW	DC-PC-TP-NUW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PV-TE-UW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PV-TE-NUW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PV-TP-UW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PV-TP-NUW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-RPC-TE-UW	INCOMP*	Plur.3
CC-PC-TP-NUW	**DC-RPC-TE-NUW**	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-RPC-TP-UW	INCOMP*	Plur.3
CC-PC-TP-NUW	**DC-RPC-TP-NUW**	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PC-TE-UW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PC-TE-NUW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PC-TP-UW	INCOMP*	Plur.3
CC-PC-TP-NUW	DC-PC-TP-NUW	INCOMP*	Plur.3
DC-PV-TE-UW	DC-PV-TE-NUW	$\supsetneq$	Plur.9
DC-PV-TE-UW	DC-PV-TP-UW	INCOMP	Plur.12 + Plur.26

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
DC-PV-TE-UW	DC-PV-TP-NUW	INCOMP	Plur.2 + Plur.26
DC-PV-TE-UW	DC-RPC-TE-UW	INCOMP	Plur.10 + Plur.25
DC-PV-TE-UW	**DC-RPC-TE-NUW**	INCOMP	Plur.10 + Plur.25
DC-PV-TE-UW	DC-RPC-TP-UW	INCOMP	Plur.16 + Plur.25
DC-PV-TE-UW	**DC-RPC-TP-NUW**	INCOMP	Plur.2 + Plur.25
DC-PV-TE-UW	DC-PC-TE-UW	INCOMP	Plur.10 + Plur.25
DC-PV-TE-UW	DC-PC-TE-NUW	INCOMP	Plur.10 + Plur.25
DC-PV-TE-UW	DC-PC-TP-UW	INCOMP	Plur.13 + Plur.25
DC-PV-TE-UW	DC-PC-TP-NUW	INCOMP	Plur.2 + Plur.25
DC-PV-TE-NUW	DC-PV-TP-UW	INCOMP	Plur.19 + Plur.10
DC-PV-TE-NUW	DC-PV-TP-NUW	INCOMP	Plur.2 + Plur.26
DC-PV-TE-NUW	DC-RPC-TE-UW	INCOMP	Plur.19 + Plur.10
DC-PV-TE-NUW	**DC-RPC-TE-NUW**	INCOMP	Plur.19 + Plur.10
DC-PV-TE-NUW	DC-RPC-TP-UW	INCOMP	Plur.16 + Plur.25
DC-PV-TE-NUW	**DC-RPC-TP-NUW**	INCOMP	Plur.2 + Plur.25
DC-PV-TE-NUW	DC-PC-TE-UW	INCOMP	Plur.19 + Plur.10
DC-PV-TE-NUW	DC-PC-TE-NUW	INCOMP	Plur.19 + Plur.10
DC-PV-TE-NUW	DC-PC-TP-UW	INCOMP	Plur.19 + Plur.10
DC-PV-TE-NUW	DC-PC-TP-NUW	INCOMP	Plur.2 + Plur.25
DC-PV-TP-UW	DC-PV-TP-NUW	$\supsetneq$	Plur.2
DC-PV-TP-UW	DC-RPC-TE-UW	INCOMP	Plur.5 + Plur.25
DC-PV-TP-UW	**DC-RPC-TE-NUW**	INCOMP	Plur.5 + Plur.25
DC-PV-TP-UW	DC-RPC-TP-UW	INCOMP	Plur.13 + Plur.25
DC-PV-TP-UW	**DC-RPC-TP-NUW**	INCOMP	Plur.2 + Plur.25
DC-PV-TP-UW	DC-PC-TE-UW	INCOMP	Plur.5 + Plur.25
DC-PV-TP-UW	DC-PC-TE-NUW	INCOMP	Plur.5 + Plur.25
DC-PV-TP-UW	DC-PC-TP-UW	INCOMP	Plur.13 + Plur.25
DC-PV-TP-UW	DC-PC-TP-NUW	INCOMP	Plur.2 + Plur.25
DC-PV-TP-NUW	DC-RPC-TE-UW	INCOMP	Plur.2 + Plur.5
DC-PV-TP-NUW	**DC-RPC-TE-NUW**	INCOMP	Plur.2 + Plur.5
DC-PV-TP-NUW	DC-RPC-TP-UW	INCOMP	Plur.2 + Plur.5
DC-PV-TP-NUW	**DC-RPC-TP-NUW**	INCOMP	Plur.1 + Plur.5
DC-PV-TP-NUW	DC-PC-TE-UW	INCOMP	Plur.2 + Plur.5
DC-PV-TP-NUW	DC-PC-TE-NUW	INCOMP	Plur.2 + Plur.5
DC-PV-TP-NUW	DC-PC-TP-UW	INCOMP	Plur.2 + Plur.5
DC-PV-TP-NUW	DC-PC-TP-NUW	INCOMP	Plur.1 + Plur.5
DC-RPC-TE-UW	**DC-RPC-TE-NUW**	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-UW	DC-RPC-TP-UW	$\supsetneq$	Plur.13 + Theorem 3.1
DC-RPC-TE-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Plur.2 + Theorem 3.1
DC-RPC-TE-UW	DC-PC-TE-UW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-UW	DC-PC-TE-NUW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-UW	DC-PC-TP-UW	$\supsetneq$	Plur.13 + Theorem 3.1
DC-RPC-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Plur.2 + Theorem 3.1
DC-RPC-TE-NUW	DC-RPC-TP-UW	$\supsetneq$	Plur.13 + Theorem 3.1

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
DC-RPC-TE-NUW	**DC-RPC-TP-NUW**	$\supsetneq$	Plur.2 + Theorem 3.1
DC-RPC-TE-NUW	DC-PC-TE-UW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-NUW	DC-PC-TE-NUW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-NUW	DC-PC-TP-UW	$\supsetneq$	Plur.13 + Theorem 3.1
DC-RPC-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Plur.2 + Theorem 3.1
DC-RPC-TP-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Plur.2 + Theorem 3.1
DC-RPC-TP-UW	DC-PC-TE-UW	$\subsetneq$	Plur.13 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-UW	DC-PC-TE-NUW	$\subsetneq$	Plur.13 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-UW	DC-PC-TP-UW	INCOMP	Plur.16 + Plur.11
DC-RPC-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Plur.2 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TE-UW	$\subsetneq$	Plur.2 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TE-NUW	$\subsetneq$	Plur.2 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TP-UW	$\subsetneq$	Plur.2 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TP-NUW	EQ	Hemaspaandra et al. (2020)
DC-PC-TE-UW	DC-PC-TE-NUW	EQ	Hemaspaandra et al. (2020)
DC-PC-TE-UW	DC-PC-TP-UW	$\supsetneq$	Plur.13 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Plur.2 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TE-NUW	DC-PC-TP-UW	$\supsetneq$	Plur.13 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Plur.2 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Plur.2
CC-AC-UW	CC-AC-NUW	$\subsetneq$	Plur.37
CC-AC-UW	DC-AC-UW	INCOMP	Plur.33 + Plur.27
CC-AC-UW	DC-AC-NUW	INCOMP	Plur.33 + Plur.27
CC-AC-NUW	DC-AC-UW	INCOMP	Plur.33 + Plur.27
CC-AC-NUW	DC-AC-NUW	INCOMP	Plur.33 + Plur.27
DC-AC-UW	DC-AC-NUW	$\supsetneq$	Plur.37
CC-DC-UW	CC-DC-NUW	$\subsetneq$	Plur.38
CC-DC-UW	CC-DV-UW	INCOMP	Plur.41 + Plur.31
CC-DC-UW	CC-DV-NUW	INCOMP	Plur.38 + Plur.41
CC-DC-UW	DC-DC-UW	INCOMP	Plur.38 + Plur.30
CC-DC-UW	DC-DC-NUW	INCOMP	Plur.34 + Plur.30
CC-DC-UW	DC-DV-UW	INCOMP	Plur.38 + Plur.30
CC-DC-UW	DC-DV-NUW	INCOMP	Plur.34 + Plur.30
CC-DC-NUW	CC-DV-UW	INCOMP	Plur.38 + Plur.31
CC-DC-NUW	CC-DV-NUW	INCOMP	Plur.41 + Plur.31
CC-DC-NUW	DC-DC-UW	INCOMP	Plur.34 + Plur.30
CC-DC-NUW	DC-DC-NUW	INCOMP	Plur.34 + Plur.30
CC-DC-NUW	DC-DV-UW	INCOMP	Plur.34 + Plur.30
CC-DC-NUW	DC-DV-NUW	INCOMP	Plur.34 + Plur.30
CC-DV-UW	CC-DV-NUW	$\subsetneq$	Plur.38
CC-DV-UW	DC-DC-UW	INCOMP	Plur.38 + Plur.30
CC-DV-UW	DC-DC-NUW	INCOMP	Plur.34 + Plur.30
CC-DV-UW	DC-DV-UW	INCOMP	Plur.38 + Plur.30
CC-DV-UW	DC-DV-NUW	INCOMP	Plur.34 + Plur.30

