Coarse-Grained Complexity for Dynamic Algorithms

Sayan Bhattacharya[*] Danupon Nanongkai[†] Thatchaphol Saranurak[‡]

Abstract

To date, the only way to argue *polynomial lower bounds* for dynamic algorithms is via *fine-grained complexity arguments*. These arguments rely on strong assumptions about *specific problems* such as the Strong Exponential Time Hypothesis (SETH) and the Online Matrix-Vector Multiplication Conjecture (OMv). While they have led to many exciting discoveries, dynamic algorithms still miss out some benefits and lessons from the traditional "coarse-grained" approach that relates together classes of problems such as P and NP. In this paper we initiate the study of coarse-grained complexity theory for dynamic algorithms. Below are among questions that this theory can answer.

What if dynamic Orthogonal Vector (OV) is easy in the cell-probe model? A research program for proving *polynomial unconditional lower bounds* for dynamic OV in the cell-probe model is motivated by the fact that many conditional lower bounds can be shown via reductions from the dynamic OV problem (e.g. [Abboud, V.-Williams, FOCS 2014]). Since the cell-probe model is more powerful than word RAM and has historically allowed smaller upper bounds (e.g. [Larsen, Williams, SODA 2017; Chakraborty, Kamma, Larsen, STOC 2018]), it might turn out that dynamic OV is easy in the cell-probe model, making this research direction infeasible. Our theory implies that if this is the case, there will be very interesting algorithmic consequences: If dynamic OV can be maintained in polylogarithmic worst-case update time in the cell-probe model, then so are several important dynamic problems such as k-edge connectivity, $(1+\epsilon)$-approximate mincut, $(1+\epsilon)$-approximate matching, planar nearest neighbors, Chan's subset union and 3-vs-4 diameter. The same conclusion can be made when we replace dynamic OV by, e.g., subgraph connectivity, single source reachability, Chan's subset union, and 3-vs-4 diameter.

Lower bounds for k-edge connectivity via dynamic OV? The ubiquity of reductions from dynamic OV raises a question whether we can prove conditional lower bounds for, e.g., k-edge connectivity, approximate mincut, and approximate matching, via the same approach. Our theory provides a method to refute such possibility (the so-called *non-reducibility*). In particular, we show that there are no "efficient" reductions (in both cell-probe and word RAM models) from dynamic OV to k-edge connectivity under an assumption about the classes of dynamic algorithms whose analogue in the static setting is widely believed. We are not aware of any existing assumptions that can play the same role. (The NSETH of Carmosino et al. [ITCS 2016] is the closest one, but is not enough.) To show similar results for other problems, one only need to develop efficient randomized *verification protocols* for such problems.

1 Introduction

In a *dynamic problem*, we first get an input instance for *preprocessing*, and subsequently we have to handle a sequence of *updates* to the input. For example, in the graph connectivity problem [35, 42], an n-node graph G is given to an algorithm to preprocess. Then the algorithm has to answer whether G is connected or not after each edge insertion and deletion to G. (Some dynamic problems also consider *queries*. (For example, in the connectivity problem an algorithm may be queried whether two nodes are in the same connected component or not.) Since queries can be phrased as input updates themselves, we will focus only on updates in this paper. Algorithms that handle dynamic problems are known as *dynamic algorithms*. The *preprocessing time* of a dynamic algorithm is the time it takes to handle the initial input, whereas the worst-case *update time* of a dynamic algorithm is the *maximum* time it takes to handle any update. Although dynamic algorithms are also analyzed in terms of their *amortized update times*, we emphasize that the results in this paper deal only with worst-case update times. A holy grail for many dynamic problems – especially those concering dynamic *graphs* under edge deletions and insertions – is to design algorithms with *polylogarithmic update times*. From this perspective, the computational status of many classical dynamic problems still remain widely open.

Example: Family of Connectivity Problems. A famous example of a widely open question is for the family of connectivity problems: (i) The problem of maintaining whether the input dynamic graph is connected (the dynamic *connectivity* problem) admits a *randomized* algorithm with polylogarithmic worst-case update time. It is an active, unsettled line of research to determine whether it admits deterministic polylogarithmic worst-case update time (e.g. [28, 26, 35, 37, 67, 58, 42, 73, 52, 53]). (ii) The problem of maintaining whether the input dynamic graph can be disconnected by deleting an edge (the dynamic 2-*edge connectivity* problem) admits polylogarithmic amortized update time [37, 38], but its worst-case update time

[*]University of Warwick, UK.

[†]KTH Royal Institute of Technology, Sweden.

[‡]Toyota Technological Institute at Chicago, USA. Work partially done while at KTH Royal Institute of Technology.

(even with randomization) remains polynomial [68]. (iii) For dynamic k-edge connectivity with $k \geq 3$, the best update time – amortization and randomization allowed – suddenly jumps to $\tilde{O}(\sqrt{n})$ where $\tilde{O}$ hides polylogarithmic terms. Indeed, it is a major open problem to maintain a $(1+\epsilon)$-approximation to the value of global minimum cut in a dynamic graph in polylogarithmic update time [68]. Doing so for k-edge connectivity with $k = O(\log n)$ is already sufficient to solve the general case.

Other dynamic problems that are not known to admit polylogarithmic update times include approximate matching, shortest paths, diameter, max-flow, etc. [68, 61]. Thus, it is natural to ask: *can one argue that these problems do not admit efficient dynamic algorithms?*

A traditionally popular approach to answer the question above is to use information-theoretic arguments in the *bit-probe/cell-probe model*. In this model of computation, all the operations are free *except* memory accesses. (In more details, the bit-probe model concerns the number of bits accessed, while the cell-probe model concerns the number of accessed cells, typically of logarithmic size.) Lower bounds via this approach are usually *unconditional*, meaning that it does not rely on any assumption. Unfortunately, this approach could only give small lower bounds so far; and getting a super-polylogarithmic lower bound for *any* natural dynamic problem is an outstanding open question is this area [46].

More recent advances towards answering this question arose from a new area called *fine-grained complexity*. While traditional complexity theory (henceforth we refer to it as *coarse-grained complexity*) focuses on classifying problems based on resources and relating together resulting classes (e.g. P and NP), fine-grained complexity gives us *conditional* lower bounds in the word RAM model based on various assumptions about specific problems. For example, assumptions that are particularly useful for dynamic algorithms are the Strong Exponential Time Hyposis (SETH), which concerns the running time for solving SAT, and the Online Matrix-Vector Multiplication Conjecture (OMv), which concerns the running time of certain matrix multiplication methods (more at, e.g., [57, 3, 34]). In sharp contrast to cell-probe lower bounds, these assumptions often lead to polynomial lower bounds in the word RAM model, many of which are tight.

While the fine-grained complexity approach has led to many exciting lower bound results, there are a number of traditional results in the static setting that seem to have *no* analogues in the dynamic setting. For example, one reason that makes the $P \neq NP$ assumption so central in the static setting is that proving and disproving it will both lead to stunning consequences: If the assumption is false then hundreds of problems in NP and bigger complexity classes like the polynomial hierarchy (PH) admit efficient algorithms; otherwise the situation will be the opposite.[1] In contrast, we do not see any immediate consequence to dynamic algorithms if someone falsified SETH, OMv, or any other assumptions.[2] As another example, comparing complexity classes allows us to speculate on various situations such as non-reducibility (e.g. [4, 43, 19]), the existence of NP-intermediate problems [44] and the derandomization possibilities (e.g. [40]). (See more in Section 4.) We cannot anticipate results like these in the dynamic setting without the coarse-grained approach, i.e. by considering analogues of P, NP, BPP and other complexity classes that are defined based on computational resources.

Our Main Contributions. We initiate a systematic study of coarse-grained complexity theory for dynamic problems in the bit-probe/cell-probe model of computation. We now mention a couple of concrete implications that follow from this study.

Consider the *dynamic Orthogonal Vector* (OV) problem (see Definition 2.4). Lower bounds conditional on SETH for many natural problems (e.g. Subgraph connectivity, ST-reachability, Chan's subset union, 3-vs-4 Diameter) are based on reductions from dynamic OV [3]. This suggests two research directions: (I) Prove strong *unconditional* lower bounds for many natural problems in one shot by proving a polynomial cell-probe lower bound for dynamic OV. (II) Prove lower bounds conditional on SETH for the family of connectivity problems mentioned in the previous page via reductions (in the word RAM model) from dynamic OV. Below are some questions about the feasibility of these research directions that our theory can answer. We are not aware of any other technique in the existing literature that can provide similar conclusions.

(I) What if dynamic OV is easy in the cell-probe model? For the first direction, there is a risk that dynamic OV might turn out to admit a polylogarithmic update time algorithm in the cell-probe model. This is

[1] For further consequences see, e.g., [1, 20, 21] and references therein.

[2] An indirect consequence would be that some barriers were broken and we might hope to get better upper bounds. This is however different from when P=NP where many hard problems would immediately admit efficient algorithms. Note that some consequences of falsifying SETH have been shown recently (e.g. [2, 71, 31, 22, 62, 41, 23]); however, we are not aware of any consequence to dynamic algorithms. It might also be interesting to note that Williams [72] estimates the likelihood of SETH to be only 25%.

because lower bounds in the word RAM model do not necessarily extend to the cell-probe model. For example, it was shown by Larsen and Williams [47] and later by Chakraborty et al [17] that the OMv conjecture [34] is false in the cell-probe model.

Will all the efforts be wasted if dynamic OV turns out to admit polylogarithmic update time in the cell-probe model? Our theory implies that this will also lead to a very interesting algorithmic consequence: If dynamic OV admits polylogarithmic update time in the cell-probe model, so do several important dynamic problems such as k-edge connectivity, $(1+\epsilon)$-approximate mincut, $(1+\epsilon)$-approximate matching, planar nearest neighbors, Chan's subset union and 3-vs-4 diameter. The same conclusion can be made when we replace dynamic OV by, e.g., subgraph connectivity, single source reachability, Chan's subset union, and 3-vs-4 diameter (see Theorem 2.1). Thus, there will be interesting consequences regardless of the outcome of this line of research.

Roughly, we reach the above conclusions by proving in the dynamic setting an analogue of the fact that if P=NP (in the static setting), then the polynomial hierarchy (PH) collapses. This is done by carefully defining the classes P^{dy}, NP^{dy} and PH^{dy} as dynamic analogues of P, NP, and PH, so that we can prove such statements, along with NP^{dy}-completeness and NP^{dy}-hardness results for natural dynamic problems including dynamic OV. We sketch how to do this in Sections 2, 5.

(II) Lower bounds for k-edge connectivity via dynamic OV? As discussed above, whether dynamic k-edge connectivity admits polylogarithmic update time for $k \in [3, O(\log n)]$ is a very important open question. There is a hope to answer this question negatively via reductions (in the word RAM model) from dynamic OV. Our theory provides a method to refute such a possibility (the so-called *non-reducibility*). First, note that any reduction from dynamic OV in the word RAM model will also hold in the (stronger) cell-probe model. Armed with this simple observation, we show that there are no "efficient" reductions from dynamic OV to k-edge connectivity under an assumption about the complexity classes for dynamic problems in the cell-probe model, namely $\mathsf{PH}^{dy} \not\subseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$ (see Theorem 2.2). We defer defining the classes AM^{dy} and coAM^{dy}, but note two things. (i) Just as the classes AM and coAM (where AM stands for *Arthur-Merlin*) extend NP in the static setting, the classes AM^{dy} and coAM^{dy} extend the class NP^{dy} in a similar manner. (ii) In the static setting it is widely believed that PH $\not\subseteq$ AM $\cap$ coAM because otherwise the PH collapses. Roughly, the phrase "*efficient reduction*" from problems X to Y refers to a way of processing each update for problem X by quickly feeding polylogarithmic number of updates as input to an algorithm for Y. All reductions from dynamic OV in the literaturethat we are aware of are efficient reductions.

