Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture*

Monika Henzinger†
University of Vienna, Austria
Faculty of Computer Science

Sebastian Krinninger†
University of Vienna, Austria
Faculty of Computer Science

Danupon Nanongkai‡
KTH Royal Institute of Technology

Thatchaphol Saranurak§
KTH Royal Institute of Technology

Abstract

Consider the following *Online Boolean Matrix-Vector Multiplication* problem: We are given an $n \times n$ matrix M and will receive n column-vectors of size n, denoted by $v_1, \ldots, v_n$, one by one. After seeing each vector v_i, we have to output the product Mv_i before we can see the next vector. A naive algorithm can solve this problem using $O(n^3)$ time in total, and its running time can be slightly improved to $O(n^3/\log^2 n)$ [Williams SODA'07].

We show that a conjecture that there is no *truly subcubic* ($O(n^{3-\epsilon})$) time algorithm for this problem can be used to exhibit the underlying polynomial time hardness shared by many dynamic problems. For a number of problems, such as subgraph connectivity, Pagh's problem, d-failure connectivity, decremental single-source shortest paths, and decremental transitive closure, this conjecture implies tight hardness results. Thus, proving or disproving this conjecture will be very interesting as it will either imply several tight unconditional lower bounds or break through a common barrier that blocks progress with these problems. This conjecture might also be considered as strong evidence against any further improvement for these problems since refuting it will imply a major breakthrough for combinatorial Boolean matrix multiplication and other long-standing problems if the term "combinatorial algorithms" is interpreted as "non-Strassen-like algorithms" [Ballard et al. SPAA'11].

The conjecture also leads to hardness results for problems that were previously based on diverse problems and conjectures – such as 3SUM, combinatorial Boolean matrix multiplication, triangle detection, and multiphase – thus providing a uniform way to prove polynomial hardness results for dynamic algorithms; some of the new proofs are also simpler or even become trivial. The conjecture also leads to stronger and new, non-trivial, hardness results, e.g., for the fully dynamic densest subgraph and diameter problems.

*A preliminary version of this paper was presented at the 47th ACM Symposium on Theory of Computing (STOC 2015).

†Work partially done while visiting the Simons Institute for the Theory of Computing. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Grant Agreement number 340506.

‡Work partially done while at University of Vienna, Austria.

§Work partially done while at University of Vienna, Austria and Saarland University, Germany.

Contents

1 Introduction

Consider the following problem called *Online Boolean Matrix-Vector Multiplication* (OMv): Initially, an algorithm is given an integer n and an $n \times n$ Boolean matrix M. Then, the following protocol repeats for n rounds: At the i^{th} round, it is given an n-dimensional column vector, denoted by v_i, and has to compute Mv_i. It has to output the resulting column vector before it can proceed to the next round. We want the algorithm to finish the computation as quickly as possible.

This problem is a generalization of the classic *Matrix-Vector Multiplication* problem (Mv), which is the special case with only one vector. The main question is whether we can *preprocess* the matrix in order to make the multiplication with n sequentially given vectors faster than n matrix-vector multiplications. This study dates back to as far as 1955 (e.g., [Mot55]), but most major theoretical work has focused on structured matrices; see, e.g., [Pan01, Wil07] for more information. A naive algorithm can multiply the matrix with each vector in $O(n^2)$ time, and thus requires $O(n^3)$ time in total. It was long known that the matrix can be preprocessed in $O(n^2)$ time in order to compute Mv_i in $O(n^2/\log n)$ time, implying an $O(n^3/\log n)$ time algorithm for OMv; see, e.g., [Sav74, SU86] and a recent extension in [LZ09]. More recently, Williams [Wil07] showed that the matrix can be preprocessed in $O(n^{2+\epsilon})$ time in order to compute Mv_i in $O(n^2/\epsilon \log^2 n)$ time for any $0 < \epsilon < 1/2$, implying an $O(n^3/\log^2 n)$ time algorithm for OMv. This is the current best running time for OMv. In this light, it is natural to conjecture that this problem does not admit a so-called *truly subcubic time algorithm*:

Conjecture 1.1 (OMv Conjecture)**.** *For any constant $\epsilon > 0$, there is no $O(n^{3-\epsilon})$-time algorithm that solves OMv with an error probability of at most $1/3$.*

In fact, it can be argued that this conjecture is implied by the standard *combinatorial Boolean matrix multiplication* (BMM) conjecture which states that there is no truly subcubic ($O(n^{3-\epsilon})$) time *combinatorial* algorithm for multiplying two $n \times n$ Boolean matrices if the term “combinatorial algorithms” (which is not a well-defined term) is interpreted in a certain way – in particular if it is interpreted as “non-Strassen-like algorithms”, as defined in [BDH$^+$12], which captures all known fast matrix multiplication algorithms; see Section 1.2 for further discussion. Thus, breaking Conjecture 1.1 is arguably at least as hard as making a breakthrough for Boolean matrix multiplication and other long-standing open problems (e.g., [DHZ00, VWW10, AVW14, RT13, HKN13]). This conjecture is also supported by an algebraic lower bound [Blä14].

1.1 OMv-Hardness for Dynamic Algorithms

We show that the OMv conjecture can very well capture the underlying polynomial time hardness shared by a large number of dynamic problems, leading to a unification, simplification, and strengthening of previous results. By dynamic algorithm we mean an algorithm that allows a change to the input. It usually allows three operations: (1) *preprocessing*, which is called when the input is first received, (2) *update*, which is called for every input update, and (3) *query*, which is used to request an answer to the problem. For example, in a typical dynamic graph problem, say *s*-*t* shortest path, we will start with an empty graph at the preprocessing step. Each update operation consists of an insertion or deletion of one edge.

The algorithm has to answer a query by returning the distance between s and t at that time. Corresponding to the three operations, we have *preprocessing time*, *update time*, and *query time*. There are two types of bounds on the update time: *worst-case bounds*, which bound the time that *each individual* update takes in the worst case, and *amortized bounds* which bound the time taken by *all* updates and then averaging it over all updates. The bounds on query time can be distinguished in the same way. We call a dynamic algorithm *fully dynamic* if any of its updates can be undone (e.g., an edge insertion can later be undone by an edge deletion); otherwise, we call it *partially dynamic*. We call a partially dynamic graph algorithm *decremental* if it allows only edge deletions, and *incremental* if it allows only edge insertions. For this type of algorithm, the update time is often analyzed in terms of *total update time*, which is the total time needed to handle *all* insertions or deletions.

Previous hardness results for dynamic problems are often based on diverse conjectures, such as those for 3SUM, combinatorial Boolean matrix multiplication (BMM), triangle detection, all-pairs shortest paths, and multiphase (we provide their definitions in Appendix A for completeness). This sometimes made hardness proofs quite intricate since there are many conjectures to start from, which often yield different hardness results, and in some cases none of these results are tight. Our approach results in stronger bounds which are tight for some problems. Additionally, we show that a number of previous proofs can be unified as they can now start from only one problem, that is OMv, and can be done in a much simpler way (compare, e.g., the hardness proof for Pagh's problem in this paper and in [AVW14]). Thus proving the hardness of a problem via OMv should be a simpler task.

We next explain our main results and the differences to prior work: As shown in Figure 1, we obtain more than 15 new tight[1] hardness results[2]. (Details of these results are provided in Tables 4 and 8 for tight results and Tables 5 to 7 for improved results. We also provide a summary of the problem definitions in Tables 1 to 3.) (1) Generally speaking, for most previous hardness results in [Pat10, AVW14, KPP16] that rely on various conjectures, except those relying on the Strong Exponential Time Hypothesis (SETH), our OMv conjecture implies hardness bounds on the amortized time per operation that are the same or better. (2) We also obtain new results such as those for vertex color distance oracles (studied in [HLW+11, Che12] and used to tackle the minimum Steiner tree problem [LOP+15]), restricted top trees with edge query problem (used to tackle the minimum cut problem in [FKN+14]), and the dynamic densest subgraph problem [BHN+15]. (3) Some minor improvement can in fact immediately be obtained since our conjecture implies a very strong bound for Pătraşcu's *multiphase* problem [Pat10], giving improved bounds for many problems considered in [Pat10]. We can, however, improve these bounds even more by avoiding a reduction via the multiphase problem. (We discuss this further in Section 1.2.)

[1] Our results are tight in one of the following ways: (1) the query time of the existing algorithms cannot be improved without significantly increasing the update time, (2) the update time of the existing algorithms cannot be improved without significantly increasing the query time, (3) the update and query time of the existing algorithms cannot be improved simultaneously, and (4) the approximation guarantee cannot be improved without significantly increasing both query and update time.

[2] For the s-t reachability problem, our result does not subsume the result based on the Boolean matrix multiplication (BMM) conjecture because the latter result holds only for combinatorial algorithms, and it is in fact larger than an upper bound provided by the non-combinatorial algorithm of Sankowski [San04] (see Section 1.2 for a discussion). Also note that the result based on the triangle detection problem which is not subsumed by our result holds only for a more restricted notion of amortization (see Section 1.2). This explains the solid lines in Figure 1.

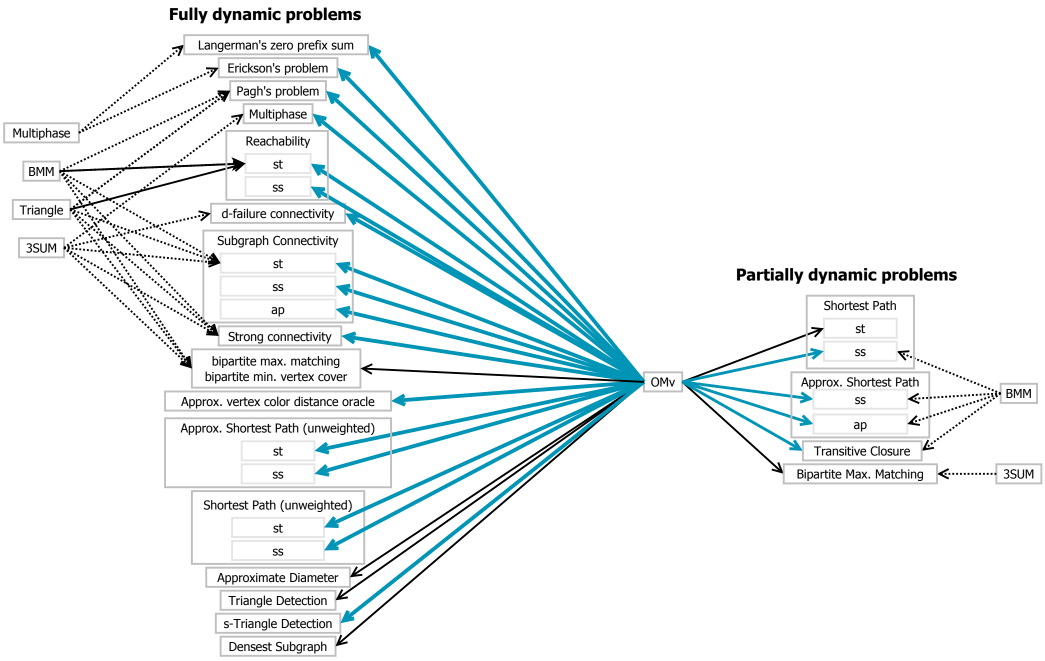

Figure 1: Overview of our and previous hardness results. An arrow from a conjecture to a problem indicates that there is a hardness result based on the conjecture. A thick blue arrow indicates that the hardness result is tight, i.e., there is a matching upper bound. A dotted arrow means that the result is subsumed in our paper. (Footnote 2 discusses results that are not subsumed.) Note that all hardness results based on BMM hold only for combinatorial algorithms.

The conjecture leads to an improvement for all problems whose hardness was previously based on 3SUM. (4) A few other improvements follow from converting previous hardness results that hold only for *combinatorial* algorithms into hardness results that hold for *any* algorithm. We note that removing the term "combinatorial" is an important task as there are algebraic algorithms that can break through some bounds for combinatorial algorithms. (We discuss this more in Section 1.2.) (5) Interestingly, all our hardness results hold even when we allow an *arbitrary polynomial preprocessing time*. This type of results was obtained earlier only in [AVW14] for hardness results based on SETH. (6) Since the OMv conjecture can replace all other conjectures except SETH, these two conjectures together are sufficient to show that all hardness results for dynamic problems known so far hold even for arbitrary polynomial preprocessing time. (7) We also note that all our results hold for a very general type of amortized running time; e.g., they hold even when there is a large (polynomial) number of updates and, for graph problems, even when we start with an empty graph. No previous hardness results, except those obtained via SETH, hold for this case.

We believe that the *universality* and *simplicity* of the OMv conjecture will be important not only in proving tight hardness results for well-studied dynamic problems, but also in developing faster algorithms; for example, as mentioned earlier it can be used to show the limits of some specific approaches to attack the minimum Steiner tree and minimum cut problems [LOP+15, FKN+14]. Below is a sample of our results. A list of all of them and detailed proofs can be found in later sections.

Subgraph Connectivity. In this problem, introduced by Frigioni and Italiano [FI00], we are given a graph G, and we have to maintain a subset S of nodes where the updates are adding and removing a node of G to and from S, which can be viewed as turning nodes on and off. The queries are to determine whether two nodes s and t are in the same connected component in the subgraph induced by S. The best upper bound in terms of m is an algorithm with $\tilde{O}(m^{4/3})$ preprocessing time, $\tilde{O}(m^{2/3})$ amortized update time and $\tilde{O}(m^{1/3})$ worst-case query time [CPR11]. There is also an algorithm with $\tilde{O}(m^{6/5})$ preprocessing time, $\tilde{O}(m^{4/5})$ worst-case update time and $\tilde{O}(m^{1/5})$ worst-case query time [Dua10]. An upper bound in terms of n is an algorithm with $\tilde{O}(m)$ preprocessing time, $\tilde{O}(n)$ worst-case update time, and $O(1)$ worst-case query time[3].

For hardness in terms of m, Abboud and Vassilevska Williams [AVW14] showed that the 3SUM conjecture can rule out algorithms with $m^{4/3-\epsilon}$ preprocessing time, $m^{\alpha-\epsilon}$ amortized update time and $m^{2/3-\alpha-\epsilon}$ amortized query time, for any constants $1/6 \leq \alpha \leq 1/3$ and $0 < \epsilon < \alpha$. In this paper, we show that the OMv conjecture can rule out algorithms with *polynomial* preprocessing time, $m^{\alpha-\epsilon}$ amortized update time and $m^{1-\alpha-\epsilon}$ amortized query time[4], for any $0 \leq \alpha \leq 1$. This matches the upper bound of [CPR11] when we set $\alpha = 2/3$.

For hardness in terms of n, Pǎtraşcu [Pat10] showed that, assuming the hardness of his *multiphase problem*, there is no algorithm with $n^{\delta-\epsilon}$ worst-case update time and query time, for some constant $0 < \delta \leq 1$. By assuming the combinatorial BMM conjecture, Abboud and

[3]This update time is achieved by using $O(n)$ updates for dynamic connectivity data structure under edge updates by [KKM13]. The query time needs only $O(1)$ time because this data structure internally maintains a spanning forest, so we can label vertices in each component in the spanning forest in time $O(n)$ after each update.

[4]We note the following detail: The 3SUM-hardness result of Abboud and Vassilevska Williams holds when $m \leq n^{1.5}$ and our hardness result holds when $m \leq \min\{n^{1/\alpha}, n^{1/(1-\alpha)}\}$

Vassilevska Williams [AVW14] could rule out combinatorial algorithms with $n^{1-\epsilon}$ amortized update time, and $n^{2-\epsilon}$ query time. These two bounds cannot rule out some improvement over [KKM13], e.g., a non-combinatorial algorithm with $n^{1-\epsilon}$ amortized update time, and $O(1)$ amortized query time. In this paper, we show that the OMv conjecture can rule out *any* algorithm with *polynomial* preprocessing time, $n^{1-\epsilon}$ amortized update time and $n^{2-\epsilon}$ amortized query time. Thus, there is no algorithm that can improve the upper bound of [KKM13] without significantly increasing the query time.

Decremental Shortest Paths. In the decremental single-source shortest paths problem, we are given an unweighted undirected graph G and a source node s. Performing an update means to delete an edge from the graph. A query will ask for the distance from s to some node v. The best exact algorithm for this problem is due to the classic result of Even and Shiloach [ES81] and requires $O(m)$ preprocessing time, $O(mn)$ total update time, and $O(1)$ query time. Very recently, Henzinger, Krinninger, and Nanongkai [HKN14a] showed a $(1+\epsilon)$-approximation algorithm with $O(m^{1+o(1)})$ preprocessing time, $O(m^{1+o(1)})$ total update time, and $O(1)$ query time. Roditty and Zwick [RZ11] showed that the combinatorial BMM conjecture implies that there is no combinatorial *exact* algorithm with $mn^{1-\epsilon}$ preprocessing time and $mn^{1-\epsilon}$ total update time if we need $\tilde{O}(1)$ query time. This leaves the open problem whether we can develop a faster exact algorithm for this problem using algebraic techniques (e.g., by adapting Sankowski's techniques [San04, San05a, San05b]). Our OMv conjecture implies that this is not possible: there is no exact algorithm with polynomial preprocessing and $mn^{1-\epsilon}$ total update time if we need $\tilde{O}(1)$ query time.

For the decremental all-pairs shortest paths problem on undirected graphs, $(1+\epsilon)$-approximation algorithms with $\tilde{O}(mn)$ total update time are also known in both unweighted and weighted cases [RZ12, Ber13]. For combinatorial algorithms, this is tight even in the static setting under the combinatorial BMM conjecture [DHZ00]. Since fast matrix multiplication can be used to break this bound in the static setting when the graph is dense, the question whether we can do the same in the dynamic setting was raised by Bernstein [Ber13]. In this paper, we show that this is impossible under the OMv conjecture. (Our hardness result holds for any algorithm with approximation ratio less than two.)

Pagh's Problem. In this problem, we want to maintain a family X of at most k sets $\{X_i\}_{1\le i\le k}$ over $[n]$. An update is by adding the set $X_i \cap X_j$ to X. We have to answer a query of the form "Does element j belong to set X_i?". A trivial solution to this problem requires $O(kn)$ preprocessing time, $O(n)$ worst-case update time and $O(1)$ worst-case query time. Previously, Abboud and Williams [AVW14] showed that, assuming the combinatorial BMM conjecture, for any $n \le k \le n^2$ there is no combinatorial algorithm with $k^{3/2-\epsilon}$ preprocessing time, $k^{1/2-\epsilon}$ amortized update time, and $k^{1/2-\epsilon}$ amortized query time. They also obtained hardness for non-combinatorial algorithms but the bounds are weaker. Our OMv conjecture implies that for any $k = poly(n)$ there is no algorithm with $\mathrm{poly}(k, n)$ preprocessing time, $n^{1-\epsilon}$ update time, and $k^{1-\epsilon}$ query time, matching the trivial upper bound. Note that our hardness holds against *all* algorithms, including non-combinatorial algorithms. Also note that while the previous proof in [AVW14] is rather complicated (it needs, e.g., a universal hash function), our proof is almost trivial.

