# The Dodgson Winner Problem

Dhamma Kimpara

August 17, 2020

## 1 Introduction

Suppose you are a theoretical computer scientist, wandering in the wilds of theory-land and you come upon a complexity class. It has not been seen before. It seems intuitive and natural. So you decide to submit it to the complexity zoo [Aaronson et al., 2005]. But the worry is whether this class will turn out to capture the complexity of important real-world problems. In other words, is anyone or any problems going to come around to your new section of the zoo and join your party? After all, there are 545 classes and counting in the zoo!

This was the case for the class $\Theta_2^P$ in the mid-1990s. $\Theta_2^P$ or $\mathsf{P}_{\|}^{\mathsf{NP}}$, the class of languages decidable by a polynomial time Turing machine with parallel access to $\mathsf{NP}$, was introduced in 1982 by Papadimitriou and Zachos [1982]. The location of this class in the polynomial hierarchy is:

$$\Sigma_1^P \cup \Pi_1^P \subseteq \Theta_2^p \subseteq \Delta_2^P \subseteq \Sigma_2^P \cap \Pi_2^P, \quad \text{or}$$

$$\mathsf{NP} \cup \mathsf{coNP} \subseteq \mathsf{P}_{\|}^{\mathsf{NP}} \subseteq \mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{NP}^{\mathsf{NP}} \cap \mathsf{coNP}^{\mathsf{NP}}$$

By the mid 1990s, the theoretical importance of $\Theta_2^P$ was recognized in complexity theory. Wagner [1990] established half a dozen characterizations of $\Theta_2^P$, several complete problems and a toolkit for establishing $\Theta_2^P$-hardness. Hemachandra [1987] and Köbler, Johannes et al. [1987] showed that $\Theta_2^P$ was equivalent to the class of problems that can be solved by $\mathcal{O}(\log(n))$ sequential Turing queries to $\mathsf{NP}$. Furthermore, if $\mathsf{NP}$ contains some $\Theta_2^P$-hard problem, then the polynomial hierarchy collapses to $\mathsf{NP}$. However, these connections and complete problems lived in the pure theory section of the zoo and did not yet have the appeal of problems from "real world" settings.

Along came the Dodgson winner problem, invented in 1876 by Charles Lutwidge Dodgson, better known under the pen name of Lewis Carroll. Hemaspaandra et al. [1999] proved that this problem was $\Theta_2^P$-complete. This was the first "real-world" problem proven complete for the class $\Theta_2^P$.

Dodgson's election system takes the following form. An election is a finite number of voters, who each cast a linear order over a common finite set of candidates. Note that linear orders are "tie-free". The winner is determined by whichever candidate is closest to being a Cordorcet winner, a criteria used by other election systems. A Condorcet winner is a candidate $a$ who for every other candidate $b$, is preferred to $b$ by strictly more than half of the voters. We naturally want election systems to be Condorcet-consistent, i.e. the system has the property where if $a$ is a Condorcet winner, $a$ is the one and only winner in the election. Dodgson's election system is Condorcet-consistent [Brandt et al., 2016]

The winner(s) in a Dodgson election is defined as the candidate(s) who are the "closest" to being Condorcet winners. The winners are the candidates that have the lowest Dodgson score. The Dodgson score of a candidate $a$ is the smallest number of sequential exchanges of adjacent candidates in preference orders such that after those exchanges $a$ is a Cordorcet winner.

Note that it is remarkable that we find $\Theta_2^P$-complete problems that were defined 100 years before complexity theory itself existed. DodgsonWinner is also extremely natural when compared with other complete

problems in this class such as determining if the maximum size clique in a graph is of odd size (odd-max-clique). In this project we will use Comp-SAT, which was recently shown to be complete for this class [Lukasiewicz and Malizia, 2017].

The rest of this project presents the theory of and a practical algorithm for the Dodgson winner problem. We present slightly modified proof of completeness of the Dodgson winner problem based on new results for the class $\Theta_2^P$. Then we present, implement, and examine a heuristic algorithm that is self-knowingly correct for most practical instances of the problem.

# 2 Preliminaries

## 2.1 Problem Setting

We formally introduce the Dodgson Winner problem, or the decision problem of checking whether a particular candidate is a Dodgson winner in a Dodgson election. First, we formally define a Dodgson Election, or an instance of the decision problem DodgsonWinner, which we will define later.

**Definition 1.** Dodgson Triple.
A triple, $\langle C, c, V \rangle$, where $C$ is the set of candidates $1, \ldots, m$, a candidate $c \in C$, and a set of $n$ strict (ie. irreflexive and anti-symmetric) preference orders, one per voter, over all candidates in $C$.

**Definition 2.** Preference relation.
For vote $v$ and candidates $c, d \in C$. $c <_v d$ means that in vote $v$, $d$ is preferred to $c$ and $c \prec_v d$ means that $c <_v d$ and $\nexists e \ \in C \ s.t. \ c <_v e <_v d$.

A natural criterion for the winner of an election is the concept of a Condorcet Winner. This concept is used in other election systems such as Young's election, which is related to Dodgson's system.

**Definition 3.** Condorcet Winner [Condorcet and de Caritat, 1785].
In an election, with a set of candidates $C$ and $n$ votes or strict preference orders $V$, a candidate $a \in C$ is a Condorcet Winner if for every other candidate $b$, $a > b$ by strictly more than half of the voters.

However, many elections will not have a Condorcet winner, so Dodgson's idea was that whichever candidate was the "closest" to being a Cordorcet winner should be the winner. It is also possible to have multiple winners, all with equal scores (closeness) in this system. A candidates closeness to being a Cordorcet winner is defined as the Dodgson Score of that candidate.

**Definition 4.** Dodgson Score.
First define a switch as an exchange of two adjacent preferences in the preference order of one voter. Then, the Dodgson Score of a candidate is the smallest number of sequential switches needed to make the candidate a Condorcet winner. The Dodgson Score of any Cordorcet Winner is 0. We denote the Dodgson score of a Dodgson triple as $Score(\langle C, c, V \rangle)$

This notion of "closeness" is based on finding the minimum edit distance between the initial votes and a region in the space of all votes. The edits are sequential and defined as adjacent exchanges in the preference orders. The candidates with the minimum Dodgson Scores are the winners of the Dodgson election.

