Providing an Alternate Version of the Logspace Hierarchy to Recreate the Polynomial Hierarchy

Marina Knittel[†], Jana Novotná[‡], Jakub Pekárek[‡],
Václav Rozhoň[‡], Štěpán Šimsa[‡], Jakub Svoboda[‡]

***Abstract*—The polynomial hierarchy is a central topic in computer science today, particularly because of its close relationship with the infamous question of whether or not P=NP. For instance, we know that P=NP implies that the polynomial hierarchy collapses. Therefore studying the polynomial hierarchy, for instance finding new ways to define its structure, is an important part of research in theoretical computer science. This project aimed to discover a new hierarchical structure that has a strong correspondence with the polynomial hierarchy. In the end, we discovered a structure based off of the logspace hierarchy that is equivalent to the polynomial hierarchy when given an additional quantifier. The main consequence of this result is that we now have a new way to consider the polynomial hierarchy. Additionally, we were able to define a new set of complete problems for various levels of the hierarchy. More generally, we hope this work can act as a stepping stone for further study in formulations of complexity classes that have somehow relate to the polynomial hierarchy. The results also provide a strong framework for the discovery of new natural complete problems for the polynomial hierarchy phrased in terms of the properties of our model as opposed to the standard model. The ultimate goal is to generally improve our understanding of the various components of the polynomial hierarchy, especially P and NP, in the hopes of answering broader questions such as whether or not P=NP.**

***Index Terms*—complexity theory, polynomial hierarchy, P, NP**

I. Introduction

We provide an alternate model for the logspace hierarchy by altering the function of a canonical Turing machine. This machine can be most simply characterized as a typical logspace machine with two modifications to the way it interprets nondeterministic tapes. First, it can only read the nondeterministic tape once, from left to right. Second, instead of including one alternating nondeterministic tape, it has simultaneous access to one nondeterministic tape for every quantifier in the given problem.

Most importantly, we show that this model is equivalent to the polynomial hierarchy (PH) at higher levels. To do this, we first show that the new hierarchy is embedded in PH by simulating the kth level of PH with a machine representing the $(k+1)$th level of our hierarchy. Then we show that these two levels are actually equivalent by adapting the proof that NL is in P and applying it to higher levels of our hierarchy. This is sufficient to prove that the hierarchies are equivalent for higher levels of our modified logspace hierarchy. To further classify the model, we also characterize a typical complete problem, SAT with logarithmic pathwidth, and extend it to all levels of the hierarchy.

NSF grant CCF-1559855
† Department of Computer Science, Harvey Mudd College
‡ Faculty of Mathematics and Physics, Charles University

II. Preliminaries

This paper introduces the notion of a new hierarchy based off of a modified Turing machine. We then relate the hierarchy to another intensively studied hierarchy, the polynomial hierarchy, or PH, shown in Figure 1. In this section, we discuss the fundamentals of PH, as well as introduce other concepts used throughout the paper.

A. The Polynomial Hierarchy

The base of PH is the complexity class P, containing all languages that can be solved by a polynomial time machine. The basic polynomial time machine for PH consists of a Turing machine with an input tape and a polynomial-sized work tape, and is allowed to run for polynomial time.

$$\begin{tikzcd}
 & \vdots & \\
\Sigma_3^P & & \Pi_3^P \\
 & \Delta_3^P \arrow[ul] \arrow[ur] & \\
\Sigma_2^P \arrow[uu] \arrow[ur] & & \Pi_2^P \arrow[uu] \arrow[ul] \\
 & \Delta_2^P \arrow[ul] \arrow[ur] & \\
\mathbf{NP} = \Sigma_1^P \arrow[uu] \arrow[ur] & & \Pi_1^P = \mathbf{coNP} \arrow[uu] \arrow[ul] \\
 & \Sigma_0^{\mathrm{P}} = \mathbf{P} = \Pi_0^{\mathrm{P}} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Fig. 1. The polynomial hierarchy. Arrows represent containment. The pattern continues upwards. This paper focuses on classes Σ_k^P and Π_k^P [3].