Fully Dynamic Weighted Diameter Approximation. In this problem, we are give a weighted undirected graph. An update operation adds or deletes a weighted edge. The query asks for the diameter of the graph. For the unweighted case, Abboud and Vassilevska Williams [AVW14] showed that the Strong Exponential Time Hypothesis (SETH) rules out any $(4/3-\epsilon)$-approximation algorithm with polynomial preprocessing time, $n^{2-\epsilon}$ update and query time. Nothing was known for the weighted setting. In this paper, we show that for the weighted case, OMv rules out any $(2-\epsilon)$-approximation algorithm with polynomial preprocessing time, $n^{1/2-\epsilon}$ update time and $n^{1-\epsilon}$ query time. This result is among a few that require a rather non-trivial proof.

1.2 Discussions

OMv vs. Combinatorial BMM. The combinatorial BMM conjecture states that there is no truly subcubic *combinatorial* algorithm for multiplying two $n \times n$ Boolean matrices. There are two important points to discuss here. First, it can be easily observed that any reduction from the OMv problem can be turned into a reduction from the combinatorial BMM problem since, although we get two matrices at once in the BMM problem, we can always pretend that we see one column of the second matrix at a time (this is the OMv problem). This means that bounds obtained via the OMv conjecture will *never* be stronger than bounds obtained via the combinatorial BMM conjecture. However, the latter bounds will hold *only for combinatorial algorithms*, leaving the possibility of an improvement via an algebraic algorithm. This possibility cannot be overlooked since there are examples where an algebraic algorithm can break through the combinatorial hardness obtained by assuming the combinatorial BMM conjecture. For example, it was shown in [AVW14] that the combinatorial BMM conjecture implies that there is no combinatorial algorithm with $n^{3-\epsilon}$ preprocessing time, $n^{2-\epsilon}$ update time, and $n^{2-\epsilon}$ query time for the fully dynamic s-t reachability and bipartite perfect matching problems. However, we can break these bounds using Sankowski's algebraic algorithm [San04, San07] which requires n^{ω} preprocessing time, $n^{1.449}$ worst-case update time, and $O(1)$ worst case query time, where ω is the exponent of the best known matrix multiplication algorithm (currently, $\omega < 2.3728639$ [Gal14]).[5]

Second, it can be argued that the combinatorial BMM conjecture actually implies the OMv conjecture, if the term "combinatorial algorithm" is interpreted in a certain way. Note that while this term has been used very often (e.g., [DHZ00, VWW10, AVW14, RT13, HKN13]), it is not a well-defined term. Usually it is vaguely used to refer as an algorithm that is different from the "algebraic" approach originated by Strassen [Str69]; see, e.g., [BVW08, BW12, BKM95]. One formal way to interpret this term is by using the term "Strassen-like algorithm", as defined by Ballard et al. [BDH+12]. Roughly speaking, a Strassen-like algorithm divides *both* matrices into constant-size blocks and utilizes an algorithm for multiplying two blocks in order to recursively multiply matrices of arbitrary size (see [BDH+12, Section 5.1] for a detailed definition[6]). As pointed out in [BDH+12],

[5]Furthermore, the exponent 1.449 is the result of balancing the terms $n^{1+\epsilon}$ and $n^{\omega(1,\epsilon,1)-\epsilon}$, where $\omega(1,\epsilon,1)$ is the exponent of the best known algorithm [Gal12] for multiplying an $n \times n^{\epsilon}$ matrix with an $n^{\epsilon} \times n$ matrix. The value 1.449 is obtained by a linear interpolation of the values of $\omega(1,\epsilon,1)$ reported in [Gal12], which upperbounds $\omega(1,\epsilon,1)$.

[6]Note that Ballard et al. also need to include a technical assumption in the Strassen-like algorithms that they consider to prove their results (see [BDH+12, Section 5.1.1]). This assumption is irrelevant to us.

this is the structure of all the fast matrix multiplication algorithms that were obtained since Strassen's, including the recent breakthroughs by Stothers [Sto10] and Vassilevska Williams [VW12]. Since OMv reveals one column of the second matrix at a time, it naturally disallows an algorithm to utilize block multiplications, and thus Strassen-like algorithms cannot be used to solve OMv.

We note that the OMv problem actually excludes even some combinatorial BMM algorithms; e.g., the $O(n^3(\log\log n)^2/\log^{2.25} n)$-time algorithm of Bansal and Williams [BW12] cannot be used to solve OMv. Finally, even if one wants to interpret the term "combinatorial algorithm" differently and argue that the combinatorial BMM conjecture does not imply the OMv conjecture, we believe that breaking the OMv conjecture will still be a breakthrough since it will yield a fast matrix multiplication algorithm that is substantially different from those using Strassen's approach.

OMv vs. Multiphase. Pătraşcu [Pat10] introduced a dynamic version of set disjointness called *multiphase problem*, which can be rephrased as a variation of the Matrix-Vector multiplication problem as follows (see [Pat10] for the original definition). Let k, n, and τ be some parameters. First, we are given a $k \times n$ Boolean matrix M and have $O(nk \cdot \tau)$ time to preprocess M. Second, we are given an n-dimensional vector v and have $O(n \cdot \tau)$ additional computation time. Finally, we are given an integer $1 \leq i \leq n$ and must output $(Mv)[i]$ in $O(\tau)$ time. Pătraşcu conjectured that if there are constants $\gamma > 0$ and $\delta > 0$ such that $k = n^\gamma$, then any solution to the multiphase problem in the Word RAM model requires $\tau = n^\delta$, and used this conjecture to prove polynomial time hardness for several dynamic problems. How strong these hardness bounds are depends on how hard one believes the multiphase problem to be. By a trivial reduction, the OMv conjecture implies that the multiphase conjecture holds with $\delta = 1$ when $\gamma = 1$. (We found it quite surprising that viewing the multiphase problem as a matrix problem, instead of a set problem as originally stated, can give an intuitive explanation for a possible value of δ.) This implies the strongest bound possible for the multiphase problem. Moreover, while hardness based on the multiphase problem can only hold for a *worst-case* time bound, it can be shown that under a general condition we can make them hold for an *amortized* time bound too if we instead assume the OMv conjecture (see Section 5.1 for details). Thus, with the OMv conjecture it seems that we do not need the multiphase conjecture anymore. Note that, as argued before, we can also conclude that the combinatorial BMM conjecture implies the multiphase conjecture. To the best of our knowledge this is the first connection between these conjectures.

OMv vs. 3SUM and SETH. As mentioned earlier, all previous hardness results that were based on the 3SUM conjecture can be strengthened through the OMv conjecture. However, we do not have a general mechanism that can always convert any hardness proof based on 3SUM into a proof based on OMv. Finding such a mechanism would be interesting.

Techniques for proving hardness for dynamic algorithms based on SETH were very recently introduced in [AVW14]. Results from these techniques are the only ones that cannot be obtained through OMv. SETH together with OMv seems to be enough to prove all the hardness results known to date. It would be very interesting if the number of conjectures one has to start with can be reduced to one.

Remark On the Notion of Amortization. We emphasize that there are two different ways to define the notion of amortized update time. First, we can define it as an amortized update time when we *start from an empty graph*; equivalently, the update time has to be amortized over all edges that ever appear. The second way is to allow the algorithm to preprocess an arbitrary input graph and amortize the update time over all updates (not counting the edges in the initial graph as updates); for example, one can start from a graph with n^2 edges and have only n updates. The first definition is more common in the analysis of dynamic algorithms but it is harder to prove hardness results for it. Our hardness results hold for this type of amortization; in fact, they hold even when there is a large (polynomial) number of updates. Many previous hardness results hold only for the second type of amortization, e.g., the results in [AVW14] that are not based on SETH and 3SUM.

1.3 Notation

All matrices and vectors in this paper are Boolean and $\tilde{O}(\cdot)$ hides logarithmic factors in $O(\cdot)$. We will also use the following non-standard notation.

Definition 1.2 ($\tilde{\tilde{o}}$ Notation)**.** *For any parameters n_1, n_2, n_3, we say that a function $f(n_1, n_2, n_3) = \tilde{\tilde{o}}(n_1^{c_1} n_2^{c_2} n_3^{c_3})$ iff there exists some constant $\epsilon > 0$ such that $f = O(n_1^{c_1-\epsilon} n_2^{c_2} n_3^{c_3} + n_1^{c_1} n_2^{c_2-\epsilon} n_3^{c_3} + n_1^{c_1} n_2^{c_2} n_3^{c_3-\epsilon})$. We use the analogous definition for functions with one or two parameters.*

1.4 Organization

Instead of starting from OMv, our reductions will start from an intermediate problem called OuMv. We describe this and prove necessary results in Section 2. In Section 3 we prove hardness results for the amortized update time of fully dynamic algorithms and the worst-case update time of partially dynamic algorithms. In Section 4 we prove hardness results for the total update time of partially dynamic problems. In Section 5 we provide further discussions.

2 Intermediate Problems

In this section we show that the OMv conjecture implies that OMv is hard even when there is a polynomial preprocessing time and different dimension parameters (Section 2.1). Then in Section 2.2, we present the problem whose hardness can be proved assuming the OMv conjecture, namely the *online vector-matrix-vector multiplication* (OuMv) problem, which is the key starting points for our reductions in later sections.

2.1 OMv with Polynomial Preprocessing Time and Arbitrary Dimensions

We first define a more general version of the OMv problem: (1) we allow the algorithm to preprocess the matrix before the vectors arrive and (2) we allow the matrix to have arbitrary dimensions with a promise that the size of minimum dimension is not too "small" compared to the size of maximum dimension.

Definition 2.1 (γ-OMv)**.** *Let $\gamma > 0$ be a fixed constant. An algorithm for the γ-OMv problem is given* parameters *n_1, n_2, n_3 as input with a promise that $n_1 = \lfloor n_2^{\gamma} \rfloor$. Next, it is given a*

Name and short name		Input	Update	Query
Subgraph Connectivity	st-SubConn	A fixed undirected graph G, and a subset S of its vertices	Insert/remove a node into/from S	Are s and t connected in $G[S]$ the subgraph induced by S?
	ss-SubConn			For any v, are s and v connected in $G[S]$ the subgraph induced by S?
	ap-SubConn			For any u, v, are u and v connected in $G[S]$ the subgraph induced by S?
Reachability	st-Reach	A directed graph	Edge insertions/deletions	Is t reachable from s?
	ss-Reach			For any v, is v reachable from s?
	ap-Reach (Transitive Closure or TC)			For any u, v, is v reachable from u?
Shortest Path (undirected)	st-SP	An undirected unweighted graph	Edge insertions/deletions	Find the distance $d(s, t)$.
	ss-SP			Find the distance $d(s, v)$, for any v.
	ap-SP			Find the distance $d(u, v)$, for any u, v.
Triangle Detection		An undirected graph	Edge insertions/deletions	Is there a triangle in the graph?
s-Triangle Detection				Is there a triangle containing s in the graph?

Table 1: Definitions of dynamic graph problems (1)

Name	Input	Update	Query
Densest Subgraph	An undirected graph	Edge insertions/deletions	What is the density $\lvert E(S)\rvert/\lvert S\rvert$ of the densest subgraph S?
d-failure connectivity [DP10]	A fixed undirected graph	Roll back to original graph. Then remove any d vertices from the graph.	Is s connected to t, for any given (s, t)?
Vertex color distance oracle [Che12, LOP+15]	A fixed undirected graph	Change the color of any vertex	Given s and color c, find the shortest distance from s to any c-colored vertex.
Diameter	An undirected graph	Edge insertions/deletions	Find the diameter of the graph.
Strong Connectivity	A directed graph	Edge insertions/deletions	Is the graph strongly connected?

Table 2: Definitions of dynamic graph problems (2)

Name	Input	Update	Query
Pagh's Problem	A family $\mathcal{X}$ of sets $X_1, X_2, \ldots, X_k \subseteq [n]$	Given i, j, insert $X_i \cap X_j$ into $\mathcal{X}$	Given index i and $j \in [n]$, is $j \in X_i$?
Langerman's Zero Prefix Sum	An array $A[1 \ldots n]$ of integers	Set $A[i] = x$ for any i and x	Is there a k s.t. $\sum_{i=1}^{k} A[i] = 0$?
Erickson's Problem	A matrix of integers of size $n \times n$	Increment all values in a specified row or column	Find the maximum value in the matrix

Table 3: Definitions of dynamic non-graph problems

matrix M of size $n_1 \times n_2$ that can be preprocessed. Let $p(n_1, n_2)$ denote the preprocessing time*. After the preprocessing, an online sequence of vectors $v^1, \ldots, v^{n_3}$ is presented one after the other and the task is to compute each Mv^t before v^{t+1} arrives. Let $c(n_1, n_2, n_3)$ denote the* computation time *over the whole sequence.*

Note that the γ-OMv problem can be trivially solved with $O(n_1n_2n_3)$ total computing time and without preprocessing time. Obviously, the OMv conjecture implies that this running time is (almost) tight when $n_1 = n_2 = n_3 = n$. Interestingly, it also implies that this running time is tight for other values of n_1, n_2, and n_3:

Theorem 2.2. *For any constant $\gamma > 0$, Conjecture 1.1 implies that there is no algorithm for γ-OMv with parameters n_1, n_2, n_3 using preprocessing time $p(n_1, n_2) = poly(n_1, n_2)$ and computation time $c(n_1, n_2, n_3) = \tilde{o}(n_1n_2n_3)$ that has an error probability of at most $1/3$.*

The rest of this section is devoted to proving the above theorem. The proof proceeds in two steps. First, we show that, assuming Conjecture 1.1, there is no algorithm for γ-OMv when the preprocessing time is $(n_1n_2)^{1+\epsilon}$ for any constant $\epsilon < 1/2$ and the computation time is $\tilde{o}(n_1n_2n_3)$.

Lemma 2.3. *For any constant $\gamma > 0$ and integer n, fix any n_1, n_2 where $n_1 = n_2^\gamma$, $\max\{n_1, n_2\} = n$ and $n_3 = n$.*[7] *Suppose there is an algorithm $\mathcal{A}$ for γ-OMv with parameters n_1, n_2, n_3 using preprocessing time $p(n_1, n_2)$ and computation time $c(n_1, n_2, n_3)$ that has an error probability of at most δ. Then there is an algorithm $\mathcal{B}$ for OMv with parameter n using (no preprocessing time and) computation time $\tilde{O}(\frac{n^2}{n_1n_2}p(n_1, n_2) + \frac{n^2}{n_1n_2}c(n_1, n_2, n) + \frac{n^3}{n_2})$ that has an error probability of at most δ.*

Proof. We will construct $\mathcal{B}$ by using $\mathcal{A}$ as a subroutine. We partition M into blocks $\{M_{x,y}\}_{1 \le x \le n/n_1, 1 \le y \le n/n_2}$ where $M_{x,y}$ is of size $n_1 \times n_2$.[8] We feed $M_{x,y}$ to an instance $I_{x,y}$ of $\mathcal{A}$ and preprocess using $\frac{n^2}{n_1n_2}p(n_1, n_2)$ time. For each vector v^t, we partition it into blocks $\{v^t_y\}_{1 \le y \le n/n_2}$ each of size n_2. For each x, y, we compute $M_{x,y}v^t_y$ using the instance $I_{x,y}$ for all $t \le n$. The total time for computing $M_{x,y}v^t_y$, for all x, y, t, is $\frac{n^2}{n_1n_2}c(n_1, n_2, n)$. We keep the error probability to remain at most δ by a standard application of the Chernoff bound:

[7] Actually, we need $n_1 = \lfloor n_2^\gamma \rfloor$ but from now we will always omit it and assume that n_2^γ is an integer. This affects the running time of the statement by at most a constant factor.

[8] Here we assume that n_1 and n_2 divides n and we will similarly assume this whenever we divide a matrix into a blocks. This assumption can be removed easily: for each "boundary" blocks $M_{x,y}$ where $x = \lceil n/n_1 \rceil$ or $y = \lceil n/n_2 \rceil$, we keep the size $M_{x,y}$ to be $n_1 \times n_2$ but it may overlap with other block. This will affect the running time by at most a constant factor.

repeat the above procedure for computing $M_{x,y}v_y^t$, for each x, y, t, $O(\log(n_1n_2n_3))$ many times and take the most frequent answer.

Let $c^t = Mv^t$. We write c^t as blocks $\{c_x^t\}_{1\le x\le n/n_1}$ each of size n_1. Since $c_x^t = \bigvee_y M_{x,y}v_y^t$ (bit-wise OR) for each x, t, we can compute c_x^t in time $\frac{n}{n_2}n_1$ for each x, t (there are $\frac{n}{n_2}$ many y's and $M_{x,y}v_y^t$ is a vector of size n_1). The total time for this, over all x, t, is $\frac{n}{n_1}n \times \frac{n}{n_2}n_1 = \frac{n^3}{n_2}$. $\square$

Corollary 2.4. *For any constants $\gamma > 0$ and $\epsilon < 1/2$, Conjecture 1.1 implies that there is no algorithm for γ-OMv with parameters n_1, n_2, n_3 using preprocessing time $p(n_1, n_2) \le (n_1n_2)^{1+\epsilon}$ and computation time $c(n_1, n_2, n_3) = \tilde{o}(n_1n_2n_3)$ that has an error probability of at most $1/3$.*

Proof. Suppose there is such an algorithm. Then by Lemma 2.3, we can solve OMv with parameter n in time $\tilde{O}(\frac{n^2}{n_1n_2}p(n_1, n_2) + \frac{n^2}{n_1n_2}c(n_1, n_2, n) + \frac{n^3}{n_2})$ with error probability at most $1/3$ where $n_1 = n_2^\gamma$ and $\max\{n_1, n_2\} = n$. We have $\frac{n^2}{n_1n_2}p(n_1, n_2) \le n^2(n_1n_2)^\epsilon \le n^{2+2\epsilon} = \tilde{o}(n^3)$, and $\frac{n^2}{n_1n_2}c(n_1, n_2, n) + \frac{n^3}{n_2} = \tilde{o}(n^3)$ where the last equality holds because $n_1, n_2 \ge \min\{n^\gamma, n^{1/\gamma}\}$. The total time is $\tilde{o}(n^3)$ and the error probability is at most $1/3$ contradicting Conjecture 1.1. $\square$

For the second step, we show that the hardness of γ-OMv even when the preprocessing time is $p(n_1, n_2) = \text{poly}(n_1, n_2)$.