Remark: We define our complexity classes in the cell-probe model, whereas the reductions from dynamic OV are in the word RAM model. This does not make any difference, however, since any reduction in the word RAM reduction continues to have the same guarantees in the (stronger) cell-probe model.

To show a similar non-reducibility result for any problem X, one needs to prove that $X \in \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$, which boils down to developing efficient randomized *verification protocols* for such problems. We explain this in more details in Section 2.

We are not aware of any existing assumptions that can lead the same conclusion as above. To our knowledge, the only conjecture that can imply results of this nature is the Nondeterministic Strong Exponential Time Hypothesis (NSETH) of [16]. However, it needs a stronger property of k-edge connectivity that is not yet known to be true. (In particular, Theorem 2.2 follows from the fact that k-edge connectivity is in $\mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$. To use NSETH, we need to show that it is in $\mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$.) Moreover, even if such a property holds it would only rule out deterministic reductions since NSETH only holds for deterministic algorithms.

Paper Organization. In Section 2, we explain our contributions in details, including the conclusions above and beyond. We discuss related works and future directions in Sections 3 and 4. An overview of our main NP^{dy}-completeness proof is in Section 5.

2 Our Contributions in Details

We show that coarse-grained complexity results similar to the static setting can be obtained for dynamic problems in the bit-probe/cell-probe model of computation,[3] provided the notion of "nondeterminism" is carefully defined. Recall that the cell-probe model is similar to the word RAM model, but the time complexity is measured by the number of memory reads and writes (probes); other operations are free. Like in the static setting, we only consider *decision* dynamic problems, meaning that the output after each update is either "yes" or "no". Note the following remarks.

- Readers who are familiar with the traditional complexity theory may wonder why we do not consider the Turing machine. This is because the Turing machine is not suitable for implementing dynamic al-

[3] Throughout the paper, we use the cell-probe and bit-probe models interchangeably since the complexity in these models are the same up to polylogarithmic factors.

gorithms, since we cannot access an arbitrary tape location in $O(1)$ time. There is no efficient algorithm even for a basic data structure like the binary search tree.

- Our results for decision problems extend naturally to *promised problems* which are useful when we discuss approximation algorithms. We do not discuss promised problems here to keep the discussions simple.

- Readers who are familiar with the oblivious adversaries assumption for randomized dynamic algorithms may wonder if we consider this assumption here. This assumption plays no role for decision problems, since an algorithm that is correct with high probability (w.h.p.) under this assumption is also correct w.h.p. without the assumption (in other words, its output reveals no information about its randomness). Because of this, we do not discuss this assumption in this paper.

We start with our main results which can be obtained with appropriate definitions of complexity classes $\mathsf{P}^{dy} \subseteq \mathsf{NP}^{dy} \subseteq \mathsf{PH}^{dy}$ for dynamic problems: These classes are described in details later. For now they should be thought of as analogues of the classes P, NP and PH (polynomial hierarchy).

THEOREM 2.1. *(P^{dy} vs. NP^{dy}) Below, the phrase "efficient algorithms" refers to dynamic algorithms that are deterministic and require polylogarithmic worst-case update time and polynomial space to handle a polynomial number of updates in the bit-probe/cell-probe model.*

1. *The dynamic orthogonal vector (OV) problem is "NP^{dy}-complete", and there are a number of dynamic problems that are "NP^{dy}-hard" in the sense that if $\mathsf{P}^{dy} \neq \mathsf{NP}^{dy}$, then they admit* no *efficient algorithms. These problems include decision versions of Subgraph connectivity, ST-reachability, Chan's subset union, and 3-vs-4 Diameter (see Tables 2, 3 for more).*

2. *If $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$ then $\mathsf{P}^{dy} = \mathsf{PH}^{dy}$, meaning that all problems in PH^{dy} (which contains the class NP^{dy}) admit efficient algorithms. These problems include decision versions of k-edge Connectivity, $(1+\epsilon)$-approximate Matching,*[4] *$(1+\epsilon)$-approximate mincut, Planar nearest neighbors, Chan's subset union and 3-vs-4 Diameter (see Tables 2, 4 for more).*

[4]Technically speaking, $(1+\epsilon)$-approximate matching is a *promised* or *gap* problem in the sense that for some input instance all answers are correct. It is in $\mathsf{promise}-\mathsf{PH}^{dy}$ which is bigger than PH^{dy}. We can make the same conclusion for promised problems: If $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$, then all problems in $\mathsf{promise}-\mathsf{PH}^{dy}$ admit efficient algorithms.

Thus, proving or disproving $\mathsf{P}^{dy} \neq \mathsf{NP}^{dy}$ will both lead to interesting consequences: If $\mathsf{P}^{dy} \neq \mathsf{NP}^{dy}$, then many dynamic problems do not admit efficient algorithms. Otherwise, if $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$, then many problems admit efficient algorithms which are not known or even believed to exist.

Remark: We can obtain similar results in the word-RAM model, but we need a notion of "efficient algorithms" that is slightly non-standard in that a quasi-polynomial preprocessing time is allowed. (In contrast, all our results hold in the standard cell-probe setting.) We postpone discussing word-RAM results to later in the paper to avoid confusions.

As another showcase, our study implies a way to show *non-reducibility*, like below.

THEOREM 2.2. *Assuming $\mathsf{PH}^{dy} \not\subseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$, the k-edge connectivity problem cannot be NP^{dy}-hard. Consequently, there is no "efficient reduction" from the dynamic Orthogonal Vector (OV) problem to k-edge connectivity.*

From the discussion in Section 1, recall that the k-edge connectivity problem is currently known to admit a polylogarithmic amortized update time algorithm for $k \leq 2$, and a $O(\sqrt{n}\,\mathrm{polylog}(n))$ update time algorithm for $k \in [3, O(\log n)]$. It is a very important open problem whether it admits polylogarithmic worst-case update time. Theorem 2.2 rules out a way to prove lower bounds and suggest that an efficient algorithm might exist.

A more important point beyond the k-edge connectivity problem is that one can prove a similar result for any dynamic problem X by showing that $X \in \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$ or, even better, $X \in \mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$. See Section 4 for some candidate problems for X. This is easier than showing a dynamic algorithm for X itself. Thus, this method is an example of the by-products of our study that we expect to be useful for developing algorithms and lower bounds for dynamic problems in future. See Section 2.4 for more details. As noted in Section 1, we are not aware of any existing technique that is capable of deriving a non-reducibility result of this kind.

The key challenge in deriving the above results is to come up with the right set of definitions for various dynamic complexity classes. We provide some of these definitions and discussions here, but defer more details to later in the paper.

2.1 Defining the Complexity Classes P^{dy} and NP^{dy}

Class P^{dy}. We start with P^{dy}, the class of dynamic problems that admit "efficient" algorithms in the cell-probe model. For any dynamic problem, define its *update size* to be the number of bits needed to describe each update. Note that we have not yet defined what dynamic problems are formally. Such a definition is needed for a proper, rigorous description of our complexity classes, and can be found in the full version of the paper. For an intuition, it suffices to keep in mind that most dynamic graph problems – where each update is an edge deletion or insertion – have logarithmic update size (since it takes $O(\log n)$ bits to specify an edge in an n-node graph).

Definition 2.1. (P^{dy}; brief) *A dynamic problem with polylogarithmic update size is in P^{dy} if it admits a deterministic algorithm with polylogarithmic worst-case update time for handling a sequence of polynomially many updates.*

Examples of problems in P^{dy} include connectivity on plane graphs and predecessor; for more, see Table 1. Note that one can define P^{dy} more generally to include problems with larger update sizes. Our complexity results hold even with this more general definition. However, since our results are most interesting for problems with polylogarithmic update size, we focus on this case in this paper to avoid cumbersome notations.

Class NP^{dy} and nondeterminism with rewards. Next, we introduce our complexity class NP^{dy}. Recall that in the static setting the class NP consists of the set of problems that admit efficiently verifiable proofs or, equivalently, that are solvable in polynomial time by a nondeterministic algorithm. Our notion of nondeterminism is captured by the proof-verification definition where, after receiving a proof, the verifier does not only output YES/NO, but also a *reward*, which is supposed to be maximized at every step.

Before defining NP^{dy} more precisely, we remark that the notion of reward is a key for our NP^{dy}-completeness proof. Having the constraint about rewards potentially makes NP^{dy} contains less problems. Interestingly, all natural problems that we are aware of remains in NP^{dy} even with this constraint. This might not be a big surprise, when it is realized that in the static setting imposing a similar constraint about the reward does *not* make the class (static) NP smaller; see more discussions below. We now define NP^{dy} more precisely.

Definition 2.2. (NP^{dy}; brief) *A dynamic problem Π with polylogarithmic update size is in NP^{dy} if there is a verifier that can do the following over a sequence of polynomially many updates: (i) after every update, the verifier takes the update and a polylogarithmic-size* proof *as an input, and (ii) after each update, the verifier outputs in polylogarithmic time a pair (x, y), where $x \in \{YES, NO\}$and y is an integer (representing a reward) with the following properties.*[5]

1. *If the current input instance is an YES-instance and the verifier has so far always received a proof that maximizes the reward at every step, then the verifier outputs $x = YES$.*
2. *If the current input instance is a NO-instance, then the verifier outputs $x = NO$ regardless of the sequence of proofs it has received so far.*

To digest the above definition, first consider the *static* setting. One can redefine the class NP for static problems in a similar fashion to Definition 2.2 by removing the preprocessing part and letting the only update be the whole input. Let us refer to this new (static) complexity class as "*reward-NP*". To show that a static problem is in reward-NP, a verifier has to output some reward in addition to the YES/NO answer. Since usually a proof received in the static setting is a solution itself, a natural choice for reward is the *cost* of the solution (i.e., the proof). For example, a "proof" in the maximum clique problem is a big enough clique, and in this case an intuitive reward would be the size of the clique given as a proof. Observe that this is sufficient to show that max clique is in reward-NP. In fact, it turns out that *in the static setting the complexity classes NP and reward-NP are equal.* (Proof sketch: Let Π be any problem in the original static NP and V be a corresponding verifier. We extend V to V' which outputs $x = YES/NO$ as V and outputs $y = 1$ as a reward if $x = YES$ and $y = 0$ otherwise. It is not hard to check that V' satisfies conditions in Definition 2.2.)

To further clarify Definition 2.2, we now consider examples of some well-known dynamic problems that happen to be in NP^{dy}.

Example 1. (Subgraph Detection) *In the dynamic subgraph detection problem, an n-node and k-node graphs G and H are given at the preprocessing, for some $k = \text{polylog}(n)$. Each update is an edge insertion or deletion in G. We want an algorithm to output YES if and only if G has H as a subgraph.*

This problem is in NP^{dy} due to the following verifier: the verifier outputs $x = YES$ if and only if the proof (given after each update) is a mapping of the edges in H to the edges in a subgraph of G that is isomorphic to H. With output $x = YES$, the verifier gives reward $y = 1$. With output $x = NO$, the verifier gives reward

[5] Later in the paper, we use $x = 1$ and $x = 0$ to represent $x = YES$ and $x = NO$, respectively.