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-DV-NUW	DC-DC-UW	INCOMP	Plur.34 + Plur.30
CC-DV-NUW	DC-DC-NUW	INCOMP	Plur.34 + Plur.30
CC-DV-NUW	DC-DV-UW	INCOMP	Plur.34 + Plur.30
CC-DV-NUW	DC-DV-NUW	INCOMP	Plur.34 + Plur.30
DC-DC-UW	DC-DC-NUW	$\supsetneq$	Plur.38
DC-DC-UW	DC-DV-UW	INCOMP	Plur.42 + Plur.32
DC-DC-UW	DC-DV-NUW	INCOMP	Plur.38 + Plur.31
DC-DC-NUW	DC-DV-UW	INCOMP	Plur.38 + Plur.32
DC-DC-NUW	DC-DV-NUW	INCOMP	Plur.43 + Plur.32
DC-DV-UW	DC-DV-NUW	$\supsetneq$	Plur.38
CC-AV-UW	CC-AV-NUW	$\subsetneq$	Plur.39
CC-AV-UW	DC-AV-UW	INCOMP	Plur.39 + Plur.28
CC-AV-UW	DC-AV-NUW	INCOMP	Plur.35 + Plur.28
CC-AV-NUW	DC-AV-UW	INCOMP	Plur.35 + Plur.28
CC-AV-NUW	DC-AV-NUW	INCOMP	Plur.35 + Plur.28
DC-AV-UW	DC-AV-NUW	$\supsetneq$	Plur.39
CC-UAC-UW	CC-UAC-NUW	$\subsetneq$	Plur.40
CC-UAC-UW	DC-UAC-UW	INCOMP	Plur.40 + Plur.29
CC-UAC-UW	DC-UAC-NUW	INCOMP	Plur.36 + Plur.29
CC-UAC-NUW	DC-UAC-UW	INCOMP	Plur.36 + Plur.29
CC-UAC-NUW	DC-UAC-NUW	INCOMP	Plur.36 + Plur.29
DC-UAC-UW	DC-UAC-NUW	$\supsetneq$	Plur.40

A.2.2 Veto Tables

Table A.5: List of separation witnesses in veto. We note the computer-generated entries with a "†" superscript.

ID	C	S	V	U	k
Veto.1	$\{a, b\}$	-	$\{a > b, b > a\}$	-	-
Veto.2	$\{a, b, c\}$	-	$\{a > b > c, a > b > c\}$	-	-
Veto.3	$\{a, b, c\}$	-	$\{a > b > c, c > a > b, c > b > a, c > b > a\}$	-	-
Veto.4	$\{a, b, c\}$	-	$\{a > b > c\}$	-	-
Veto.5	$\{a, b\}$	-	$\{a > b\}$	-	-
Veto.6	$\{a, b\}$	-	$\{b > a\}$	-	-
Veto.7	$\{a, b, c\}$	-	$\{a > b > c, a > c > b, b > c > a, b > c > a\}$	-	-
Veto.8	$\{a, b, c\}$	-	$\{a > b > c, a > b > c, c > a > b, c > b > a, c > b > a\}$	-	-
Veto.9	$\{a, b, c, d\}$	-	$\{a > b > c > d, a > b > c > d, b > d > c > a\}$	-	-
Veto.10	$\{a, b, c\}$	-	$\{b > c > a, c > b > a\}$	-	-
Veto.11	$\{a, b, c\}$	-	$\{b > a > c\}$	-	-
Veto.12	$\{a, b, c, d\}$	-	$\{a > b > c > d, b > c > d > a, c > a > d > b\}$	-	-

ID	C	S	V	U	k
Veto.13	$\{a,b,c\}$	-	$\{a>b>c, a>b>c, a>b>c, c>a>b, c>a>b, c>b>a, c>b>a\}$	-	-
Veto.14	$\{a,b,c,d\}$	-	$\{c>d>a>b, c>d>a>b, d>b>a>c\}$	-	-
Veto.15	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, b>c>a>d, c>a>b>d, c>d>b>a\}$	-	-
Veto.16†	$\{a,b,c,d\}$	-	$\{a>b>c>d, b>a>c>d, b>a>c>d, b>a>c>d, c>b>a>d, d>c>a>b, d>c>a>b, d>c>a>b, d>c>b>a, d>c>b>a\}$	-	-
Veto.17	$\{a,b,c\}$	-	$\{b>a>c, c>a>b\}$	-	-
Veto.18	$\{a,b,c\}$	-	$\{a>b>c, a>b>c, a>b>c, a>c>b, a>c>b, a>c>b, b>c>a, b>c>a\}$	-	-
Veto.19	$\{a,b,c,d\}$	-	$\{a>b>c>d, b>d>a>c, c>d>a>b\}$	-	-
Veto.20	$\{a,b,c\}$	-	$\{a>b>c, c>a>b\}$	-	-
Veto.21	$\{a,b\}$	$\{c\}$	$\{b>a>c\}$	-	1
Veto.22	$\{a,b,c\}$	$\{d\}$	$\{b>a>c>d\}$	-	1
Veto.23	$\{a,b\}$	$\{c\}$	$\{a>b>c\}$	-	1
Veto.24	$\{a,b\}$	$\{c\}$	$\{b>c>a, c>b>a\}$	-	1
Veto.25	$\{a,b,c,d\}$	-	$\{d>c>a>b\}$	-	1
Veto.26	$\{a,b,c\}$	-	$\{a>b>c, a>b>c, c>a>b, c>b>a, c>b>a\}$	-	0
Veto.27	$\{a,b,c\}$	-	$\{a>c>b, a>c>b\}$	-	1
Veto.28	$\{a,b,c\}$	-	$\{c>a>b, c>a>b\}$	-	1
Veto.29	$\{a,b\}$	-	$\{a>b\}$	-	1
Veto.30	$\{a,b,c\}$	-	$\{c>b>a, c>a>b, b>a>c\}$	-	1
Veto.31	$\{a,b,c\}$	-	$\{a>c>b, a>c>b, a>c>b, c>b>a, c>b>a\}$	-	1
Veto.32	$\{a,b,c,d\}$	-	$\{a>b>c>d, a>b>c>d, b>d>c>a\}$	-	0
Veto.33	$\{a,b,c\}$	-	$\{b>c>a, b>c>a, b>a>c, b>a>c, a>c>b\}$	-	2
Veto.34	$\{a,b,c\}$	-	$\{c>b>a, c>b>a, b>c>a\}$	-	1
Veto.35	$\{a,b,c\}$	-	$\{c>a>b\}$	-	1
Veto.36	$\{a,b,c\}$	-	$\{c>a>b\}$	$\{c>a>b\}$	1
Veto.37	$\{a,b\}$	-	$\{a>b\}$	$\{a>b\}$	1
Veto.38	$\{a,b\}$	-	$\{b>a\}$	$\{b>a\}$	1
Veto.39	$\{a,b,c\}$	$\{d\}$	$\{d>c>a>b\}$	-	-
Veto.40	$\{a,b\}$	$\{c\}$	$\{a>c>b, a>b>c\}$	-	-
Veto.41	$\{a,b\}$	$\{c\}$	$\{c>b>a\}$	-	-
Veto.42	$\{a,b,c\}$	-	$\{a>b>c, a>c>b\}$	-	-
Veto.43†	$\{a,b,c,d\}$	-	$\{c>d>b>a, a>b>c>d, b>d>a>c, b>d>c>a, a>b>d>c, a>b>d>c\}$	-	-
Veto.44	$\{a,b,c\}$	-	$\{a>b>c\}$	$\emptyset$	0
Veto.45	$\{a,b\}$	-	$\{a>b\}$	-	0
Veto.46	$\{a,b,c,d,e\}$	-	$\{b>c>d>e>a, b>c>d>e>a, d>b>c>a>e, e>b>c>a>d, e>c>d>a>b, e>b>d>a>c\}$	-	2

Table A.6: Table of separations and collapses (here denoted by EQ) in veto voting. The Classification column partitions each separation into one of the three cases, $\subsetneq$, $\supsetneq$, and INCOMP. (For cases of INCOMP for which we happen to have established that the separation holds with strong incomparability, we have noted that with a "$*$" superscript.)