Lemma 2.5. *For any constant $\gamma > 0$ and integers n_1, n_2, n_3 where $n_1 = n_2^\gamma$, fix any k_1, k_2 where $k_1 \le n_1, k_2 \le n_2$, and $k_1 = k_2^\gamma$. Suppose there is an algorithm $\mathcal{A}$ for γ-OMv with parameters k_1, k_2, n_3, preprocessing time $p(k_1, k_2)$ and computation time $c(k_1, k_2, n_3)$ that has an error probability of at most δ. Then there is an algorithm $\mathcal{B}$ for γ-OMv with parameters n_1, n_2, n_3, preprocessing time $\tilde{O}(\frac{n_1n_2}{k_1k_2}p(k_1, k_2))$ and computation time $\tilde{O}(\frac{n_1n_2}{k_1k_2}c(k_1, k_2, n_3) + \frac{n_1n_2n_3}{k_2})$ that has an error probability of at most δ.*

Proof. We will construct $\mathcal{B}$ by using $\mathcal{A}$ as a subroutine. We partition M into blocks $\{M_{x,y}\}_{1\le x\le n_1/k_1, 1\le y\le n_2/k_2}$ where $M_{x,y}$ is of size $k_1 \times k_2$. We feed $M_{x,y}$ to an instance $I_{x,y}$ of $\mathcal{A}$ and then preprocess. This takes $\frac{n_1n_2}{k_1k_2}p(k_1, k_2)$ total preprocessing time.

Once the vector v^t arrives, we partition it into blocks $\{v_y^t\}_{1\le y\le n_2/k_2}$ each of size k_2. For each x, y, we compute $M_{x,y}v_y^t$ using the instance $I_{x,y}$. The total computation time, over all t, for doing this will be $\frac{n_1n_2}{k_1k_2}c(k_1, k_2, n_3)$. By repeating the procedure for a logarithmic number of times as in Lemma 2.3, the error probability remains at most δ.

Let $c^t = Mv^t$. We write c^t as blocks $\{c_x^t\}_{1\le x\le n_1/k_1}$ each of size k_1. Since $c_x^t = \bigvee_y M_{x,y}v_y^t$ for each x, t, we can compute c_x^t in time $\frac{n_2}{k_2}k_1$ for each x, t. The total time for this, over all x, t, is $\frac{n_1}{k_1}n_3 \times \frac{n_2}{k_2}k_1 = \frac{n_1n_2n_3}{k_2}$. $\square$

We conclude the proof of Theorem 2.2.

Proof of Theorem 2.2. We construct an algorithm $\mathcal{B}$ for γ-OMv with parameters n_1, n_2, n_3 that contradicts Conjecture 1.1 by using an algorithm $\mathcal{A}$ from the statement of Theorem 2.2 as a subroutine. That is, $\mathcal{A}$ is an algorithm for γ-OMv with parameters k_1, k_2, k_3, preprocessing time $(k_1k_2)^c$ for some constant c, and computation time $\tilde{o}(k_1k_2k_3)$ with error probability

1/3. Let $\epsilon < \min\{1/2, c\}$ be a constant. We choose k_1, k_2 such that $k_1 = n_1^{\epsilon/c}$, $k_2 = n_2^{\epsilon/c}$ and $k_3 = n_3$.

Note that $k_1 \le n_1$, $k_2 \le n_2$ and $k_1 = k_2^{\gamma}$. So we can apply Lemma 2.5 and get $\mathcal{B}$ which has error probability 1/3 and, ignoring polylogarithmic factors, uses preprocessing time $\frac{n_1 n_2}{k_1 k_2} p(k_1, k_2) \le n_1 n_2 \cdot p(k_1, k_2) \le (n_1 n_2)^{1+\epsilon}$ and computation time $\frac{n_1 n_2}{k_1 k_2} c(k_1, k_2, n_3) + \frac{n_1 n_2 n_3}{k_2} = \frac{n_1 n_2}{n_1^{\epsilon/c} n_2^{\epsilon/c}} \tilde{\tilde{o}}(n_1^{\epsilon/c} n_2^{\epsilon/c} n_3) + \frac{n_1 n_2 n_3}{n_2^{\epsilon/c}} = \tilde{\tilde{o}}(n_1 n_2 n_3)$. This contradicts Conjecture 1.1 by Corollary 2.4. □

2.2 The Online Vector-Matrix-Vector Multiplication Problem (OuMv)

Although we base our results on the hardness of OMv, the starting point of most of our reductions is a slightly different problem called online vector-matrix-vector multiplication problem. In this problem, we multiply the matrix with two vectors, one from the left and one from the right.

Definition 2.6 (γ-OuMv problem)**.** *Let $\gamma > 0$ be a fixed constant. An algorithm for the γ-OuMv problem is given* parameters n_1, n_2, n_3 *as its input with the promise that $n_1 = \lfloor n_2^{\gamma} \rfloor$. Next, it is given a matrix M of size $n_1 \times n_2$ that can be preprocessed. Let $p(n_1, n_2)$ denote the* preprocessing time*. After the preprocessing, an online sequence of vector pairs $(u^1, v^1), \ldots, (u^{n_3}, v^{n_3})$ is presented one after the other and the task is to compute each $(u^t)^\top M v^t$ before (u^{t+1}, v^{t+1}) arrives. Let $c(n_1, n_2, n_3)$ denote the* computation time *over the whole sequence. The γ-uMv problem with parameters n_1, n_2 is the special case of γ-OuMv where $n_3 = 1$.*

We also write OuMv and uMv to refer to, respectively, γ-OuMv and γ-uMv without the promise. Our reductions will exploit the fact that the result of this multiplication is either 0 or 1; thus using only 1 bit as opposed to n bits in OMv. Starting from OuMv instead of OMv will thus give simpler reductions and better lower bounds on the query time. Using a technique for finding "witnesses", which will be defined below, when the result of a vector-matrix-vector multiplication is 1, we can reduce the γ-OMv problem to the γ-OuMv problem and establish the following hardness for γ-OuMv.

Theorem 2.7. *For any constant $\gamma > 0$, Conjecture 1.1 implies that there is no algorithm for γ-OuMv with parameters n_1, n_2, n_3 using preprocessing time $p(n_1, n_2) = poly(n_1, n_2)$ and computation time $c(n_1, n_2, n_3) = \tilde{\tilde{o}}(n_1 n_2 n_3)$ that has an error probability of at most 1/3.*

An γ-uMv algorithm with preprocessing time $p(n_1, n_2)$ and computation time $c(n_1, n_2)$ implies an γ-OuMv algorithm with preprocessing time $\tilde{O}(p(n_1, n_2))$, computation time $\tilde{O}(n_3 c(n_1, n_2))$ and the same error probability by a standard application of the Chernoff bound as in the proof of Lemma 2.3. Therefore, we also get the following:

Corollary 2.8. *For any constant $\gamma > 0$, Conjecture 1.1 implies that there is no algorithm for γ-uMv with parameters n_1, n_2 using preprocessing time $p(n_1, n_2) = \text{poly}(n_1, n_2)$ and computation time $c(n_1, n_2) = \tilde{\tilde{o}}(n_1 n_2)$ that has an error probability of at most 1/3.*

The rest of this section is devoted to the proof of Theorem 2.7.

Definition 2.9 (Witness of OuMv)**.** *We say that any index i is a* witness *for a pair of vectors (u^t, v^t) in an instance I of OuMv if $u_i^t \wedge (Mv^t)_i = 1$, i.e., the i-th entries of vectors u^t and Mv^t are both one.*

Observe that $(u^t)^\top Mv^t = 1$ if and only if there is a witness for (u^t, v^t). The problem of *listing all the witnesses of γ-OuMv* is defined similarly as γ-OuMv except that for each vector pair (u^t, v^t), we have to list all witnesses of (u^t, v^t) (i.e., output every index i such that i is a witness) before (u^{t+1}, v^{t+1}) arrives. We first show a reduction from the problem of listing all the witnesses of γ-OuMv to the γ-OuMv problem itself. The reduction is similar to [VWW10, Lemma 3.2].

Lemma 2.10. *Fix any constant $\gamma > 0$ and integers n_1, n_2 and n_3. Suppose there is an algorithm $\mathcal{A}$ for γ-OuMv with parameters n_1, n_2, n_3 preprocessing time $p(n_1, n_2)$ and computation time $c(n_1, n_2, n_3)$ that has an error probability of at most δ. Then there is an algorithm $\mathcal{B}$ for listing all witnesses of γ-OuMv with the same parameters using preprocessing time $\tilde{O}(p(n_1, n_2))$ and computation time $\tilde{O}((1 + \sum_t \tilde{O}(w_t)/n_3) \cdot c(n_1, n_2, n_3))$ that has an error probability of at most δ, where w_t is the number of witnesses of (u^t, v^t).*

Proof. We will show a reduction for deterministic algorithms. This reduction can be extended to work for randomized algorithms as well by a standard application of the Chernoff bound as in the proof of Lemma 2.3. We construct $\mathcal{B}$ using $\mathcal{A}$ as a subroutine. Let M be the input matrix of $\mathcal{B}$. We use $\mathcal{A}$ to preprocess M.

For any vector pair (u, v), we say the "*query* (u, v)" to $\mathcal{A}$ returns true if, by using $\mathcal{A}$, we get $u^\top Mv = 1$. For any a set of indices I of entries of u, let u_I be the n_1-dimensional vector where $(u_I)_i = (u)_i$ for all $i \in I$ and $(u_I)_i = 0$ otherwise. Suppose that I contains w many witnesses of (u, v). We now describe a method to identify all witnesses of (u, v) in I using $1 + O(w \log n_1)$ queries. Note that the witnesses of (u, v) contained in I are exactly the witnesses of (u_I, v). We check if $w = 0$ by querying (u_I, v) one time. If $w > 0$, then we identify an arbitrary witness of (u_I, v) one by one using binary search. More precisely, if I is of size one, then return the only index $i \in I$ which must be a witness. Otherwise, let $I_0 \subset I$ be a set of size $\lfloor |I|/2 \rfloor$. If the query (u_{I_0}, v) return true, then recurse on I_0. Otherwise, recurse on $I \setminus I_0$. This takes $O(\log n_1)$ queries because $|I| \le n_1$. Once we find a witness $i \in I$, we do the same procedure on $I \setminus \{i\}$ until we find all w witnesses. Therefore, the total number of queries for finding w many witnesses of (u, v) in I is $1 + O(w \log n_1)$.

Once (u^t, v^t) arrives, we list all witnesses of (u^t, v^t) using $1 + O(w_t \log n_1)$ queries by the above procedure where $I = [n_1]$. The total number of queries is $n_3 + \sum_t \tilde{O}(w_t)$. However, $\mathcal{A}$ is an algorithm for γ-OuMv with parameters n_1, n_2, n_3. So once there are n_3 queries to $\mathcal{A}$, we need to roll back $\mathcal{A}$ to the state right after preprocessing. Hence, we need to roll back $\frac{n_3 + \sum_t \tilde{O}(w_t)}{n_3} = 1 + \sum_t \tilde{O}(w_t)/n_3$ times. Therefore, the total computation time is $(1 + \sum_t \tilde{O}(w_t)/n_3) c(n_1, n_2, n_3)$. □

Next, using Lemma 2.10, we can show the reduction from γ-OMv to γ-OuMv.

Lemma 2.11. *For any constant $\gamma > 0$ and integers n_1, n_2, n_3 where $n_1 = n_2^\gamma$, fix k_1, k_2 and k_3 such that $k_1 k_2 = n_2$, $k_1 = k_2^\gamma$ and $k_3 = n_3$. Suppose there is an algorithm $\mathcal{A}$ for γ-OuMv with parameters k_1, k_2, k_3 and preprocessing time $p(k_1, k_2)$ and computation time $c(k_1, k_2, k_3)$ that has an error probability of at most δ. Then there is an algorithm $\mathcal{B}$ for*

γ-OMv with parameters n_1, n_2, n_3 using preprocessing time $\tilde{O}(n_1 \cdot p(k_1, k_2))$ and computation time $\tilde{O}(n_1 \cdot c(k_1, k_2, n_3))$ that has an error probability of at most δ.

Proof. Again, we show a reduction for deterministic algorithms. This can be extended to work for randomized algorithms by a standard application of the Chernoff bound as in the proof of Lemma 2.3. By plugging $\mathcal{A}$ into Lemma 2.10, we have an algorithm $\mathcal{A}'$ for listing all witnesses of γ-OuMv with parameters k_1, k_2, k_3. We will formulate $\mathcal{B}$ using $\mathcal{A}'$ as a subroutine.

Let M be an input matrix M of $\mathcal{B}$ of size $n_1 \times n_2$, we partition M into blocks $\{M_{x,y}\}_{1 \le x \le n_1/k_1, 1 \le y \le n_2/k_2}$ each of size $k_1 \times k_2$. For each x and y, let $I_{x,y}$ be an instance of $\mathcal{A}'$ and we feed $M_{x,y}$ into $I_{x,y}$ to preprocess. The total preprocessing time is $\frac{n_1 n_2}{k_1 k_2} \cdot p(k_1, k_2) = n_1 \cdot p(k_1, k_2)$.

In the γ-OMv problem, for any $t \le n_3$, once v^t arrives, we need to compute $b^t = Mv^t$ before v^{t+1} arrives. To do so, we first partition v^t into blocks $\{v^t_y\}_{1 \le y \le n_2/k_2}$ each of size k_2. We write $b^t = \{b^t_x\}_{1 \le x \le n_1/k_1}$ as blocks each of size k_1. Note that $b^t_x = \bigvee_y M_{x,y} v^t_y$ ($\bigvee$ means bit-wise OR).

To compute b^t_x, the procedure iterates over all values for y from 1 to n_2/k_2. When $y = 1$, we set $u^t_{x,1}$ to be the all-ones vector. Let $W_{x,y,t}$ be the set of witnesses (u^t_x, v^t_y). We feed $(u^t_{x,y}, v^t_y)$ to the instance $I_{x,y}$ for listing the witnesses $W_{x,y,t}$. For all $i \in W_{x,y,t}$, we now know that $(b^t_x)_i = 1$. To find other indices i such that $(b^t_x)_i = 1$, we set $u^t_{x,y+1}$ to be same as $u^t_{x,y}$ except that $(u^t_{x,y+1})_i = 0$ for all found witnesses $i \in W_{x,y,t}$. Then we proceed with $y \leftarrow y + 1$. We repeat this until $y = n_2/k_2$. Then b^t_x is completely computed. Once this procedure is done for all x, b^t is completely computed. We repeat until $t = n_3$, and we are done.

Now, we denote $w_{x,y} = \sum_t |W_{x,y,t}|$. By Lemma 2.10 the computation time of the instance $I_{x,y}$ is $\frac{n_3 + \tilde{O}(w_{xy})}{n_3} c(k_1, k_2, n_3)$. Summing over all x, y, we have a total running time of

$$\sum_{1 \le x \le \frac{n_1}{k_1}, 1 \le y \le \frac{n_2}{k_2}} \frac{n_3 + \tilde{O}(w_{xy})}{n_3} c(k_1, k_2, n_3)\,.$$

To conclude that the computation time is $\tilde{O}(n_1 \cdot c(k_1, k_2, n_3))$, it is enough to show that $\sum_{x,y} w_{x,y} \le n_1 n_3$. Note that for a fixed t, the witness sets $W_{x,y,t}$ are disjoint for different x and y as we set the entries in the 'u'-vectors to 0 for witnesses that we already found. Furthermore for every $i \in \bigcup_y W_{x,y,t}$, we have $(b^t_x)_i = 1$. As the number of 1-entries of b^t is at most n_1, we have $\sum_{x,y} |W_{x,y,t}| \le n_1$. So $\sum_{x,y,t} |W_{x,y,t}| \le n_1 n_3$. Hence $\sum_{x,y} w_{x,y} \le n_1 n_3$. $\square$

Now we are ready to prove the main theorem.

Proof of Theorem 2.7. We will construct an algorithm $\mathcal{B}$ for γ-OMv with parameters n_1, n_2, n_3 that contradicts Conjecture 1.1 by using an algorithm $\mathcal{A}$ for γ-OuMv from the statement of Theorem 2.7 as a subroutine. That is, $\mathcal{A}$ is an algorithm for γ-OuMv with parameters k_1, k_2, k_3 using preprocessing time $p(k_1, k_2) = \text{poly}(k_1, k_1)$ and computation time $c(k_1, k_2, k_3) = \tilde{o}(k_1 k_2 k_3)$ that has an error probability of at most $1/3$. We choose k_1, k_2 such that $k_1 k_2 = n_2$, $k_1 = k_2^\gamma$ and $k_3 = n_3$.

By Lemma 2.11, ignoring polylogarithmic factors, $\mathcal{B}$ has error probability $1/3$ and uses preprocessing time $\text{poly}(n_1, n_2)$ and computation time $n_1 \cdot c(k_1, k_2, n_3) = n_1 \tilde{o}(k_1 k_2 n_3) = \tilde{o}(n_1 n_2 n_3)$ which contradicts Conjecture 1.1 by Theorem 2.2. $\square$

2.2.1 Interpreting OuMv as Graph Problems and a satisfiability problem

In this section, we show that OuMv can be viewed as graph problems and satisfiability problem, namely *edge query*, *independent set query* and *2-CNF query*, as stated in Theorem 2.12. This section is independent from the rest and will not be used later. The edge query problem, however, can be helpful when one wants to show a reduction from OuMv to graph problems. For example, it is implicit in the reduction from OuMv to the subgraph connectivity problem. Moreover, the independent set query problem was shown to have an $O(n^2n'/\log^2 n)$ time algorithm by Williams [Wil07] using his OMv algorithm. The 2-CNF query problem was shown to be equivalent to the independent set query problem in [BW12]. So it is interesting that these problem are equivalent to OuMv and thus cannot be solved much faster (i.e., in $\tilde{o}(n^2n')$ time) assuming the OMv conjecture.

Theorem 2.12. *For any integers n and n', consider the following problems.*

- *OuMv with parameters $n_1, n_2 = \Theta(n)$ and $n_3 = n'$, preprocessing time $p(n_1, n_2)$ and computation time $c(n_1, n_2, n_3)$.*
- *Independent set (respectively vertex cover) query [BW12]: preprocess a graph $G = (V, E)$, when $|V| = n$, in time $p_{ind}(n)$. Then given a sequence of sets $S^1, \ldots, S^{n'} \subseteq V$, decide if S^t is an independent set (respectively a vertex cover) before S^{t+1} arrives in total time $c_{ind}(n, n')$.*
- *2-CNF query [BW12]: preprocess a 2-CNF F on n variables in time $p_{cnf}(n)$. Then given a sequence of assignments $X^1, \ldots, X^{n'}$, decide if $F(X^t) = 1$ before X^{t+1} arrives in total time $c_{cnf}(n, n')$.*
- *Edge query: preprocess a graph $G = (V, E)$, when $|V| = n$, in time $p_{edge}(n)$. Then given a sequence of set pairs $(S^1, T^1), \ldots, (S^{n'}, T^{n'})$, decide if there is an edge $(a, b) \in E(S^t, T^t)$, before S^{t+1} arrives in total time $c_{edge}(n, n')$.*

We have that $p(n_1, n_2) = \Theta(p_{ind}(n)) = \Theta(p_{cnf}(n)) = \Theta(p_{edge}(n))$ and $c(n_1, n_2, n_3) = \Theta(c_{ind}(n, n')) = \Theta(c_{cnf}(n, n')) = \Theta(c_{edge}(n, n'))$.