$y = 0$. *Observe that the proof is of polylogarithmic size (since* $k = \text{polylog}(n)$*), and the verifier can calculate its outputs* (x, y) *in polylogarithmic time. Observe further that the properties stated in Definition 2.2 are satisfied: if the current input instance is a YES-instance, then the reward-maximizing proof is a mapping between* H *and the subgraph of* G *isomorphic to* H*, causing the verifier to output* $x = YES$*; otherwise, no proof will make the verifier output* $x = YES$*.*

The above example is in fact too simple to show the strength of our definition that allows NP^{dy} to include many natural problems (for one thing, y is simply 0/1 depending on x). The next example demonstrates how the definition allows us to develop more sophisticated verifiers for other problems.

Example 2. (Connectivity) *In the* dynamic connectivity *problem, an* n*-node graph* G *is given at the preprocessing. Each update is an edge insertion or deletion in* G*. We want an algorithm to output YES if and only if* G *is connected.*

This problem is in NP^{dy} *due to the following verifier. After every update, the verifier maintains a forest* F *of* G*. A proof (given after each update) is an edge insertion to* F *or an* $\perp$ *symbol indicating that there is no update to* F*. It handles each update as follows.*

- *After an edge* e *is inserted into* G*, the verifier checks if* e *can be inserted into* F *without creating a cycle. This can be done in* $O(\text{polylog}(n))$ *time using a link/cut tree data structure [65]. It outputs reward* $y = 0$*. (No proof is needed in this case.)*
- *After an edge* e *is deleted from* G*, the verifier checks if* F *contains* e*. If not, it outputs reward* $y = 0$ *(no proof is needed in this case). If* e *is in* F*, the verifier reads the proof (given after* e *is deleted). If the proof is* $\perp$ *it outputs reward* $y = 0$*. Otherwise, let the proof be an edge* e'*. The verifier checks if* $F' = F \setminus \{e\} \cup \{e'\}$ *is a forest; this can be done in* $O(\text{polylog}(n))$ *time using a link/cut tree data structure [65]. If* F' *is a forest, the verifier sets* $F \leftarrow F'$ *and outputs reward* $y = 1$*; otherwise, it outputs reward* $y = -1$*.*

After each update, the verifier outputs $x = YES$ *if and only if* F *is a spanning tree of* G*.*

Observe that if the verifier gets a proof that maximizes the reward after every update, the forest F *will always be a spanning forest (since inserting an edge* e' *to* F *has higher reward than giving* $\perp$ *as a proof). Thus, the verifier will always output* $x = YES$ *for YES-instances in this case. It is not hard to see that the verifier never outputs* $x = YES$ *for NO-instances, no matter what proof it receives.*

In short, a proof for the connectivity problem is the maximal spanning forest. Since such proof is too big to specify and verify after every update, our definition allows such proof to be updated over input changes. (This is as opposed to specifying the densest subgraph from scratch every time as in Example 1.) Allowing this is crucial for most problems to be in NP^{dy}, but create difficulties to prove NP^{dy}-completeness. We remedy this by introducing rewards.

Note that if there is no reward in Definition 2.2, then it is even easier to show that dynamic connectivity and other problems are in NP^{dy}. Having an additional constraint about rewards potentially makes less problems verifiable. Luckily, all natural problems that we are aware of that were verifiable without rewards remain verifiable with rewards. Problems in NP^{dy} include decision/gap versions of $(1+\epsilon)$-approximate matching, planar nearest neighbor, and dynamic 3SUM; see Table 2 for more. The concept of rewards (introduced while defining the class NP^{dy}) will turn to be crucial when we attempt to show the existence of a complete problem in NP^{dy}. See Section 2.2 and Section 5 for more details.

It is fairly easy to show that $\mathsf{P}^{dy} \subseteq \mathsf{NP}^{dy}$, and we conjecture that $\mathsf{P}^{dy} \neq \mathsf{NP}^{dy}$.

Previous nondeterminism in the dynamic setting. The idea of nondeterministic dynamic algorithms is not completely new. This was considered by Husfeldt and Rauhe [39] and their follow-ups [58, 56, 75, 45, 69], and has played a key role in proving cell-probe lower bounds in some of these papers. As discussed in [39], although it is straightforward to define a nondeterministic dynamic algorithm as the one that can make nondeterministic choices to process each update and query, there are different ways to handle how nondeterministic choices affect the states of algorithms which in turns affect how the algorithms handle future updates (called the "side effects" in [39]). For example, in [39] nondeterminism is allowed only for answering a query, which happens to occur only once at the very end. In [58], nondeterministic query answering may happen throughout, but an algorithm is allowed to write in the memory (thus change its state) only if all nondeterministic choices lead to the same memory state.

In this paper we define a different notion of nondeterminism and thus the class NP^{dy}. It is more general than the previous definitions in that if a dynamic problem admits an efficient nondeterministic algorithm according to the previous definitions, it is in our NP^{dy}. In a nutshell, the key differences are that (i) we allow nondeterministic steps while processing both updates and queries and (ii) different choices of nondeterminism can affect the algorithm's states in different ways;

however, we distinct different choices by giving them different rewards. These differences allow us to include more problems to our NP^{dy} (we do not know, for example, if dynamic connectivity admits nondeterministic algorithms according to previous definitions).

2.2 NP^{dy}-Completeness Here, we sketch the idea behind our NP^{dy}-completeness and hardness results. We begin by introducing a problem is called *dynamic narrow DNF evaluation problem* (in short, DNF^{dy}), as follows.

Definition 2.3. (DNF^{dy}; informally) *Initially, we have to preprocess (i) an m-clause n-variable DNF formula*[6] *where each clause contains $O(\text{polylog}(m))$ literals, and (ii) an assignment of (boolean) values to the variables. Each update changes the value of one variable. After each update, we have to answer whether the DNF formula is true or false.*

It is fairly easy to see that $\mathrm{DNF}^{dy} \in \mathsf{NP}^{dy}$: After each update, if the DNF formula happens to be true, then the proof only needs to point towards one satisfied clause, and the verifier can quickly check if this clause is satisfied or not since it contains only $O(\text{polylog}(m))$ literals. Surprisingly, it turns out that this is also a complete problem in the classs NP^{dy}.

Theorem 2.3. (NP^{dy}-completeness of DNF^{dy}) *The DNF^{dy} problem is NP^{dy}-complete. This means that $\mathrm{DNF}^{dy} \in \mathsf{NP}^{dy}$, and if $\mathrm{DNF}^{dy} \in \mathsf{P}^{dy}$, then $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$.*

To start with, recall the following intuition for proving NP-completeness in the static setting (e.g. [6, Section 6.1.2] for details): Since *Boolean circuits* can simulate polynomial-time Turing machine computation (i.e. $P \subseteq P/\mathsf{poly}$), we view the computation of the verifier V for any problem Π in NP as a circuit C. The input of C is the proof that V takes as an input. Then, determining whether there is an input (proof) that satisfies this circuit (known as *CircuitSAT*) is NP-complete, since such information will allow the verifier to find a desired proof on its own. Attempting to extend this intuition to the dynamic setting might encounter the following roadblocks.

1. Boolean circuits cannot efficiently simulate algorithms in the RAM model without losing a linear factor in running time.Furthermore, an alternative such as circuits with "indirect addressing" gates seems useless, because this complex gate makes the model more complicated. This makes it more difficult to prove NP^{dy}-hardness.

[6]Recall that a DNF formula is in the form $C_1 \vee \cdots \vee C_m$, where each "clause" C_i is a conjunction (AND) of literals.

2. Since the verifier has to work through several updates in the dynamic setting, the YES/NO output from the verifier alone is insufficient to indicate proofs that can be useful for *future* updates. For example, suppose that in Example 2 the connectivity verifier is allowed to output only $x \in \{YES, NO\}$, and we get rid of the concept of a reward. Consider a scenario where an edge e (which is part of F) gets deleted from G, and G was disconnected even before this deletion. In this case, the verifier can indicate no difference between having e' (i.e. finding a reconnecting edge) and $\perp$ (i.e. doing nothing) as a proof (because it has to output $x = 0$ in both cases). Having e' as a proof, however, is more useful for the future, since it helps maintain a spanning forest.

It so happens that we can solve (ii) if the verifier additionally outputs an integer y as a reward. Asking more from the verifier makes less problems verifiable (thus smaller NP^{dy}). Luckily, all natural problems we are aware of that were verifiable without rewards remain verifiable with rewards!

To solve (i), we use the fact that in the cell-probe model a polylogarithmic-update-time algorithm can be modeled by a polylogarithmic-depth *decision assignment tree* [49], which naturally leads to a complete problem about a decision tree (we leave details here; see Section 5 for more). It turns out that we can reduce from this problem to DNF^{dy} (Definition 2.3); the intuition being that each bit in the main memory corresponds to a boolean variable and each root-to-leaf path in the decision assignment tree can be thought of as a DNF clause. The only downside of this approach is that a polylogarithmic-depth decision tree has quasi-polynomial size. A straightforward reduction would cause quasi-polynomial space in the cell-probe model. By exploiting the special property of DNF^{dy} and the fact that the cell-probe model only counts the memory access, we can avoid this space blowup by "hardwiring" some space usage into the decision tree and reconstruct some memory when needed.

The fact that the DNF^{dy} problem is NP^{dy}-complete (almost) immediately implies that many well-known dynamic problems are NP^{dy}-hard. To explain why this is the case, we first recall the definition of the *dynamic sparse orthogonal vector* (OV^{dy}) problem.

Definition 2.4. (OV^{dy}) *Initially, we have to preprocess a collection of m vectors $V = \{v_1, \ldots, v_m\}$ where each $v_j \in \{0,1\}^n$, and another vector $u \in \{0,1\}^n$. It is guaranteed that each $v_j \in \{0,1\}^n$ has at most $O(\text{polylog}(m))$ many nonzero entries. Each update flips the value of one entry in the vector u. After each up-*

date, we have to answer if there is a vector $v \in V$ that is orthogonal to u (i.e., if $u^T v = 0$).

The key observation is that the OV^{dy} problem is equivalent to the DNF^{dy} problem, in the sense that OV^{dy} $\in$ P^{dy} iff DNF^{dy} $\in$ P^{dy}. The proof is relatively straightforward (the vectors v_j and the individual entries of u respectively correspond to the clauses and the variables in DNF^{dy}), and we defer it to the full version of the paper. In [3], Abboud and Williams show SETH-hardness for all of the problems in Table 3. In fact, they actually show a reduction from OV^{dy} to these problems. Therefore, we immediately obtain the following result.

COROLLARY 2.1. *All problems in Table 3 are NP^{dy}-hard.*

2.3 Dynamic Polynomial Hierarchy By introducing the notion of *oracles*, it is not hard to extend the class NP^{dy} into *polynomial-hierarchy* for dynamic problems, denoted by PH^{dy}. Roughly, PH^{dy} is the union of classes Σ_i^{dy} and Π_i^{dy}, where (i) $\Sigma_1^{dy} = \mathsf{NP}^{dy}$, $\Pi_1^{dy} = \mathsf{coNP}^{dy}$, and (ii) we say that a dynamic problem is in class Σ_i^{dy} (resp. Π_i^{dy}) if we can show that it is in NP^{dy} (resp. coNP^{dy}) assuming that there are efficient dynamic algorithms for problems in Σ_{i-1}. The details appear in the full version of the paper.