We now define the formal decision problems with which we will examine from a theory of computation lens. The first problem, DodgsonScore, will be used to show that the main problem, DodgsonWinner, is $\Theta_2^P$-complete.

**Decision Problem.** DodgsonScore.
**Instance:** $k \in \mathbb{N}$. A Dodgson Election and Candidate $\langle C, c, V \rangle$.
**Decide:** Is the Dodgson Score of candidate $c$ less than or equal to $k$?

**Decision Problem.** DodgsonWinner.
Instance: A Dodgson Election and Candidate $\langle C, c, V \rangle$
Decide: Is $c$ a winner of the election? In other words, does $c$ have the minimum Dodgson Score in the election?

## 2.2 $\Theta_2^P$ in the Polynomial Hierarchy

Here we formally introduce the class $\Theta_2^P$, for which we will show DodgsonWinner is complete for. This was the first natural "real-world" problem that was shown to be complete for this class. As briefly talked about in the introduction, $\Theta_2^P$ and the general classes $\Theta_k^P$ hold a particular position in the polynomial hierarchy. $\Theta_k^P$ however is not usually talked about in regular definitions of the polynomial hierarchy so we will also see in this section how they fit in.

First we introduce the polynomial hierarchy (PH). PH generalizes the classes NP and coNP. PH is formed by the classes $\Sigma_k^P$, $\Pi_k^P$, and $\Delta_k^P$. It is defined as follows: $\Sigma_0^P = \Pi_0^P = \Delta_0^P = \mathsf{P}$ and for all $k \geq 1$, $\Sigma_k^P = \mathsf{NP}^{\Sigma_{k-1}^P}$, $\Delta_k^P = \mathsf{P}^{\Sigma_{k-1}^P}$, and $\Pi_k^P = \mathsf{co}\Sigma_{k-1}^P$. Note that $\Sigma_1^P = \mathsf{NP}^{\Sigma_0^P} = \mathsf{NP}^P = \mathsf{NP}$ and $\Pi_1^P = \mathsf{co}\Sigma_0^P = \mathsf{coNP}^P$. The union of all the classes in the PH is denoted PH. An open problem in computer science is whether or not the polynomial hierarchy collapses. The polynomial hierarchy is said to collapse if for some $k$, $\Sigma_k^P$ or $\Pi_k^P = \mathsf{PH}$. It is strongly believed that the polynomial hierarchy does not collapse.

Papadimitriou and Zachos [1982] proposed that $\Theta_k^P = \mathsf{P}^{\Sigma_{k-1}^P[O(\log n)]}$ to be included in the standard definition of PH as well, where the bound in the brackets represents the bound on the allowed number of oracle calls to the oracle class preceding the brackets. Note that Hemachandra [1987] showed that this is equivalent to parallel access to $\Sigma_{k-1}^P$. Lukasiewicz and Malizia [2017] poses that $\Theta_0^P = \mathsf{P}$ and observe that $\Theta_1^P = P$. Also $\Theta_k^P$ is a deterministic class, since the machine making oracle calls is deterministic. Thus $\Theta_k^P = \mathsf{co}\Theta_k^P$. We state the generalized relationship of the classes in terms of $k$:

$$\Sigma_k^P \cup \Pi_k^P \subseteq \Theta_{k+1}^P \subseteq \Delta_{k+1}^P \subseteq \Sigma_{k+1}^P \cap \Pi_{k+1}^P$$

Now our specific problem, DodgsonWinner, is complete for $\Theta_2^P$:

**Definition 5.** $\Theta_2^P$:
The class of problems solvable with polynomial-time parallel access to an NP oracle. This is equivalent to $\mathcal{O}(\log(n))$ sequential queries to an NP oracle [Hemachandra, 1987, Köbler, Johannes et al., 1987].

While the classes $\Sigma_k^P$ and $\Pi_k^P$ in the polynomial hierarchy have their complete problem as quantified boolean formulas with $k-1$ alternations of quantifiers ($\mathrm{QBF}_k$ or $\mathrm{QSAT}_k$), the class $\Theta_2^P$ did not yet have such a characterization by some formulation of SAT until recently. The complete problem for the class $\Theta_2^P$, with this similar flavor, is Comp-SAT, introduced in Lukasiewicz and Malizia [2016]. In the same work, a general characterization for the classes $\Theta_{k+1}^P$ was also proved, given by the following problem:

**Decision Problem.** Comp-$\mathrm{PH}_k$
**Instance**: Let $A_1$ and $A_2$ be either both $\Sigma_k^P$-complete or $\Pi_k^P$-complete problems. Then let $\langle A, B\rangle$ be sets of instances of $A_1$ and $A_2$, respectively.
**Decide**: Is the number of yes-instances in $A$ greater than or equal to the number of yes-instances in $B$.

**Theorem 1.** [Lukasiewicz and Malizia, 2017] Comp-$\mathrm{PH}_k$ is $\Theta_{k+1}^P$-complete.

For a formulation via $\mathrm{QBF}_k$, let $A_1$ and $A_2$ be both the same appropriate $\mathrm{QBF}_k$ problem that is complete for $\Sigma_k^P$ or $\Pi_k^P$. Note that Lukasiewicz and Malizia [2017] state this latter problem as complete, calling it Comp-Valid$_k$, however this result is based on Theorem 1. The completeness of this problem makes sense because $\Theta_{k+1}^P$ is intuitively the class of problems solvable with parallel oracle access to the $k$th level of the polynomial hierarchy. For the class $\Theta_2^P$, of which DodgsonWinner is complete, the canonical problem is Comp-SAT.

**Decision Problem.** Comp-SAT or Comp-Valid$_1$ [Lukasiewicz and Malizia, 2017]
**Instance**: A pair $\langle A, B\rangle$ of sets of 3CNF formulas.
**Decide**: Is the number of satisfiable formulas in $A$ greater than or equal to the number of satisfiable formulas in $B$.