Another observation that might be useful in proving OMv hardness results is the following.

Theorem 2.13. *In the OuMv problem, we can assume that the matrix M is symmetric, and each vector pair (u^t, v^t) is such that either $u^t = v^t$ or the supports of u^t and v^t are disjoint (i.e., the inner product between u^t and v^t is 0).*

The rest of this section is devoted to proving the above theorems. First, we need this fact.

Proposition 2.14. *Consider a Boolean matrix $M \in \{0, 1\}^{n_1 \times n_2}$ and Boolean vectors $u \in \{0, 1\}^{n_1}$ and $v \in \{0, 1\}^{n_2}$. Let $M' = \begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix}$, $w = \begin{bmatrix} u \\ v \end{bmatrix}$, $x = \begin{bmatrix} u \\ 0 \end{bmatrix}$ and $y = \begin{bmatrix} 0 \\ v \end{bmatrix}$ where $w, x, y \in \{0, 1\}^{n_1+n_2}$. Then $u^\top M v = w^\top M' w = x^\top M' y$.*

Proof. It is easy to verify that $w^\top M' w = w^\top \begin{bmatrix} Mv \\ M^T u \end{bmatrix} = (u^\top M v) \vee (v^\top M^\top u) = u^\top M v$. Similarly, $x^\top M' y = x^\top \begin{bmatrix} Mv \\ 0 \end{bmatrix} = u^\top M v$. $\square$

Theorem 2.13 follows immediately from Proposition 2.14. Now we prove Theorem 2.12.

Proof of Theorem 2.12. Let $\mathcal{A}_{\mathsf{OMv}}, \mathcal{A}_{ind}$ and $\mathcal{A}_{edge}$ be the algorithms for OMv, independent set query and edge query respectively.

(OuMv ⇒ independent set query) Given an input graph $G = (V, E)$ of $\mathcal{A}_{ind}$, preprocess the adjacency matrix M of G using $\mathcal{A}_{\mathsf{OMv}}$. Once S^t arrives, let v^t be the indicator vector of S^t (i.e., for all $i \in V$, $(v^t)_i = 1$ if $i \in S^t$, otherwise $(v^t)_i = 0$). Observe that $(v^t)^\top M v^t = 0$ iff S^t is independent. So we can use $\mathcal{A}_{\mathsf{OMv}}$ to answer the query of $\mathcal{A}_{ind}$.

(OuMv ⇒ edge query) Given an input graph G of $\mathcal{A}_{edge}$, preprocess the adjacency matrix M of G using $\mathcal{A}_{\mathsf{OMv}}$. Once (S^t, T^t) arrives, let u^t and v^t be the indicator vectors of S^t and T^t respectively. Observe that $(u^t)^\top M v^t = 1$ iff there is an edge $(a, b) \in E(S^t, T^t)$. So we can use $\mathcal{A}_{\mathsf{OMv}}$ to answer the query of $\mathcal{A}_{edge}$.

(independent set query ⇒ OuMv) Given an input matrix M of $\mathcal{A}_{\mathsf{OMv}}$, let G the graph defined by the adjacency matrix $M' = \begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix}$. We preprocess G using $\mathcal{A}_{ind}$. Once (u^t, v^t) arrives, let S^t be the set indicated by $w = \begin{bmatrix} u^t \\ v^t \end{bmatrix}$ (i.e., $i \in S^t$ iff w_i=1). We have that S^t is independent iff $(u^t)^\top M v^t = w^\top M' w = 0$ by Proposition 2.14. So we can use $\mathcal{A}_{ind}$ to answer the query of $\mathcal{A}_{\mathsf{OMv}}$.

(edge query ⇒ OuMv) Given an input matrix M of $\mathcal{A}_{\mathsf{OMv}}$, let G be the graph defined by the adjacency matrix $M' = \begin{bmatrix} 0 & M \\ M^T & 0 \end{bmatrix}$. We preprocess G using $\mathcal{A}_{edge}$. Once (u^t, v^t) arrives, let S^t, T^t be the sets indicated by $x = \begin{bmatrix} u^t \\ 0 \end{bmatrix}$ and $y = \begin{bmatrix} 0 \\ v^t \end{bmatrix}$. There is an edge $(a, b) \in E(S^t, T^t)$ iff $(u^t)^\top M v^t = x^\top M' y = 1$ by Proposition 2.14. So we can use $\mathcal{A}_{edge}$ to answer the query of $\mathcal{A}_{\mathsf{OMv}}$.

(independent set query ⇔ 2-CNF query) See [BW12, Section 2.3]. □

Note that we can use an OMv algorithm to solve the dominating set query problem, defined in a similar way as independent set query problem. Indeed, let M be the adjacency matrix of G, and v be an indicator vector of S. We have that $Mv \vee v$ (bit-wise OR) is the all-one vector iff S is a dominating set. However, it is not clear if the reverse reduction exists.

3 Hardness for Amortized Fully Dynamic and Worst-case Partially Dynamic Problems

In this section, we give reductions from our intermediate problems to various dynamic problems. In Section 3.1, we give conditional lower bounds for those graph problems whose algorithms cannot have the update time $u(m) = \tilde{o}(\sqrt{m})$ and the query time $q(m) = \tilde{o}(m)$ simultaneously. In Section 3.2, we give the bounds for those problems that cannot have the update time $u(m) = \tilde{o}(m^{1-\delta})$ and the query time $q(m) = \tilde{o}(m^\delta)$ simultaneously, for any constant $0 < \delta < 1$. In Sections 3.3 and 3.4, we give the lower bounds for the remaining graph and non-graph problems, whose lower bound parameters of update/query time are in a different form (see Figure 2). We devote Sections 3.5 and 3.6 to proving the lower bounds for approximating the diameter of a weighted graph and the densest subgraph problem, respectively, because their reductions are more involved.

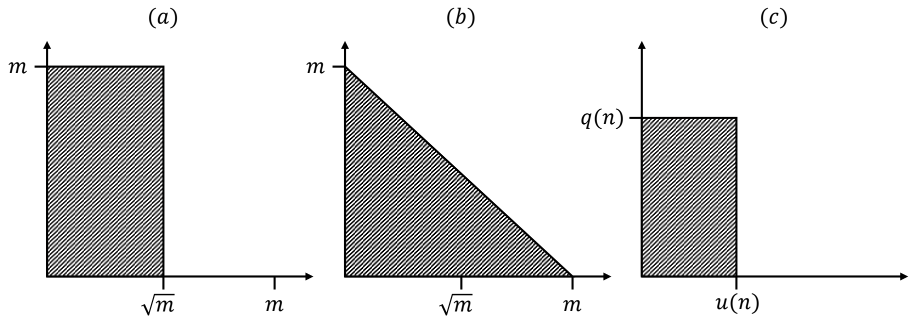

Figure 2: Horizontal and vertical axes represent the update and query time of dynamic algorithms respectively. The shaded areas indicate the ranges of update/query time whose existence of dynamic algorithm with such parameters would contradict Conjecture 1.1. Chart (a) and (b) are for problems in Section 3.1 and Section 3.2 respectively. Chart (c) is for problems in Sections 3.3 to 3.6 where the lower bound parameters are in many different forms.

Our hardness results, compared to previously known bounds, for fully dynamic problems are summarized in Table 5 and Table 6. Our tight hardness results are summarized in Table 4.

Given a matrix $M \in \{0,1\}^{n_1 \times n_2}$, we denote by $G_M = ((L,R),E)$ the bipartite graph where $L = \{l_1, \ldots, l_{n_1}\}$, $R = \{r_1, \ldots, r_{n_2}\}$, and $E = \{(r_j, l_i) \mid M_{ij} = 1\}$.

In this section, our proofs usually follow two simple steps. First, we show the reductions in lemmas that given a dynamic algorithm $\mathcal{A}$ for some problem, one can solve uMv by running the preprocessing step of $\mathcal{A}$ on some graph and then making some number of updates and queries. Then, we conclude in corollaries that if either 1) $\mathcal{A}$ has low worst-case update/query time, or 2) $\mathcal{A}$ has low amortized update/query time and $\mathcal{A}$ is fully dynamic, then this contradicts Conjecture 1.1.

3.1 Lower Bounds for Graph Problems with High Query Time

To show hardness of the problems in this section, we reduce from uMv where $n_1 = n_2 = \sqrt{m}$. The idea is that when u and v arrive, we make the update operations of the dynamic algorithm $\mathcal{A}$ to "handle" both u and v. Then make only 1 query to $\mathcal{A}$ to answer 1-uMv. Since the reduction is efficient in the number of queries, we get a high lower bound of query time.

s-t Subgraph Connectivity (st-SubConn)

[9] implies the same bound for all reachablity problems (including transitive closure), strong connectivity, bipartite perfect matching, size of maximum matching, minimum vertex cover, maximum independent set on bipartite graph, size of st-maxflow on undirected unit capacity. See some reductions from [AVW14]

[10] implies all shortest path problems (ss-SP,ap-SP) with same approximation factor

[11] implies the same bound for st-Reach

[12] implies the same bound for ap-SubConn, ss-Reach, transitive closure.

[13] implies the same bound for ap-SP with same approximation factor.

Problems	Upper Bounds			Lower Bounds			Remark
	$p(m,n)$	$u(m,n)$	$q(m,n)$	$p(m,n)$	$u(m,n)$	$q(m,n)$	
ss-SubConn, ap-SubConn	$m^{4/3}$	$m^{2/3}$	$m^{1/3}$	*poly*	$m^{2/3-\epsilon}$	$m^{1/3-\epsilon}$	Upper: [CPR11], amortized only; Lower: when $m = O(n^{3/2})$
	$m^{6/5}$	$m^{4/5}$	$m^{1/5}$	*poly*	$m^{4/5-\epsilon}$	$m^{1/5-\epsilon}$	Upper: [Dua10]; Lower: when $m = O(n^{5/4})$
	1	1	m	*poly*	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$	Lower: when $m = O(n^2)$
	m	n	1	*poly*	$n^{1-\epsilon}$	$n^{2-\epsilon}$	Upper: [KKM13]; Lower: $m = \Theta(n^2)$
st-SubConn, (unweighted) st-SP, st-Reach, s-triangle detection, strong connectivity	1	1	m	*poly*	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$	Lower: when $m = O(n^2)$
(unweighted) ss-SP, ss-Reach	1	m	1	*poly*	$m^{1-\epsilon}$	$m^{\epsilon'}$	Lower: when $m = O(n^{1/(1-\epsilon)})$, $\epsilon' < \epsilon$
	1	1	m	*poly*	$m^{1/2-\epsilon}$	$m^{1-\epsilon}$	Lower: when $m = O(n^2)$
Erickson's problem	1	n	1	*poly*	$n^{1-\epsilon}$	$n^{1-\epsilon}$	Upper: binary search tree
d-failure connectivity	$mn^{1/c}$	d^{2c+4}	d	$poly(n)$	$poly(d)$	$d^{1-\epsilon}$	Upper: [DP10]
3-approx vertex color distance oracle	$m\sqrt{n}$	$\sqrt{n}$	1	*poly*	$n^{1-\epsilon}$	$n^{2-\epsilon}$	Upper: [HLW+11, Che12, LOP+15]; Lower: when $(3-\epsilon)$-approx
Pagh's problem over k sets in a universe $[n]$	1	n	1	*poly*	$n^{1-\epsilon}$	$k^{1-\epsilon}$	
Multiphase over k sets in a universe $[n]$	1	$\tau \leq \max\{k,n\}$		*poly*	$\tau \geq \min\{k,n\}^{1-\epsilon}$		

Table 4: Our tight results along with the matching upper bounds (or better upper bounds when worse approximation ratio is allowed). The polylogarithmic factors are omitted. The lower bounds state that there is no algorithm achieving stated preprocessing time, amortized update time, and query time simultaneously, unless the OMv conjecture fails. The matching upper bounds of update time are all worst-case time except the bound by [CPR11]. The upper bounds without remark are the naive ones.

Problems	$p(m,n)$	$u(m,n)$	$q(m,n)$	Conj.	Reference	Remark
st-SubConn	$m^{4/3}$	$m^{\delta-\epsilon}$	$m^{2/3-\delta-\epsilon}$	3SUM	[AVW14]	Choose any $\delta \in [1/6, 1/3]$. $m \leq O(n^{1.5})$
	$m^{1+\delta-\epsilon}$	$m^{\delta-\epsilon}$	$m^{2\delta-\epsilon}$	Triangle	[AVW14]	For some $\delta < 0.41$ depending on the conjecture
	$n^{3-\epsilon}$	$n^{1-\epsilon}$	$n^{2-\epsilon}$	BMM	[AVW14]	When $m = \Theta(n^2)$
st-Reach	$m^{4/3}$	$m^{\delta-\epsilon}$	$m^{2/3-\delta-\epsilon}$	3SUM	[AVW14]	Choose any $\delta \in [1/6, 1/3]$. $m \leq O(n^{1.5})$
	$\mathbf{m^{1+\delta-\epsilon}}$	$\mathbf{m^{2\delta-\epsilon}}$	$\mathbf{m^{2\delta-\epsilon}}$ (*)	Triangle	[AVW14]	Only lower bound for amortized time over $O(n)$ updates; for some $\delta < 0.41$ depending on the conjecture
	$\mathbf{n^{3-\epsilon}}$	$\mathbf{n^{2-\epsilon}}$	$\mathbf{n^{2-\epsilon}}$ (*)	BMM	[AVW14]	Only lower bound for amortized time over $O(n)$ updates; $m = \Theta(n^2)$
st-SubConn[9], st-Reach, unweighted (α, β)st-SP where $3\alpha + \beta < 5$ [10], Triangle Detection, s-Triangle Detection[11], etc. (See footnotes)	poly($\mathbf{n}$)	$\mathbf{m^{1/2-\epsilon}}$	$\mathbf{m^{1-\epsilon}}$	OMv	Corollary 3.4	Choose any $m \leq n^2$
ss-SubConn[12], unweighted (α, β)ss-SP where $2\alpha, \beta < 4$[13], unweighted (α, β)vertex color distance oracle where $\alpha + \beta < 3$, etc. (See footnotes)	poly($\mathbf{n}$)	$\mathbf{m^{\delta-\epsilon}}$	$\mathbf{m^{1-\delta-\epsilon}}$	OMv	Corollary 3.8	Choose any $\delta \in (0,1)$, and $m \leq \min\{n^{1/\delta}, n^{1/(1-\delta)}\}$
unweighted $(3-\epsilon)$st-SP, $(2-\epsilon)$diameter on weighted Graphs, Densest Subgraph of size at least 5	poly($\mathbf{n}$)	$\mathbf{n^{1/2-\epsilon}}$	$\mathbf{n^{1-\epsilon}}$	OMv	Corollaries 3.10, 3.22 and 3.26	
d-failure Connectivity	$n^{2-\epsilon}$	$(dn)^{1/2-\epsilon}$	$d^{1/2-\epsilon}$	3SUM	[KPP16]	
	poly($\mathbf{n}$)	$\mathbf{(dn)^{1-\epsilon}}$	$\mathbf{d^{1-\epsilon}}$	OMv	Corollary 3.13	Choose any $\delta \in (0, 1/2]$, $d = m^\delta$, $m = \Theta(n^{1/(1-\delta)})$
	poly($\mathbf{n}$)	$\mathbf{d^{1/\delta}}$	$\mathbf{d^{1-\epsilon}}$	OMv	Corollary 3.12	Choose any $\delta \in (0, 1/2]$, $d = m^\delta$, $m \leq \Theta(n^{1/(1-\delta)})$

Table 5: Amortized lower bounds for fully dynamic graph problems and worst-case lower bounds for partially dynamic graph problems. Bounds which are not subsumed are highlighted. Each row states that there is no algorithm achieving stated preprocessing time, update time, and query time *simultaneously*, unless the conjecture fails. Except for $(2-\epsilon)$-approx diameter on weighted graphs, all the lower bounds also hold for the worst-case update time of partially dynamic algorithms. Bounds marked with the asterisk (*) hold when the update time is amortized over *only* $O(n)$ *updates*. It is not clear how to get this parameter for an update time amortized over any polynomially many updates like all our bounds.

Problems	$p(m,n)$	$u(m,n)$	$q(m,n)$	Conj.	Reference	Remark
Langerman's	$n^{1+\delta-\epsilon}$	$n^{\delta/2-\epsilon}$	$n^{\delta/2-\epsilon}$	multi phase	[Pat10]	If $\tau \geq n^{\delta-\epsilon}$, $\delta \in (0,1)$
	poly($\mathbf{n}$)	$\mathbf{n^{1/2-\epsilon}}$	$\mathbf{n^{1/2-\epsilon}}$	OMv	Corollary 3.17	
Pagh's over k sets in a universe $[n]$	$k^{1+\delta-\epsilon}$	$k^{\delta-\epsilon}$	$k^{\delta-\epsilon}$	Triangle	[AVW14]	$n \leq k \leq n^2$, Choose $\delta \in (1/3, 0.41)$ depending on the conjecture
	$k^{3/2-\epsilon}$	$k^{1/2-\epsilon}$	$k^{1/2-\epsilon}$	BMM	[AVW14]	$n \leq k \leq n^2$
	$k^{1/3-\epsilon}$	$k^{1/3-\epsilon}$	$k^{1/3-\epsilon}$	3SUM	[AVW14]	$k = \Theta(n^{1.5+\epsilon'})$
	poly($\mathbf{n}$)	$\mathbf{n^{1-\epsilon}}$	$\mathbf{k^{1-\epsilon}}$	OMv	Corollary 3.15	$k \leq poly(n)$
Erickson over a matrix of size $n \times n$	$n^{2+\delta-\epsilon}$	$n^{\delta-\epsilon}$	$n^{\delta-\epsilon}$	multi phase	[Pat10]	If $\tau \geq n^{\delta-\epsilon}$, $\delta \in (0,1)$
	poly($\mathbf{n}$)	$\mathbf{n^{1-\epsilon}}$	$\mathbf{n^{2-\epsilon}}$	OMv	Corollary 3.19	
Multiphase over k sets in a universe $[n]$	$n^{4-\epsilon}$	$\tau \geq n^{1/2-\epsilon}$		3SUM	[Pat10]	$k = \Theta(n^{5/2})$
	poly($\mathbf{k}, \mathbf{n}$)	$\tau \geq \mathbf{\min\{k,n\}^{1-\epsilon}}$		OMv	Corollary 5.3	

Table 6: Amortized lower bounds for fully dynamic non-graph problems. Bounds which are not subsumed are highlighted. Each row states that there is no algorithm achieving stated preprocessing time, update time, and query time *simultaneously*, unless the corresponding conjecture fails. The lower bounds based on the multiphase problem are only for worst case time and their parameters are implicit from [Pat10].