EXAMPLE 3. (k- AND ($<k$)-EDGE CONNECTIVITY) *In the* dynamic k-edge connectivity *problem, an n-node graph $G = (V, E)$ and a parameter $k = O(\text{polylog}(n))$ is given at the time of preprocessing. Each update is an edge insertion or deletion in G. We want an algorithm to output YES if and only if G has connectivity* at least *k, i.e. removing at most $k-1$ edges will* not *disconnect G. We claim that this problem is in Π_2^{dy}. To avoid dealing with coNP^{dy}, we consider the complement of this problem called* dynamic (¡k)-edge connectivity, *where $x = YES$ if and only if G has connectivity less than k. We show that (¡k)-edge connectivity is in Σ_2^{dy}.*

We already argued in Example 2 that dynamic connectivity is in $\mathsf{NP}^{dy} = \Sigma_1^{dy}$. Assuming that there exists an efficient (i.e. polylogarithmic-update-time) algorithm $\mathcal{A}$ for dynamic connectivity, we will show that (¡k)-edge connectivity is in NP^{dy}. Consider the following verifier $\mathcal{V}$. After every update in G, the verifier $\mathcal{V}$ reads a proof that is supposed to be a set $S \subseteq E$ of at most $k-1$ edges. $\mathcal{V}$ then sends the update to $\mathcal{A}$ and also tells $\mathcal{A}$ to delete the edges in S from G. If $\mathcal{A}$ says that G is not connected at this point, then the verifier $\mathcal{V}$ outputs $x = YES$ with reward $y = 1$; otherwise, the verifier $\mathcal{V}$ outputs $x = NO$ with reward $y = 0$. Finally, $\mathcal{V}$ tells $\mathcal{A}$ to add the edges in S back in G.

Observe that if G has connectivity less than k and the verifier always receives a proof that maximizes the reward, then the proof will be a set of edges disconnecting the graph and $\mathcal{V}$ will answer YES. Otherwise, no proof can make $\mathcal{V}$ answer YES. Thus the dynamic (¡k)-edge connectivity problem is in NP^{dy} if $\mathcal{A}$ exists. In other words, the problem is in Σ_2^{dy}.

By arguments similar to the above example, we can show that other problems such as Chan's subset union and small diameter are in PH^{dy}; see Table 4 for more.

The theorem that plays an important role in our main conclusion (Theorem 2.1) is the following.

THEOREM 2.4. *If $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$, then $\mathsf{PH}^{dy} = \mathsf{P}^{dy}$.*

To get an idea how to proof the above theorem, observe that if $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$, then $\mathcal{A}$ in Example 3 exists and thus dynamic (¡k)-edge connectivity are in $\Sigma_1^{dy} = \mathsf{NP}^{dy}$ by the argument in Example 2; consequently, it is in P^{dy}! This type of argument can be extended to all other problems in PH^{dy}.

2.4 Other Results and Remarks In previous subsections, we have stated two complexity results, namely NP^{dy}-completeness/hardness and the collapse of PH^{dy} when $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$. With right definitions in place, it is not a surprise that more can be proved. For example, we obtain the following results:

1. If $\mathsf{NP}^{dy} \subseteq \mathsf{coNP}^{dy}$, then $\mathsf{PH}^{dy} = \mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$.
2. If $\mathsf{NP}^{dy} \subseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$, then $\mathsf{PH}^{dy} \subseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$.

Here, coNP^{dy}, AM^{dy}, and coAM^{dy} are analogues of complexity classes coNP, AM, and coAM. The details appear in the full version of the paper.

While the coarse-grained complexity results in this paper are mostly resource-centric (in contrast to fine-grained complexity results that are usually centered around problems), we also show that this approach is helpful for understanding the complexity of specific problems as well, in the form of *non-reducibility*. In particular, the following results are shown in the full version of the paper:

1. Assuming $\mathsf{PH}^{dy} \neq \mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$, the two statements *cannot* hold at the same time.
   (a) Connectivity is in coNP^{dy}. (This would be the case if it is in P^{dy}.)

(b) One of the following problems is NP^{dy}-hard: approximate minimum spanning forest (MSF), d-weight MSF,bipartiteness, and k-edge connectivity.

2. k-edge connectivity is in $\mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$. Consequently, assuming $\mathsf{PH}^{dy} \not\subseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$, k-edge connectivity cannot be NP^{dy}-hard.

Note that both $\mathsf{PH} \neq \mathsf{NP} \cap \mathsf{coNP}$ and $\mathsf{PH} \not\subseteq \mathsf{AM} \cap \mathsf{coAM}$ are widely believed in the static setting since refuting them means collapsing PH. While we can show that PH^{dy} would also collapse if $\mathsf{PH}^{dy} = \mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$, it remains open whether this is the case for $\mathsf{PH}^{dy} \subseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$; in particular is $\mathsf{PH}^{dy} \supseteq \mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$?

When a problem Y cannot be NP^{dy}-hard, there is no efficient reduction from an NP^{dy}-hard problem X to Y, where an efficient reduction is roughly to a way to handle each update for problem X by making polylogarithmic number of updates to an algorithm for Y (such reduction would make Y an NP^{dy}-hard problem). Consequently, this rules out efficient reductions from dynamic OV, since it is NP^{dy}-complete. As a result, this rules out a common way to prove lower bounds based on SETH, since previously this was mostly done via reductions from dynamic OV [3]. (A lower bound for dynamic diameter is among a very few exception [3].)

2.5 Relationship to Fine-Grained Complexity

As noted earlier, it turns out that the dynamic OV problem is NP^{dy}-complete. Since most previous reductions from SETH to dynamic problems (in the word RAM model) are in fact reductions from dynamic OV [3], and since any reduction in the word RAM model applies also in the (stronger) cell-probe model, we get many NP^{dy}-hardness results for free. In contrast, our results above imply that the following two statements are equivalent: (i) "problem Π cannot be NP^{dy}-hard" and (ii) "there is no *efficient* reduction from dynamic OV to Π", where "efficient reductions" are reduction that only polynomially blow up the instance size (all reductions in [3] are efficient). In other words, we may not expect reductions from SETH that are similar to the previous ones for k-edge connectivity, bipartiteness, etc.

Finally, we emphasize that the coarse-grained approach should be viewed as a complement of the fine-grained approach, as the above results exemplify. We do not expect to replace results from one approach by those from another.

2.6 Complexity classes for dynamic problems in the word RAM model As an aside, we managed to define complexity classes and completeness results for dynamic problems in the word RAM model as well. We refer to P^{dy} and NP^{dy} as $\mathsf{RAM}-\mathsf{P}^{dy}$ and $\mathsf{RAM}-\mathsf{NP}^{dy}$ in the word-RAM model. One caveat is that for technical reasons we need to allow for quasipolynomial preprocessing time and space while defining the complexity classes $\mathsf{RAM}-\mathsf{P}^{dy}$ and $\mathsf{RAM}-\mathsf{NP}^{dy}$. We discuss this in more details in the full version of the paper.

3 Related Work

There are several previous attempts to classify dynamic problems. First, there is a line of works called "dynamic complexity theory" (see e.g. [24, 70, 63]) where the general question asks whether a dynamic problem is in the class called DynFO. Roughly speaking, a problem is in DynFO if it admits a dynamic algorithm expressible by a first-order logic. This means, in particular, that given an update, such algorithm runs in $O(1)$ parallel time, but might take arbitrary poly(n) works when the input size is n. A notion of reduction is defined and complete problems of DynFO and related classes are proven in [36, 70]. However, as the total work of algorithms from this field can be large (or even larger than computing from scratch using sequential algorithms), they do not give fast dynamic algorithms in our sequential setting. Therefore, this setting is somewhat irrelevant to our setting.

Second, a problem called the *circuit evaluation* problem has been shown to be complete in the following sense. First, it is in P (the class of static problems). Second, if the dynamic version of circuit evaluation problem, which is defined as DNF^{dy} where a DNF-formula is replaced with an arbitrary circuit, admits a dynamic algorithm with polylogarithmic update time, then for any static problem $L \in \mathsf{P}$, a dynamic version of L also admits a dynamic algorithm with polylogarithmic update time. This idea is first sketched informally since 1987 by Reif [60]. Miltersen et al. [50] then formalized this idea and showed that other P-complete problems listed in [51, 32] also are complete in the above sense.[7] The drawback about this completeness result is that the dynamic circuit evaluation problem is extremely difficult. Similar to the case for static problems that reductions from EXP-complete problems to problems in NP are unlikely, reductions from the dynamic circuit evaluation problem to other natural dynamic problems studied in the field seem unlikely. Hence, this does not give a framework for proving hardness for other dynamic problems.

Our result can be viewed as a more fine-grained completeness result than the above. As we show that a very special case of the dynamic circuit evaluation

[7] But they also show that this is not true for all P-complete problems.

problem which is DNF^{dy} is already a complete problem. An important point is that DNF^{dy} is simple enough that reductions to other natural dynamic problems are possible.

Finally, Ramalingam and Reps [59] classify dynamic problems according to some measure,[8] but did not give any reduction and completeness result.

4 Future Directions

One byproduct of our paper is a way to prove non-reducibility. It is interesting to use this method to shed more light on the hardness of other dynamic problems. To do so, it suffices to show that such problem is in $\mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$ (or, even better, in $\mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$). One particular problem is whether connectivity is in $\mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$. It is known to be in $\mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$ due to the randomized algorithm of Kapron et al [42]. It is also in NP^{dy} (see Example 2). The main question is whether it is in coNP^{dy}. (Techniques from [53, 73, 52] almost give this, with verification time $n^{o(1)}$ instead of polylogarithmic.) Having connectivity in $\mathsf{NP}^{dy} \cap \mathsf{coNP}^{dy}$ would be a strong evidence that it is in P^{dy}, meaning that it admits a deterministic algorithm with polylogarithmic update time. Achieving such algorithm will be a major breakthrough. Another specific question is whether the promised version of the $(2-\epsilon)$ approximate matching problem is in $\mathsf{AM}^{dy} \cap \mathsf{coAM}^{dy}$. This would rule out efficient reductions from dynamic OV to this problem. Whether this problem admits a randomized algorithm with polylogarithmic update time is a major open problem. Other problems that can be studied in this direction include approximate minimum spanning forest (MSF), d-weight MSF,bipartiteness, dynamic set cover, dominating set, and st-cut.

It is also very interesting to rule out efficient reductions from the following variant of the OuMv conjecture: At the preprocessing, we are given a boolean $n \times n$ matrix M and boolean n-dimensional row and column vectors u and v. Each update changes one entry in either u or v. We then have to output the value of uMv. Most lower bounds that are hard under the OMv conjecture [34] are via efficient reductions from this problem. It is interesting to rule out such efficient reductions since SETH and OMv are two conjectures that imply most lower bounds for dynamic problems.

Now that we can prove completeness and relate some basic complexity classes of dynamic problems, one big direction to explore is whether more results from coarse-grained complexity for static problems can be reconstructed for dynamic problems. Below are a few samples.