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TE-UW	CC-PV-TE-NUW	$\subsetneq$	Veto.1
CC-PV-TE-UW	CC-PV-TP-UW	INCOMP	Veto.2 + Veto.3
CC-PV-TE-UW	CC-PV-TP-NUW	INCOMP	Veto.1 + Veto.3
CC-PV-TE-UW	CC-RPC-TE-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-UW	CC-RPC-TE-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TE-UW	CC-RPC-TP-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-UW	CC-RPC-TP-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TE-UW	CC-PC-TE-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-UW	CC-PC-TE-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TE-UW	CC-PC-TP-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-UW	CC-PC-TP-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TE-UW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-UW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	CC-PV-TP-UW	INCOMP	Veto.1 + Veto.2
CC-PV-TE-NUW	CC-PV-TP-NUW	INCOMP	Veto.4 + Veto.7
CC-PV-TE-NUW	CC-RPC-TE-UW	INCOMP	Veto.1 + Veto.2
CC-PV-TE-NUW	CC-RPC-TE-NUW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-NUW	CC-RPC-TP-UW	INCOMP	Veto.1 + Veto.2
CC-PV-TE-NUW	CC-RPC-TP-NUW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-NUW	CC-PC-TE-UW	INCOMP	Veto.1 + Veto.2
CC-PV-TE-NUW	CC-PC-TE-NUW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-NUW	CC-PC-TP-UW	INCOMP	Veto.1 + Veto.2
CC-PV-TE-NUW	CC-PC-TP-NUW	INCOMP	Veto.4 + Veto.8
CC-PV-TE-NUW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TE-NUW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TE-NUW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
breakpage CC-PV-TE-NUW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	CC-PV-TP-NUW	$\subsetneq$	Veto.1
CC-PV-TP-UW	CC-RPC-TE-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TP-UW	CC-RPC-TE-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TP-UW	CC-RPC-TP-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TP-UW	CC-RPC-TP-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TP-UW	CC-PC-TE-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TP-UW	CC-PC-TE-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TP-UW	CC-PC-TP-UW	INCOMP	Veto.4 + Veto.8
CC-PV-TP-UW	CC-PC-TP-NUW	INCOMP	Veto.1 + Veto.8
CC-PV-TP-UW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-UW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	CC-RPC-TE-UW	INCOMP	Veto.1 + Veto.9
CC-PV-TP-NUW	CC-RPC-TE-NUW	INCOMP	Veto.10 + Veto.8
CC-PV-TP-NUW	CC-RPC-TP-UW	INCOMP	Veto.1 + Veto.9
CC-PV-TP-NUW	CC-RPC-TP-NUW	INCOMP	Veto.11 + Veto.9
CC-PV-TP-NUW	CC-PC-TE-UW	INCOMP	Veto.1 + Veto.9
CC-PV-TP-NUW	CC-PC-TE-NUW	INCOMP	Veto.10 + Veto.8
CC-PV-TP-NUW	CC-PC-TP-UW	INCOMP	Veto.1 + Veto.9
CC-PV-TP-NUW	CC-PC-TP-NUW	INCOMP	Veto.12 + Veto.8
CC-PV-TP-NUW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TP-NUW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PV-TP-NUW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	CC-RPC-TE-NUW	$\subsetneq$	Veto.1
CC-RPC-TE-UW	CC-RPC-TP-UW	INCOMP	Veto.10 + Veto.13
CC-RPC-TE-UW	CC-RPC-TP-NUW	INCOMP	Veto.1 + Veto.10
CC-RPC-TE-UW	CC-PC-TE-UW	INCOMP	Veto.12 + Veto.14
CC-RPC-TE-UW	CC-PC-TE-NUW	INCOMP	Veto.1 + Veto.15
CC-RPC-TE-UW	CC-PC-TP-UW	INCOMP	Veto.10 + Veto.13
CC-RPC-TE-UW	CC-PC-TP-NUW	INCOMP	Veto.1 + Veto.10
CC-RPC-TE-UW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-UW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	CC-RPC-TP-UW	INCOMP	Veto.1 + Veto.13
CC-RPC-TE-NUW	CC-RPC-TP-NUW	INCOMP	Veto.10 + Veto.13
CC-RPC-TE-NUW	CC-PC-TE-UW	INCOMP	Veto.1 + Veto.14
CC-RPC-TE-NUW	CC-PC-TE-NUW	INCOMP	Veto.11 + Veto.15
CC-RPC-TE-NUW	CC-PC-TP-UW	INCOMP	Veto.1 + Veto.13
CC-RPC-TE-NUW	CC-PC-TP-NUW	INCOMP	Veto.10 + Veto.11
CC-RPC-TE-NUW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TE-NUW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	CC-RPC-TP-NUW	$\subsetneq$	Veto.1

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-RPC-TP-UW	CC-PC-TE-UW	INCOMP	Veto.10 + Veto.12
CC-RPC-TP-UW	CC-PC-TE-NUW	INCOMP	Veto.1 + Veto.15
CC-RPC-TP-UW	CC-PC-TP-UW	INCOMP	Veto.12 + Veto.14
CC-RPC-TP-UW	CC-PC-TP-NUW	INCOMP	Veto.1 + Veto.15
CC-RPC-TP-UW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-UW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	CC-PC-TE-UW	INCOMP	Veto.1 + Veto.10
CC-RPC-TP-NUW	CC-PC-TE-NUW	INCOMP	Veto.10 + Veto.15
CC-RPC-TP-NUW	CC-PC-TP-UW	INCOMP	Veto.1 + Veto.14
CC-RPC-TP-NUW	CC-PC-TP-NUW	INCOMP	Veto.11 + Veto.15
CC-RPC-TP-NUW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-RPC-TP-NUW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	CC-PC-TE-NUW	$\subsetneq$	Veto.1
CC-PC-TE-UW	CC-PC-TP-UW	INCOMP	Veto.10 + Veto.13
CC-PC-TE-UW	CC-PC-TP-NUW	INCOMP	Veto.1 + Veto.10
CC-PC-TE-UW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PC-TE-UW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-UW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	CC-PC-TP-UW	INCOMP	Veto.1 + Veto.16
CC-PC-TE-NUW	CC-PC-TP-NUW	INCOMP	Veto.10 + Veto.16
CC-PC-TE-NUW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TE-NUW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	CC-PC-TP-NUW	$\subsetneq$	Veto.1
CC-PC-TP-UW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-UW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PV-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	**DC-PV-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PV-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PV-TP-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-RPC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-RPC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PC-TE-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PC-TE-NUW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PC-TP-UW	INCOMP	Veto.5 + Veto.6
CC-PC-TP-NUW	DC-PC-TP-NUW	INCOMP	Veto.5 + Veto.6
DC-PV-TE-UW	**DC-PV-TE-NUW**	EQ	Theorem 3.2
DC-PV-TE-UW	DC-PV-TP-UW	$\supsetneq$	Veto.42 + Theorem 3.3