The next level in PH contains NP and coNP. Languages in NP can be represented as problems in P with an additional existential variable input. Similarly, languages for coNP problems look like languages in P with a universal variable. Such problems are written as follows.

Definition II.1. *A language L is in NP if there exists a polynomial time Turing machine M such that for input x*

$$x \in L \iff \exists\, a\, M(x,a) = 1.$$

Similarly, a language L is in coNP if there exists a polynomial time Turing machine M such that for input x

$$x \in L \iff \forall\, a\, M(x,a) = 1.$$

This addition of a single nondeterministic (existential or universal) variable in the hierarchy's base language can be used to define the first level complexity classes of that hierarchy. If that variable is existential, then we denote this class Σ, and if it's universal, then we denote it Π. In the case of PH, we can denote NP as Σ_1^P and coNP as Π_1^P, where the superscript P denotes we are in PH.

Further addition of alternating nondeterministic variables can be used to define the higher levels of hierarchies. For PH, [1] describes the levels as follows.

Definition II.2. *A language L is in Σ_k^p for some k if there exists a polynomial time Turing machine M such that for input x*

$$x \in L \iff \exists\, a_1 \forall\, a_2 \ldots Q\, a_k\, M(x, a_1, \ldots, a_k) = 1,$$

where Q is $\exists$ if k is odd, else Q is $\forall$.

Similarly, language L is in Π_k^p for some k if there exists a polynomial time Turing machine M such that for input x

$$x \in L \iff \forall\, a_1 \exists\, a_2 \ldots Q\, a_k\, M(x, a_1, \ldots, a_k) = 1,$$

where Q is $\exists$ if k is even, else Q is $\forall$.

The complexity classes that arise from this definition make up PH. Some important known features of the general hierarchical format is that for any k, we know that $\Sigma_k, \Pi_k \subseteq \Sigma_{k+1}, \Pi_{k+1}$. In the case of PH, we have P $= \Sigma_0 = \Pi_0$, thus P $\subseteq$ NP, coNP.

Note that our new hierarchy is based off the logspace hierarchy. The logspace hierarchy can be defined simlarly to PH, with the exception that all the aforementioned Turing machines should use only logarithmic space. The levels of the logspace hierarchy are denoted Σ_k^L and Π_k^L, where L signifies these classes are in the logspace hierarchy.

B. Configuration Graphs

Configuration graphs are useful representations of the functionality of various machines. In this paper, we will only discuss configuration graphs in terms of Turing machines.

When a Turing machine is run, the machine keeps track of its internal state at every timestep. In the vocabulary of configuration graphs, we call these states "congfigurations". At each timestep, the machine uses the information stored in the configuration and any applicable tape (the input tape or nondeterministic tapes) to determine what configuration to move to next. The set of such states and transition rules is sufficient to describe the functionality of the Turing machine.

A configuration graph encodes this information by creating one vertex for each configuration of the Turing machine. For every possible transition between configurations of the machine, we add a directed edge from the starting configuration to the ending configuration and note on that edge the conditions of the relevant tapes required for the machine to take that path. This completes the configuration graph.

In a sense, the configuration graph is simply a new way to visualize the Turing machine. Because it encodes every possible state and transition of the Turing machine, configuration graphs can be used to simulate the Turing machine on any input. This technique will be useful for some of the proofs in the Results section of this paper.

C. Bounded Pathwidth SAT

Canonically, the term "pathwidth" refers to a graph property. It describes the "width" of something known as a path decomposition. However, the term can extend to the SAT problem to define a property of boolean expressions. When considering an instance of SAT, pathwidth refers to a quality of the path decomposition of the boolean expression. The decomposition is as follows.

Consider an instance of SAT, I. We will construct a path graph where each vertex, which we will refer to as a bag, on the path contains a subset of the variables that appear in I. Additionally, we restrict these bags such that if any variable appears in two bags along the path, it must also appear in all bags in between. That way, variables must only lie in a single stretch of contiguous bags.