Lemma 3.1. *Given a partially dynamic algorithm $\mathcal{A}$ for st-SubConn, one can solve* 1-uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(m)$ edges and $\Theta(\sqrt{m})$ vertices, and then performing $O(\sqrt{m})$ turn-on operations (or $O(\sqrt{m})$ turn-off operations) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. We only prove the decremental case, because the incremental case is symmetric. Given M, we construct the bipartite graph G_M and add to it vertices s, t, and edges $(t, l_i), (r_j, s)$ for all $r_j \in R, l_i \in L$. Thus, the total number of edges is at most $n_1 n_2 + n_1 + n_2 = O(m)$. In the beginning, every vertex is "turned on", i.e., included in the set S of the st-SubConn algorithm

Once u and v arrive, we turn off l_i iff $u_i = 0$ and turn off r_j iff $v_j = 0$. We have $u^\top M v = 1$ iff s is connected to t. In total, we need to do at most $n_1 + n_2 = O(\sqrt{m})$ updates and 1 query. □

Distinguishing between 3 and 5 for s-t distance (st-SP (3 vs. 5))

Lemma 3.2. *Given a partially dynamic algorithm $\mathcal{A}$ for (α,β)-approximate st-SP with $3\alpha + \beta < 5$, one can solve* 1-uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(m)$ edges and $\Theta(\sqrt{m})$ vertices, and then making $O(\sqrt{m})$ insertions (or $O(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. We only prove the decremental case, because the incremental case is symmetric. Given M, we construct the bipartite graph G_M and add to it vertices s, t, and edges $(t, l_i), (r_j, s)$ for all $r_j \in R, l_i \in L$. Thus, the total number of edges is at most $n_1 n_2 + n_1 + n_2 = O(m)$.

Once u and v arrive, we delete (t, l_i) iff $u_i = 0$ and delete (r_j, s) iff $v_j = 0$. If $u^\top M v = 1$, then $d(s,t) = 3$, otherwise $d(s,t) \geq 5$. In total, we need to do at most $n_1 + n_2 = O(\sqrt{m})$ updates and 1 query. □

Triangle Detection and Triangle Detection at vertex s

Lemma 3.3. *Given a partially dynamic algorithm $\mathcal{A}$ for (s-)triangle detection, one can solve* 1-uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(m)$ edges and $\Theta(\sqrt{m})$ vertices, and then making $O(\sqrt{m})$ insertions (or $O(\sqrt{m})$ deletions) and 1 query, where m is such that $n_1 = n_2 = \sqrt{m}$.*

Proof. We only prove the decremental case. Given M, we construct the bipartite graph G_M and add to it a vertex s and edges $(s, l_i), (r_j, s)$ for all $r_j \in R, l_i \in L$. Thus, the total number of edges is at most $n_1 n_2 + n_1 + n_2 = O(m)$.

Once u and v arrive, we delete (s, l_i) iff $u_i = 0$ and delete (r_j, s) iff $v_j = 0$. We have $u^\top M v = 1$ iff there is a triangle in a graph iff there is a triangle incident to s. In total, we need to do $n_1 + n_2 = O(\sqrt{m})$ updates and 1 query. ☐

Corollary 3.4. *For any n and $m \leq n^2$, unless Conjecture 1.1 fails, there is no partially dynamic algorithm $\mathcal{A}$ for the problems in the list below for graphs with n vertices and m edges with preprocessing time $p(m) = poly(m)$, worst update time $u(m) = \tilde{o}(\sqrt{m})$ and query time $q(m) = \tilde{o}(m)$ that has an error probability of at most $1/3$. Moreover, this is true also for fully dynamic algorithms with amortized update time. The problems are:*

- *st-SubConn*
- *st-SP (3 vs. 5)*
- *(s-)triangle detection*

Proof. Suppose there is such a partially dynamic algorithm $\mathcal{A}$. That is, on a graph with n_0 vertices and $m_0 = O(n_0^2)$ edges, $\mathcal{A}$ has worst-case update time $u(m_0) = \tilde{o}(\sqrt{m_0})$ and query time $q(m_0) = \tilde{o}(m_0)$. We will construct an algorithm $\mathcal{B}$ for 1-uMv with parameters n_1 and n_2 which contradicts Conjecture 1.1. Using Lemmas 3.1 to 3.3, by running $\mathcal{A}$ on a graph with $n_0 = \Theta(\sqrt{m})$ vertices and $m_0 = O(m)$ edges where m is such that $n_1 = n_2 = \sqrt{m}$ (note that, indeed, $m_0 = O(n_0^2)$), $\mathcal{B}$ has preprocessing time poly(m) and computation time $O(\sqrt{m}u(m) + q(m)) = \tilde{o}(m)$ which contradicts Conjecture 1.1 by Corollary 2.8.

Next, suppose that $\mathcal{A}$ is fully dynamic and only guarantees an amortized bound. We will construct an algorithm $\mathcal{C}$ for 1-OuMv with parameters n_1, n_2, and n_3 which again contradicts Conjecture 1.1 by running $\mathcal{A}$ on the same graph as for solving 1-uMv while the number of updates and queries needed is multiplied by $O(n_3)$. This can be done because $\mathcal{A}$ is fully dynamic. So, for each vector pair (u, v) for $\mathcal{C}$, if $\mathcal{A}$ makes k updates to the graph, then $\mathcal{A}$ can undo these updates with another k updates so that the updated graph is the same as right after the preprocessing. Recall that, by the notion of amortization, if there are t updates, then $\mathcal{A}$ takes $O((t + m_0) \cdot u(m_0))$ time where m_0 is a number of edges ever appearing in the graph. By choosing $n_3 = \sqrt{m}$, we have that $\mathcal{C}$ has preprocessing time poly(m) and computation time $O((\sqrt{m}n_3 + m)u(m) + n_3 q(m)) = \tilde{o}(m\sqrt{m})$ which contradicts Conjecture 1.1 by Theorem 2.7. ☐

Note that st-SubConn is reducible to the following problems in a way that preserves the parameters of the lower bounds (see [AVW14] for the first three reductions):

- st-Reach,
- Strong connectivity,
- Bipartite perfect matching,
- Size of bipartite maximum matching (and, hence, vertex cover),

- st-maxflow in undirected and unit capacity graph (see [Mąd11, Theorem 3.6.1]).

Therefore, these problems have the same lower bound.

3.2 Lower Bounds for Graph Problems with a Trade-off

To show hardness of the problems in this section, we reduce from uMv where $n_1 = m^\delta$ and $n_2 = m^{1-\delta}$ for any constant $\delta \in (0, 1)$. When u and v arrive, we make $m^{1-\delta}$ updates of the dynamic algorithm $\mathcal{A}$ to handle only v. Then make m^δ queries to $\mathcal{A}$ to find the value of uMv. Since the choice of δ is free, we get a trade-off lower bound between update time and query time.

Through out this subsection, $\delta \in (0, 1)$ is any constant.

Single Source Subgraph Connectivity (ss-Subconn)

Lemma 3.5. *Given a partially dynamic algorithm $\mathcal{A}$ for ss-SubConn, after polynomial preprocessing time, one can solve $(\frac{1-\delta}{\delta})$-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $\Theta(m^{1-\delta} + m^\delta)$ nodes and $O(m)$ edges, then making $O(m^{1-\delta})$ insertions (or $O(m^{1-\delta})$ deletions) and $O(m^\delta)$ queries, where m is such that $m^\delta = n_1$ (so $m^{1-\delta} = n_2$).*

Proof. We only prove the decremental case because the incremental case is symmetric. Given M, we construct the bipartite graph G_M, with an additional vertex s and edges (r_j, s) for all $j \leq n_2$. Thus, the total number of edges is $n_1 n_2 + n_2 = O(m)$. In the beginning, every node is turned on. Once u and v arrive, we turn off r_j iff $v_j = 0$. If $u^\top M v = 1$, then s is connected to l_i for some i where $u_i = 1$. Otherwise, s is not connected to l_i for all i where $u_i = 1$. We distinguish these two cases by querying, for every $1 \leq i \leq n_2$, whether s and l_i are connected. In total, we need to do $n_2 = m^{1-\delta}$ updates and $n_1 = m^\delta$ queries. □

Distinguishing between 2 and 4 for distances from s (ss-SP (2 vs. 4))

Lemma 3.6. *Given a partially dynamic algorithm $\mathcal{A}$ for (α, β)-approximate ss-SP with $2\alpha + \beta < 4$, one can solve $(\frac{1-\delta}{\delta})$-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(m)$ edges and $O(m^\delta + m^{1-\delta})$ vertices, and then making $O(m^{1-\delta})$ insertions (or $O(m^{1-\delta})$ deletions) and $O(m^\delta)$ queries, where m is such that $m^\delta = n_1$ (so $m^{1-\delta} = n_2$).*

Proof. We only prove the decremental case. Given M, we construct the bipartite graph G_M and add to it a vertex s and edges (r_j, s) for all $r_j \in R$. Thus, the total number of edges $n_1 n_2 + n_2 = O(m)$.

Once u and v arrive, we disconnect s from r_j iff $v_j = 0$. We have that if $u^\top M v = 1$, then $d(s, l_i) = 2$ for some i where $u_i = 1$, otherwise $d(s, l_i) \geq 4$ for all i where $u_i = 1$. In total, we need to do $n_2 = O(m^{1-\delta})$ updates and $n_1 = O(m^\delta)$ queries. □

Vertex-color Distance Oracle (1 vs. 3) Vertex-color distance oracles are studied in [HLW+11, Che12]. Given a graph G, one can change the color of any vertex and must handle the query that, for any vertex u and color c, return $d(u, c)$ the distance from u to the nearest vertex with color c. Chechik [Che12] showed, for any integer $k \geq 2$, a dynamic oracle with update time $\tilde{O}(n^{1/k})$ and query time $O(k)$ which $(4k-5)$ approximates the distance. Lacki et al. [LOP+15] extended the result when $k=2$ by handling additional operations and used it as a subroutine to get an algorithm for dynamic $(6+\epsilon)$-approximate Steiner tree.

Lemma 3.7. *Given a dynamic algorithm $\mathcal{A}$ for (α,β)-approximate vertex-color distance oracle with $\alpha+\beta<3$, one can solve $(\frac{1-\delta}{\delta})$-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(m)$ edges and $O(m^\delta+m^{1-\delta})$ vertices, and then making $O(m^{1-\delta})$ vertex-color changes and $O(m^\delta)$ queries, where m is such that $m^\delta=n_1$ (so $m^{1-\delta}=n_2$).*

Proof. Given M, we construct the bipartite graph G_M. Set the color of l_i for all $l_i \in L$ to c. The colors of the other vertices (i.e., those in R) are set to $c' \neq c$.

Once u and v arrive, we set the color of l_i to $c' \neq c$ iff $u_i=0$. If $u^\top Mv=1$, then $d(r_j,c)=1$ for some j where $v_j=1$. Otherwise, $d(r_j,c)\geq 3$ for all j where $v_j=1$. In total, we need to do $n_1=O(m^\delta)$ updates and $n_2=O(m^{1-\delta})$ queries. □

Corollary 3.8. *For any n, $m\leq O(\min\{n^{1/\delta},n^{1/(1-\delta)}\})$, and constant $\delta\in(0,1)$, unless Conjecture 1.1 fails, there is no partially dynamic algorithm $\mathcal{A}$ for the problems in the list below for graphs with n vertices and at most m edges with preprocessing time $p(m)=poly(m)$, worst-case update time $u(m)=\tilde{o}(m^\delta)$, and query time $q(m)=\tilde{o}(m^{1-\delta})$ that has an error probability of at most $1/3$. Moreover, this is true also for fully dynamic algorithms with amortized update time. The problems are:*

- *ss-SubConn*
- *ss-SP (2 vs. 4)*
- *Vertex-color Distance Oracle (1 vs. 3)*

Proof. Suppose there is such a partially dynamic algorithm $\mathcal{A}$. That is, on a graph with n_0 vertices and $m_0=O(\min\{n_0^{1/\delta},n_0^{1/(1-\delta)}\})$ edges, $\mathcal{A}$ has worst-case update time $u(m_0)=\tilde{o}(m_0^\delta)$ and query time $q(m_0)=\tilde{o}(m_0^{1-\delta})$. We will give an algorithm $\mathcal{B}$ for $(\frac{1-\delta}{\delta})$-uMv with parameters n_1 and n_2 which contradicts Conjecture 1.1. Using Lemmas 3.5 to 3.7, by running $\mathcal{A}$ on a graph with $n_0=\Theta(m^\delta+m^{1-\delta})$ vertices and $m_0=O(m)$ edges, where m is such that $m^\delta=n_1$ and so $m^{1-\delta}=n_2$ (note that, indeed, $m_0=O(\min\{n_0^{1/\delta},n_0^{1/(1-\delta)}\})$), $\mathcal{B}$ has preprocessing time poly(m) and computation time $O(m^{1-\delta}u(m)+m^\delta q(m))=\tilde{o}(m)$ which contradicts Conjecture 1.1 by Corollary 2.8.

The argument for fully dynamic algorithm is similar as in the proof of Corollary 3.4. □

These results show that improving the approximation ratio of 3 of vertex-color distance oracle will cost too much; i.e., we will need $\Omega(n)$ update or query time in a dense graph assuming Conjecture 1.1 by setting $\delta=1/2$ and $m=n^2$. In particular, one cannot improve the approximation ratio of dynamic Steiner tree with sub-linear update time by improving the approximation ratio of vertex-color distance oracle.

3.3 Lower Bounds for Graph Problems with other Parameters

$(3-\epsilon)$-approximate s-t Shortest Path ($(3-\epsilon)$st-SP) By subdividing edges, we can get a weaker lower bound, but better approximation factor, for distance related problems.

Lemma 3.9. *Given a partially dynamic algorithm $\mathcal{A}$ for $(3-\epsilon)$-approximate st-SP, one can solve* uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(n)$ vertices and then making $O(\sqrt{n})$ insertions (or $O(\sqrt{n})$ deletions) and 1 query, where n is such that $n_1 = n_2 = \sqrt{n}$.*

Proof. We only prove the decremental case, because the incremental case is symmetric. Given M, we construct the bipartite graph G_M and add to it vertices s, t, and edges $(t, l_i), (r_j, s)$ for all $r_j \in R, l_i \in L$. Furthermore we replace each edge e in G_M by a path P_e of length $\frac{4}{\epsilon}$. Thus, the total number of vertices is at most $O(n_1 n_2/\epsilon) = O(n/\epsilon)$.

Once u and v arrive, we delete (t, l_i) iff $u_i = 0$ and delete (r_j, s) iff $v_j = 0$. If $u^\top M v = 1$, then $d(s,t) = 2 + \frac{4}{\epsilon}$, otherwise $d(s,t) \geq 2 + 3 \cdot \frac{4}{\epsilon}$. One can verify that $\frac{2+3\cdot\frac{4}{\epsilon}}{2+\frac{4}{\epsilon}} > 3 - \epsilon$ for any $\epsilon > 0$. In total, we need to do $n_1 + n_2 = O(\sqrt{n})$ updates and 1 query. □

Corollary 3.10. *Unless Conjecture 1.1 fails, there is no partially dynamic algorithm for $(3-\epsilon)$ st-SP on a graph with n vertices with preprocessing time $p(n) = poly(n)$, worst-case update time $u(n) = \tilde{\tilde{o}}(\sqrt{n})$, and query time $q(n) = \tilde{\tilde{o}}(n)$ that has an error probability of at most $1/3$. Moreover, this is true also for fully dynamic algorithm with amortized update time.*

***d*-failure Connectivity** d-failure connectivity problem is a "1-batch-update" version of dynamic subgraph connectivity. The update, for turning off up to d vertices, comes in one batch. Then one can query whether two nodes s and t are connected. We want the update time $u(d)$ for a batch of size d and the query time $q(d)$ to depend mainly on d.

Lemma 3.11. *Let $\delta \in (0, 1/2]$ be a fixed constant. Given an algorithm $\mathcal{A}$ for d-failure connectivity, one can solve $(\frac{\delta}{1-\delta})$-*uMv *with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(m)$ edges and $\Theta(m^{1-\delta})$ vertices, then making 1 batch of $O(m^\delta)$ updates and $O(m^{1-\delta})$ queries, where m is such that $m^{1-\delta} = n_1$ (so $m^\delta = n_2$).*

Proof. Given M, we construct the bipartite graph G_M and add to it a vertex s and edges (r_j, s) for all $r_j \in R$. There are $n_1 n_2 = O(m)$ edges and $n_1 + n_2 = \Theta(m^\delta + m^{1-\delta}) = \Theta(m^{1-\delta})$ vertices.

Once u and v arrive, we turn off all r_j where $v_j = 0$ in one batch of $O(m^\delta)$ updates. $u^\top M v = 1$ iff, for some i, $u_i = 1$, s is connected to u_i. This can be checked using at most $m^{1-\delta}$ queries. □

Corollary 3.12. *For any n, $m = O(n^{1/(1-\delta)})$, and constant $\delta \in (0, 1/2]$, unless Conjecture 1.1 fails, there is no algorithm for d-failure connectivity for a graph with n vertices and at most m edges with preprocessing time $p(n) = poly(n)$, update time $u(d) = \tilde{\tilde{o}}(d^{1/\delta})$, and query time $q(d) = \tilde{\tilde{o}}(d)$ that has an error probability of at most $1/3$, when $d = m^\delta$.*

Proof. Suppose there is such an algorithm $\mathcal{A}$. By Lemma 3.11, we can solve $(\frac{\delta}{1-\delta})$-uMv with parameters $n_1 = m^{1-\delta}$ and $n_2 = m^\delta$ by running $\mathcal{A}$ in time $O(u(m^\delta) + m^{1-\delta} q(m^\delta)) = \tilde{\tilde{o}}(m)$. This contradicts Conjecture 1.1 by Corollary 2.8. □

Corollary 3.12 implies that Duan and Pettie's result [DP10] with preprocessing time $\tilde{O}(d^{1-2/c} m n^{1/c-1/(c\log(2d))})$, update time $\tilde{O}(d^{2c+4})$ and query time $O(d)$, for any integer

$c \geq 1$, is tight in the sense that we cannot improve the query time significantly as long as we want to have update time polynomial in d (because we can choose δ to be any possible constant close to zero). However, improving the update time does not contradict the OMv conjecture.

With the same argument we also get the following lower bound.