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
DC-PV-TE-UW	DC-PV-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.3
DC-PV-TE-UW	DC-RPC-TE-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-UW	**DC-RPC-TE-NUW**	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-UW	DC-RPC-TP-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Veto.1 + Theorem 3.4
DC-PV-TE-UW	DC-PC-TE-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-UW	DC-PC-TE-NUW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-UW	DC-PC-TP-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.4 + Hemaspaandra et al. (2020)
DC-PV-TE-NUW	DC-PV-TP-UW	$\supsetneq$	Veto.42 + Theorem 3.3
DC-PV-TE-NUW	DC-PV-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.3
DC-PV-TE-NUW	DC-RPC-TE-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-NUW	**DC-RPC-TE-NUW**	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-NUW	DC-RPC-TP-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-NUW	**DC-RPC-TP-NUW**	$\supsetneq$	Veto.1 + Theorem 3.4
DC-PV-TE-NUW	DC-PC-TE-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-NUW	DC-PC-TE-NUW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-NUW	DC-PC-TP-UW	$\supsetneq$	Veto.42 + Theorem 3.4
DC-PV-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.4 + Hemaspaandra et al. (2020)
DC-PV-TP-UW	DC-PV-TP-NUW	$\supsetneq$	Veto.1
DC-PV-TP-UW	DC-RPC-TE-UW	INCOMP	Veto.17 + Veto.18
DC-PV-TP-UW	**DC-RPC-TE-NUW**	INCOMP	Veto.17 + Veto.18
DC-PV-TP-UW	DC-RPC-TP-UW	INCOMP	Veto.4 + Veto.20
DC-PV-TP-UW	**DC-RPC-TP-NUW**	INCOMP	Veto.1 + Veto.19
DC-PV-TP-UW	DC-PC-TE-UW	INCOMP	Veto.17 + Veto.18
DC-PV-TP-UW	DC-PC-TE-NUW	INCOMP	Veto.17 + Veto.18
DC-PV-TP-UW	DC-PC-TP-UW	INCOMP	Veto.20 + Veto.18
DC-PV-TP-UW	DC-PC-TP-NUW	INCOMP	Veto.1 + Veto.19
DC-PV-TP-NUW	DC-RPC-TE-UW	INCOMP	Veto.1 + Veto.18
DC-PV-TP-NUW	**DC-RPC-TE-NUW**	INCOMP	Veto.1 + Veto.18
DC-PV-TP-NUW	DC-RPC-TP-UW	INCOMP	Veto.1 + Veto.18
DC-PV-TP-NUW	**DC-RPC-TP-NUW**	INCOMP	Veto.11 + Veto.18
DC-PV-TP-NUW	DC-PC-TE-UW	INCOMP	Veto.1 + Veto.18
DC-PV-TP-NUW	DC-PC-TE-NUW	INCOMP	Veto.1 + Veto.18
DC-PV-TP-NUW	DC-PC-TP-UW	INCOMP	Veto.1 + Veto.18
DC-PV-TP-NUW	DC-PC-TP-NUW	INCOMP	Veto.11 + Veto.18
DC-RPC-TE-UW	**DC-RPC-TE-NUW**	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-UW	DC-RPC-TP-UW	$\supsetneq$	Veto.4 + Theorem 3.1
DC-RPC-TE-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Veto.1 + Theorem 3.1
DC-RPC-TE-UW	DC-PC-TE-UW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-UW	DC-PC-TE-NUW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-UW	DC-PC-TP-UW	$\supsetneq$	Veto.17 + Theorem 3.1
DC-RPC-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.1
DC-RPC-TE-NUW	DC-RPC-TP-UW	$\supsetneq$	Veto.4 + Theorem 3.1

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
DC-RPC-TE-NUW	**DC-RPC-TP-NUW**	$\supsetneq$	Veto.1 + Theorem 3.1
DC-RPC-TE-NUW	DC-PC-TE-UW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-NUW	DC-PC-TE-NUW	EQ	Hemaspaandra et al. (2020)
DC-RPC-TE-NUW	DC-PC-TP-UW	$\supsetneq$	Veto.17 + Theorem 3.1
DC-RPC-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.1
DC-RPC-TP-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Veto.1
DC-RPC-TP-UW	DC-PC-TE-UW	$\subsetneq$	Veto.4 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-UW	DC-PC-TE-NUW	$\subsetneq$	Veto.4 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-UW	DC-PC-TP-UW	INCOMP	Veto.4 + Veto.43
DC-RPC-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TE-UW	$\subsetneq$	Veto.1 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TE-NUW	$\subsetneq$	Veto.1 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TP-UW	$\subsetneq$	Veto.1 + Hemaspaandra et al. (2020)
DC-RPC-TP-NUW	DC-PC-TP-NUW	EQ	Hemaspaandra et al. (2020)
DC-PC-TE-UW	DC-PC-TE-NUW	EQ	Hemaspaandra et al. (2020)
DC-PC-TE-UW	DC-PC-TP-UW	$\supsetneq$	Veto.17 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TE-NUW	DC-PC-TP-UW	$\supsetneq$	Veto.17 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Veto.1 + Theorem 3.1 + Hemaspaandra et al. (2020)
DC-PC-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Veto.1
CC-AC-UW	CC-AC-NUW	$\subsetneq$	Veto.22
CC-AC-UW	DC-AC-UW	INCOMP	Veto.22 + Veto.24
CC-AC-UW	DC-AC-NUW	INCOMP	Veto.21 + Veto.23
CC-AC-NUW	DC-AC-UW	INCOMP*	Veto.26
CC-AC-NUW	DC-AC-NUW	INCOMP*	Veto.32
DC-AC-UW	DC-AC-NUW	$\supsetneq$	Veto.22
CC-DC-UW	CC-DC-NUW	$\subsetneq$	Veto.25
CC-DC-UW	CC-DV-UW	INCOMP	Veto.27 + Veto.30
CC-DC-UW	CC-DV-NUW	INCOMP	Veto.25 + Veto.31
CC-DC-UW	DC-DC-UW	INCOMP*	Veto.26
CC-DC-UW	DC-DC-NUW	INCOMP	Veto.25 + Veto.34
CC-DC-UW	DC-DV-UW	INCOMP*	Veto.45
CC-DC-UW	DC-DV-NUW	INCOMP	Veto.25 + Veto.34
CC-DC-NUW	CC-DV-UW	INCOMP	Veto.27 + Veto.46
CC-DC-NUW	CC-DV-NUW	INCOMP	Veto.35 + Veto.31
CC-DC-NUW	DC-DC-UW	INCOMP	Veto.25 + Veto.34
CC-DC-NUW	DC-DC-NUW	INCOMP	Veto.25 + Veto.34
CC-DC-NUW	DC-DV-UW	INCOMP	Veto.25 + Veto.34
CC-DC-NUW	DC-DV-NUW	INCOMP	Veto.25 + Veto.34
CC-DV-UW	CC-DV-NUW	$\subsetneq$	Veto.27
CC-DV-UW	DC-DC-UW	INCOMP*	Veto.45
CC-DV-UW	DC-DC-NUW	INCOMP	Veto.25 + Veto.28
CC-DV-UW	DC-DV-UW	INCOMP*	Veto.45
CC-DV-UW	DC-DV-NUW	INCOMP	Veto.25 + Veto.34

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-DV-NUW	DC-DC-UW	INCOMP*	Veto.26
CC-DV-NUW	DC-DC-NUW	INCOMP	Veto.25 + Veto.34
CC-DV-NUW	DC-DV-UW	INCOMP*	Veto.26
CC-DV-NUW	DC-DV-NUW	INCOMP	Veto.25 + Veto.34
DC-DC-UW	DC-DC-NUW	$\supsetneq$	Veto.27
DC-DC-UW	DC-DV-UW	INCOMP	Veto.29 + Veto.31
DC-DC-UW	DC-DV-NUW	INCOMP	Veto.27 + Veto.33
DC-DC-NUW	DC-DV-UW	INCOMP	Veto.27 + Veto.31
DC-DC-NUW	DC-DV-NUW	INCOMP	Veto.28 + Veto.33
DC-DV-UW	DC-DV-NUW	$\supsetneq$	Veto.27
CC-AV-UW	CC-AV-NUW	$\subsetneq$	Veto.44
CC-AV-UW	DC-AV-UW	INCOMP	Veto.37 + Veto.38
CC-AV-UW	DC-AV-NUW	INCOMP	Veto.37 + Veto.38
CC-AV-NUW	DC-AV-UW	INCOMP	Veto.37 + Veto.38
CC-AV-NUW	DC-AV-NUW	INCOMP	Veto.37 + Veto.38
DC-AV-UW	DC-AV-NUW	$\supsetneq$	Veto.36
CC-UAC-UW	CC-UAC-NUW	$\subsetneq$	Veto.39
CC-UAC-UW	DC-UAC-UW	INCOMP	Veto.39 + Veto.40
CC-UAC-UW	DC-UAC-NUW	INCOMP	Veto.40 + Veto.41
CC-UAC-NUW	DC-UAC-UW	INCOMP	Veto.40 + Veto.41
CC-UAC-NUW	DC-UAC-NUW	INCOMP	Veto.39 + Veto.41
DC-UAC-UW	DC-UAC-NUW	$\supsetneq$	Veto.39

A.2.3 Approval-Voting Tables

Table A.7: List of separation witnesses in approval voting. (Example Appr.7 can also be found on page 220 of the Handbook of Approval Voting by Baumeister et al. 2010.)