The purpose of these bags is to provide solutions for the clauses in SAT. In order to represent the expression for I, there must be a mapping from each clause in I to a bag in the path such that all variables that appear in the clause also appear in the bag. Now we introduce the term pathwidth with respect to an instance of SAT.

Definition II.3. *Consider I, an instance of* SAT. *Enumerate all the possible pathwidth decompositions of I. Let $w_{i,j}$ be the width of the ith bag in the jth path decomposition of I. Then the* ***pathwidth*** *of I is*

$$\textit{pathwidth} = \min_j \max_i w_{i,j}.$$

We now define the logarithmic bounded pathwidth SAT problem.

Definition II.4. LBPSAT *is the set of* SAT *problems with pathwidth at most $\mathcal{O}(\log n)$, where the path decomposition of the* SAT *instance is provided in the input.*

In other words, for an instance I of SAT to be a LBPSAT problem, it must have a path decomposition such that all bags have size at most $\mathcal{O}(\log n)$. And this decomposition must be passed in as part of the input. This problem will be further addressed in the discussion about complete problems for our modified logspace hierarchy.

III. Results

In our attempts to modify the logspace hierarchy, we looked to change the functionality of the underlying machine defining the hierarchy. The attempted modifications were made with the problem of classifying the group isomorphism problem in mind. However, the resulting model did not aid in the study of the group isomorphism problem, and simply resulted in an additional way to classify PH. Therefore, while it is useful to keep in mind the origins of this research, it is not important to further discuss the motivations behind the initial formulation of the new hierarchy. In this section, we will simply define the hierarchy and state our findings.

A. The Modified Logspace Hierarchy

Our hierarchy is a modified version of the logspace hierarchy. The modifications stem from an alteration made to the underlying machine present in the formulation of the different levels of the hierarchy. We made two fundamental changes to the functionality of this machine.

First, the machine can only read its nondeterministic tapes once, from left to right. This will decrease the strength of the machine, as the machine will have less flexibility when attempting to utilize the information of the nondeterministic tapes.

Second, the machine's nondeterministic tapes appear in a slightly different format. The PH machine, and consequently the traditional logspace machine, has a single nondeterministic tape. If the complexity class in question has multiple quantifiers, then the tape alternates between universal and existential tapes where appropriate. The head can move back and forth on this tape as much as it needs within the time specifications of the machine. In our version, the machine has a single unique tape for each quantifier. The tapes do not alternate. The machine has a head for each tape, and thus can switch between the tapes without additional work. This modification makes our machine stronger, as it can more quickly switch between reading the different nondeterministic variables.

Based off this definition, it is not directly clear whether or not our machine is ultimately stronger or weaker than the typical logspace machine. However, as previously stated, we discovered that this machine was equivalent to the machine in PH when you add an additional quantifier.

We will denote the levels of our hierarchy as $\hat{\Sigma}_k^L$ and $\hat{\Pi}_k^L$, where L again represents we are working in logspace, and the hat designates our new machine.

B. Connecting the Modified Logspace hierarchy to PH

Next, we aim to show that the modified logspace hierarchy with an additional quantifier is equivalent to PH. Before we do this, we introduce another modified logspace hierarchy. Components of this heirarchy are denoted $\bar{\Sigma}_k^L$ and $\bar{\Pi}_k^L$, which are the same as $\hat{\Sigma}_k^L$ and $\hat{\Pi}_k^L$, except the machine can now iterate back and forth over its multiple existential tapes. Thus it is stronger than our hierarchy.

Regarding this hierarchy, we claim the following.

Lemma III.1. $\Sigma_k^P = \bar{\Sigma}_k^L$.

The proof of this is not completely novel and relevant for this paper. One direction, $\Sigma_k^P \subseteq \bar{\Sigma}_k^L$, uses the same argument as Theorem III.4. The argument for $\bar{\Sigma}_k^L \subseteq \Sigma_k^P$ stems from the fact that logspace machines are generally speaking weaker than polynomial time machines. This will help us with the following Lemma.