Corollary 3.13. *For any n, $m = \Theta(n^{1/(1-\delta)})$, and constant $\delta \in (0, 1/2]$, unless Conjecture 1.1 fails, there is no algorithm for d-failure connectivity for a graph with n vertices and at most m edges with preprocessing time $p(n) = poly(n)$, update time $u(n,d) = \tilde{\tilde{o}}(dn)$, and query time $q(d) = \tilde{\tilde{o}}(d)$ that has an error probability of at most $1/3$, when $d = m^\delta$.*

3.4 Lower Bounds for Non-graph Problems

In this section, we mimic the reductions from [Pat10] to show hardness of non-graph problems. However, our results imply amortized lower bounds while the results in [Pat10] are for worst-case lower bounds.

Pagh's problem.

Lemma 3.14. *Given an algorithm $\mathcal{A}$ for Pagh's problem (cf. Table 3), for any constant $\gamma > 0$, one can solve γ-uMv with parameters k and n by running the preprocessing step of $\mathcal{A}$ on k initial sets in the universe $[n]$ using $\text{poly}(k, n)$ preprocessing time, k updates (adding k more sets into the family) and n queries.*

Proof. Given a matrix M, let $\bar{M}$ be the matrix defined by $\bar{M}_{i,j} = 1 - M_{i,j}$ for all i and j. Let M_i be the i-th row of M and treat it as a subset of $[n]$ i.e., $j \in M_i$ iff $M_{i,j} = 1$. We similarly treat the i-th row $\bar{M}_i$ of $\bar{M}$ as a set. Note that $u^\top M e_j = 1$ iff $j \in \bigcup_{i:u_i=1} M_i$ iff $j \notin \bigcap_{i:u_i=1} \bar{M}_i$. Thus, at the beginning, we compute $\bar{M}_i$ for each $i \leq k$. Once u and v arrive, we compute $\bigcap_{u_i=1} \bar{M}_i$ using k updates. There is a j with $v_j = 1$ such that $j \notin \bigcap_{u_i=1} \bar{M}_i$ iff $u^\top M v = 1$. We need n queries to check if such a j exists. □

Corollary 3.15. *For any constant $\gamma > 0$, Conjecture 1.1 implies that there is no dynamic algorithm $\mathcal{A}$ for Pagh's problem maintaining k sets over the universe $[n]$ where $k = n^\gamma$, with $\text{poly}(k, n)$ preprocessing time, $u(k, n) = \tilde{\tilde{o}}(n)$ amortized update time, and $q(k, n) = \tilde{\tilde{o}}(k)$ query time that has an error probability of at most $1/3$.*

Proof. Recall that, by the notion of amortization, if $\mathcal{A}$ initially maintains k sets, then the total update time of $\mathcal{A}$ is $O((t + k) \cdot u(t + k, n))$. By Lemma 3.14, we can solve uMv with parameters k and n by running $\mathcal{A}$ and making k updates and n queries in time $O(2k \cdot u(2k, n) + n \cdot q(2k, n)) = \tilde{\tilde{o}}(nk)$. This contradicts Conjecture 1.1 by Corollary 2.8. □

Langerman's Zero Prefix Sum problem

Lemma 3.16. *Given an algorithm $\mathcal{A}$ for Langerman's problem (cf. Table 3), one can solve 1-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on an array of size $O(n)$, and then making $O(\sqrt{n})$ updates and $O(\sqrt{n})$ queries where n is such that $n_1 = n_2 = \sqrt{n}$*

$2(n_2-j)+1$	0	1	1	2	0	$\cdots$	2	0	$-2n_2$
	0	1	1	2	0	$\cdots$	1	1	$-2n_2$
	$-2n_2$	2	0	2	0	$\cdots$	1	1	0
	$-2n_2$	1	1	2	0	$\cdots$	2	0	0
	0	2	0	1	1	$\cdots$	2	0	$-2n_2$

Figure 3: The array in the reduction from uMv to Langerman's zero prefix sum problem

Proof. Given a matrix M, we construct an array R of size $1+n_1\cdot(2n_2+2)=O(n)$. For convenience, we will imagine that R is arranged as a two-dimensional array $\{R_{i,j}\}_{i\in[n_1],j\in[2n_2+2]}$ ($R_{i,2n_2+2}$ is before $R_{i+1,1}$) with one additional entry R_0 at the beginning.

For all $i\le n_1$, we set $R_{i,1}=0$ and $R_{i,2n_2+2}=-2n_2$. For each entry of $M_{i,j}$, if $M_{i,j}=1$, then we set $R_{i,2j}=1$ and $R_{i,2j+1}=1$. If $M_{i,j}=0$, then we set $R_{i,2j}=2$ and $R_{i,2j+1}=0$. Note that $\sum_j R_{i,j}=0$ for all i, so the rows of R are "independent".

Once u and v arrive, we swap the values of $R_{i,1}$ and $R_{i,2n_2+2}$ for all i where $u_i=1$ by setting $R_{i,1}=-2n_2$ and $R_{i,2n_2+2}=0$. For each j where $v_j=1$, we set $R_0=2(n_2-j)+1$ and query for a zero prefix sum. See Figure 3. In total, we need to do $O(n_1+n_2)=O(\sqrt{n})$ updates and $O(n_2)=O(\sqrt{n})$ queries.

To show correctness, we claim that a zero prefix sum exists iff $u^\top Me_j=1$ where e_j has 1 at only the j-th entry. First, the prefix sums cannot reach zero at row i of R if $u_i=0$, because in that row i, each number is positive except $R_{i,2n_2+2}=-2n_2$ which just resets the sum within the row to zero. Second, for each row i where $u_i=1$, the prefix sum from R_0 to $R_{i,1}$ is $-2j+1$. Then each pair of entries in the row increments the sum by 2. The prefix sum reaches zero iff $M_{i,j}=1$. If $M_{i,j}=0$, then $R_{i,2j}=2$ so the prefix sums to $R_{i,2j-1}$ and $R_{i,2j}$ are -1 and 1, respectively. The prefix sum then stays positive until row i finishes. If $M_{i,j}=1$, then $R_{i,2j}=1$ and the prefix sum from R_0 to $R_{i,2j}$ is exactly 0. $\square$

Corollary 3.17. *Conjecture 1.1 implies that there is no algorithm for Langerman's problem on an array of size n with preprocessing time $p(n)=poly(n)$, amortized update time $u(n)=\tilde{\tilde{o}}(\sqrt{n})$, and query time $q(m)=\tilde{\tilde{o}}(\sqrt{n})$ that has an error probability of at most $1/3$.*

Proof. Suppose there is such an algorithm $\mathcal{A}$. By Lemma 3.16 and by "resetting" the array when the new vector pair arrives, one can solve OuMv with parameters $n_1=\sqrt{n}$, $n_2=\sqrt{n}$, and $n_3=\sqrt{n}$ in time $O(n\cdot u(n)+n\cdot q(n))=\tilde{\tilde{o}}(n\sqrt{n})$. This contradicts Conjecture 1.1 by Theorem 2.7. Note that by the choice of $n_3=\sqrt{n}$ as the third parameter of OuMv we perform $\Theta(n)$ updates to $\mathcal{A}$, which allows use to use the amortized update time $u(n)$ of $\mathcal{A}$ in this argument. $\square$

Erickson's problem

Lemma 3.18. *Given an algorithm $\mathcal{A}$ for Erickson's problem (cf. Table 3), one can solve 1-OuMv with parameters n_1,n_2, and n_3 by running the preprocessing step of $\mathcal{A}$ on a matrix of size $n\times n$ and then making $O(n\cdot n_3)$ updates and n_3 queries, where n is such that $n_1=n_2=n$.*

Proof. Given a Boolean matrix M, $\mathcal{A}$ runs on the same matrix but treats it as an integer matrix. Once u^t and v^t arrive, we increment the row i iff $u^t_i=1$ and increment the column j

iff $v_j^t = 1$. Before u^{t+1} and v^{t+1} arrive, we increment the remaining rows i where $u_i^t = 0$ and remaining column where $v_j^t = 0$. Therefore, we have that $(u^t)^\top M v^t = 1$ iff the maximum value in the matrix is $2t + 1$. □

Corollary 3.19. *Unless Conjecture 1.1 fails, there is no algorithm for Erickson's problem on a matrix of size $n \times n$ with preprocessing time $p(n) = poly(n)$, amortized update time $u(n) = \tilde{\tilde{o}}(n)$, and query time $q(n) = \tilde{\tilde{o}}(n^2)$ that has an error probability of at most $1/3$.*

Proof. Otherwise, we can solve 1-OuMv with parameters $n_1 = n$, $n_2 = n$, and n_3 using polynomial preprocessing time and $\tilde{\tilde{o}}(n_1 n_2 n_3)$ computation time which contradicts Conjecture 1.1 by Theorem 2.7. □

3.5 $(2 - \epsilon)$ Approximate Diameter on Weighted Graphs

We show that, for any $\epsilon > 0$, it is OMv-hard to maintain a $(2-\epsilon)$-approximation of the diameter in a weighted graph under both insertions and deletions with $\tilde{\tilde{o}}(\sqrt{n})$ update time and $\tilde{\tilde{o}}(n)$ query time, even if the edge weights are only 0 and 1. This reduction is inspired by a lower bound in distributed computation [FHW12]. It is different from previous reductions in that we can show the hardness for this problem only in the fully dynamic setting and not in the partially dynamic setting.

Lemma 3.20. *For any $\gamma > 0$, Given a fully dynamic algorithm $\mathcal{A}$ for $(2-\epsilon)$-approximate diameter on a $\{0,1\}$-weighted undirected graph, one can solve γ-uMv with parameters n_1 and n_2 by running the preprocessing step of $\mathcal{A}$ on a graph with $O(n_1\sqrt{n_2})$ vertices, and then making $n_2 + O(n_1\sqrt{n_2})$ updates and n_1 queries to $\mathcal{A}$.*

Proof. First, let us define an undirected graph H^v, called *vector graph*, from a vector v of size n_2. A vector graph $H^v = (B^v \cup C^v)$ has two halves, called *upper* and *lower halves* denoted by B^v and C^v respectively. B^v and C^v are both cliques of size $\sqrt{n_2}$. Let $\{b_x^v\}_{1\le x\le\sqrt{n_2}}$ and $\{c_x^v\}_{1\le x\le\sqrt{n_2}}$ be the vertices of B^v and C^v respectively. We include an edge (b_x^v, c_y^v) iff $v_{(x-1)\sqrt{n_2}+y} = 0$. The weight of all edges in H^v is 1.

Given a matrix M of size $n_1 \times n_2$, let M_i be the i-th row of M. We construct a vector graph H^{M_i} for each $1 \le i \le n_1$, and another vector graph H^v where v is the zero vector. There are two special vertices a and z. Connect a to all vertices in H^v with weight one. Connect z to a and all vertices in H^{M_i} with weight zero, for every $1 \le i \le n_1$.

Once (u, v) arrives, we update H^v to be the vector graph of v in n_2 updates. Then we work in stages i, for each i where $u_i = 1$. Before going to the next stage, we undo all the updates. In stage i, 1) disconnect z from each vertex in H^{M_i}, 2) connect a to each vertex in H^{M_i} with weight one, and 3) add 0-weight *matching edges* $(b_x^{M_i}, b_x^v)$ and $(c_y^{M_i}, c_y^v)$ for all $x, y \le \sqrt{n_2}$. All three steps need $O(\sqrt{n_2})$ updates. Let $G(M_i, v)$ denote the resulting graph (see Figure 4 for example). We query the diameter of $G(M_i, v)$ and will use the result to solve γ-uMv. After finishing all stages, there are $n_2 + O(n_1\sqrt{n_2})$ updates and n_1 queries in total.

If there is some stage i where the diameter of the graph $G(M_i, v)$ is 2, then report $u^\top M v = 1$. Otherwise report 0. The following claim justifies this answer.

Claim 3.21. *The diameter of $G(M_i, v)$ is 1 if $M_i^\top v = 0$. Otherwise, the diameter is 2.*

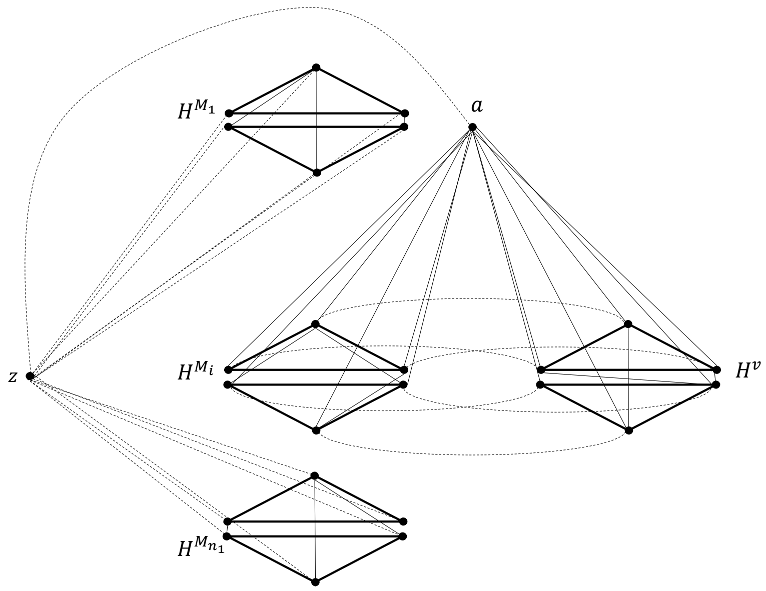

Figure 4: An example of $G(M_i, v)$ from the proof of Lemma 3.20. Dashed lines are edges of weight zero. Other lines are edges of weight one.

Proof. First, every edge incident to z has weight 0. So we can treat all the adjacent vertices of z, which are exactly a and those in $H^{M_{i'}}$ where $i' \neq i$, as a single vertex. Therefore, we just have to analyze the distance among vertices in H^{M_i}, H^v and the vertex a.

Note that a is connected to all vertices in H^{M_i} and H^v by 1-weight edges. The distance between vertices among the upper halves B^{M_i} and B^v is at most 1, because B^{M_i} and B^v are cliques and there are matching edges of weight zero. Similarly, for the lower halves C^{M_i} and C^v. Now, we are left with analyzing the distance between a vertex in the upper halves and another vertex in the lower ones. There are two cases.

If $M_i^\top v = 0$, then for each $x, y \leq \sqrt{n_2}$, there is either an edge $(b_x^{M_i}, c_y^{M_i})$ or an edge (b_x^v, c_y^v). So given any $b_x^{M_i}$ and $c_y^{M_i}$, there is either a path $(b_x^{M_i}, b_x^v, c_y^v, c_y^{M_i})$ or a path $(b_x^{M_i}, c_y^{M_i})$ both of weight 1. Since $d(b_x^{M_i}, c_y^{M_i}) = 1$, the distance among $b_x^{M_i}, b_x^v, c_y^{M_i}, c_y^v$ is at most 1. Since this is true for any x, y, the diameter of the graph is 1.

If $M_i^\top v = 1$, then there are some x, y such that neither the edge $(b_x^{M_i}, c_y^{M_i})$ nor the edge (b_x^v, c_y^v) exists. To show that $d(b_x^{M_i}, c_y^{M_i}) \geq 2$, it is enough to show that $d(b_x^{M_i}, c_y^{M_i}) > 1$ because every non-zero edge weight is 1. The set of vertices with distance 1 from $b_x^{M_i}$ includes exactly the neighbors of $b_x^{M_i}$ in H^{M_i} and their "matching" neighbors in H^v, the neighbors of b_x^v in H^v and their "matching" neighbors in H^{M_i}, and the vertex a, which combines z and all vertices in $H^{M_{i'}}$ where $i' \neq i$. But this set does not include $c_y^{M_i}$. Therefore $d(b_x^{M_i}, c_y^{M_i}) \geq 2$ and we are done. □

□

Corollary 3.22. *Assuming Conjecture 1.1, there is no fully dynamic algorithm for $(2-\epsilon)$-approximate diameter on $\{0,1\}$-weighted graphs with n vertices with preprocessing time*

$p(n) = poly(n)$, *amortized update time* $u(n) = \tilde{\tilde{o}}(\sqrt{n})$, *and query time* $q(n) = \tilde{\tilde{o}}(n)$ *that has an error probability of at most* $1/3$.

Proof. Suppose such a fully dynamic algorithm $\mathcal{A}$ exists. By Lemma 3.20 and by "undoing" the operations as in the proof of Corollary 3.4, we can solve 2-OuMv with parameters $n_1 = \sqrt{n}$, $n_2 = n$, and $n_3 = n$ by running $\mathcal{A}$ on a graph with $\Theta(n_1\sqrt{n_2}) = \Theta(n)$ vertices, and then making $O(n_2 + n_1\sqrt{n_2}) \times n_3 = O(n^2)$ updates and $n_1 n_3 = O(n\sqrt{n})$ queries in total. The computation time is $O(n^2 u(n) + n\sqrt{n}q(n)) = \tilde{\tilde{o}}(n^2\sqrt{n})$, contradicting Conjecture 1.1 by Theorem 2.7. Note that we choose $n_3 = n\sqrt{n}$ to make sure that the number of updates is at least the number of edges in the graph, so that we can use the amortized time bound. □

3.6 Densest Subgraph Problem

In this section, we show a non-trivial reduction from 1-uMv to the densest subgraph problem and hence show the hardness of this problem.

Theorem 3.23. *Given a partially dynamic* $\mathcal{A}$ *for maintaining the density of the densest subgraph, one can solve 1-uMv with parameters* $n_1 = n$ *and* $n_2 = n$ *by running the preprocessing step of* $\mathcal{A}$ *on a graph with* $\Theta(n^3)$ *vertices, and then making* $\Theta(n)$ *updates and 1 query.*

Problem definition We are given an undirected input graph $G = (V, E)$ with vertices V and edges E. For every subset of vertices $S \subseteq V$, let $G(S) = (S, E(S))$ denote the subgraph of G induced by the vertices in S, i.e., we have $E(S) = \{(u, v) \in E \mid u, v \in S\}$. The *density* of any subset of vertices $S \subseteq V$ is defined as $\rho(S) = |E(S)|/|S|$. For the reduction described in the following let M be a Boolean matrix of size $n \times n$ and set $k = 6n$.

Preprocessing. We construct the graph G as follows:

- *Bit graphs for* M*.* For each bit $m_{i,j}$ of M, construct a graph $B_{i,j}$ consisting of k vertices. There are two special vertices in $B_{i,j}$, called *special vertex 1* and *special vertex 2*. If the bit $m_{i,j}$ is set, connect the nodes in $B_{i,j}$ by a path of $k - 1$ edges in $B_{i,j}$ from special vertex 1 to special vertex 2. If the bit $m_{i,j}$ is not set, insert no edges into $B_{i,j}$.

- *Row graph for M.* For each row i of M, construct a graph R_i consisting of 3 vertices. One of these vertices is special. Add an edge from the special vertex of R_i to special vertex 1 of $B_{i,j}$ for all $1 \leq j \leq n$.

- *Column graph for* M*.* For each column j of M, construct a graph C_j consisting of 3 vertices. One of these vertices is special. Add an edge from the special vertex of C_j to special vertex 2 of $B_{i,j}$ for all $1 \leq i \leq n$.