**Theorem 2.** [Lukasiewicz and Malizia, 2017] Comp-SAT is $\Theta_2^P$-complete.

# 3 The Complexity of the Dodgson Winner Problem

Our results largely follow the proof in Hemaspaandra et al. [1999], except instead of using a technical lemma by Wagner [1990], we instead begin the reduction with the recent result that Comp-SAT is $\Theta_2^P$-complete [Lukasiewicz and Malizia, 2017]. All lemmas and theorems in this section unless cited otherwise are from Hemaspaandra et al. [1999]. We have omitted proofs of technical lemmas and have used our own proof of Lemma 3, which is the main hardness proof.

We first state the main result of this section and a big-picture corollary concerning the polynomial hierarchy.

**Theorem 3.** DodgsonWinner is $\Theta_2^P$-complete

It follows that though DodgsonWinner is NP-hard [Bartholdi et al., 1989], it cannot be NP-complete unless the polynomial hierarchy collapses.

**Corollary 1.** If DodgsonWinner is NP-complete then PH = NP.

## 3.1 Outline of Proof of Theorem 3

First we briefly outline the reduction from Comp-SAT to DodgsonWinner. All of the following reductions are many-one reductions. Let us start with a Comp-SAT instance $\langle A, B\rangle$. For each 3CNF formula $x$ in $A$ and $B$, reduce $x$ into the corresponding instance of ThreeDimensionalMatching (3DM), which is possible because 3DM is NP-complete. Now, for each 3DM problem, we reduce it into an instance of DodgsonScore. Using a merger lemma, we will merge these two sets of DodgsonScore instances into two elections, one for $A$ and one for $B$, thus an yielding instance of 2ElectionRanking (2ER). Then 2ER will be reduced to DodgsonWinner. For the formal proof we will use the problems Comp-3DM and Comp-DodgsonScore which are similar to Comp-SAT. Alternatively, one could replace some of these reductions with polynomial time truth-table reductions.

A reference diagram is provided:

$$\text{Comp-SAT} \leq_m^p \text{Comp-3DM} \leq_m^p \text{Comp-DodgsonScore} \leq_m^p \text{2ER} \leq_m^p \text{DodgsonWinner}$$

This reduction can be further truncated by using Theorem 1 after showing that DodgsonScore is NP-complete. We would be immediately be able to establish that Comp-DodgsonScore is $\Theta_2^P$-hard. However we leave in the other steps to reduce from a candidate for a canonical problem for the class $\Theta_2^P$. Once the reduction above is proved, it remains to show membership in $\Theta_2^P$ in order to prove Theorem 3.

**Theorem 4.** DodgsonWinner $\in \Theta_2^P$.

*Proof.* Ask in parallel all the DodgsonScore queries, one for each candidate in $C$. Each query is an NP query since DodgsonScore is NP-complete by the second reduction outlined above (lemma 1) and observing that given a yes-answer, it can be justified in polynomial time by identifying appropriate switches and tabulating the vote. We now have the exact Dodgson Score for each candidate (do binary search over the potential scores, which is polynomial in the input size). It is easy to decide whether or not the given candidate $c$ has the minimum score (find the max of the scores and compare the $c$'s score to the max $\sim \mathcal{O}(n)$). □

The rest of the section will be devoted to proving the reduction outlined above.

## 3.2 Outline of relevant lemmas

In order to prove the reduction outlined in section 3.1, we first state the necessary lemmas. Proofs of the non-technical lemmas will be provided in section 3.3. What these lemmas provide are ways to add election scores, and how to merge elections. The main part of the reduction is reducing from Comp-SAT to 2ER from which a merger lemma is needed to reduce to DodgsonWinner.

The first part of the reduction is performed by the following lemma.

**Lemma 1.** There exists an NP-complete problem $A$ and a polynomial-time computable function $f$ that reduces $A$ to DodgsonScore in such a way that, $\forall x \in \Sigma^*, f(x) = \langle\langle C, c, V\rangle, k\rangle$ is an instance of DodgsonScore with an odd number of voters and

1. if $x \in A$ then $Score(\langle C, c, V\rangle) = k$, and
2. if $x \notin A$ then $Score(\langle C, c, V\rangle) = k + 1$.

**Lemma 2.** There exists a polynomial-time computable function $DodgsonSum$ such that $\forall k$ and for all $\langle C_1, c_1, V_1\rangle, \langle C_2, c_2, V_2\rangle, \ldots, \langle C_k, c_k, V_k\rangle$ satisfying $||V_j||$ is odd for all $j$, it holds that

$$DodgsonSum(\langle\langle C_1, c_1, V_1\rangle, \langle C_2, c_2, V_2\rangle, \ldots, \langle C_k, c_k, V_k\rangle\rangle)$$

is a Dodgson triple having an odd number of voters and such that

$$\sum_j Score(\langle C_j, c_j, V_j\rangle) = Score(DodgsonSum(\langle\langle C_1, c_1, V_1\rangle, \langle C_2, c_2, V_2\rangle, \ldots, \langle C_k, c_k, V_k\rangle\rangle))$$

Theorem 2, Lemma 1 and Lemma 2, establish the $\Theta_2^P$-hardness of a problem related to DodgsonWinner. We now define this decision problem:

**Decision Problem.** TwoElectionRanking (2ER)
Instance: A pair of Dodgson triples $\langle\langle C, c, V\rangle, \langle D, d, W\rangle\rangle$ both having an odd number of voters such that $c \neq d$.
Decide: Is $Score(\langle C, c, V\rangle) \leq Score(\langle D, d, W\rangle)$?

**Lemma 3.** 2ER is $\Theta_2^P$-hard.

We now need to make the results so far applicable to DodgsonWinner, so we need another merger lemma to merge two elections into a single election.

**Lemma 4.** There exists a polynomial-time computable function $Merge$ such that, for all Dodgson triples, $\langle C, c, V\rangle$ and $\langle D, d, W\rangle$ for which $c \neq d$ and both having an odd number of voters, there exist $\hat{C}$ and $\hat{V}$ such that

1. $Merge(\langle C, c, V\rangle, \langle D, d, W\rangle)$ is an instance of DogsonWinner,
2. $Merge(\langle C, c, V\rangle, \langle D, d, W\rangle) = \langle \hat{C}, c, \hat{V}\rangle$,
3. $Score(\langle \hat{C}, c, \hat{V}\rangle) = Score(\langle C, c, V\rangle) + 1$,
4. $Score(\langle \hat{C}, d, \hat{V}\rangle) = Score(\langle D, d, W\rangle) + 1$ and,
5. for each $e \in \hat{C} \setminus \{c, d\}$, $Score(\langle \hat{C}, c, \hat{V}\rangle) < Score(\langle \hat{C}, e, \hat{V}\rangle)$

### 3.3 Proof of select lemmas

Proof sketch: Lemma 1 or 3DM $\leq_m^p$ DodgsonScore
This reduction differs from Bartholdi et al. [1989] in that this reduction has additional properties that are required by the lemma. We will reduce from ThreeDimensionalMatching to DodgsonScore. This reduction has many technical details so we provide only a very brief sketch.