We now show the containment of corresponding classes in the two hierarchies in one direction.

Lemma III.2. *For all $k \geq 0$, it holds that $\Sigma_k^P \subseteq \hat{\Sigma}_{k+1}^L$ and $\Pi_k^P \subseteq \hat{\Pi}_{k+1}^L$.*

Proof. Referring to Lemma III.1, it is sufficient to show that $\bar{\Sigma}_k^L \subseteq \hat{\Sigma}_{k+1}^L$. Consider a machine $\bar{M}$ for $\bar{\Sigma}_k^L$. We will construct a machine, $\hat{M}$ for $\hat{\Sigma}_{k+1}^L$, to simulate this machine.

Denote the tapes for $\bar{M}$ as $\bar{t}_1, \ldots, \bar{t}_k$, and for $\hat{M}$ as $\hat{t}_1, \ldots, \hat{t}_{k+1}$. We designate each $\hat{M}$ tape as a tuple containing multiple parts. For some general tape, let $\hat{t} = (i, x_1, \ldots, x_{p(n)})$, where $p(n)$ is the running time of $\bar{M}$ (we know $p(n)$ is a polynomial because $\bar{M}$ will run in polynomial time). Note $\hat{t}_1$ will not have the first part of the tuple, i, and $\hat{t}_{k+1}$ will only consist of the final part, $x_1, \ldots, x_{p(n)}$. For our proof, each i will act as an index into the previous nondeterministic tape, and all x_j for each tape will act as copies of the corresponding nondeterministic tapes for $\bar{M}$. This description will be clarified in the following paragraphs.

Now we will describe the construction of $\hat{M}$. It will start by reading the i portion of each of its nondeterministic tapes and storing it on its work tape. The purpose of i is to index bits in the x_j sections of the previous existential tape. Since x_j is a copy of a nondeterministic tape of $\bar{M}$, then it is at most $p(n)$ long. To index that, you need $\log(p(n)) = \mathcal{O}(\log n)$ bits. Thus, all i can be stored on the $\hat{M}$ logspace work tape.

Next, our machine acts as $\bar{M}$ on the input. However, whenever the iterator on some $\bar{t}_j$ head moves left (backwards), the head on $\hat{t}_j$ moves from its current component, x_k, to the next, x_{k+1}, to read that bit. As we read blocks, we store the bits in the current x_k that we know we will read in the next x_{k+1}. So when we move on to x_{k+1}, we can compare the corresponding bits in x_k and x_{k+1}.

When we compare these bits, we check equality. If the bits are equal, move on. Otherwise, if we are reading an existential tape ($\hat{t}_j$ for odd j), $\hat{M}$ rejects. Else, we are reading a universal tape, and $\hat{M}$ accepts. If neither of these events occur before $\hat{M}$ gets to $\hat{t}_{k+1}$, then it accepts if $\hat{t}_{k+1}$ is existential, and rejects if it's univeral.

We claim this accepts if and only if $\bar{M}$ accepts. If $\bar{M}$ accepts, we show that $\hat{M}$ accepts. Consider $\hat{t}_1$, an existential tape. Let every x_j on this tape contain the contents of $\bar{t}_1$. Then the x_j's for $\hat{t}_1$ are all the same, so $\hat{M}$ will not reject due to reading incongruous bits on this tape.

Now look at $\hat{t}_2$, a universal tape. If not all x_j's are equivalent, then there is some x_j and x_{j+1} that differ by at least one bit. Set the i portion of the $\hat{t}_3$ tape such $\hat{M}$ compares these two bits. Then $\hat{M}$ will accept, as desired. Otherwise, all x_j's are equivalent, and $\hat{M}$ will neither accept nor reject.