Observe that G has $O(n^3)$ vertices.

Revealing u and v We execute the following $O(n)$ edge operations and one query.

- For each i where $u_i = 1$, turn the row graph R_i into a triangle by inserting $O(n)$ edges.

- For each j where $v_j = 1$, turn the column graph C_i into a triangle by inserting $O(n)$ edges.
- Then ask for the size of the densest subgraph.

This describes the reduction of Theorem 3.23. Note that we only need a partially dynamic algorithm. Before proving the correctness of this reduction below, we observe the following easy lemma.

Lemma 3.24. *For all numbers a, b, c, d, and r we have:*

1. *If $\frac{a}{b} \geq r$ and $\frac{c}{d} \geq r$, then $\frac{a+c}{b+d} \geq r$.*
2. *If $\frac{a}{b} \geq r$ and $\frac{c}{d} \leq r$, then $\frac{a-c}{b-d} \geq r$.*

Theorem 3.25. *There exists a subset $S \subseteq V$ with density $\rho(S) \geq \frac{k+7}{k+6}$ if and only if $u^\top M v = 1$.*

Proof. Assume first that $u^\top M v = 1$. So there are indices i and j such that $u_i M_{i,j} v_j = 1$. Consider the subgraph consisting of the union of R_i, $B_{i,j}$ and C_j. It consists of $k+6$ vertices and $k+7$ edges, i.e., it has density $\frac{k+7}{k+6}$.

Now assume that $u^\top M v = 0$ and let $S \subseteq V$. To show that $\rho(S) \leq \frac{k+7}{k+6}$ we will make the following assumptions:

(1) For every i and j, either the full bit graph $B_{i,j}$ together with two edges leaving the bit graph is contained in S or no node of $B_{i,j}$ is contained in S.

(2) For every row i of a set bit (i.e., where $u_i = 1$), either the full row graph R_i is contained in $G(S)$ or no node of R_i is contained in $G(S)$

(3) For every row i of an unset bit (i.e., where $u_i = 0$), either the special node of the row graph R_i is contained in S or no node of R_i is contained in $G(S)$

(4) For every column j of a set bit (i.e., where $v_j = 1$), either the full column graph R_j is contained in $G(S)$ or no node of C_j is contained in $G(S)$

(5) For every column j of an unset bit (i.e., where $v_j = 0$), either the special node of the column graph C_j is contained in S or no node of C_j is contained in $G(S)$

These assumptions can be made without loss of generality as we argue in the following.

(1) Suppose S contains some subset U of nodes of $B_{i,j}$ and either U does not contain all nodes of $B_{i,j}$ or one of the special nodes of $B_{i,j}$ is not contained in S. Then by removing U from S we remove some $q \leq k$ nodes and at most q edges from $G(S)$. Thus, we are removing a piece of density at most 1. If $\rho(S) \geq \frac{k+7}{k+6} \geq 1$, then removing U from S will not decrease $\rho(S)$ by Part 2 of Lemma 3.24 (using $r = \frac{k+7}{k+6}$, $\rho(S) = \frac{a}{b}$, $c \leq q$, and $d = q$). Therefore we may assume without loss of generality that S does not contain U.

(2) If only one of the nodes of R_i is contained in S, then by adding the two other nodes we add 2 nodes and 3 edges to $G(S)$. As $\frac{3}{2} > \frac{k+7}{k+6}$, doing so will not decrease the density of S to below $\frac{k+7}{k+6}$ by Part 1 of Lemma 3.24 (using $r = \frac{k+7}{k+6}$, $\rho(S) = \frac{a}{b}$, $c = 3$, and $d = 2$) and thus we may assume without loss of generality that all the nodes of R_i are contained in S.

Similarly, if only two of the nodes of R_i are contained in S, then by adding the third node we add 1 node and 2 edges to $G(S)$. Again, by Lemma 3.24, we may assume without loss of generality that all three nodes of R_i are contained in S.

(3) There are no edges incident to the non-special nodes of R_i. By removing the non-special nodes of R_i from S we only increase $\rho(S)$. Thus, we may assume without loss of generality, that only the special node is contained in S.

(4) and (5) follow the same arguments as (2) and (3).

Using these assumptions we conclude that $G(S)$ has the following structure: it contains some full bit graphs (i.e., paths) of set bits, each with two outgoing edges, some full row or column graphs (i.e., triangles) of set bits, and some special nodes of row or column graphs of unset bits. In the rest of this proof we will use the following notation: x denotes the number of bit graphs contained in S, y denotes the number of row or column graphs of set bits contained in S, and z denotes the number of row or column graphs of unset bits contained in C. Thus, $G(S)$ has the density

$$\rho(S) = \frac{3y + (k+1)x}{3y + z + kx} \,.$$

The inequality $\rho(S) < \frac{k+7}{k+6}$, which we want to prove, is now equivalent to

$$6x < (k+7)z + 3y \,.$$

Consider some bit graph contained in $G(S)$. As argued above, this graph has one edge going to a row graph and one edge going to a column graph. As $u^\top M v = 0$ at least one of those edges must go to a row or column graph of an unset bit. In this way we assign at most n bit graphs to every unset row or column bit and it follows that $x \leq nz$. As we have defined $k = 6n$ we obtain

$$6x \leq 6nz = kz < (k+7)z + 3y$$

as desired. □

This complete the proof of Theorem 3.23.

Corollary 3.26. *Unless Conjecture 1.1 fails, there is no partially dynamic algorithm $\mathcal{A}$ for maintaining the density of the densest subgraph on a graph with n vertices with polynomial preprocessing time, worst-case update time $u(n) = \tilde{\tilde{o}}(n^{1/3})$, and query time $q(n) = \tilde{\tilde{o}}(n^{2/3})$ that has an error probability of at most $1/3$. Moreover, this is true also for fully dynamic algorithms with amortized update time.*

Proof. Suppose that such a partially dynamic algorithm $\mathcal{A}$ exists. By Theorem 3.23 and by scaling down the parameter from n to $n^{1/3}$, we can solve 1-uMv with parameters $n^{1/3}$ and $n^{1/3}$, by running $\mathcal{A}$ on a graph with $\Theta(n)$ vertices, in time $O(n^{1/3}u(n) + q(n)) = \tilde{\tilde{o}}(n^{2/3})$ contradicting Conjecture 1.1 by Corollary 2.8.

If $\mathcal{A}$ is fully dynamic, the argument is similar as in the proof of Corollary 3.4. □

Problems	$p(m,n)$	$u(m,n)$	$q(m,n)$	Conj.	Reference	Remark
Bipartite Max Matching	$m^{4/3-\epsilon}$	$m^{4/3-\epsilon}$	$m^{1/3-\epsilon}$	3SUM	[KPP16]	$m = \Theta(n)$; only for incremental case.[14]
	poly	$\mathbf{m^{3/2-\epsilon}}$	$\mathbf{m^{1-\epsilon}}$	OMv	Corollary 4.4	$m = \Theta(n)$
unweighted st-SP	**poly**	$\mathbf{m^{3/2-\epsilon}}$	$\mathbf{m^{1-\epsilon}}$	OMv	Corollary 4.2	$m = O(n^2)$
unweighted ss-SP	**poly**	$\mathbf{m^{3/2-\epsilon}}$	$\mathbf{m^{1-\epsilon}}$	OMv	Corollary 4.2	$m = O(n^2)$
	$(mn)^{1-\epsilon}$	$(mn)^{1-\epsilon}$	$m^{\delta'-\epsilon}$	BMM	[RZ11]	Choose any $\delta' \in (0, 1/2]$, $m = \Theta(n^{1/(1-\delta')})$
	poly	$\mathbf{(mn)^{1-\epsilon}}$	$\mathbf{m^{\delta'-\epsilon}}$	OMv	Corollary 4.8	
unweighted (α, β)ap-SP, $2\alpha + \beta < 4$	$(mn)^{1-\epsilon}$	$(mn)^{1-\epsilon}$	$m^{\delta'-\epsilon}$	BMM	[DHZ00]	
	poly	$\mathbf{(mn)^{1-\epsilon}}$	$\mathbf{m^{\delta'-\epsilon}}$	OMv	Corollary 4.8	
Transitive Closure	$(mn)^{1-\epsilon}$	$(mn)^{1-\epsilon}$	$m^{\delta'-\epsilon}$	BMM	[DHZ00]	
	poly	$\mathbf{(mn)^{1-\epsilon}}$	$\mathbf{m^{\delta'-\epsilon}}$	OMv	Corollary 4.8	

Table 7: Lower bounds for total update time of partially dynamic problems. Bounds which are not subsumed are highlighted. Each row states that there is no algorithm achieving stated preprocessing time, total update time, and query time *simultaneously*, unless the conjecture fails. Lower bounds based on BMM apply to only combinatorial algorithms.

4 Hardness for Total Update Time of Partially Dynamic Problems

Our lower bounds, compared to previously known bounds, for the total update time of partially dynamic problems are summarized in Table 7. Tight results are summarized in Table 8

Given a matrix $M \in \{0,1\}^{n_1 \times n_2}$, we denote a bipartite graph $G_M = ((L, R), E)$ where $L = \{l_1, \dots, l_{n_1}\}$, $R = \{r_1, \dots, r_{n_2}\}$, and $E = \{(r_j, l_i) \mid M_{ij} = 1\}$.

In this section, our proofs again follow two simple steps as in Section 3. First, we show the reductions in lemmas that given a partially dynamic algorithm $\mathcal{A}$ for some problem, one can solve OuMv by running the preprocessing step of $\mathcal{A}$ on some graph and then making some number of updates and queries. Then, we conclude in corollaries that if $\mathcal{A}$ has low total update update time and query time then this contradicts Conjecture 1.1.

In the proofs of the lemmas of this section, we only usually show the reduction from OuMv to the decremental algorithm, because it is symmetric in the incremental setting.

s-*t* Shortest Path (st-SP)

Lemma 4.1. *Given an incremental (respectively decremental) dynamic algorithm $\mathcal{A}$ for st-SP, one can solve 1-OuMv with parameters n_1, n_2, and n_3 by running the preprocessing step of $\mathcal{A}$ on a graph with $\Theta(\sqrt{m})$ vertices and $O(m)$ edges which is initially empty (respectively initially has $\Theta(m)$ edges), and then making $\sqrt{m}$ queries, where m is such that $n_1 = n_2 = n_3 = \sqrt{m}$.*

Proof. Given an input matrix M of 1-OuMv, we construct a bipartite graph G_M, and also two paths P and Q with n_3 vertices each. Let $P = (p_1, p_2, \dots, p_{n_3})$ where $p_1 = s$, and

[14] Kopelowitz, Pettie, and Porat [KPP16] show a higher lower bound for the amortized update time of incremental algorithms. But they allow reverting the insertion which is not allowed in our setting.

Problem	Upper Bounds			Lower Bounds			Problem	Remark
	$p(m,n)$	$u(m,n)$	$q(m,n)$	$p(m,n)$	$u(m,n)$	$q(m,n)$		
dec. unweighted Exact ss-SP	m	mn	1	$poly$	$mn^{1-\epsilon}$	$m^{\delta'-\epsilon}$	dec. unweighted Exact ss-SP	Upper: [ES81]
dec. unweighted $(1+\epsilon)$ ss-SP	$m^{1+o(1)}$	$m^{1+o(1)}$	1					Upper: [HKN14a]
dec. unweighted $(1+\epsilon,0)$ ap-SP	mn	mn	1	$poly$	$mn^{1-\epsilon}$	$m^{\delta'-\epsilon}$	dec. (α,β) ap-SP, $2\alpha+\beta<4$	Upper: [HKN13, RZ12, Ber13]
dec. unweighted $(1+\epsilon,2)$ ap-SP	$n^{5/2}$	$n^{5/2}$	1					Upper: [HKN13]
dec. unweighted $(2+\epsilon,0)$ ap-SP	$n^{5/2}$	$n^{5/2}$	1					
dec. Transitive Closure	mn	mn	1	$poly$	$mn^{1-\epsilon}$	$m^{\delta'-\epsilon}$	dec. transitive closure	Upper: [Lac13]

Table 8: Our tight results along with the matching upper bounds (or better upper bounds when worse approximation ratio is allowed). The polylogarithmic factors are omitted. The lower bounds state that there is no algorithm achieving stated preprocessing time, total update time, and query time *simultaneously*, unless the conjecture fails. The table shows that one cannot improve the approximation factor of decremental $(1+\epsilon)$ ss-SP/$(1+\epsilon,2)$ ap-SP/$(2+\epsilon,0)$ ap-SP without scarifying fast running time. All lower bounds hold, for any $\delta' \in (0,1/2]$, when $m = \Theta(n^{1/(1-\delta')})$. $\epsilon > 0$ is any constant.

$Q = (q_1, q_2, \ldots, q_{n_3})$ where $q_1 = s'$. Add the edge (p_t, l_i) and (q_t, r_j) for all $i \leq n_1, j \leq n_2$ and $t \leq n_3$. There are $\Theta(n_1+n_2+n_3) = \Theta(\sqrt{m})$ vertices and $O(n_1 n_2 + n_3) = O(m)$ edges.

Once u^t and v^t arrive, we disconnect p_t from l_i iff $u^t_i = 0$, and disconnect q_t from r_j iff $v^t_j = 0$. We have that if $(u^t)^\top M v^t = 1$, then $d(s,s') = 2t+1$, otherwise $d(s,s') \geq 2t+2$. This is because, before u^{t+1} and v^{t+1} arrive, we disconnect p_t from l_i for all $i \leq n_1$ and disconnect q_t from r_j for all $j \leq n_2$. So for each t, we need 1 query, and hence $n_3 = \sqrt{m}$ queries in total. □

Corollary 4.2. *For any n and $m = O(n^2)$, Conjecture 1.1 implies that there is no partially dynamic st-SP algorithm $\mathcal{A}$ on a graph with n vertices and at most m edges with polynomial preprocessing time, total update time $\tilde{\tilde{o}}(m^{3/2})$, and query time $\tilde{\tilde{o}}(m)$ that has an error probability of at most $1/3$.*

Proof. Suppose there is such an algorithm $\mathcal{A}$. By Lemma 4.1, we construct an algorithm $\mathcal{B}$ for 1-OuMv with parameters $n_1 = \sqrt{m}$, $n_2 = \sqrt{m}$, and $n_3 = \sqrt{m}$ by running $\mathcal{A}$ on a graph with $n_0 = \Theta(\sqrt{m})$ vertices and $m_0 = O(m)$ edges. Note that $m_0 = O(n_0^2)$. Since $\mathcal{A}$ uses polynomial preprocessing time, total update time $\tilde{\tilde{o}}(m^{3/2})$ and total query time $O(\sqrt{m}q(m)) = \tilde{\tilde{o}}(m^{3/2})$ $\mathcal{B}$ has polynomial preprocessing time and $\tilde{\tilde{o}}(m^{3/2})$ computation time contradicting Conjecture 1.1 by Theorem 2.7. □

Note that, when $m = \Theta(n^2)$, Corollary 4.2 implies that there is no algorithm with $\tilde{\tilde{o}}(n^3)$ total update time and $\tilde{\tilde{o}}(n^2)$ query time. There is a matching upper bound of total update time $O(mn) = O(n^3)$ due to [ES81].

Bipartite Maximum Matching

Lemma 4.3. *Given an incremental (respectively decremental) dynamic algorithm for bipartite maximum matching, one can solve 1-*OuMv *with parameters* n_1*,* n_2*, and* n_3 *by running* $\mathcal{A}$ *on a graph with* $\Theta(m)$ *vertices and* $\Theta(m)$ *edges which is initially empty (respectively initially has* $\Theta(m)$ *edges), and then making* $\sqrt{m}$ *queries, where* m *is such that* $n_1 = n_2 = n_3 = \sqrt{m}$*.*

Proof. Given an input matrix M of 1-OuMv, we perform the following preprocessing. First, we construct a bipartite graph G_M which has $O(n_1n_2) = O(m)$ edges. There are also additional sets of vertices 1) $L' = \{l'_1, \dots, l'_{n_1}\}$ and $R' = \{r'_1, \dots, r'_{n_2}\}$, 2) $X_t = \{x_{t,1}, \dots, x_{t,n_1}\}$ and $X'_t = \{x'_{t,1}, \dots, x'_{t,n_1}\}$ for all $t \leq n_3$, and 3) $Y_t = \{y_{t,1}, \dots, y_{t,n_1}\}$ and $Y'_t = \{y'_{t,1}, \dots, y'_{t,n_1}\}$ for all $t \leq n_3$. These are all vertices in the graph, so there are $\Theta(n_1 + n_2 + n_1n_3 + n_2n_3) = \Theta(m)$ vertices in total. Next, we add edges (l_i, l'_i), (r_j, r'_j), $(x_{t,i}, x'_{t,i})$ and $(y_{t,j}, y'_{t,j})$ for each $i \leq n_1, j \leq n_2, t \leq n_3$. These edges form a perfect matching of size $\Theta(m)$. Finally, we add edges $(x_{t,i}, l_i)$ and $(y_{t,j}, r_j)$ for each $i \leq n_1, j \leq n_2, t \leq n_3$. These $\Theta(m)$-many edges do not change the size of matching. In total, there are $\Theta(m)$ edges.

Once (u^t, v^t) arrives, we delete the edge $(x_{t,i}, x'_{t,i})$ for each $u_i = 1$ and the edge $(y_{t,j}, y'_{t,j})$ for each $v_j = 1$. Let d_t be the number of edges we delete in this way. Observe that $(u^t)^\top M v^t = 1$ iff there is an edge (l_i, r_j) for some i, j where $u_i = 1, v_j = 1$ iff there is an augmenting path from $x_{t,i}$ to $y_{t,j}$ for some i, j. So if $(u^t)^\top M v^t = 1$, then the size of maximum matching is decreased by at most $d_t - 1$ Otherwise, the size of maximum matching is decreased by d_t. Before (u^{t+1}, v^{t+1}) arrives, we delete all edges $(x_{t,i}, l_i)$ and $(y_{t,j}, l_j)$ for each $i \leq n_1, j \leq n_2$. Therefore, the graph now has a perfect matching again. So for each t, we need 1 query, and hence $n_3 = \sqrt{m}$ queries in total. □

Corollary 4.4. *For any* n *and* $m = O(n)$*, Conjecture 1.1 implies that there is no partially dynamic algorithm for bipartite maximum matching on a graph with* n *vertices and at most* m *edges with preprocessing time* $p(n) = poly(n)$*, total update time* $u(m, n) = \tilde{o}(m^{3/2})$*, and query time* $q(m) = \tilde{o}(m)$ *that has an error probability of at most* $1/3$*.*

Proof. Same argument as Corollary 4.2. □

Note that the hardness proof in Corollary 4.4 applies only to sparse graphs.