1. *Derandomization:* Making dynamic algorithms deterministic is an important issue. Derandomization efforts have so far focused on specific problems (e.g. [52, 53, 10, 11, 9, 13, 12, 14]). Studying this issue via the class BPP^{dy} might lead us to the more general understanding. For example, the *Sipser-Lautermann theorem* [64, 48] states that $BPP \subseteq \Sigma_2 \cap \Pi_2$, Yao [74] showed that the existence of some pseudorandom generators would imply that $P = BPP$, and Impagliazzo and Wigderson [40] suggested that $BPP = P$ (assuming that any problem in $E = DTIME(2^{O(n)})$ has circuit complexity $2^{\Omega(n)}$). We do not know anything similar to these for dynamic problems.

2. *NP-Intermediate:* Many static problems (e.g. graph isomorphism and factoring) are considered good candidates for being NP-intermediate, i.e. being neither in P nor NP-complete. This paper leaves many natural problems in NP^{dy} unproven to be NP^{dy}-complete. Are these problems in fact NP^{dy}-intermediate? The first step towards this question might be proving an analogues of *Ladner's theorem* [44], i.e. that an NP^{dy}-intermediate dynamic problem exists, assuming $\mathsf{P}^{dy} \neq \mathsf{NP}^{dy}$. It is also interesting to prove the analogue of the *time-hierarchy theorems*, i.e. that with more time, more dynamic problems can be solved. (Both theorems are proved by diagonalization in the static setting.)

3. This work and lower bounds from fine-grained complexity has focused mostly on *decision* problems. There are also *search dynamic problems*, which always have valid solutions, and the challenge is how to maintain them. These problems include maximal matching, maximal independent set, minimal dominating set, coloring vertices with $(\Delta + 1)$ or more colors, and coloring edges with $(1 + \epsilon)\Delta$ or more colors, where Δ is the maximum degree (e.g. [8, 12, 7, 66, 25, 33, 54]). These problems do not seem to correspond to any decision problems. Can we define complexity classes for these problems and argue that some of them might not admit polylogarithmic update time? Analogues of TFNP and its subclasses (e.g. PPAD) might be helpful here.

There are also other concepts that have not been discussed in this paper at all, such as interactive proofs, probabilistically checkable proofs (PCP), counting problems (e.g. Toda's theorem), relativization and other barriers. Finally, in this paper we did not discuss amortized update time. It is a major open problem whether

[8] They measure the complexity dynamic algorithms by comparing the update time with the size of *change in input and output* instead of the size of input itself.

similar results, especially an analogue of NP-hardness, can be proved for algorithms with amortized update time.

5 An Overview of the NP^{dy}-Completeness Proof

In this section, we present an overview of one of our main technical contributions (the proof of Theorem 2.3) at a finer level of granularity. In order to explain the main technical insights we focus on a nonuniform model of computation called the *bit-probe* model, which has been studied since the 1970's [30, 49].

5.1 Dynamic Complexity Classes P^{dy} and NP^{dy}

We begin by reviewing (informally) the concepts of a dynamic problem and an algorithm in the bit-probe model. Consider any dynamic problem $\mathcal{D}_n$. Here, the subscript n serves as a reminder that the bit-probe model is nonuniform and it also indicates that each instance I of this problem can be specified using n bits. We will will mostly be concerned with dynamic *decision* problems, where the *answer* $\mathcal{D}_n(I) \in \{0, 1\}$ to every instance I can be specified using a single bit. We say that I is an YES instance if $\mathcal{D}_n(I) = 1$, and a NO instance if $\mathcal{D}_n(I) = 0$. An algorithm $\mathcal{A}_n$ for this dynamic problem $\mathcal{D}_n$ has access to a memory $\texttt{mem}_n$, and the total number of bits available in this memory is called the *space complexity* of $\mathcal{A}_n$. The algorithm $\mathcal{A}_n$ works in steps $t = 0, 1, \ldots,$ in the following manner.

Preprocessing: At step $t = 0$ (also called the preprocessing step), the algorithm gets a *starting instance* $I_0 \in \mathcal{D}_n$ as input. Upon receiving this input, it initializes the bits in its memory $\texttt{mem}_n$ and then it *outputs* the *answer* $\mathcal{D}_n(I_0)$ to the *current instance* I_0.

Updates: Subsequently, at each step $t \geq 1$, the algorithm gets an *instance-update* (I_{t-1}, I_t) as input. The sole purpose of this instance-update is to change the current instance from I_{t-1} to I_t. Upon receiving this input, the algorithm probes (reads/writes) some bits in the memory $\texttt{mem}_n$, and then outputs the answer $\mathcal{D}_n(I_t)$ to the current instance $I_t \in \mathcal{D}_n$. The *update time* of $\mathcal{A}_n$ is the maximum number of bit-probes it needs to make in $\texttt{mem}_n$ while handling an instance-update.

One way to visualize the above description as follows. An adversary keeps constructing an *instance-sequence* $(I_0, I_1, \ldots, I_k, \ldots)$ one step at a time. At each step t, the algorithm $\mathcal{A}_n$ gets the corresponding instance-update (I_{t-1}, I_t), and at this point it is only aware of the prefix $(I_0, \ldots, I_t)$. Specifically, the algorithm does not know the future instance-updates. After receiving the instance-update at each step t, the algorithm has to output the answer to the current instance $\mathcal{D}_n(I_t)$. This framework is flexible enough to capture dynamic problems that allow for both *update* and *query* operations, because we can easily model a query operation as an instance-update. Furthermore, w.l.o.g. we assume that an instance-update in a dynamic problem $\mathcal{D}_n$ can be specified using $O(\log n)$ bits.

For technical reasons, we will work under the following assumption. This assumption will be implicitly present in the definitions of the complexity classes P^{dy} and NP^{dy} below.

Assumption 1. *A dynamic algorithm $\mathcal{A}_n$ for a dynamic problem $\mathcal{D}_n$ has to handle at most* $\text{poly}(n)$ *many instance-updates.*

We now define the complexity class P^{dy}.

Definition 5.1. (Class P^{dy}) *A dynamic decision problem $\mathcal{D}_n$ is in P^{dy} iff there is an algorithm $\mathcal{A}_n$ solving $\mathcal{D}_n$ which has update time $O(\text{polylog}(n))$ and space-complexity $O(\text{poly}(n))$.*

In order to define the class NP^{dy}, we first introduce the notion of a *verifier* in Definition 5.2. Subsequently, we introduce the class NP^{dy} in Definition 5.3. We have already discussed the intuitions behind these concepts in Section 1 after the statement of Definition 2.2.

Definition 5.2. (Dynamic verifier) *We say that a dynamic algorithm $\mathcal{V}_n$ with space-complexity $O(\text{poly}(n))$ is a* verifier *for a dynamic decision problem $\mathcal{D}_n$ iff it works as follows.*

Preprocessing: *At step $t = 0$, the algorithm $\mathcal{V}_n$ gets a* starting instance *$I_0 \in \mathcal{D}_n$ as input, and it outputs an ordered pair (x_0, y_0) where $x_0 \in \{0, 1\}$ and $y_0 \in \{0, 1\}^{\text{polylog}(n)}$.*

Updates: *Subsequently, at each step $t \geq 1$, the algorithm $\mathcal{V}_n$ gets an* instance-update *(I_t, I_{t-1}) and a* proof *$\pi_t \in \{0, 1\}^{\text{polylog}(n)}$ as input, and it outputs an ordered pair (x_t, y_t) where $x_t \in \{0, 1\}$ and $y_t \in \{0, 1\}^{\text{polylog}(n)}$. The algorithm $\mathcal{V}_n$ has $O(\text{polylog}(n))$ update time, i.e., it makes at most $O(\text{polylog}(n))$ bit-probes in the memory during each step t. Note that the output (x_t, y_t) depends on the instance-sequence $(I_0, \ldots, I_t)$ and the* proof-sequence *$(\pi_1, \ldots, \pi_t)$ seen so far.*

Definition 5.3. (Class NP^{dy}) *A decision problem $\mathcal{D}_n$ is in NP^{dy} iff it admits a verifier $\mathcal{V}_n$ which satisfy the following properties. Fix any instance-sequence $(I_0, \ldots, I_k)$. Suppose that the verifier $\mathcal{V}_n$ gets I_0 as input at step $t = 0$ and the ordered pair $((I_{t-1}, I_t), \pi_t)$ as input at every step $t \geq 1$. Then:*

1. *For every proof-sequence $(\pi_1, \ldots, \pi_k)$, we have $x_t = 0$ for each $t \in \{0, \ldots, k\}$ where $\mathcal{D}_n(I_t) = 0$.*

Dynamic Problems	**Preprocess**	**Update**	**Queries**	**Ref.**
Numbers				
Sum/max	a set S of numbers	insert/delete a number in S	return $\sum_{x \in S} x$ or $\max_{x \in S} x$	Folklore
Predecessor			given x, return the maximum $y \in S$ where $y \leq x$.	
Geometry				
2-dimensional range counting	a set S of points on a plane	insert/delete a point in S	given $[x_1, x_2] \times [y_1, y_2]$, return $\lvert S \cap ([x_1, x_2] \times [y_1, y_2]) \rvert$	[55, after Theorem 7.6.3]
Incremental planar nearest neighbor		insert a point to S	given a point q, return $p \in S$ which is closest to q	[55, Theorem 7.3.4.1]
Vertical ray shooting	a set S of segments on a plane	insert/delete a segment in S	given a point q, return the segment immediately above q	[18, Theorem 3.7]
Graphs				
Dynamic problems on forests	a forest F	insert/delete an edge in F s.t. F remains a forest	given two nodes u and v, decide if u and v are connected in F	[65, 35, 5]
			given a node u, return the size of the tree containing u	
		many more kinds of updates	many more kinds of query	
Connectivity on plane graphs	a plane graph G (i.e. a planar graph on a fixed embedding)	insert/delete an edge in G such that G has no crossing on the embedding	given two nodes u and v, decide if u and v are connected in G	[28, 27]
2-edge connectivity on plane graphs			given two nodes u and v, decide if u and v are 2-edge connected in G	[29]
$(2+\epsilon)$-approx. size of maximum matching	a general graph G	insert/delete an edge in G	decide whether the size of maximum matching is at most k or at least $(2+\epsilon)k$ for some k and constant $\epsilon > 0$	[15]

Table 1: Problems in P^{dy}. Some problems are strictly promise problems, but our class can be extended easily to include them.

Dynamic Problems	**Preprocess**	**Update**	**Queries**
Connectivity	an undirected unweighted graph G	insert/delete an edge in G	given two nodes u and v, decide if u and v are connected in G
$(1+\epsilon)$-approx. size of maximum matching	an undirected unweighted graph G	insert/delete an edge in G	decide whether the size of maximum matching is at most k or at least $(1+\epsilon)k$ for some k and constant $\epsilon > 0$
Subgraph detection	a graph G and H where $\lvert V(H)\rvert = \text{polylog}(\lvert V(G)\rvert)$	insert/delete an edge in G	decide whether H is a subgraph of G
uMv (entry update)	$u, v \in \{0,1\}^n$ and $M \in \{0,1\}^{n\times n}$	update an entry of u or v	decide whether $u^T Mv = 1$ (multiplication over Boolean semi-ring).
3SUM	a set S of numbers	insert/delete a number in S	decide whether there is $a, b, c \in S$ where $a + b = c$
Planar nearest neighbor	a set S of points on a plane	insert a point to S	given a point q, return $p \in S$ which is closest to q
Erikson's problem [57]	a matrix M	choose a row or a column and increment all number of such row or column	given k, is the maximum entry in M at least k?
Langerman's problem [57]	an array A	given (i, x), set $A[i] = x$	is there a k such that $\sum_{i=1}^{k} A[i] = 0$?