The machine continues in this manner. Consider the case where $\hat{M}$ reaches the $\hat{t}_k$ and has not yet accepted. If $\hat{t}_k$ is universal, $\hat{M}$ may accept following the logic of the previous paragraph. Otherwise, note that $\hat{M}$ will have accurately simulated $\bar{M}$ as it reads all of its nondeterministic tapes. Thus, at the end, because $\bar{M}$ accepts, $\hat{M}$ accepts.

Thus we have shown one direction of the bidirectional implication. Next, consider when $\bar{M}$ rejects. The proof of this follows the previous proofs, except we want to look at where $\hat{M}$ rejects and show it won't accept. For the existential tapes, we can equivalently view them as universal tapes and show they won't reject. Universal tapes can similarly be viewed as existential tapes. Then, the same result holds. Thus, the bidirectional implication holds.

Therefore we have constructed a machine $\hat{M}$ to simulate $\bar{M}$. Thus $\bar{\Sigma}_k^L \subseteq \hat{\Sigma}_{k+1}^L$, and, consequently, $\Sigma_k^P \subseteq \hat{\Sigma}_{k+1}^L$, as desired. ☐

Next, we show containment in the reverse direction.

Lemma III.3. *For all $k \geq 0$, it holds that $\hat{\Sigma}_{k+1}^L \subseteq \Sigma_k^P$ and $\hat{\Pi}_{k+1}^L \subseteq \Pi_k^P$.*

Proof. To prove this, we only consider the Σ classes and extend it to Π. This work is based off the widely known proof from [2] that the complexity class NL under the classical logspace model is a subset of P. It can apply to the modified hierarchy with little alteration to show that any machine for $\hat{\Sigma}_{k+1}^L$ can be simulated by a machine for Σ_k^P for all k.

Consider some k. Using a Σ_k^P machine, we will attempt to construct a given $\hat{\Sigma}_{k+1}^L$ machine using its configuration graph. Let's call the $\hat{\Sigma}_{k+1}^L$ machine we're trying to simulate M_L, and the Σ_k^P machine we're constructing to run the simulation M_P.

Consider when the k alternations on the nondeterministic tape of M_P contain the contents of the first k nondeterministic tapes of M_L. Given this information, M_P can hardwire the contents of these k tapes into the configuration graph of M_L. Now there is only one nondeterministic tape affecting the configuration graph of M_L.

We're left with a configuration graph for NL or coNL (depending on the remaining quantifier for M_L), and a P machine (M_P without its nondeterministic tape). Since we know NL, coNL $\subseteq$ P, clearly this final P machine can be used to simulate the NL or coNL machine. Apply this simulation to construct our simulation for M_L using N_P.

Thus, we have shown that for any $\hat{\Sigma}_{k+1}^L$ machine, there is a Σ_k^P machine that can simulate it. A similar argument shows that for every $\hat{\Pi}_{k+1}^L$ machine, there is a Π_k^P machine that can simulate it. Thus, we have shown that $\hat{\Sigma}_{k+1}^L \subseteq \Sigma_k^P$ and $\hat{\Pi}_{k+1}^L \subseteq \Pi_k^P$. ☐

Combining the results for the Lemma III.2 and III.3, we can directly conclude the following.

Theorem III.1. *For all $k \geq 0$, it holds that $\Sigma_k^P = \hat{\Sigma}_{k+1}^L$ and $\Pi_k^P = \hat{\Pi}_{k+1}^L$.*

Therefore, our modified logspace hierarchy with an additional quantifier is equivalent to PH.

C. Complete Problems for the Modified Logspace Hierarchy

One way to characterize a hierarchy is to define complete problems for the complexity classes at different levels of the hierarchy. While we already know many complete problems for the different levels of the modified logspace hierarchy—they're the same as the ones in the correspond levels of PH—, understanding the complete problems that naturally fall out of our model might give us a new way to look at PH.

In our studies, we came across a simple formulation for a complete problem for $\hat{\Sigma}_1^L$, LBPSAT, which can be extended to define a complete problem for all other levels of the hierarchy.