ID	C	S	V	U	k
Appr.1	$\{a, b\}$	-	$\{10\}$	-	-
Appr.2	$\{a, b\}$	-	$\{10, 01\}$	-	-
Appr.3	$\{a, b, c\}$	-	$\{101, 110\}$	-	-
Appr.4	$\{a, b, c\}$	-	$\{110, 110, 010, 101, 101, 001\}$	-	-
Appr.5	$\{a, b, c\}$	-	$\{100, 011, 011\}$	-	-
Appr.6	$\{a, b, c\}$	-	$\{100, 110, 011, 011\}$	-	-
Appr.7	$\{a, b, c\}$	-	{100, 100, 100, 100, 100, 110, 010, 010, 010, 010, 001, 001, 001, 001, 001, 001, 001}	-	-
Appr.8	$\{a, b, c\}$	-	{100, 100, 100, 100, 010, 010, 010, 010, 010, 001, 001, 001}	-	-
Appr.9	$\{a, b, c, d\}$	-	{1001, 1001, 1001, 1000, 0100, 0100, 0100, 0100, 0100, 0010, 0010, 0010, 0010, 0010}	-	-
Appr.10	$\{a, b\}$	-	$\{10\}$	-	0

ID	C	S	V	U	k
Appr.11	$\{a,b\}$	$\{c\}$	$\{111\}$	-	1
Appr.12	$\{a,b\}$	$\{c\}$	$\{010\}$	-	1
Appr.13	$\{a,b\}$	$\{c\}$	$\{100\}$	-	1
Appr.14	$\{a,b,c\}$	-	$\{111\}$	-	1
Appr.15	$\{a,b\}$	-	$\{11\}$	-	1
Appr.16	$\{a,b,c\}$	-	$\{011\}$	-	1
Appr.17	$\{a,b,c\}$	-	$\{011,011\}$	-	1
Appr.18	$\{a,b,c\}$	-	$\{100,111\}$	-	1
Appr.19	$\{a,b,c\}$	-	$\{100,011\}$	-	1
Appr.20	$\{a,b\}$	-	$\{10,01,01,01\}$	-	1
Appr.21	$\{a,b,c,d\}$	-	$\{1000,0111,0111\}$	-	2
Appr.22	$\{a,b\}$	-	$\{10,10,01\}$	-	2
Appr.23	$\{a\}$	-	$\{1\}$	-	1
Appr.24	$\{a,b\}$	-	$\{01\}$	$\{10\}$	1
Appr.25	$\{a,b\}$	-	$\{10\}$	$\{10\}$	1
Appr.26	$\{a,b\}$	-	$\{01\}$	$\{01\}$	1
Appr.27	$\{a,b\}$	-	$\{10,01\}$	$\emptyset$	0
Appr.28	$\{a,b\}$	$\{c\}$	$\{100\}$	-	-
Appr.29	$\{a,b,c\}$	-	$\{101,110\}$	-	0
Appr.30	$\{a,b\}$	$\emptyset$	$\{10,01\}$	-	-
Appr.31	$\{a,b,c,d,$ $e,f,g,h\}$	-	$\{10111100,10111100,11100000,01000001,01000001,00010001,00001011,00000111,$ $10111110,11011110\}$	-	-
Appr.32	$\{a,b,c,d,$ $e,f,g,h,$ $i,j\}$	-	$\{1110111100,1110111100,1110111100,1111000001,0001000001,0001000001,0001000001,$ $0000100001,0000010011,0000001011,0000000111,1011111110,1101111110,1110111110\}$	-	-

Table A.8: Table of separations and collapses (here denoted by EQ) in approval voting. The Classification column partitions each separation into one of the three cases, $\subsetneq$, $\supsetneq$, and INCOMP. (For cases of INCOMP for which we happen to have established that the separation holds with strong incomparability, we have noted that with a "$*$" superscript.)

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TE-UW	CC-PV-TE-NUW	$\subsetneq$	Appr.2
CC-PV-TE-UW	CC-PV-TP-UW	INCOMP	Appr.5 + Appr.31
CC-PV-TE-UW	CC-PV-TP-NUW	INCOMP	Appr.2 + Appr.5
CC-PV-TE-UW	**CC-RPC-TE-UW**	INCOMP	Appr.6 + Appr.9
CC-PV-TE-UW	**CC-RPC-TE-NUW**	INCOMP	Appr.2 + Appr.6
CC-PV-TE-UW	**CC-RPC-TP-UW**	$\supsetneq$	Appr.5 + Corollary 3.10
CC-PV-TE-UW	**CC-RPC-TP-NUW**	INCOMP	Appr.2 + Appr.5
CC-PV-TE-UW	CC-PC-TE-UW	INCOMP	Appr.6 + Appr.9

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TE-UW	CC-PC-TE-NUW	INCOMP	Appr.2 + Appr.6
CC-PV-TE-UW	CC-PC-TP-UW	$\supsetneq$	Appr.5 + Corollary 3.10
CC-PV-TE-UW	CC-PC-TP-NUW	INCOMP	Appr.2 + Appr.5
CC-PV-TE-UW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PV-TE-UW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PV-TE-UW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PV-TE-UW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PV-TE-UW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PV-TE-UW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PV-TE-UW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PV-TE-UW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PV-TE-UW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PV-TE-UW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PV-TE-UW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PV-TE-UW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PV-TE-NUW	CC-PV-TP-UW	INCOMP	Appr.2 + Appr.32
CC-PV-TE-NUW	CC-PV-TP-NUW	INCOMP*	Appr.5
CC-PV-TE-NUW	**CC-RPC-TE-UW**	INCOMP	Appr.2 + Appr.9
CC-PV-TE-NUW	**CC-RPC-TE-NUW**	INCOMP	Appr.6 + Appr.5
CC-PV-TE-NUW	**CC-RPC-TP-UW**	$\supsetneq$	Appr.2 + Corollary 3.10
CC-PV-TE-NUW	**CC-RPC-TP-NUW**	INCOMP*	Appr.5
CC-PV-TE-NUW	CC-PC-TE-UW	INCOMP	Appr.2 + Appr.9
CC-PV-TE-NUW	CC-PC-TE-NUW	INCOMP	Appr.6 + Appr.5
CC-PV-TE-NUW	CC-PC-TP-UW	$\supsetneq$	Appr.2 + Corollary 3.10
CC-PV-TE-NUW	CC-PC-TP-NUW	INCOMP*	Appr.5
CC-PV-TE-NUW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PV-TE-NUW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PV-TE-NUW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PV-TE-NUW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PV-TE-NUW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PV-TP-UW	CC-PV-TP-NUW	$\subsetneq$	Appr.2
CC-PV-TP-UW	**CC-RPC-TE-UW**	INCOMP	Appr.5 + Appr.8
CC-PV-TP-UW	**CC-RPC-TE-NUW**	INCOMP	Appr.2 + Appr.8
CC-PV-TP-UW	**CC-RPC-TP-UW**	$\supsetneq$	Appr.7 + Corollary 3.10
CC-PV-TP-UW	**CC-RPC-TP-NUW**	INCOMP	Appr.2 + Appr.8
CC-PV-TP-UW	CC-PC-TE-UW	INCOMP	Appr.5 + Appr.8
CC-PV-TP-UW	CC-PC-TE-NUW	INCOMP	Appr.2 + Appr.8