Table 2: Problems in NP^{dy} that are not known to be in P^{dy}. Some problems are strictly promise problems, but our class can be extended easily to include them.

Dynamic Problems	**Preprocess**	**Update**	**Queries**
Pagh's problem with emptiness query [3]	A collection X of sets $X_1, \ldots, X_k \subseteq [n]$	given i, j, insert $X_i \cap X_j$ into X	given i, is $X_i = \emptyset$?
Chan's subset union problem [3]	A collection of sets $X_1, \ldots, X_n \subseteq [m]$. A set $S \subseteq [n]$.	insert/deletion an element in S	is $\cup_{i\in S} X_i = [m]$?
Single source reachability Count (#s-reach)	a directed graph G and a node s	insert/delete an edge	count the nodes reachable from s.
2 Strong components (SC2)	a directed graph G	insert/delete an edge	are there more than 2 strongly connected components?
st-max-flow	a capacitated directed graph G and nodes s and t	insert/delete an edge	the size of s-t max flow.
Subgraph global connectivity	a fixed undirected graph G	turn on/off a node	is a graph induced by turned on nodes connected?
3 vs. 4 diameter	an undirected graph G	insert/delete an edge	is a diameter of G 3 or 4?
ST-reachability	a directed graph G and sets of node S and T	insert/delete an edge	is there $s \in S$ and $t \in T$ where s can reach t?

Table 3: Problems that are NP^{dy}-hard.

Dynamic Problems	Preprocess	Update	Queries
Small dominating set	a graph G	insert/delete an edge	Is there a dominating set of size at most k?
Small vertex cover	a graph G	insert/delete an edge	Is there a vertex cover of size at most k?
Small maximal independent set	a graph G	insert/delete an edge	Is there a maximal independent set of size at most k?
Small maximal matching	a graph G	insert/delete an edge	Is there a maximal matching of size at most k?
Chan's Subset Union Problem	a collection of sets $X_1, \dots, X_n$ from universe $[m]$, and a set $S \subseteq [n]$	insert/delete an index in S	is $\cup_{i \in S} X_i = [m]$?
3 vs. 4 diameter	a graph G	insert/delete an edge	Is the diameter of G 3 or 4?
Euclidean k-center	a point set $X \subseteq \mathbb{R}^d$ and a threshold $T \in \mathbb{R}$	insert/delete a point	Is there a set $C \subseteq X$ where $\lvert C \rvert \leq k$ and $\max_{u \in X} \min_{v \in C} d(u, v) \leq T$
k-edge connectivity	a graph G	insert/delete an edge	Is G k-edge connected?

Table 4: Problems in PH^{dy} that are not known to be in NP^{dy}. The parameter k in every problem must be at most polylog(n) where n is the size of the instance.

2. *If the proof-sequence* $(\pi_1, \dots, \pi_k)$ *is* reward-maximizing *(defined below), then we have* $x_t = 1$ *for each* $t \in \{0, \dots, k\}$ *with* $\mathcal{D}_n(I_t) = 1$*,*

The proof-sequence $(\pi_1, \dots, \pi_k)$ *is* reward-maximizing *iff the following holds. At each step* $t \geq 1$*, given the past history* $(I_0, \dots, I_t)$ *and* $(\pi_1, \dots, \pi_{t-1})$*, the proof* π_t *is chosen in such a way that maximizes the value of* y_t*. We say that such a proof* π_t *is* reward-maximizing*.*

Just as in the static setting, we can easily prove that $\mathsf{P}^{dy} \subseteq \mathsf{NP}^{dy}$ and we conjecture that $\mathsf{P}^{dy} \neq \mathsf{NP}^{dy}$. The big question left open in this paper is to resolve this conjecture.

Corollary 5.1. *We have* $\mathsf{P}^{dy} \subseteq \mathsf{NP}^{dy}$*.*

5.2 A complete problem in NP^{dy} One of the main results in this paper shows that a natural problem called *dynamic narrow DNF evaluation* (denoted by DNF^{dy}) is NP^{dy}-complete. Intuitively, this means that (a) DNF$^{dy} \in \mathsf{NP}^{dy}$, and (b) if DNF$^{dy} \in \mathsf{P}^{dy}$ then $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$.[9] We now give an informal description of this problem.

Dynamic narrow DNF evaluation (DNF^{dy}): An instance I of this problem consists of a triple $(\mathcal{Z}, \mathcal{C}, \phi)$, where $\mathcal{Z} = \{z_1, \dots, z_N\}$ is a set of N variables, $\mathcal{C} = \{C_1, \dots, C_M\}$ is a set of M DNF clauses, and $\phi : \mathcal{Z} \to \{0, 1\}$ is an assignment of values to the variables. Each clause C_j is a conjunction (AND) of at most polylog(N) literals, where each literal is of the form z_i or $\neg z_i$ for some variable $z_i \in \mathcal{Z}$. This is an YES instance if at least one clause $C \in \mathcal{C}$ is true under the assignment ϕ, and this is a NO instance if every clause in $\mathcal{C}$ is false under the assignment ϕ. Finally, an *instance-update* changes the assignment ϕ by flipping the value of exactly one variable in $\mathcal{Z}$.

It is easy to see that the above problem is in NP^{dy}. Specifically, if the current instance is an YES instance, then a proof π_t simply points to a specific clause $C_j \in \mathcal{C}$ that is true under the current assignment ϕ. The proof π_t can be encoded using $O(\log M)$ bits. Furthermore, since each clause contains at most polylog(N) literals, the verifier can check that the clause C_j specified by the proof π_t is true under the assignment ϕ in $O(\text{polylog}(N))$ time. On the other hand, no proof can fool the verifier if the current instance is a NO instance (where every clause is false). All these observations can be formalized in a manner consistent with Definition 5.3. We will prove the following theorem.

Theorem 5.1. *The DNF^{dy} problem described above is* NP^{dy}*-complete.*

In order to prove Theorem 5.1, we consider an intermediate dynamic problem called First-DNF^{dy}.

First-DNF^{dy}: An instance I of First-DNF^{dy} consists of a tuple $(\mathcal{Z}, \mathcal{C}, \phi, \prec)$. Here, the symbols $\mathcal{Z}, \mathcal{C}$ and

[9] To be more precise, condition (b) means that every problem in P^{dy} is P^{dy}-reducible to DNF^{dy}.

ϕ denote exactly the same objects as in the DNF^{dy} problem described above. In addition, the symbol $\prec$ denotes a total order on the set of clauses $\mathcal{C}$. The answer to this instance I is defined as follows. If every clause in $\mathcal{C}$ is false under the current assignment ϕ, then the answer to I is 0. Otherwise, the answer to I is the *first clause* $C_j \in \mathcal{C}$ according to the total order $\prec$ that is true under ϕ. It follows that First-DNF^{dy} is *not* a decision problem. Finally, as before, an instance-update for the First-DNF^{dy} changes the assignment ϕ by flipping the value of exactly one variable in $\mathcal{Z}$.

We prove Theorem 5.1 as follows. (1) We first show that First-DNF^{dy} is NP^{dy}-hard. Specifically, if there is an algorithm for First-DNF^{dy} with polylog update time and polynomial space complexity, then $\mathsf{P}^{dy} = \mathsf{NP}^{dy}$. We explain this in more details in Section 5.2.1. (2) Using a standard binary-search trick, we show that there exists an $O(\text{polylog}(n))$ time *reduction* from First-DNF^{dy} to DNF^{dy}. Specifically, this means that if DNF$^{dy} \in \mathsf{P}^{dy}$, then we can use an algorithm for DNF^{dy} as a subroutine to design an algorithm for First-DNF^{dy} with polylog update time and polynomial space complexity. Theorem 5.1 follows from (1) and (2), and the observation that DNF$^{dy} \in \mathsf{NP}^{dy}$.

5.2.1 NP^{dy}-hardness of First-DNF^{dy} Consider any dynamic decision problem $\mathcal{D}_n \in \mathsf{NP}^{dy}$. Thus, there exists a verifier $\mathcal{V}_n$ for $\mathcal{D}_n$ with the properties mentioned in Definition 5.3. Throughout Section 5.2.1, we assume that there is an algorithm for First-DNF^{dy} with polynomial space complexity and polylog update time. Under this assumption, we will show that there exists an algorithm $\mathcal{A}_n$ for $\mathcal{D}_n$ that also has $O(\text{poly}(n))$ space complexity and $O(\text{polylog}(n))$ update time. This will imply the NP^{dy}-hardness of First-DNF^{dy}.

The high-level strategy: The algorithm $\mathcal{A}_n$ will use the following two *subroutines*: (1) The verifier $\mathcal{V}_n$ for $\mathcal{D}_n$ as specified in Definition 5.2 and Definition 5.3, and (2) A dynamic algorithm $\mathcal{A}^*$ that solves the First-DNF^{dy} problem with polylog update time and polynomial space complexity.

To be more specific, consider any instance-sequence $(I_0, \ldots, I_k)$ for the problem $\mathcal{D}_n$. At step $t = 0$, after receiving the starting instance I_0, the algorithm $\mathcal{A}_n$ calls the subroutine $\mathcal{V}_n$ with the same input I_0. The subroutine $\mathcal{V}_n$ returns an ordered pair (x_0, y_0). At this point, the algorithm $\mathcal{A}_n$ outputs the bit x_0. Subsequently, at each step $t \geq 1$, the algorithm $\mathcal{A}_n$ receives the instance-update (I_{t-1}, I_t) as input. It then calls the subroutine $\mathcal{A}^*$ in such a manner which ensures that $\mathcal{A}^*$ returns a reward-maximizing proof π_t for the verifier $\mathcal{V}_n$ (see Definition 5.3). This is explained in more details below. The algorithm $\mathcal{A}_n$ then calls the verifier $\mathcal{V}_n$ with the input $((I_{t-1}, I_t), \pi_t)$, and the verifier returns an ordered pair (x_t, y_t). At this point, the algorithm $\mathcal{A}_n$ outputs the bit x_t.

To summarize, the algorithm $\mathcal{A}_n$ uses $\mathcal{A}^*$ as a dynamic subroutine to construct a reward-maximizing proof-sequence $(\pi_1, \ldots, \pi_k)$ – one step at a time. Furthermore, after each step $t \geq 1$, the algorithm $\mathcal{A}_n$ calls the verifier $\mathcal{V}_n$ with the input $((I_{t-1}, I_t), \pi_t)$. The verifier $\mathcal{V}_n$ returns (x_t, y_t), and the algorithm $\mathcal{A}_n$ outputs x_t. Item (1) in Definition 5.3 implies that the algorithm $\mathcal{A}_n$ outputs 0 on all the NO instances (where $\mathcal{D}_n(I_t) = 0$). Since the proof-sequence $(\pi_1, \ldots, \pi_k)$ is reward-maximizing, item (2) in Definition 5.3 implies that the algorithm $\mathcal{A}_n$ outputs 1 on all the YES instances (where $\mathcal{D}_n(I_t) = 1$). So the algorithm $\mathcal{A}_n$ always outputs the correct answer and solves the problem $\mathcal{D}_n$. We now explain how the algorithm $\mathcal{A}_n$ calls the subroutine $\mathcal{A}^*$, and then analyze the space complexity and update time of $\mathcal{A}_n$. The key observation is that we can represent the verifier $\mathcal{V}_n$ as a collection of decision trees, and each root-to-leaf path in each of these trees can be modeled as a DNF clause.