Lemma III.4. LBPSAT *is complete for $\hat{\Sigma}_k^L$.*

Proof. First, we show that LBPSAT is in $\hat{\Sigma}_1^L$ by defining a machine M for $\hat{\Sigma}_1^L$ that can solve LBPSAT.

Consider some instance I of LBPSAT. We now describe the construction of machine M. We can specify that the input to M, the LBPSAT boolean expresssion, must be provided in a certain order. Specifically, the clauses must appear in the same order that their corresponding bags appear in the path decomposition of I. This is important to describe how M will act on the input.

Because we are working with $\hat{\Sigma}_1^L$, we know M has one existential tape. Machine M will accept when this existential tape contains the assignments that satisfy the input. The variable assignments on the existential tape are provided in the same order that the variables appear on the input tape.

Machine M will act as follows. It reads directly through the clauses of the input tape. Since M is provided the path decomposition in the input tape, it knows two things every time it comes across a new bag: all new variables it needs to store, and all variables it needs to drop. When it reaches a clause in a new bag, it "erases" all dropped variables from and writes all new variables to its work tape.

As it reads through the clauses of I, it checks what the value of each clause is. If any clause evaluates to 0, M stops and outputs 0. Otherwise, if M finds no clauses evaluate to 0, M outputs a 1.

Clearly, M outputs the correct value for the given input. We now confirm that it is a machine for $\hat{\Sigma}_1^L$. The machine clearly only reads the existential tape once from left to right. So all we have to do is confirm that it only uses logarithmic space.

We can think of the bags in the path decomposition of I as the set of variables on M's work tape. As soon as M reaches the end of the bag, it erases the variables it no longer needs and overwrites those cells with the new variables it comes across (which come from the next bag in the path). Thus it only needs to store one bag of variables at a time. Since the number of variables in any bag of I is at most $\mathcal{O}(\log n)$, the number of cells M uses on its work tape is at most $\mathcal{O}(\log n)$. Thus it satisfies space limit, and therefore is a machine for $\hat{\Sigma}_1^L$.

Next, we must show that LBPSAT is hard for $\hat{\Sigma}_1^L$. This will be based off of the proof in [2] that SAT is complete for NP. Apply the same deconstruction of the machine M for $\hat{\Sigma}_1^L$

into boolean expressions. Call this boolean expression I. We will prove I is an instance of LBPSAT.

Because we know M has a worktape of size $\mathcal{O}(\log n)$, we have some additional information about the clauses in I. Consider all the clauses at some timestep t. These clauses specify precisely the state of M at timestep t. A table of the elements required to describe M for each state and the number of bits required in their representation is depicted in Table 1.

TABLE I
CONTENTS REQUIRED FOR EACH STATE OF M

Element	**Bits in Representation**
Input tape head location	$\mathcal{O}(\log n)$
Work tape head location	$\mathcal{O}(\log(\log n))$
Work tape contents	$\mathcal{O}(\log n)$

As exhibited by Table 1, the number of bits required to store the entire state of the machine at any given time, besides the initial and final timesteps, is $\mathcal{O}(\log n)$. Thus the total number of variables that can be present in the clauses at any such timestep is $\mathcal{O}(\log n)$. Note that most of the clauses for the initial and final states requite a constant number of bits. The one exception is the initial contents of the work tape, which clearly only contains $\mathcal{O}(\log n)$ bits as well. Therefore, for all states, only $\mathcal{O}(\log n)$ variables are required. And note that since all these variables are time-dependent, they cannot be present in any other timesteps.

Therefore, we could construct a path for I where we create a single bag for each timestep. This bag contains precisely the variables in the clauses corresponding to that timestep. So each bag only contains $\mathcal{O}(\log n)$ variables. Additionally, since variables only show up in the clauses corresponding to a single timestep, the variables will also only show up in a single bag. In other words, each variable is stored in a single contiguous stretch of bags (or a single bag). Thus I is an instance of LBPSAT.