**Decision Problem.** ThreeDimensionalMatching (3DM)
**Input:** Sets $M, W, X, Y$, where $M \subseteq W \times X \times Y$ and $W, X, Y$ are disjoint, nonempty sets having the same number of elements.
**Decide:** Does $M$ contain a matching, i.e. a subset $M' \subseteq M$ such that $\|M'\| = \|W\|$ and no two elements of $M'$ agree in any coordinate?

Now given an instance of 3DM as outlined above, let $C = W \cup X \cup Y \cup \{c, s, t\}$ where $c, s, t \notin W \cup X \cup Y$. Let $V$ consist of voters simulating elements of $M$ and $\|M\|-1$ dummy voters. Then, $M$ containing a matching corresponds to $Score(\langle C, c, V\rangle) = 3q$ and $M$ not containing a matching corresponds to $Score(\langle C, c, V\rangle) = 3q + 1$. □

*Proof.* Lemma 3
We will reduce from Comp-SAT to 2ER. Let $\langle A, B\rangle$ be a Comp-SAT instance.

For each 3CNF formula $x \in A$ or $B$, reduce $x$ into the corresponding 3DM instance $x'$ and add $x'$ to $A'$ or $B'$ if $x \in A$ or $x \in B$, respectively. In effect we are reducing Comp-SAT to an instance of Comp-3DM, $\langle A', B'\rangle$. It is easy to see that solving $\langle A', B'\rangle$ solves $\langle A, B\rangle$. One can also see that Comp-3DM is $\Theta_2^P$-complete because it shares the structure where two lists of NP-hard problems are compared.

Now we perform a similar reduction from Comp-3DM to Comp-DodgsonScore. For each $x' \in A'$ or $B'$, use the function in Lemma 1 to reduce $x'$ into the corresponding DodgsonScore instance $\langle C, c, V\rangle$ and add $\langle C, c, V\rangle$ to $A''$ or $B''$ if $x' \in A'$ or $x' \in B'$, respectively. It is similarly easy to see that solving $\langle A'', B''\rangle$ solves $\langle A', B'\rangle$. If $x'$ is a yes-instance of 3DM then by Lemma 1,

$$Score(f(x')) = Score(x'') = Score(\langle C, c, V\rangle) = k$$

where $f$ is the function described in Lemma 1. Thus $\langle C, c, V\rangle$ is also a yes-instance of DodgsonScore. If $x'$ is a no-instance of 3DM then $Score(\langle C, c, V\rangle) = k+1$ and the corresponding triple $\langle C, c, V\rangle$ is a no-instance of DodgsonScore.

Now we reduce from Comp-DodgsonScore to 2ER using Lemma 2. First note that the direction of the inequality of the decision problem changes in this reduction by the nature of Lemma 1. Now to begin the reduction, we merge all the Dodgson elections in $A''$ and $B''$ into $\langle C, c, V\rangle$ and $\langle D, d, W\rangle$, respectively. This is done using the *DodgsonSum* function in Lemma 2, which we can use because Lemma 1 ensures that conditions of the Lemma are met by each election.

For example if $A'' = \langle C_1, c_1, V_1\rangle, \langle C_2, c_2, V_2\rangle, \ldots, \langle C_k, c_k, V_k\rangle$ then

$$\langle C, c, V\rangle = DodgsonSum(\langle\langle C_1, c_1, V_1\rangle, \langle C_2, c_2, V_2\rangle, \ldots, \langle C_k, c_k, V_k\rangle\rangle))$$

and

$$\sum_j Score(\langle C_j, c_j, V_j\rangle) = Score(\langle C, c, V\rangle).$$

Now $\langle A'', B''\rangle$ is a yes-instance of Comp-DodgsonScore if by definition, the number of satisfied DodgsonScore instances in $A''$ being greater than that of $B''$. This again is equivalent to the sum of the Dodgson scores of $A''$ being less than that of $B''$ since we fix $k$ to be the same for each of the reductions using Lemma 1. Let $\|A\|_{yes}$ be the number of yes-instances in a set of decision problems $A$. Using the reduction of $\langle A'', B''\rangle$ outlined above,

$$\begin{aligned}
&\langle A'', B''\rangle \text{ is a yes-instance of Comp-DodgsonScore} \\
&\iff \|A''\|_{yes} \geq \|B''\|_{yes} \\
&\iff \|\{x \in A'' | Score(x) \leq k\}\| \geq \|\{x \in B'' | Score(x) \leq k\}\| \\
&\iff Score(\langle C, c, V\rangle) \leq Score(\langle D, d, W\rangle) \qquad (1) \\
&\iff \langle\langle C, c, V\rangle, \langle D, d, W\rangle\rangle \text{ is a yes-instance of 2ER}
\end{aligned}$$

For line (1), note that by the reduction used from Lemma 1, the elections in $\langle A'', B''\rangle$ can have score of either $k$ or $k+1$. Hence we have shown a reduction from Comp-DodgsonScore to 2ER.

Combining the many-one reductions above, we have shown that:

$$\text{Comp-SAT} \leq_m^p \text{Comp-3DM} \leq_m^p \text{Comp-DodgsonScore} \leq_m^p \text{2ER}.$$

So by Theorem 2, 2ER is $\Theta_2^P$-hard and the Lemma is proved. □

### 3.4 Proof that DodgsonWinner is $\Theta_2^P$-complete.

*Proof.* Theorem 3
By Theorem 4, DodgsonWinner $\in \Theta_2^P$. We now show that 2ER $\leq_m^p$ DodgsonWinner and so by Lemma 3, the theorem then follows.