The decision trees that define the verifier $\mathcal{V}_n$: Let $\texttt{mem}_{\mathcal{V}_n}$ denote the memory of the verifier $\mathcal{V}_n$. We assume that during each step $t \geq 1$, the instance-update (I_{t-1}, I_t) is written in a designated region $\texttt{mem}^{(0)}_{\mathcal{V}_n} \subseteq \texttt{mem}_{\mathcal{V}_n}$ of the memory, and the proof π_t is written in another designated region $\texttt{mem}^{(1)}_{\mathcal{V}_n} \subseteq \texttt{mem}_{\mathcal{V}_n}$ of the memory. Each bit in $\texttt{mem}_{\mathcal{V}_n}$ can be thought of as a boolean variable $z \in \{0, 1\}$. We view the region $\texttt{mem}_{\mathcal{V}_n} \setminus \texttt{mem}^{(1)}_{\mathcal{V}_n}$ as a collection of boolean variables $\mathcal{Z} = \{z_1, \ldots, z_N\}$ and the contents of $\texttt{mem}_{\mathcal{V}_n} \setminus \texttt{mem}^{(1)}_{\mathcal{V}_n}$ as an assignment $\phi : \mathcal{Z} \to \{0, 1\}$. For example, if $\phi(z_j) = 1$ for some $z_j \in \mathcal{Z}$, then it means that the bit z_j in $\texttt{mem}_{\mathcal{V}_n} \setminus \texttt{mem}^{(1)}_{\mathcal{V}_n}$ is currently set to 1. Upon receiving an input $((I_{t-1}, I_t), \pi_t)$, the verifier $\mathcal{V}_n$ makes some probes in $\texttt{mem}_{\mathcal{V}_n} \setminus \texttt{mem}^{(1)}_{\mathcal{V}_n}$ according to some pre-defined procedure, and then outputs an answer (x_t, y_t). This procedure can be modeled as a decision tree T_{π_t}. Each internal node (including the root) in this decision tree is either a "read" node or a "write" node. Each read-node has two children and is labelled with a variable $z \in \mathcal{Z}$. Each write-node has one child and is labelled with an ordered pair (z, λ), where $z \in \mathcal{Z}$ and $\lambda \in \{0, 1\}$. Finally, each leaf-node of T_{π_t} is labelled with an ordered pair (x, y), where $x \in \{0, 1\}$ and $y \in \{0, 1\}^{\text{polylog}(n)}$. Upon receiving the input $((I_{t-1}, I_t), \pi_t)$, the verifier $\mathcal{V}_n$ *traverses* this decision tree T_{π_t}. Specifically, it starts at the root of T_{π_t}, and then inductively applies the following steps until it reaches a leaf-node.

- Suppose that it is currently at a read-node of T_{π_t} labelled with $z \in \mathcal{Z}$. If $\phi(z) = 0$ (resp. $\phi(z) = 1$), then it goes to the left (resp. right) child of the node. On the other hand, suppose that it is currently at a write-node of T_{π_t} which is labelled with (z, λ). Then it writes λ in the memory-bit z (by setting $\phi(z) = \lambda$) and then moves on to the only child of this node.

Finally, when it reaches a leaf-node, the verifier $\mathcal{V}_n$ outputs the corresponding label (x, y). This is the way the verifier operates when it is called with an input $((I_{t-1}, I_t), \pi_t)$. The depth of the decision tree (the maximum length of any root-to-leaf path) is at most $\text{polylog}(n)$, since as per Definition 5.2 the verifier makes at most $\text{polylog}(n)$ many bit-probes in the memory while handling any input.

Each possible proof π for the verifier can be specified using $\text{polylog}(n)$ bits. Hence, we get a collection of $O(2^{\text{polylog}(n)})$ many decision trees $\mathcal{T} = \{T_\pi\}$ – one tree T_π for each possible input π. This collection of decision trees $\mathcal{T}$ completely characterizes the verifier $\mathcal{V}_n$.

DNF clauses corresponding to a decision tree T_{π_t}: Suppose that the proof π is given as part of the input to the verifier during some update step. Consider any root-to-leaf path P in a decision tree T_π. We can naturally associate a DNF clause C_P corresponding to this path P. To be more specific, suppose that the path P traverses a read-node labelled with $z \in \mathcal{Z}$ and then goes to its left (resp. right) child. Then we have a literal $\neg z$ (resp. z) in the clause C_P that corresponds to this read-node.[10] The clause C_P is the conjunction (AND) of these literals, and C_P is true iff the verifier $\mathcal{V}_n$ traverses the path P when π is the proof given to it as part of the input. Let $\mathcal{C} = \{C_P : P \text{ is a root-to-leaf path in some tree } T_\pi \in \mathcal{T}\}$ be the collection of all these DNF clauses.

Defining a total order $\prec$ over $\mathcal{C}$: We now define a total order $\prec$ over $\mathcal{C}$ which satisfies the following property: Consider any two root-to-leaf paths P and P' in the collection of decision trees $\mathcal{T}$. Let (x, y) and (x', y') respectively denote the labels associated with the leaf nodes of the paths P and P'. If $C_P \prec C_{P'}$, then $y \geq y'$. Thus, the paths with higher y values appear earlier in $\prec$.

Finding a reward-maximizing proof: Suppose that $(I_0, \ldots, I_{t-1})$ is the instance-sequence of $\mathcal{D}_n$ received by $\mathcal{A}_n$ till now. By induction, suppose that $\mathcal{A}_n$ has managed to construct a reward-maximizing proof-sequence $(\pi_1, \ldots, \pi_{t-1})$ till this point, and has fed this as input to the verifier $\mathcal{V}_n$ (which is used as a subroutine). At the present moment, suppose that $\mathcal{A}_n$ receives an instance-update (I_{t-1}, I_t) as input. Our goal now is to find a reward-maximizing proof π_t at the current step t.

Consider the tuple $(\mathcal{Z}, \mathcal{C}, \phi, \prec)$ where $\mathcal{Z} = \texttt{mem}_{\mathcal{V}_n} \setminus \texttt{mem}^{(1)}_{\mathcal{V}_n}$ is the set of variables, $\mathcal{C} = \{C_P : P \text{ is a root-to-leaf path in some decision tree } T_\pi\}$ is the set of DNF clauses, the assignment $\phi : \mathcal{Z} \to \{0,1\}$ reflects the current contents of the memory-bits in $\texttt{mem}_{\mathcal{V}_n} \setminus \texttt{mem}^{(1)}_{\mathcal{V}_n}$, and $\prec$ is the total ordering over $\mathcal{C}$ described above. Let $C_{P'} \in \mathcal{C}$ be the answer to this First-DNF^{dy} instance $(\mathcal{Z}, \mathcal{C}, \phi, \prec)$, and suppose that the path P' belongs to the decision tree $T_{\pi'}$ corresponding to the proof π'. A moment's thought will reveal that $\pi_t = \pi'$ is the desired reward-maximizing proof at step t we were looking for, because of the following reason. Let (x', y') be the label associated with the leaf-node in P'. By definition, if the verifier gets the ordered pair $((I_{t-1}, I_t), \pi')$ as input at this point, then it will traverse the path P' in the decision tree $T_{\pi'}$ and return the ordered pair (x', y'). Furthermore, the path P' comes first according to the total ordering $\prec$, among all the paths that can be traversed by the verifier at this point. Hence, the path P' is chosen in such a way that maximizes y', and accordingly, we conclude that $y_t = y'$ is a reward-maximizing proof at step t.

Wrapping up: Handling an instance-update (I_{t-1}, I_t). To summarize, when the algorithm $\mathcal{A}_n$ receives an instance-update (I_{t-1}, I_t), it works as follows. It first writes down in the instance-update (I_{t-1}, I_t) in $\texttt{mem}^{(0)}_{\mathcal{V}_n}$ and accordingly updates the assignment $\phi : \mathcal{Z} \to \{0,1\}$. It then calls the subroutine $\mathcal{A}^*$ on the First-DNF^{dy} instance $(\mathcal{Z}, \mathcal{C}, \phi, \prec)$. The subroutine $\mathcal{A}^*$ returns a reward-maximizing proof π_t. The algorithm $\mathcal{A}_n$ then calls the verifier $\mathcal{V}_n$ as a subroutine with the ordered pair $((I_{t-1}, I_t), \pi_t)$ as input. The verifier updates at most $\text{polylog}(n)$ many bits in $\texttt{mem}_{\mathcal{V}_n}$ and returns an ordered pair (x_t, y_t). The algorithm $\mathcal{A}_n$ now updates the assignment $\phi : \mathcal{Z} \to \{0,1\}$ to ensure that it is synchronized with the current contents of $\texttt{mem}_{\mathcal{V}_n}$. This requires $O(\text{polylog}(n))$ many calls to the subroutine $\mathcal{A}^*$ for the First-DNF^{dy} instance. Finally, $\mathcal{A}_n$ outputs the bit $x_t \in \{0,1\}$.

Bounding the update time of $\mathcal{A}_n$: Notice that after each instance-update (I_{t-1}, I_t), the algorithm $\mathcal{A}_n$ makes one call to the verifier $\mathcal{V}_n$ and at most $\text{polylog}(n)$ many calls to $\mathcal{A}^*$. By Definition 5.2, the call to $\mathcal{V}_n$ requires $O(\text{polylog}(n))$ time. Furthermore, we have assumed that $\mathcal{A}^*$ has polylog update time. Hence, each call to $\mathcal{A}^*$ takes $O(\text{polylog}(N, M)) = O(\text{polylog}(2^{\text{polylog}(n)})) = O(\text{polylog}(n))$ time. Since the algorithm $\mathcal{A}_n$ makes at most $\text{polylog}(n)$ many calls to $\mathcal{A}^*$, the total time spent

[10] W.l.o.g. we can assume that no two read nodes on the same path are labelled with the same variable.

in these calls is still $O(\text{polylog}(n))$. Thus, we conclude that $\mathcal{A}_n$ has $O(\text{polylog}(n))$ update time.

Bounding the space complexity of $\mathcal{A}_n$: The space complexity of $\mathcal{A}_n$ is dominated by the space complexities of the subroutines $\mathcal{V}_n$ and $\mathcal{A}^*$. As per Definition 5.2, the verifier $\mathcal{V}_n$ has space complexity $O(\text{poly}(n))$.

We next bound the memory space used by the subroutine $\mathcal{A}^*$. Note that in the First-DNF^{dy} instance, we have a DNF clause $C_P \in \mathcal{C}$ for every root-to-leaf path P of every decision tree T_π. Since a proof π consists of polylog(n) bits, there are at most $O(2^{\text{polylog}(n)})$ many decision trees of the form T_π. Furthermore, since every root-to-leaf path is of length at most polylog(n), each decision tree T_π has at most $O(2^{\text{polylog}(n)})$ many root-to-leaf paths. These two observations together imply that the set of clauses $\mathcal{C}$ is of size at most $O\left(2^{\text{polylog}(n)} \cdot 2^{\text{polylog}(n)}\right) = O(2^{\text{polylog}(n)})$. Furthermore, as per Definition 5.2 there are at most $O(\text{poly}(n))$ many bits in the memory $\texttt{mem}_{\mathcal{V}_n}$, which means that there are at most $O(\text{poly}(n))$ many variables in $\mathcal{Z}$. Thus, the First-DNF^{dy} instance $(\mathcal{Z}, \mathcal{C}, \phi, \prec)$ is defined over a set of $N = \text{poly}(n)$ variables and a set of $M = 2^{\text{polylog}(n)}$ clauses (where each clause consists of at most polylog(n) many literals). We have assumed that $\mathcal{A}^*$ has quasipolynomial space complexity. Thus, the total space needed by the subroutine $\mathcal{A}^*$ is $O(2^{\text{polylog}(N,M)}) = O(2^{\text{polylog}(n)})$.