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PV-TP-UW	CC-PC-TP-UW	$\supsetneq$	Appr.7 + Corollary 3.10
CC-PV-TP-UW	CC-PC-TP-NUW	INCOMP	Appr.2 + Appr.8
CC-PV-TP-UW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PV-TP-UW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PV-TP-UW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PV-TP-UW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PV-TP-UW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PV-TP-UW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PV-TP-UW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PV-TP-UW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PV-TP-UW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PV-TP-UW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PV-TP-UW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PV-TP-UW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PV-TP-NUW	**CC-RPC-TE-UW**	INCOMP	Appr.2 + Appr.5
CC-PV-TP-NUW	**CC-RPC-TE-NUW**	INCOMP	Appr.5 + Appr.8
CC-PV-TP-NUW	**CC-RPC-TP-UW**	$\supsetneq$	Appr.2 + Corollary 3.10
CC-PV-TP-NUW	**CC-RPC-TP-NUW**	$\supsetneq$	Appr.7 + Corollary 3.15
CC-PV-TP-NUW	CC-PC-TE-UW	INCOMP	Appr.2 + Appr.5
CC-PV-TP-NUW	CC-PC-TE-NUW	INCOMP	Appr.5 + Appr.8
CC-PV-TP-NUW	CC-PC-TP-UW	$\supsetneq$	Appr.2 + Corollary 3.10
CC-PV-TP-NUW	CC-PC-TP-NUW	$\supsetneq$	Appr.7 + Corollary 3.15
CC-PV-TP-NUW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PV-TP-NUW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PV-TP-NUW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PV-TP-NUW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PV-TP-NUW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-RPC-TE-UW	**CC-RPC-TE-NUW**	$\subsetneq$	Appr.2
CC-RPC-TE-UW	**CC-RPC-TP-UW**	$\supsetneq$	Appr.5 + Corollary 3.10
CC-RPC-TE-UW	**CC-RPC-TP-NUW**	INCOMP	Appr.2 + Appr.5
CC-RPC-TE-UW	CC-PC-TE-UW	EQ	Corollary 3.18
CC-RPC-TE-UW	CC-PC-TE-NUW	$\subsetneq$	Appr.2
CC-RPC-TE-UW	CC-PC-TP-UW	$\supsetneq$	Appr.5 + Corollary 3.10
CC-RPC-TE-UW	CC-PC-TP-NUW	INCOMP	Appr.2 + Appr.5
CC-RPC-TE-UW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-RPC-TE-UW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-PV-TP-UW	INCOMP*	Appr.1

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-RPC-TE-UW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-RPC-TE-UW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-RPC-TE-UW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-RPC-TE-UW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-RPC-TE-NUW	**CC-RPC-TP-UW**	$\supsetneq$	Appr.2 + Corollary 3.10
CC-RPC-TE-NUW	**CC-RPC-TP-NUW**	$\supsetneq$	Appr.5 + Corollary 3.15
CC-RPC-TE-NUW	CC-PC-TE-UW	$\supsetneq$	Appr.2
CC-RPC-TE-NUW	CC-PC-TE-NUW	EQ	Theorem 3.17
CC-RPC-TE-NUW	CC-PC-TP-UW	$\supsetneq$	Appr.2 + Corollary 3.10
CC-RPC-TE-NUW	CC-PC-TP-NUW	$\supsetneq$	Appr.5 + Corollary 3.15
CC-RPC-TE-NUW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-RPC-TE-NUW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-RPC-TE-NUW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-RPC-TE-NUW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-RPC-TE-NUW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-RPC-TP-UW	**CC-RPC-TP-NUW**	$\subsetneq$	Appr.2
CC-RPC-TP-UW	CC-PC-TE-UW	$\subsetneq$	Appr.5 + Corollary 3.10
CC-RPC-TP-UW	CC-PC-TE-NUW	$\subsetneq$	Appr.2 + Corollary 3.10
CC-RPC-TP-UW	CC-PC-TP-UW	EQ	Corollary 3.11
CC-RPC-TP-UW	CC-PC-TP-NUW	$\subsetneq$	Appr.2
CC-RPC-TP-UW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-RPC-TP-UW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-RPC-TP-UW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-RPC-TP-UW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-RPC-TP-UW	DC-PC-TP-NUW	INCOMP*	Appr.1

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-RPC-TP-NUW	CC-PC-TE-UW	INCOMP	Appr.2 + Appr.5
CC-RPC-TP-NUW	CC-PC-TE-NUW	$\subsetneq$	Appr.5 + Corollary 3.15
CC-RPC-TP-NUW	CC-PC-TP-UW	$\supsetneq$	Appr.2 + Corollary 3.15
CC-RPC-TP-NUW	CC-PC-TP-NUW	EQ	Corollary 3.11
CC-RPC-TP-NUW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-RPC-TP-NUW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-RPC-TP-NUW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-RPC-TP-NUW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-RPC-TP-NUW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PC-TE-UW	CC-PC-TE-NUW	$\subsetneq$	Appr.2
CC-PC-TE-UW	CC-PC-TP-UW	$\supsetneq$	Appr.5 + Corollary 3.10
CC-PC-TE-UW	CC-PC-TP-NUW	INCOMP	Appr.2 + Appr.5
CC-PC-TE-UW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PC-TE-UW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PC-TE-UW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PC-TE-UW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PC-TE-UW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PC-TE-UW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PC-TE-UW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PC-TE-UW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PC-TE-UW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PC-TE-UW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PC-TE-UW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PC-TE-UW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PC-TE-NUW	CC-PC-TP-UW	$\supsetneq$	Appr.2 + Corollary 3.10
CC-PC-TE-NUW	CC-PC-TP-NUW	$\supsetneq$	Appr.5 + Corollary 3.15
CC-PC-TE-NUW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PC-TE-NUW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PC-TE-NUW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PC-TE-NUW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PC-TE-NUW	DC-PC-TP-UW	INCOMP*	Appr.1

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-PC-TE-NUW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PC-TP-UW	CC-PC-TP-NUW	$\subsetneq$	Appr.2
CC-PC-TP-UW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PC-TP-UW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PC-TP-UW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PC-TP-UW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PC-TP-UW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PC-TP-UW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PC-TP-UW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PC-TP-UW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PC-TP-UW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PC-TP-UW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PC-TP-UW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PC-TP-UW	DC-PC-TP-NUW	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PV-TE-UW	INCOMP*	Appr.1
CC-PC-TP-NUW	**DC-PV-TE-NUW**	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PV-TP-UW	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PV-TP-NUW	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-RPC-TE-UW	INCOMP*	Appr.1
CC-PC-TP-NUW	**DC-RPC-TE-NUW**	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-RPC-TP-UW	INCOMP*	Appr.1
CC-PC-TP-NUW	**DC-RPC-TP-NUW**	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PC-TE-UW	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PC-TE-NUW	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PC-TP-UW	INCOMP*	Appr.1
CC-PC-TP-NUW	DC-PC-TP-NUW	INCOMP*	Appr.1
DC-PV-TE-UW	**DC-PV-TE-NUW**	EQ	Theorem 3.16
DC-PV-TE-UW	DC-PV-TP-UW	$\supsetneq$	Appr.3 + Theorem 3.19
DC-PV-TE-UW	DC-PV-TP-NUW	$\supsetneq$	Appr.2 + Theorem 3.19
DC-PV-TE-UW	DC-RPC-TE-UW	$\supsetneq$	Appr.3 + Corollary 3.11 + Corollary 3.21
DC-PV-TE-UW	**DC-RPC-TE-NUW**	$\supsetneq$	Appr.3 + Corollary 3.21
DC-PV-TE-UW	DC-RPC-TP-UW	$\supsetneq$	Appr.3 + Theorem 3.12 + Corollaries 3.11 and 3.21
DC-PV-TE-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.2 + Corollary 3.21 + Corollary 3.11
DC-PV-TE-UW	DC-PC-TE-UW	$\supsetneq$	Appr.3 + Corollary 3.11 + Corollary 3.21
DC-PV-TE-UW	DC-PC-TE-NUW	$\supsetneq$	Appr.3 + Corollary 3.11 + Corollary 3.21
DC-PV-TE-UW	DC-PC-TP-UW	$\supsetneq$	Appr.3 + Corollary 3.11 + Corollary 3.21
DC-PV-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Corollary 3.21 + Corollary 3.11
DC-PV-TE-NUW	DC-PV-TP-UW	$\supsetneq$	Appr.3 + Theorem 3.19
DC-PV-TE-NUW	DC-PV-TP-NUW	$\supsetneq$	Appr.2 + Theorem 3.19
DC-PV-TE-NUW	DC-RPC-TE-UW	$\supsetneq$	Appr.3 + Corollary 3.21 + Corollary 3.11
DC-PV-TE-NUW	**DC-RPC-TE-NUW**	$\supsetneq$	Appr.3 + Corollary 3.21
DC-PV-TE-NUW	DC-RPC-TP-UW	$\supsetneq$	Appr.3 + Corollaries 3.11 and 3.21 + Theorem 3.12
DC-PV-TE-NUW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.2 + Corollary 3.21 + Corollary 3.11
DC-PV-TE-NUW	DC-PC-TE-UW	$\supsetneq$	Appr.3 + Corollary 3.21 + Corollary 3.11