We now describe a polynomial time function $f$ for this reduction. Let $s_o$ be some fixed string that is not in DodgsonWinner. Then

$$f(x) = \begin{cases} Merge(x_1, x_2) & \text{if } x \in \text{2ER} \\ s_o & \text{if } x \notin \text{2ER} \end{cases}$$

Where $Merge$ is the function defined in Lemma 4 and $x_1$ and $x_2$ are the two elections in the instance of 2ER. So $f(x)$ is an instance of DodgsonWinner if and only if $x$ is an instance of 2ER.

Now we show how $Merge(x_1, x_2)$, an instance of DodgsonWinner, solves the corresponding instance $x$ of 2ER. Let $x$ be a pair of Dodgson triples, $\langle C, c, V\rangle$ and $\langle D, d, w\rangle$, where both have an odd number of voters and $d \neq c$ (so we can apply the Lemma). Then let $Merge(\langle C, c, V\rangle, \langle D, d, W\rangle) = \langle \hat{C}, c, \hat{V}\rangle$ be the corresponding instance of DodgsonWinner.

First assume that $Score(\langle C, c, V\rangle) \leq Score(\langle D, d, W\rangle)$, or the answer to the 2ER decision problem is yes. Then by Lemma 4, properties 3 and 4, $Score(\langle \hat{C}, c, \hat{V}\rangle) \leq Score(\langle \hat{C}, d, \hat{V}\rangle)$. By property 5, $Score(\langle \hat{C}, c, \hat{V}\rangle) \leq Score(\langle \hat{C}, e, \hat{V}\rangle)$ for all $e \in \hat{C} \setminus \{c, d\}$. Hence $c$ is a Dodgson winner of the election and the answer to the decision problem of this instance of DodgsonWinner is yes.

Finally, assume that $Score(\langle C, c, V\rangle) > Score(\langle D, d, W\rangle)$, or the answer to the 2ER decision problem is no. Then similarly, $Score(\langle \hat{C}, c, \hat{V}\rangle) > Score(\langle \hat{C}, d, \hat{V}\rangle)$ and so $c$ is not a Dodgson winner of the election. Hence the answer to the decision problem of this instance of DodgsonWinner is no.

□

### 3.5 Fixed Parameter Tractability

Though DodgsonWinner is $\Theta_2^P$-complete, what can be done if we fix the number of candidates or voters? Here we mention that DodgsonWinner is fixed parameter tractable in the number of editing operations, the number of candidates and the number of voters [Bartholdi et al., 1989]. First, for a fixed constant $m = \|C\|$, there is a FPT algorithm that is exponential in the fixed constant $m$:

**Proposition 1.** [Bartholdi et al., 1989] Let $m \in \mathbb{N}^+$. There exists a polynomial time algorithm $A_m$ that computes all Dodgson scores (and thus all Dodgson Winners) in Dodgson elections having at most $m$ candidates.

The above result uses integer linear programming while the results showing fixed parameter tractability in the number of editing operations utilize dynamic programming.

## 4 Practical Greedy Algorithm

In this section we present the `GreedyWinner` algorithm by Homan and Hemaspaandra [2005] which runs in polynomial time and is self-knowingly correct with high probability when the number of voters is superquadratic in the number of candidates. This probability is over a uniform random draw of each vote i.e. $m!$ possibilities for each vote.

In real-world settings, this algorithm seems adequate for most elections. It is hard to imagine a large election where the number of candidates is larger than the square root of the number of voters. For instance theoretically this algorithm can handle a small town of 2500 with at most about 50 candidates. We will see empirically that the number is more like 8000 voters for a set of 50 candidates. In the real world though, as the number of voters increases the number of candidates does not seem to grow like $\sqrt{n}$ but more so some function much slower than that.

While hard problems may not admit efficient deterministic algorithms, we can still try to create heuristic algorithms which are correct in many cases. For example, modern SAT solvers utilize are a backtracking algorithm at their core while using several heuristics such as variable selection, clause learning, and conflict directed backjumping to improve their performance.

Theoretically however, there are results that show that no NP-hard problem has a deterministic heuristic algorithm whose asymptotic error rate is subexponential [Hemaspaandra and Williams, 2012]. But an algorithm with a larger than subexponential error rate can still be useful. For this particular algorithm, `GreedyScore`, asymptotically, the algorithm succeeds with high probability when the number of candidates, $m$, goes to infinity and the number of voters is super-quadratic in $m$.

Success is when the algorithm is self-knowingly correct. Intuitively, this means that algorithm will output the value computed along with a certificate saying that the output is either definitely (provably) correct or "maybe" correct. This is opposed to other algorithms that will output an answer without such a certificate of correctness and instead have an overall probability of correctness. We first formally define what we mean by self-knowingly correct.

**Definition 6.** For sets $S$ and $T$ and function $f : S \to T$, an algorithm $\mathcal{A} : S \to T \times \{\text{"definitely"}, \text{"maybe"}\}$ is self-knowingly correct for all $f$ if, for all $s \in S$ and $t \in T$ whenever $\mathcal{A}$ on input $s$ outputs $(t, \text{"definitely"})$ it holds that $f(s) = t$.

We now explain how the algorithm works. See below for the full pseudocode for the algorithm. Our goal is to determine if a particular candidate $c \in C$ is a Dodgson winner of the election. What `GreedyWinner` does is query `GreedyScore` for the scores of every candidate and outputs "definitely" or "maybe" and if $c$ is a Dodgson winner or not. `GreedyWinner` will output "definitely" only if all the `GreedyScore` queries output "definitely".

Now, what `GreedyScore` does is to go through each vote $v \in V$ and looks at which candidate is immediately preferred to $c$ (which $d \in C$ such that $c \prec d$). If $c$ is not yet beating $d$ in terms of how many voters prefer one candidate to the other, then `GreedyScore` exchanges $c$ and $d$ in that vote $v$. Now if $c$ is a Condorcet winner once `GreedyScore` has gone through all the votes, then the algorithm outputs "definitely" and $Score(c)$.

Intuitively, a lower bound for the number of exchanges to make $c$ a Condorcet winner is the minimum number votes in which a single adjacent exchange takes place to make $c$ a winner. If the algorithm does all these exchanges and $c$ is a Condorcet winner then it is clear that the algorithm does not perform any more exchanges than the lower bound. The algorithm thus has computed the Dodgson score of $c$.