Unfortunately, the bound of $2^{\text{polylog}(n)}$ is too large for us. Instead, we will like to have a space complexity of $O(\text{poly}(n))$. Towards this end, we introduce a new subroutine $\mathcal{S}_n^*$ that acts as an *interface* between the subroutine $\mathcal{A}^*$ and the memory $\texttt{mem}_{\mathcal{A}^*}$ used by $\mathcal{A}^*$ (the details appear in the full version of the paper). Specifically, as we observed in the preceding paragraph, the memory $\texttt{mem}_{\mathcal{A}^*}$ consists of $2^{\text{polylog}(n)}$ many bits and we cannot afford to store all these bits during the execution of the algorithm $\mathcal{A}_n$. The subroutine $\mathcal{S}_n^*$ has the nice properties that (a) it has space complexity $O(\text{poly}(n))$ and (b) it can still return the content of a given bit in $\texttt{mem}_{\mathcal{A}^*}$ in $O(\text{polylog}(n))$ time. In other words, the subroutine $\mathcal{S}_n^*$ stores the contents of $\texttt{mem}_{\mathcal{A}^*}$ in an *implicit manner*, and whenever the subroutine $\mathcal{A}^*$ wants to read/write a given bit in $\texttt{mem}_{\mathcal{A}^*}$, it does that by calling the subroutine $\mathcal{S}_n^*$. This ensures that the overall space complexity of $\mathcal{A}^*$ remains $O(\text{poly}(n))$. However, the subroutine $\mathcal{S}_n^*$ will be able perform its designated task with polylog(n) update time and poly(n) space complexity only if the algorithm $\mathcal{A}_n$ is required to handle at most poly(n) many instance-updates after the preprocessing step. This is why we need Assumption 1 while defining the complexity classes P^{dy} and NP^{dy}.

To summarize, we have shown that the algorithm $\mathcal{A}_n$ has polylog update time and polynomial space complexity. This implies that the First-DNF^{dy} problem is NP^{dy}-hard.

Acknowledgement

Nanongkai and Saranurak thank Thore Husfeldt for bringing [39] to their attention.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. Nanongkai and Saranurak were also partially supported by the Swedish Research Council (Reg. No. 2015-04659).

References

[1] S. Aaronson. P=?NP. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:4, 2017. 2

[2] A. Abboud, K. Bringmann, H. Dell, and J. Nederlof. More consequences of falsifying SETH and the orthogonal vectors conjecture. In *STOC*, pages 253–266. ACM, 2018. 2

[3] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443, 2014. 2, 8, 9, 13

[4] D. Aharonov and O. Regev. Lattice problems in NP ∩ coNP. *J. ACM*, 52(5):749–765, 2005. announced at FOCS'04. 2

[5] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005. 12

[6] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. 7

[7] S. Assadi, K. Onak, B. Schieber, and S. Solomon. Fully dynamic maximal independent set with sublinear update time. In *STOC*, pages 815–826. ACM, 2018. 10

[8] S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in o(log n) update time. *SIAM J. Comput.*, 44(1):88–113, 2015. Announced at FOCS'11. 10

[9] A. Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *ICALP*, volume 80 of *LIPIcs*, pages 44:1–44:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. 10

[10] A. Bernstein and S. Chechik. Deterministic decremental single source shortest paths: beyond the o(mn) bound. In *STOC*, pages 389–397. ACM, 2016. 10

[11] A. Bernstein and S. Chechik. Deterministic partially dynamic single source shortest paths for sparse graphs. In *SODA*, pages 453–469. SIAM, 2017. 10

[12] S. Bhattacharya, D. Chakrabarty, M. Henzinger, and D. Nanongkai. Dynamic algorithms for graph coloring. In *SODA*, pages 1–20. SIAM, 2018. 10

[13] S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018. 10

[14] S. Bhattacharya, M. Henzinger, and D. Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, pages 398–411. ACM, 2016. 10

[15] S. Bhattacharya, M. Henzinger, and D. Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 470–489, 2017. 12

[16] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *ITCS*, pages 261–270. ACM, 2016. 3

[17] D. Chakraborty, L. Kamma, and K. G. Larsen. Tight cell probe bounds for succinct boolean matrix-vector multiplication. In *STOC*, pages 1297–1306. ACM, 2018. 3

[18] T. M. Chan and Y. Nekrich. Towards an optimal method for dynamic planar point location. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 390–409, 2015. 12

[19] A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. 2

[20] S. Cook. The P versus NP problem. *The millennium prize problems*, page 86, 2006. 2

[21] S. A. Cook. The importance of the P versus NP question. *J. ACM*, 50(1):27–29, 2003. 2

[22] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. 2

[23] E. Dantsin and A. Wolpert. Exponential complexity of satisfiability testing for linear-size boolean formulas. In *CIAC*, volume 7878 of *Lecture Notes in Computer Science*, pages 110–121. Springer, 2013. 2

[24] S. Datta, R. Kulkarni, A. Mukherjee, T. Schwentick, and T. Zeume. Reachability is in dynfo. In *ICALP (2)*, volume 9135 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2015. 9

[25] Y. Du and H. Zhang. Improved algorithms for fully dynamic maximal independent set. *CoRR*, abs/1804.08908, 2018. 10

[26] D. Eppstein, Z. Galil, G. F. Italiano, and A. Nissenzweig. Sparsification-a technique for speeding up dynamic graph algorithms (extended abstract). In *FOCS*, pages 60–69. IEEE Computer Society, 1992. 1

[27] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992. 12

[28] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. 1, 12

[29] G. N. Frederickson. Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees. *SIAM J. Comput.*, 26(2):484–538, 1997. 12

[30] M. L. Fredman. Observations on the complexity of generating quasi-gray codes. *SIAM J. Comput.*, 7(2):134–146, 1978. 11

[31] J. Gao, R. Impagliazzo, A. Kolokolova, and R. R. Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *SODA*, pages 2162–2181. SIAM, 2017. 2

[32] R. Greenlaw, J. Hoover, and W. L. Ruzzo. A compendium of problems complete for p (preliminary). 1991. 9

[33] M. Gupta and S. Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. 10

[34] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30, 2015. 2, 3, 10

[35] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. appeared in STOC'95. 1, 12

[36] W. Hesse and N. Immerman. Complete problems for dynamic complexity classes. In *LICS*, page 313. IEEE Computer Society, 2002. 9

[37] J. Holm, K. de Lichtenberg, and M. Thorup. Polylogarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *STOC*, pages 79–89. ACM, 1998. 1

[38] J. Holm, E. Rotenberg, and M. Thorup. Dynamic bridge-finding in $\tilde{O}(\log^2 n)$ amortized time. In *SODA*, pages 35–52. SIAM, 2018. 1

[39] T. Husfeldt and T. Rauhe. New lower bound techniques for dynamic partial sums and related problems. *SIAM J. Comput.*, 32(3):736–753, 2003. 6, 17

[40] R. Impagliazzo and A. Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *STOC*, pages 220–229. ACM, 1997. 2, 10

[41] H. Jahanjou, E. Miles, and E. Viola. Local reductions. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 749–760. Springer, 2015. 2

[42] B. M. Kapron, V. King, and B. Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013. 1, 10

[43] A. R. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Computut.*, 31(5):1501–1526, 2002. announced at STOC'99.

2
[44] R. E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975. 2, 10
[45] K. G. Larsen. Logarithmic cell probe lower bounds for non-deterministic static data structures. 6
[46] K. G. Larsen, O. Weinstein, and H. Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. *STOC*, 2018. 2
[47] K. G. Larsen and R. R. Williams. Faster online matrix-vector multiplication. In *SODA*, pages 2182–2189. SIAM, 2017. 3
[48] C. Lautemann. BPP and the polynomial hierarchy. *Inf. Process. Lett.*, 17(4):215–217, 1983. 10
[49] P. B. Miltersen. Cell probe complexity - a survey. In *In 19th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 1999. Advances in Data Structures Workshop*, 1999. 7, 11
[50] P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theor. Comput. Sci.*, 130(1):203–236, 1994. 9
[51] S. Miyano, S. Shiraishi, and T. Shoudai. *A list of P-complete problems.* Kyushu Univ., Res. Inst. of Fundamental Information Science, 1990. 9
[52] D. Nanongkai and T. Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o($n^{1/2 - \epsilon}$)-time. In *STOC*, pages 1122–1129. ACM, 2017. 1, 10
[53] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *FOCS*, pages 950–961. IEEE Computer Society, 2017. 1, 10
[54] K. Onak, B. Schieber, S. Solomon, and N. Wein. Fully dynamic MIS in uniformly sparse graphs. In *ICALP*, volume 107 of *LIPIcs*, pages 92:1–92:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. 10
[55] M. H. Overmars. *Design of Dynamic Data Structures.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1987. 12
[56] M. Patrascu. *Lower Bound Techniques for Data Structures.* PhD thesis, Cambridge, MA, USA, 2008. AAI0821553. 6
[57] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610. ACM, 2010. 2, 13
[58] M. Patrascu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. announced at STOC'04 and SODA'04. 1, 6
[59] G. Ramalingam and T. W. Reps. On the computational complexity of dynamic graph problems. *Theor. Comput. Sci.*, 158(1&2):233–277, 1996. 10
[60] J. H. Reif. A topological approach to dynamic graph connectivity. *Inf. Process. Lett.*, 25(1):65–70, 1987. 9
[61] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126. SIAM, 2007. 2
[62] R. Santhanam and S. Srinivasan. On the limits of sparsification. In *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 774–785. Springer, 2012. 2
[63] T. Schwentick and T. Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. 9
[64] M. Sipser. A complexity theoretic approach to randomness. In *STOC*, pages 330–335. ACM, 1983. 10
[65] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *STOC*, pages 114–122. ACM, 1981. 6, 12
[66] S. Solomon and N. Wein. Improved dynamic graph coloring. In *ESA*, volume 112 of *LIPIcs*, pages 72:1–72:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. 10
[67] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC*, pages 343–350. ACM, 2000. 1
[68] M. Thorup. Fully-dynamic min-cut. In *STOC*, pages 224–230. ACM, 2001. 2
[69] Y. Wang and Y. Yin. Certificates in data structures. In *ICALP (1)*, volume 8572 of *Lecture Notes in Computer Science*, pages 1039–1050. Springer, 2014. 6
[70] V. Weber and T. Schwentick. Dynamic complexity theory revisited. In *STACS*, volume 3404 of *Lecture Notes in Computer Science*, pages 256–268. Springer, 2005. 9
[71] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. Announced at STOC'10. 2
[72] R. R. Williams. Some estimated likelihoods for computational complexity. 2018. 2
[73] C. Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *STOC*, pages 1130–1143. ACM, 2017. 1, 10
[74] A. C. Yao. Theory and applications of trapdoor functions (extended abstract). In *FOCS*, pages 80–91. IEEE Computer Society, 1982. 10
[75] Y. Yin. Cell-probe proofs. *TOCT*, 2(1):1:1–1:17, 2010. 6