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
DC-PV-TE-NUW	DC-PC-TE-NUW	$\supsetneq$	Appr.3 + Corollary 3.21 + Corollary 3.11
DC-PV-TE-NUW	DC-PC-TP-UW	$\supsetneq$	Appr.3 + Corollary 3.21 + Corollary 3.11
DC-PV-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Corollary 3.21 + Corollary 3.11
DC-PV-TP-UW	DC-PV-TP-NUW	$\supsetneq$	Appr.2
DC-PV-TP-UW	DC-RPC-TE-UW	$\supsetneq$	Appr.4 + Theorem 3.20 + Corollary 3.11
DC-PV-TP-UW	**DC-RPC-TE-NUW**	$\supsetneq$	Appr.4 + Theorem 3.20
DC-PV-TP-UW	DC-RPC-TP-UW	$\supsetneq$	Appr.4 + Theorems 3.12 and 3.20 + Corollary 3.11
DC-PV-TP-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.2 + Corollary 3.15
DC-PV-TP-UW	DC-PC-TE-UW	$\supsetneq$	Appr.4 + Theorem 3.20 + Corollary 3.11
DC-PV-TP-UW	DC-PC-TE-NUW	$\supsetneq$	Appr.4 + Theorem 3.20 + Corollary 3.11
DC-PV-TP-UW	DC-PC-TP-UW	$\supsetneq$	Appr.4 + Theorem 3.20 + Corollary 3.11
DC-PV-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Corollary 3.15
DC-PV-TP-NUW	DC-RPC-TE-UW	INCOMP	Appr.2 + Appr.8
DC-PV-TP-NUW	**DC-RPC-TE-NUW**	INCOMP	Appr.2 + Appr.8
DC-PV-TP-NUW	DC-RPC-TP-UW	INCOMP	Appr.2 + Appr.8
DC-PV-TP-NUW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.4 + Theorem 3.1 + Theorem 3.20
DC-PV-TP-NUW	DC-PC-TE-UW	INCOMP	Appr.2 + Appr.8
DC-PV-TP-NUW	DC-PC-TE-NUW	INCOMP	Appr.2 + Appr.8
DC-PV-TP-NUW	DC-PC-TP-UW	INCOMP	Appr.2 + Appr.8
DC-PV-TP-NUW	DC-PC-TP-NUW	$\supsetneq$	Appr.4 + Theorem 3.1 + Theorem 3.20
DC-RPC-TE-UW	**DC-RPC-TE-NUW**	EQ	Corollary 3.11
DC-RPC-TE-UW	DC-RPC-TP-UW	EQ	Corollary 3.11 + Corollary 3.11
DC-RPC-TE-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.2 + Theorem 3.1
DC-RPC-TE-UW	DC-PC-TE-UW	EQ	Corollary 3.11
DC-RPC-TE-UW	DC-PC-TE-NUW	EQ	Corollary 3.11
DC-RPC-TE-UW	DC-PC-TP-UW	EQ	Corollary 3.11 + Corollary 3.11
DC-RPC-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Theorem 3.1
DC-RPC-TE-NUW	DC-RPC-TP-UW	EQ	Corollary 3.11
DC-RPC-TE-NUW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.2 + Theorem 3.1
DC-RPC-TE-NUW	DC-PC-TE-UW	EQ	Corollary 3.11
DC-RPC-TE-NUW	DC-PC-TE-NUW	EQ	Corollary 3.11
DC-RPC-TE-NUW	DC-PC-TP-UW	EQ	Corollary 3.11
DC-RPC-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Theorem 3.1
DC-RPC-TP-UW	**DC-RPC-TP-NUW**	$\supsetneq$	Appr.2
DC-RPC-TP-UW	DC-PC-TE-UW	EQ	Corollary 3.11 + Theorem 3.12
DC-RPC-TP-UW	DC-PC-TE-NUW	EQ	Corollary 3.11 + Theorem 3.12
DC-RPC-TP-UW	DC-PC-TP-UW	EQ	Theorem 3.12
DC-RPC-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Theorems 3.1 and 3.12 + Corollary 3.11
DC-RPC-TP-NUW	DC-PC-TE-UW	$\subsetneq$	Appr.2 + Theorem 3.1 + Corollary 3.11
DC-RPC-TP-NUW	DC-PC-TE-NUW	$\subsetneq$	Appr.2 + Theorem 3.1 + Corollary 3.11
DC-RPC-TP-NUW	DC-PC-TP-UW	$\subsetneq$	Appr.2 + Theorem 3.1 + Corollary 3.11
DC-RPC-TP-NUW	DC-PC-TP-NUW	EQ	Corollary 3.11
DC-PC-TE-UW	DC-PC-TE-NUW	EQ	Corollary 3.11
DC-PC-TE-UW	DC-PC-TP-UW	EQ	Corollary 3.11 + Corollary 3.11

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
DC-PC-TE-UW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Theorem 3.1 + Corollary 3.11
DC-PC-TE-NUW	DC-PC-TP-UW	EQ	Corollary 3.11 + Corollary 3.11
DC-PC-TE-NUW	DC-PC-TP-NUW	$\supsetneq$	Appr.2 + Theorem 3.1 + Corollary 3.11
DC-PC-TP-UW	DC-PC-TP-NUW	$\supsetneq$	Appr.2
CC-AC-UW	CC-AC-NUW	$\subsetneq$	Appr.11
CC-AC-UW	DC-AC-UW	INCOMP	Appr.13 + Appr.11
CC-AC-UW	DC-AC-NUW	INCOMP	Appr.13 + Appr.12
CC-AC-NUW	DC-AC-UW	INCOMP	Appr.13 + Appr.12
CC-AC-NUW	DC-AC-NUW	INCOMP	Appr.13 + Appr.12
DC-AC-UW	DC-AC-NUW	$\supsetneq$	Appr.11
CC-DC-UW	CC-DC-NUW	$\subsetneq$	Appr.14
CC-DC-UW	CC-DV-UW	INCOMP	Appr.15 + Appr.19
CC-DC-UW	CC-DV-NUW	INCOMP	Appr.20 + Appr.14
CC-DC-UW	DC-DC-UW	INCOMP	Appr.23 + Appr.14
CC-DC-UW	DC-DC-NUW	INCOMP*	Appr.10
CC-DC-UW	DC-DV-UW	INCOMP	Appr.23 + Appr.14
CC-DC-UW	DC-DV-NUW	INCOMP*	Appr.10
CC-DC-NUW	CC-DV-UW	INCOMP	Appr.14 + Appr.21
CC-DC-NUW	CC-DV-NUW	INCOMP	Appr.16 + Appr.20
CC-DC-NUW	DC-DC-UW	INCOMP	Appr.23 + Appr.16
CC-DC-NUW	DC-DC-NUW	INCOMP*	Appr.29
CC-DC-NUW	DC-DV-UW	INCOMP*	Appr.29
CC-DC-NUW	DC-DV-NUW	INCOMP*	Appr.29
CC-DV-UW	CC-DV-NUW	$\subsetneq$	Appr.14
CC-DV-UW	DC-DC-UW	INCOMP	Appr.23 + Appr.14
CC-DV-UW	DC-DC-NUW	INCOMP	Appr.23 + Appr.16
CC-DV-UW	DC-DV-UW	INCOMP	Appr.23 + Appr.14
CC-DV-UW	DC-DV-NUW	INCOMP	Appr.23 + Appr.16
CC-DV-NUW	DC-DC-UW	INCOMP	Appr.23 + Appr.17
CC-DV-NUW	DC-DC-NUW	INCOMP*	Appr.29
CC-DV-NUW	DC-DV-UW	INCOMP	Appr.23 + Appr.17
CC-DV-NUW	DC-DV-NUW	INCOMP*	Appr.29
DC-DC-UW	DC-DC-NUW	$\supsetneq$	Appr.14
DC-DC-UW	DC-DV-UW	$\subsetneq$	Theorem 3.6 + Appr.18
DC-DC-UW	DC-DV-NUW	INCOMP	Appr.14 + Appr.22
DC-DC-NUW	DC-DV-UW	$\subsetneq$	Theorem 3.6 + Appr.18
DC-DC-NUW	DC-DV-NUW	$\subsetneq$	Theorem 3.7 + Appr.19
DC-DV-UW	DC-DV-NUW	$\supsetneq$	Appr.14
CC-AV-UW	CC-AV-NUW	$\subsetneq$	Appr.24
CC-AV-UW	DC-AV-UW	INCOMP	Appr.25 + Appr.24
CC-AV-UW	DC-AV-NUW	INCOMP	Appr.25 + Appr.24
CC-AV-NUW	DC-AV-UW	INCOMP	Appr.25 + Appr.26
CC-AV-NUW	DC-AV-NUW	INCOMP	Appr.25 + Appr.26
DC-AV-UW	DC-AV-NUW	$\supsetneq$	Appr.27