**Theorem 5.**

1. `GreedyScore` is self-knowingly correct for $Score$.
2. `GreedyWinner` is self-knowingly correct for DodgsonWinner.
3. `GreedyScore` and `GreedyWinner` both run in polynomial time.

We now explain the probability of success of `GreedyScore` and thus `GreedyWinner`. As mentioned above, the `GreedyWinner` succeeds with high probability when $\|V\| = n$ is superquadratic in $\|C\| = m$. This is because when the $n$ is large compared to $m$, it is more likely that `GreedyScore` will have enough immediately adjacent swap votes to make up for the deficit between the candidate in question, $c$, and all the other candidates $d$. More precisely, for any two candidates $c$ and $d$, in half the ways voter $v$ can vote, $c <_v d$. But, $c \prec_v d$ in only $1/m$ of the ways. Drawing the votes uniformly at random, the number of votes where $c <_v d$ is $n/2$ and the number of votes where $c \prec_v d$ is $n/m$. Thus the probability of success is high when $n$ is large compared to $m$. The probability of success for `GreedyWinner` then utilizes the union bound and Chernoff's Theorem.

**Algorithm 1** `GreedyScore`$(\langle C, c, V\rangle)$

**Input:** A Dodgson triple $\langle C, c, V\rangle$.

**for** $d \in C \setminus \{c\}$ **do** ▷ Initialize counter variables
  $Deficit[d] \leftarrow 0$ ▷ Number of votes by which $c < d$ ($d$ beats $c$)
  $Swaps[d] \leftarrow 0$ ▷ Number of votes by which $c \prec d$ (greedily swappable votes against $d$)
**end for**
**for** each vote $v \in V$ **do** ▷ each vote is an array where $v[k] \prec v[k+1]$
  $i \leftarrow 1$
  **while** $v[i] \neq c$ **do**
    $d \leftarrow v[i]$
    $Deficit[d] \leftarrow Deficit[d] - 1$
    $i \leftarrow i + 1$
  **end while**
  **if** $i < length(v)$ **then**
    $d \leftarrow v[i+1]$
    $Swaps[d] \leftarrow Swaps[d] + 1$
  **end if**
  **for** $i \leftarrow i + 1$ to $length(v)$ **do**
    $d \leftarrow v[i]$
    $Deficit[d] \leftarrow Deficit[d] + 1$
  **end for**
**end for**
$confidence \leftarrow$ "definitely"
$score \leftarrow 0$
**for** $d \in C \setminus \{c\}$ **do** ▷ If there are more than $1 + Deficit[d]/2$ greedily swappable votes,
  **if** $Deficit[d] \geq 0$ **then** ▷ then this is the Dodgson Score of $c$.
    $score \leftarrow score + \lfloor Deficit[d]/2 \rfloor + 1$
    **if** $Deficit[d] \geq 2 \cdot Swaps[d]$ **then**
      $confidence \leftarrow$ "maybe"
      $score \leftarrow score + 1$
    **end if**
  **end if**
**end for**
**Output:** $(score,\ confidence)$

**Algorithm 2** `GreedyWinner`$(\langle C, c, V\rangle)$

**Input:** A Dodgson triple $\langle C, c, V\rangle$ where we want to test whether $c$ is a Dodgson winner in the election.
$(cscore,\ confidence) \leftarrow$ `GreedyScore`$(\langle C, c, V\rangle)$
$winner \leftarrow$ "yes"
**for** $d \in C \setminus \{c\}$ **do**
  $(dscore,\ confidence) \leftarrow$ `GreedyScore`$(\langle C, d, V\rangle)$
  **if** $dscore < cscore$ **then**
    $winner \leftarrow$ "no"
    **if** $dcon =$ "maybe" **then**
      $confidence \leftarrow$ "maybe"
    **end if**
  **end if**
**end for**
**Output:** $(winner,\ confidence)$

**Theorem 6.** For each $m, n \in \mathbb{N}^+$, the following hold. Let $C = \{1, \ldots, m\}, \|V\| = n$.

1. For each $c \in C$,

$$\Pr[\texttt{GreedyScore}(\langle C, c, V\rangle) \neq (Score(\langle C, c, V\rangle), \text{“definitely”})] < 2(m-1)e^{\frac{-n}{8m^2}},$$

where the probability is taken over drawing uniformly at random an $m$-candidate, $n$-voter Dodgson election (i.e. all $(m!)^n$ Dodgson elections having $m$ candidates and $n$ voters are equally likely to be chosen).

2.

$$\Pr[\exists c \in C | \texttt{GreedyWinner}(\langle C, c, V\rangle) \neq (DodgsonWinner(\langle C, c, V\rangle), \text{“definitely”})] < 2(m^2-m)e^{\frac{-n}{8m^2}},$$

where the probability is taken over drawing uniformly at random an $m$-candidate, $n$-voter Dodgson election.

*Proof.* First we need to establish under what conditions `GreedyScore` is self-knowingly correct.

**Claim 1.** For each $c \in C$, if for all $d \in C \setminus \{c\}$ it holds that

$$\|\{i \in [n] | c <_{v_i} d\}\| \leq \frac{2mn+n}{4m}, \text{ and} \tag{2}$$

$$\|\{i \in [n] | c \prec_{v_i} d\}\| \geq \frac{3n}{4m} \tag{3}$$

then $\texttt{GreedyScore}(\langle C, c, V\rangle) = (Score(\langle C, c, V\rangle), \text{“definitely”})$.

Proof: $\frac{2mn+n}{4m} = \frac{n}{2} + \frac{n}{4m}$ so if (2) holds then either $c$ beats $d$ and we are done. Otherwise, $d$ beats $c$ and flipping $\frac{n}{4m}$ votes where $c \prec d$ to $d \prec c$ would ensure that $c$ beats $d$. If (3) holds then there are strictly more than $\frac{n}{4m}$ flippable votes so `GreedyScore` will be able to make enough swaps to ensure that $c$ beats $d$.

Now we need to show what is the probability that these conditions are not met given a possible pair $c, d \in C, c \neq d$. We will later use the union bound to present the probability of failure of the algorithm overall.