Applying the same argument used in [2], we know I is true if and only if M outputs 1 on its input. Namely, M can be represented by a formula in LBPSAT. Thus LBPSAT is hard for $\hat{\Sigma}_1^L$, completing our proof of completeness. $\square$

It then follows that we can construct similar complete problems for other levels of the modified logspace hierarchy.

Theorem III.2. *Consider the language L such that $x \in L$ if and only if there is a machine M where $\exists a_1 \ldots Q\, a_k M(x, a_1, \ldots, a_k) = 1$, where the quantifiers alternate between existential and universal, and M is a Turing machine for $\hat{\Sigma}_k^L$. Then L is complete for $\hat{\Sigma}_k^L$.*

Similarly, if L is defined with a $\forall$ quantifer in the beginning of the equation, and M is a Turing Machine for $\hat{\Pi}_k^L$, then L is complete for $\hat{\Pi}_k^L$.

The proof of this directly stems from Lemma III.4. Note the proof does not depend on how many nondeterministic tapes M has for either direction of the proof.

And thus we have characterized a set of complete problems for all levels of the modified logspace hierarchy. Additionally, combining Theorem III.1 and Theorem III.2, we can directly conclude the following.

Theorem III.3. *Consider the languages mentioned in Theorem III.2. For any given $k \geq 2$, these languages are complete for Σ_{k-1}^P and Π_{k-1}^P respectively.*

Thus we have also provided an alternate representation for the complete problems of the polynomial hierarchy.

IV. CONCLUSION

One of the most notorious questions in computer science is whether or not P = NP. This question can be rephrased as: "Does PH collapse to P?" Gaining a deeper insight into PH, its possible representations, and problems that relate to it can help us further understand important questions like the relationship between P and NP. In addition, showing that our modified logspace hierarchy is closely tied to PH can give us more information about how complexity structures like these work. For instance, the modified logspace hierarchy collapses if and only if PH collapses. If we can find relations between other complexity classes and the modified logspace hierarchy, then that relationship should directly translate to some sort of connection with PH. Overall, this new model gives us a different perspective with which to view PH, and could be used to find new relations to PH.

It is not completely clear what direction future work in this area should take to utilize the findings of this report. The modified logspace hierarchy has a clear connection to a bounded pathwidth representation of SAT. Perhaps one might attempt to interpret previous results on this topic in the context of the new model. In addition, the modified logspace hierarchy has strong ties to the traditional logspace hierarchy, however there is a divergence between the classes after the first level of the hierarchy. Nevertheless, it may prove worthy to examine the relationship further to explore how the logspace hierarchy and PH relate.

In order to more fully characterize the modified logspace hierarchy and related problems, one could search for more natural problems that are complete for various levels of the model. This most likely implies searching for complete problems for various levels of PH and applying a pathwidth restriction or an equivalent type of modification.

ACKNOWLEDGMENT

This work would not be possible without grant CCF-1559855 for financial support. I would like to thank Dr Eric Allender and Dr Periklis Papakonstantinou for helpful discussion, as well as Dr Lazaros Gallos and DIMACS at Rutgers University for their support. An additional thanks to Kayla Cummings and Edgar Jaramillo for useful comments and suggestions.

REFERENCES

[1] S. Arora and B Barak, "The polynomial hierarchy and alternations," in *Computational Complexity: A Modern Approach*, 1st ed. Cambridge, UK: Cambridge University Press, 2007, ch. 5, sec. 2, pp. 97-99.

[2] M. R. Garey and D. S. Johnson, “The Theory of NP-Completeness,” in *Computers and Intractability: A Guide to the Theory of NP-Completeness*, 1st ed. Murray Hills, New Jersey, USA: Bell Telephone Laboratories, 1979, ch. 2, sec. 6, pp. 38-44.

[3] The Polynomial Hierarchy. (2015). [image] Available at: https://plato.stanford.edu/entries/computational-complexity/fig4.png [Accessed 24 Jul. 2017].