$\mathcal{T}$	$\mathcal{T}'$	Classification	Justification(s)
CC-UAC-UW	CC-UAC-NUW	$\subsetneq$	Appr.30
CC-UAC-UW	DC-UAC-UW	INCOMP	Appr.28 + Appr.24
CC-UAC-UW	DC-UAC-NUW	INCOMP	Appr.28 + Appr.24
CC-UAC-NUW	DC-UAC-UW	INCOMP	Appr.28 + Appr.24
CC-UAC-NUW	DC-UAC-NUW	INCOMP	Appr.28 + Appr.24
DC-UAC-UW	DC-UAC-NUW	$\supsetneq$	Appr.30

B Verifying Claimed Separations and Collapses in Complexity Theory

This section briefly summarizes additional work performed in conjunction with 15 different undergraduate students that has resulted in six technical reports (Carleton et al., 2021; Chavrimootoo and Welles, 2021; Chavrimootoo et al., 2022, 2023a,b,c) that refute claims to have resolved longstanding open problem in complexity theory. Part of the goal of these projects was to get undergraduate students acquainted with doing research in TCS and with writing research papers.

B.1 A Critique of Keum-Bae Cho's Proof that P $\subsetneq$ NP

This paper (Carleton et al., 2021) critiques a claimed proof that P $\subsetneq$ NP by Cho (2018). Their purported proof restricts its attention to resolution-based algorithms for 3-SAT, and attempts to give a lower bound on the complexity of 3-SAT by leveraging that restriction. We pinpoint the error in their main purported proof: they misuse one of their own lemmas by making an unproven generalization, thereby drawing the wrong conclusion. We also note that their

work ignores already established superpolynomial lower bounds on methods using resolution (Haken, 1985; Ben-Sasson and Wigderson, 2001).

B.2 A Critique of Kumar's "Necessary and Sufficient Condition for Satisfiability of a Boolean Formula in CNF and Its Implications on P versus NP Problem"

In their paper, Manoj (2021) claim to give a polynomial-time algorithm for CNF-SAT, thus claiming that P = NP. Their algorithm constructs a tree-like structure from input formulas and then repeatedly prunes the tree to determine whether a given formula is satisfiable. In our critique (Chavrimootoo and Welles, 2021), we give a family of boolean formulas and show that the tree-like structure constructed by Kumar's algorithm uses an exponential amount of time and space when the input is in that family, thereby making their claim erroneous.

B.3 A Critique of Sopin's "PH = PSPACE"

Sopin (2014) claims to show that PH = PSPACE by showing that the well-known PSPACE-complete problem QBF lies in Π_4^p (a level of the so-called polynomial hierarchy). Their paper first establishes a known result about Skolem functions, and then proceeds to give a purported proof that QBF $\in \Pi_4^p$. However, we argue that their purported proof fails to achieve its goal. We also argue that their attempted proof makes an implicit assumption, and we prove that the aforementioned assumption has far more dramatic consequences that what they claim to establish (Chavrimootoo et al., 2023a).

B.4 A Closer Look at Some Recent Proof Compression-Related Claims

In a series of peer-reviewed publications Gordeev and Haeusler (2019, 2020, 2022) claim to resolve a longstanding open problem in complexity theory, namely that NP = PSPACE. Roughly speaking, they (1) claim that the problem of deciding if a formula is provable in minimal propositional logic is PSPACE-complete, and (2) build a framework to show that each formula that is provable in minimal propositional logic has a proof whose length is polynomial in the size of the formula (and they do so via a notion of proof compression). We find that (1) is not known to hold, and thus they have no such proof, and (2) fails to hold because they convert sentences in minimal propositional logic to a weaker logic system that is not complete for minimal propositional logic, hence making their bound seem to hold for that weaker system.

B.5 Evaluating the Claims of "SAT Requires Exhaustive Search"

Xu and Zhou (2023) give a purported proof of the Strong Exponential Time Hypothesis (SETH)—which states that for each positive real-valued constant $c < 1$, SAT cannot be solved in $O(2^{cn})$—which is a stronger claim than P $\neq$ NP. They build their work off of a random CSP (Constraint Satisfaction Problem) model called Model RB, and claim to establish a lower bound on that problem's complexity. However, we note (Chavrimootoo et al., 2023b) that they make several assumptions that are presented without proof. Namely, they claim that an algorithm for Model RB must be a divide-and-conquer algorithm, and they leverage a downward self-reducibility approach to claim that, in conjunction with the divide-

and-conquer assumption, that such a subexponential-time algorithm for Model RB cannot exist. However, they fail to present a proof that the self-reducibility structure does exist, and so fail to reach their conclusion. Moreover, the claim that SETH is true relies on an encoding of CSPs into boolean formulas by Walsh (2000). That encoding however was designed to relate two decision problems. Model RB, however, describes a distributional problem—a pair (L, μ) where L is a language and μ is a distribution—and so the encoding of Walsh is not known to establish any such meaningful relationship.

B.6 On Czerwinski's "P $\neq$ NP Relative to a P-Complete Oracle"

Czerwinski (2023) claims to construct a P-complete oracle relative to which P and NP differ. Though there are (uncountably) many oracles relative to which P and NP differ, their paper's claim is far more ambitious than it claims to be: It is well-known that for every $A \in \mathrm{P}$, $\mathrm{P}^A \neq \mathrm{NP}^A \iff \mathrm{P} \neq \mathrm{NP}$, as $\mathrm{NP}^A = \mathrm{NP}$ and $\mathrm{P}^A = \mathrm{P}$. Czwerwinski's paper presents two undecidable sets $\mathrm{D_P}$ and $\mathrm{D_{NP}}$ and claims that any deterministic oracle machine that both accepts $\mathrm{D_P}$ and has $\mathrm{D_{NP}}$ as its oracle must make $h(n)$ (for some $h(n) = \Theta(2^n)$) queries to its oracle on each input of length n. They then attempt to use that claim to argue that an oracle machine accepting some NP set and having a P-complete set as oracle must make $h(n)$ calls to their oracle. However, we prove that $\mathrm{D_P}$ and $\mathrm{D_{NP}}$ are both Σ_1^0-complete (Chavrimootoo et al., 2023c) and thus recursively isomorphic. This in turn implies that a machine that accepts $\mathrm{D_P}$, halts on every input when its oracle is $\mathrm{D_{NP}}$, and only makes one oracle call certainly exists. And if we drop the requirement that the machine needs to halt on every (nonaccepting) input, then a machine that makes *zero* oracle calls and yet accepts $\mathrm{D_P}$ clearly exists.