**Claim 2.** For each $c, d \in C$ such that $c \neq d$,

$$\Pr[(\|\{i \in [n] | c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in [n] | c \prec_{v_i} d\}\| < \frac{3n}{4m})] < 2e^{\frac{-n}{8m^2}}.$$

Where the probability is taken over drawing uniformly at random an $m$-candidate, $n$-voter Dodgson election.

Proof sketch: Use the Chernoff theorem to show that

$$\Pr[\|\{i \in [n] | c <_{v_i} d\}\| > \frac{2mn+n}{4m}] < e^{\frac{-n}{8m^2}} \text{ and} \tag{4}$$

$$\Pr[\|\{i \in [n] | c \prec_{v_i} d\}\| < \frac{3n}{4m}] < e^{\frac{-n}{8m^2}} \tag{5}$$

by creating random variables $X_i$ for the events $c <_{v_i} d$ and $c \prec_{v_i} d$ and adjusting their values to utilize the theorem appropriately. Then use the union bound to prove Claim 2.

Now we can prove the main statements of the theorem. Item 1 follows from applying the union bound to claim 2:

$$\bigvee_{d \in C \setminus \{c\}} \Pr[(\|\{i \in [n] | c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in [n] | c \prec_{v_i} d\}\| < \frac{3n}{4m})].$$

Item 2, follows similarly using the union bound:

$$\bigvee_{c,d \in C \ s.t. \ c \neq d} \Pr[(\|\{i \in [n] | c <_{v_i} d\}\| > \frac{2mn+n}{4m}) \vee (\|\{i \in [n] | c \prec_{v_i} d\}\| < \frac{3n}{4m})].$$

Note that $\|\{c, d \in C \ s.t. \ c \neq d\}\| = (m^2 - m)$. □

# 5 Experiments

In this section we present empirical results based on our implementation the `GreedyWinner` algorithm from the previous section (see appendix for the python code). As proved in the previous section, the algorithm should succeed with high probability when $n \sim \Omega(m^2)$ so we tested values for $n$ ranging between $n = m^{1.5}$ to $m^{2.5}$ for each $m$ between 5 and 50. For $m \in [50, 100]$ we ran experiments for $n = m^{2.1}$ to $m^{2.5}$ Note that the experimental values for $m$ were not evenly spaced apart. Each pair of $n, m$ was run for 100 trials, each trial consisting of generating an election uniformly at random, picking a random candidate to test, and running `GreedyWinner` on that instance. A success was when `GreedyWinner` returned "definitely", as in it was self-knowingly correct.

At 100 candidates, the 100 trials took 9 hours, while at 5 candidates, the trials took about 3 minutes.

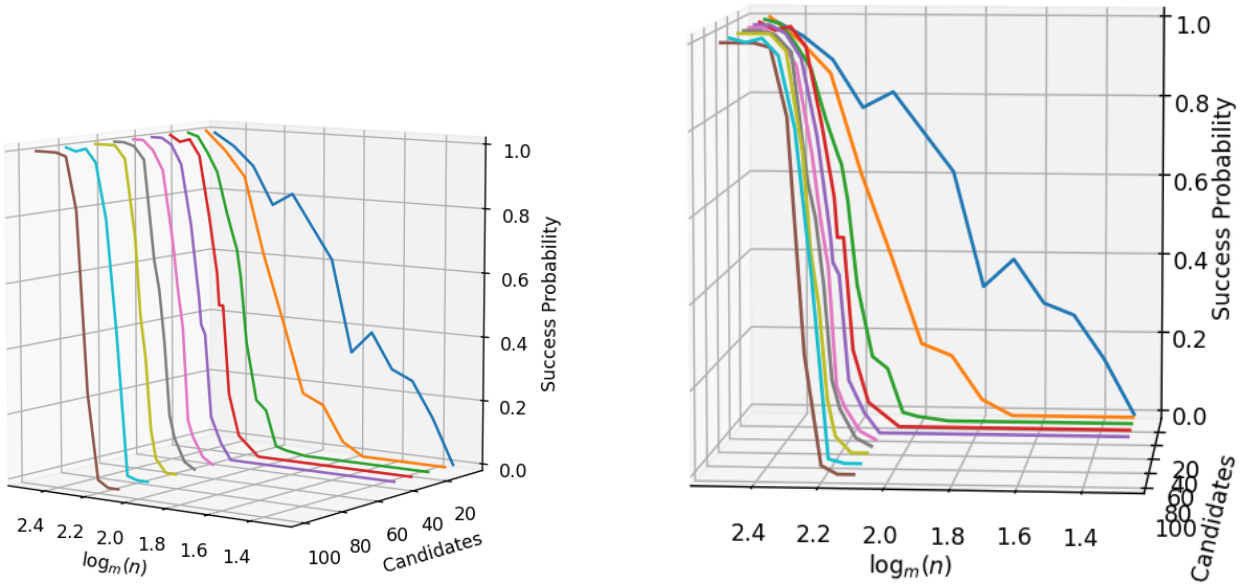


Figure 1: 3D line plot showing the success probability of the `GreedyWinner` algorithm. Each line color corresponds to a particular number of candidates. 100 trials were run for each pair of $n, m$. Error bars were omitted to not clutter the graph. For reference they are at most $\pm 0.08$.

Figure 1 confirms the theoretical result that `GreedyWinner` succeeds with high probability when $n \sim \Omega(m^2)$. Though the figure shows close to 1 probability of success when $n \geq m^{2.3}$, this can be explained by the other less significant terms in the polynomial which are showing a large effect at low values of $m$. Due to computational constraints, it was only feasible to experiment with up to 100 candidates. Observe also that when $m \leq 30$, the less significant terms have a large effect, dragging the line towards linear in $\log_m(n)$ (the exponent), as $m$ decreases.

# 6 Conclusion

In the project we have shown that DodgsonWinner is $\Theta_2^P$-complete by using recent results from Lukasiewicz and Malizia [2017]. We have also expanded upon the theory of the polynomial hierarchy and the classes $\Theta_k^P$, which is not commonly known. Finally, experimental results of a polynomial time heuristic algorithm, `GreedyWinner`, were presented.

We mention a few possible extensions. First, is there a way to use SAT solvers or Integer Linear Programming to solve DodgsonScore? This would enable a better algorithm for solving DodgsonWinner. While pursuing a backtracking based modification of `GreedyWinner`, we ran into the barrier that one would have to verify that any branch was in fact the minimum amount of swaps to make a candidate a Condorcet winner. This would be another avenue to investigate.

## References


Scott Aaronson, Greg Kuperberg, and Christopher Granade. The complexity zoo, 2005.

J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989. ISSN 01761714, 1432217X. URL http://www.jstor.org/stable/41105913.

Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia. *Handbook of computational social choice*. Cambridge University Press, 2016.

Marquis de Condorcet and MJAN de Caritat. An essay on the application of analysis to the probability of decisions rendered by a plurality of votes. *Classics of social choice*, pages 91–112, 1785.

L. A. Hemachandra. The strong exponential hierarchy collapses. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 110–122, New York, NY, USA, 1987. Association for Computing Machinery. ISBN 0897912217. doi: 10.1145/28395.28408. URL https://doi.org/10.1145/28395.28408.

Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Exact analysis of dodgson elections: Lewis carroll's 1876 voting system is complete for parallel access to NP. *CoRR*, cs.CC/9907036, 1999. URL https://arxiv.org/abs/cs/9907036.

Lane A. Hemaspaandra and Ryan Williams. An atypical survey of typical-case heuristic algorithms. *CoRR*, abs/1210.8099, 2012. URL http://arxiv.org/abs/1210.8099.

Christopher M. Homan and Lane A. Hemaspaandra. Guarantees for the success frequency of an algorithm for finding dodgson-election winners. *CoRR*, abs/cs/0509061, 2005. URL http://arxiv.org/abs/cs/0509061.

Köbler, Johannes, Schöning, Uwe, and Wagner, Klaus W. The difference and truth-table hierarchies for np. *RAIRO-Theor. Inf. Appl.*, 21(4):419–435, 1987. doi: 10.1051/ita/1987210404191. URL https://doi.org/10.1051/ita/1987210404191.

Thomas Lukasiewicz and Enrico Malizia. On the complexity of mcp-nets. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 558–564. AAAI Press, 2016.

Thomas Lukasiewicz and Enrico Malizia. A novel characterization of the complexity class thetakp based on counting and comparison. *Theoretical Computer Science*, 694: 21 – 33, 2017. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2017.06.023. URL http://www.sciencedirect.com/science/article/pii/S0304397517305352.

Christos H. Papadimitriou and Stathis K. Zachos. Two remarks on the power of counting. In Armin B. Cremers and Hans-Peter Kriegel, editors, *Theoretical Computer Science*, pages 269–275, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. ISBN 978-3-540-39421-1.

Klaus W Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

# A Code

## A.1 Algorithm

```
from math import floor

def greedy_score(C: list, c: int, V: list) -> tuple:
    """C is list of candidates 0 to m-1 (m candidates)
    V is matrix of votes, v[i][0] is least preferred, v[m] most preferred
    c is the candidate in question, int"""
    C_minus_c = [d for d in C if d != c] #candidates without c

    deficit, swaps = {}, {}
    for d in C_minus_c: #initialize counter variables
        deficit[d] = 0
        swaps[d] = 0
    for v in V:
        i = 0
        while v[i] != c:
            d = v[i]
            deficit[d] = deficit[d] - 1
            i += 1
        if i < len(v) - 1:
            d = v[i+1]
            swaps[d] = swaps[d] + 1
        i += 1
        while i <= len(v) - 1:
            d = v[i]
            deficit[d] = deficit[d] + 1
            i += 1

    confident = True
    #now calculate the score
    score = 0
    for d in C_minus_c:
        if deficit[d] >= 0:
            score += floor(deficit[d]/2) + 1
            if deficit[d] >= 2 * swaps[d]:
                confident = False
                score += 1
    return (score, confident)

def greedy_winner(C: list, c: int, V: list) -> tuple:
    winner = True
    cscore, confident = greedy_score(C,c,V)
    for d in C:
        if d != c: #only check with d that isn't c
            dscore, dcon = greedy_score(C,d,V)
            if dscore < cscore:
                winner = False
            if dcon == False:
                confident = False
```

```
    return (winner, confident)
```

## A.2 Data Collection

```
import os
from random import shuffle, choice
import numpy as np
from multiprocessing import Pool
import pandas as pd
import pickle as pk

from alg import *


#want to test the algorithm for a range of n,m
#and graph it. how many trials per run? depends on
#size of probability

def collect_data(candidates, exps):
    trials = 100

    input_list = [(e, c, trials) for e in exps for c in candidates]
    with Pool(processes=os.cpu_count() - 1) as pool:
        process = pool.map_async(run_trial, input_list)
        data = process.get()

    write_dataframe(data, str([candidates, exps]))

def run_trial(input_data):
    exp, candidates, trials = input_data
    data_dict = {'v_exp': exp,
                 'candidates': candidates,
                 'success': test_sample(candidates ** exp, candidates, trials),
                 'trials': trials}

    return data_dict

def test_sample(n, m, trials = 100):
    success = 0
    for i in range(trials):
        C, V = uniform_random(n, m)
        res = greedy_winner(C, 0, V)
        if res[1]:
            success += 1
    return success/trials

def uniform_random(n, m):
    """n, int, number of voters
    m, the number of candidates
    returns a uniform random election"""
    #make sure inputs are ints
    n, m = int(n), int(m)
```

```
    V = [list(range(m)) for i in range(n)]
    for v in V:
        shuffle(v)

    C = list(range(m))
    return C, V


def write_dataframe(data, experiment_name):
    df = pd.DataFrame(data)  # empty dicts will be stored as NaNs
    df.name = experiment_name
    df.to_pickle('./df_' + df.name)

    '''to unpickle:
    df =pd.read_pickle("./dummy.pkl")
    '''


def combine():
    abs_folder_path = get_exp_path()
    frames = []
    for filename in os.listdir(abs_folder_path):
        frames.append(pd.read_pickle("data/" + filename))
    return pd.concat(frames)

def get_exp_path():
    script_dir = os.path.dirname(__file__)
    rel_path = "../data"
    return os.path.join(script_dir, rel_path)

def lower(m):
    #returns number of voters that makes probability less than 1
    return -8 * m **2 * np.log(1 / (2 * (m-1) * m))
```