Single Source Shortest Path (ss-SP)

Lemma 4.5. *Given an incremental (respectively decremental) dynamic algorithm* $\mathcal{A}$ *for ss-SP, one can solve* $(\frac{\delta}{1-\delta})$*-*$\mathsf{OuMv}$ *with parameters* n_1*,* n_2*, and* n_3 *by running* $\mathcal{A}$ *on a graph with* $\Theta(m^\delta + m^{1-\delta})$ *vertices and* $\Theta(m)$ *edges which is initially empty (respectively initially has* $\Theta(m)$ *edges), and then making* $m^{2(1-\delta)}$ *queries, where* m *is such that* $n_1 = m^{1-\delta}, n_2 = m^\delta$ *and* $n_3 = m^{1-\delta}$*.*

Proof. Given an input matrix M of $(\frac{\delta}{1-\delta})$-$\mathsf{OuMv}$, we construct the bipartite graph G_M, and also a path $Q = (q_1, q_2, \dots, q_{n_3})$ where $q_1 = s$. Add the edge (q_t, r_j) for all $j \leq n_2, t \leq n_3$. There are $\Theta(n_1 + n_2 + n_3) = \Theta(m^\delta + m^{1-\delta})$ vertices and $\Theta(n_1n_2 + n_2n_3) = \Theta(m)$ edges.

Once u^t and v^t arrive, we disconnect q_t from r_j iff $v^t_j = 0$. We have that if $(u^t)^\top M v^t = 1$, then $d(s, l_i) = t + 1$ for some i where $u_i = 1$, otherwise $d(s, l_i) \geq t + 2$ for all i where $u_i = 1$. So n_1 queries are enough to distinguish these two cases. Before u^{t+1} and v^{t+1} arrive, we disconnect q_t from r_j for all $j \leq n_2$. So for each t, we need n_1 queries, and hence $n_1n_3 = m^{2(1-\delta)}$ queries in total. □

Distinguishing Distance among Vertices between 2 and 4 (ap-SP (2 vs. 4))

Lemma 4.6. *Given an incremental (respectively decremental) dynamic algorithm $\mathcal{A}$ for (α, β)-approximate ap-SP with $2\alpha + \beta < 4$, one can solve $(\frac{\delta}{1-\delta})$-OuMv with parameters n_1, n_2, and n_3 by running $\mathcal{A}$ on a graph with $\Theta(m^\delta + m^{1-\delta})$ vertices and $\Theta(m)$ edges which is initially empty (respectively initially has $\Theta(m)$ edges), and then making $m^{2(1-\delta)}$ queries, where m is such that $n_1 = m^{1-\delta}, n_2 = m^\delta$ and $n_3 = m^{1-\delta}$.*

Proof. Given an input matrix M of $(\frac{\delta}{1-\delta})$-OuMv, we construct a bipartite graph G_M, and another set of vertices $Q = \{q_1, \ldots, q_{n_3}\}$. Add the edge (q_t, r_j) for all $j \leq n_2, t \leq n_3$. There are $\Theta(n_1 + n_2 + n_3) = \Theta(m^\delta + m^{1-\delta})$ vertices and $\Theta(n_1 n_2 + n_2 n_3) = \Theta(m)$ edges.

Once u^t and v^t arrive, we disconnect q_t from r_j iff $v_j^t = 0$. We have that if $(u^t)^\top M v^t = 1$, then $d(q_t, l_i) = 2$ for some i where $u_i = 1$, otherwise $d(s, l_i) \geq 4$ for all i where $u_i = 1$. So for each t, we need n_1 queries, and hence $n_1 n_3 = m^{2(1-\delta)}$ queries in total. □

Transitive Closure

Lemma 4.7. *Given an incremental (respectively decremental) dynamic algorithm $\mathcal{A}$ for transitive closure, one can solve $(\frac{\delta}{1-\delta})$-OuMv with parameters n_1, n_2, and n_3 by running $\mathcal{A}$ on a graph with $\Theta(m^\delta + m^{1-\delta})$ vertices and $\Theta(m)$ edges which is initially empty (respectively initially has $\Theta(m)$ edges), and then making $m^{2(1-\delta)}$ queries, where m is such that $n_1 = m^{1-\delta}, n_2 = m^\delta$ and $n_3 = m^{1-\delta}$.*

Proof. Given an input matrix M of $(\frac{\delta}{1-\delta})$-OuMv, we construct a directed graph G_M where the edges are directed from R to L, and another set of vertices $Q = \{q_1, \ldots, q_{n_3}\}$. Add the directed edge (q_t, r_j) for all $j \leq n_2, t \leq n_3$. There are $\Theta(n_1 + n_2 + n_3) = \Theta(m^\delta + m^{1-\delta})$ vertices and $\Theta(n_1 n_2 + n_2 n_3) = \Theta(m)$ edges.

Once u^t and v^t arrive, we disconnect q_t from r_j iff $v_j^t = 0$. We have that $(u^t)^\top M v^t = 1$ iff q_t can reach l_i for some i where $u_i = 1$. So for each t, we need n_1 queries, and hence $n_1 n_3 = m^{2(1-\delta)}$ queries in total. □

Corollary 4.8. *For any n, $m = \Theta(n^{1/(1-\delta)}\})$ and constant $\delta \in (0, 1/2]$, Conjecture 1.1 implies that there is no partially dynamic algorithm for the problems listed below for a graph with n vertices and at most m edges with preprocessing time $p(m) = poly(m)$, total update time $u(m) = \tilde{\tilde{o}}(mn)$, and query time $q(m) = \tilde{\tilde{o}}(m^\delta)$ per query that has an error probability of at most $1/3$. The problems are:*

- *ss-SP*
- *ap-SP (2 vs. 4)*
- *Transitive Closure*

Proof. Suppose there is such an algorithm $\mathcal{A}$ for any problem in the list. By Lemmas 4.5 to 4.7, we construct an algorithm $\mathcal{B}$ for $(\frac{\delta}{1-\delta})$-OuMv with parameters $n_1 = m^{1-\delta}$, $n_2 = m^\delta$, and $n_3 = m^{1-\delta}$ by running $\mathcal{A}$ on a graph with $n_0 = \Theta(m^\delta + m^{1-\delta}) = \Theta(m^{1-\delta})$ vertices and $m_0 = \Theta(m)$ edges. Note that $m_0 = \Theta(n_0^{1/(1-\delta)})$. Since $\mathcal{A}$ uses polynomial preprocessing time, total update time $\tilde{\tilde{o}}(mn) = \tilde{\tilde{o}}(m^{2-\delta})$ and total query time $O(m^{2(1-\delta)} q(m)) = \tilde{\tilde{o}}(m^{2-\delta})$, $\mathcal{B}$ has polynomial preprocessing time and $\tilde{\tilde{o}}(m^{2-\delta})$ computation time contradicting Conjecture 1.1 by Theorem 2.7. □

5 Further Discussions

5.1 Multiphase Problem

The multiphase problem is introduced in [Pat10] as a problem which can easily provide hardness for various dynamic problems. Though, the lower bounds obtained are always worst-case time lower bounds.

In this section, we first prove that the OMv conjecture implies a tight lower bound for this problem. Then, we show a general approach for getting amortized lower bounds using the known reductions from the multiphase problem.

Definition 5.1 (Multiphase Problem)**.** *The multiphase problem with parameters n_1, n_2 is a problem with 3 phases. Phase 1: Preprocess a Boolean matrix M of size $n_1 \times n_2$ in time $O(n_1 n_2 \tau)$. Phase 2: Get an n_2-dimensional vector v and spend time $O(n_2 \tau)$. Phase 3: Given an index i, answer if $(Mv)_i = 1$ in time $O(\tau)$. We call τ the update time.*

We note that the equivalent problem definition described in [Pat10] uses a family of sets and a set instead of a matrix and a vector. Assuming that there is no truly subquadratic 3SUM algorithm, Pătraşcu [Pat10] showed that one cannot solve multiphase with parameters n_1, n_2 when $n_1 = n_2^{2.5}$ and $\tau = \tilde{\tilde{o}}(\sqrt{n_2})$. Based on OMv, we can easily prove a better a lower bound for τ.

Lemma 5.2. *For any $\gamma > 0$, given an algorithm for multiphase problem with parameters n_1, n_2 and τ update time, one can solve γ-OMv with parameters n_1, n_2, n_3 in time $O((n_1 n_2 + n_2 n_3 + n_1 n_3)\tau)$.*

Proof. Given a matrix M of $n_1 \times n_2$, we run Phase 1 of the multiphase algorithm in time $O(n_1 n_2 \tau)$. For every v^t, $1 \le t \le n_3$, we run n_3 many instances of Phase 2 in time $n_3 \times O(n_2 \tau)$. To compute Mv^t, we run $n_1 n_3$ many instances of Phase 3 in time $n_1 n_3 \times O(\tau)$. □

Corollary 5.3. *For any n_1, n_2, Conjecture 1.1 implies that there is no algorithm $\mathcal{A}$ for the multiphase problem with parameters n_1, n_2 such that $\tau = \tilde{\tilde{o}}(\min\{n_1, n_2\})$.*

Proof. Suppose there is such an algorithm $\mathcal{A}$. Then, by Lemma 5.2 and setting $n_3 = \min\{n_1, n_2\}$, one can solve γ-OMv with parameters n_1, n_2, n_3 in time $O(n_1 n_2 + n_2 n_3 + n_1 n_3) \times \tilde{\tilde{o}}(\min\{n_1, n_2\})$. Assume w.l.o.g. that $n_1 \le n_2$ then we get the expression $O(n_1 n_2 + n_1^2) \times \tilde{\tilde{o}}(n_1) = \tilde{\tilde{o}}(n_1^2 n_2) = \tilde{\tilde{o}}(n_1 n_2 n_3)$, which contradicts Conjecture 1.1 by Theorem 2.2. □

5.1.1 Converting Worst-case Bounds to Amortized Bounds

In the following, let $\mathcal{A}$ be a fully dynamic algorithm that maintains some object G, e.g., a graph, a matrix, an array etc. Similar to Lemma 5.2, by running many instances of phase 2 and 3 of the multiphase algorithm, we have the following.

Lemma 5.4. *For any constant $\gamma > 0$, suppose that one can solve the multiphase problem with parameters n_1, n_2 by running $\mathcal{A}$ on G of size $s(n_1, n_2)$ using $p(n_1, n_2)$ preprocessing time, and then making k_i updates/queries on phase i, for $i \in 2, 3$. Then one can solve γ-OMv problems with parameters n_1, n_2, n_3 by running $\mathcal{A}$ on G using $p(n_1, n_2)$ preprocessing time, and then using $O(k_2 n_3 + k_3 n_1 n_3)$ updates/queries.*

Corollary 5.5. *Suppose that one can solve the multiphase problem with parameters n_1, n_2, where $n_1 = n_2^{\gamma}$, by running $\mathcal{A}$ on G of size $s(n_1, n_2)$ using $\text{poly}(n_1, n_2)$ preprocessing time, and then using k_i updates/queries on phase i, for $i \in 2, 3$. Then Conjecture 1.1 implies that $\mathcal{A}$ cannot maintain an object G of size $s(n_1, n_2)$ with polynomial preprocessing time and $\tilde{\tilde{o}}(\min\{\frac{n_1 n_2}{k_2}, \frac{n_2}{k_3}\})$ amortized update and query time.*

Proof. Suppose that $\mathcal{A}$ has such an amortized update/query time. Then, by Lemma 5.4, one can solve γ-OMv with parameters n_1, n_2, n_3 using $\text{poly}(n_1, n_2)$ preprocessing time, and computation time $\tilde{\tilde{o}}(\min\{\frac{n_1 n_2}{k_2}, \frac{n_2}{k_3}\} \times O(k_2 n_3 + k_3 n_1 n_3) = \tilde{\tilde{o}}(n_1 n_2 n_3)$. This contradicts Conjecture 1.1 by Theorem 2.2. □

Example. It is shown in [Pat10] that, given a fully dynamic algorithm $\mathcal{A}$ for subgraph connectivity that runs on a graph with $O(n_1 + n_2)$ vertices, the multiphase problem with parameters n_1, n_2 can be solved using $k_2 = n_2$ operations in phase 2 and $k_3 = 1$ operations in phase 3. By Corollary 5.5 and by setting $n_1 = n_2 = n$, the amortized cost of $\mathcal{A}$ on a graph with $\Theta(n)$ vertices cannot be $\tilde{\tilde{o}}(n)$ unless the OMv conjecture fails. Note however that this lower bound is subsumed by the one we give in Corollary 3.4.

5.2 Open Problems

Of course it is very interesting to settle the OMv conjecture. Besides this, there are still many problems for which this work does not provide tight lower bounds, and it is interesting to prove such lower bounds based on the OMv or other reasonable conjectures.

Minimum Cut. Thorup and Karger [Tho00] presented a $(2 + o(1))$-approximation algorithm with polylogarithmic amortized update time. Thorup [Tho07] showed that in $\tilde{O}(\sqrt{n})$ worst-case update time the minimum cut can be maintained exactly when the minimum cut size is small and $(1 + \epsilon)$-approximately otherwise. Improving this result using amortization is mentioned as a major open problem in [Tho07]. Very recently Fakcharoenphol et al. [FKN+14] showed some related hardness results (e.g., for some subroutine used in Thorup's algorithm). However, currently there is no evidence that the minimum cut cannot be maintained in polylogarithmic update time. In fact, it is not even known if polylogarithmic update time is possible or impossible for a key subroutine in Thorup's algorithm called *min-tree cut*, where we are given edge updates on a graph and its spanning tree and have to maintain the minimum cut among the cuts obtained by removing one edge from the spanning tree. We believe that understanding this subroutine is an important step in understanding the dynamic minimum cut problem.

Approximation Algorithms for Non-Distance Problems. In this paper we provide hardness results for several approximation algorithms for distance-related problems. However, we could not extend the techniques to non-distance graph problems in undirected graphs such as approximating maximum matching, minimum cut, maximum flow, and maximum densest subgraph.

Total Update Time for Partially Dynamic Algorithms. While there are many hardness results in the partially dynamic setting in this and previous work, quite a few problems are still open. Of particular interest are problems that are known to be easy when one type of updates is allowed but become challenging when the other type of updates is allowed. For example, the single-source reachability problem can be solved in $O(m)$ total update time in the incremental setting [Ita86] but the best time in the decremental setting is still larger than $mn^{0.9}$ (e.g., [HKN14b, HKN15]). This is also the case for the minimum cut problem where the incremental setting can be $(1+\epsilon)$-approximated in $\tilde{O}(m)$ total update time [RH97] while the current best decremental $(1+\epsilon)$-approximation algorithm requires $\tilde{O}(m+n\sqrt{n})$ total update time [Tho07], and the topological ordering problem which is trivial in the decremental setting but challenging otherwise (e.g., [HKM+12, BFG09]). Since known hardness techniques – including those presented in this paper – usually work for both the incremental and the decremental setting, proving non-trivial hardness results for the above problems seems to be challenging.

Worst-Case Update Time. While there are fully dynamic algorithms with polylogarithmic *amortized* update time for many problems, not much is known for *worst-case* update time. The only exception that we are aware of is the connectivity problem (due to the recent breakthrough of Kapron et al. [KKM13]). Other basic graph problems, such as minimum spanning tree, 2-edge connectivity, biconnectivity, maximum matching, and densest subgraph are not known to have polylogarithmic worst-case update time. A polynomial hardness result for worst-case update time for these problems based on a natural assumption will be very interesting. The challenge in obtaining this is that such a result must hold only for the worst-case update time and not for the amortized one. Such results were published previously (e.g., those in [AVW14, Pat10, KPP16]), but most of these results are now known to hold for amortized update time as well assuming OMv and SETH (some exceptions are those for partially dynamic problems).

Deterministic Algorithms. Derandomizing the current best randomized algorithms is an important question for many problems, e.g., approximate decremental single-source and all-pairs shortest paths [Ber13, HKN14a] and worst-case connectivity and spanning tree [KKM13]. This is important since deterministic algorithms do not have to limit the power of the adversary generating the sequence of updates and queries. Proving that derandomization is impossible for some problems will be very interesting. The challenge is that such hardness results must hold only for deterministic algorithm and not for randomized algorithms.

Trade-off between Query and Update Time In this paper we present hardness results with a trade-off between query and update time for several problems. Are these hardness results tight? This seems to be a non-trivial question since not much is known about the upper bounds for these problems. A problem for which it is particularly interesting to study this question is the subgraph connectivity problem, since it is the starting point of many reductions that lead to hardness results with a trade-off. In this paper, we show that for any $0 < \alpha < 1$ getting an $O(m^\alpha)$ update time requires a query time of $\Omega(m^{1-\alpha})$. This matches two known upper bounds in [Dua10, CPR11] when $\alpha = 4/5$ and $\alpha = 2/3$. It is reasonable to conjecture that there is a matching upper bound for all $0 < \alpha < 1$; however, it is not clear if this is true or not.

Acknowledgement

D. Nanongkai would like to thank Parinya Chalermsoomsook, Bundit Laekhanukit, and Virginia Vassilevska Williams for comments. T. Saranurak would like to thank Markus Bläser and Gorav Jindal for discussions.

References

[AVW14] Amir Abboud and Virginia Vassilevska Williams. "Popular conjectures imply strong lower bounds for dynamic problems". In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 434–443 (cit. on pp. 3, 4, 6–10, 19, 21–23, 40, 45).

[BDH+12] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. "Graph Expansion and Communication Costs of Fast Matrix Multiplication". In: *Journal of the ACM* 59.6 (2012), 32:1–32:23 (cit. on pp. 1, 3, 8).

[Ber13] Aaron Bernstein. "Maintaining Shortest Paths Under Deletions in Weighted Directed Graphs". In: *Symposium on Theory of Computing (STOC)*. 2013, pp. 725–734 (cit. on pp. 7, 35, 40).

[BFG09] Michael A. Bender, Jeremy T. Fineman, and Seth Gilbert. "A New Approach to Incremental Topological Ordering". In: *Symposium on Discrete Algorithms (SODA)*. 2009, pp. 1108–1115 (cit. on p. 40).

[BHN+15] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. "Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams". In: *Symposium on Theory of Computing (STOC)*. 2015, pp. 173–182 (cit. on p. 4).

[BKM95] Julien Basch, Sanjeev Khanna, and Rajeev Motwani. *On Diameter Verification and Boolean Matrix Multiplication*. Tech. rep. Stanford University, 1995 (cit. on p. 8).

[Blä14] Markus Bläser. "Lower Bounds for Online Matrix-Vector Multiplication". Manuscript. 2014 (cit. on p. 3).

[BVW08] Guy E. Blelloch, Virginia Vassilevska, and Ryan Williams. "A New Combinatorial Approach for Sparse Graph Problems". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2008, pp. 108–120 (cit. on p. 8).

[BW12] Nikhil Bansal and Ryan Williams. "Regularity Lemmas and Combinatorial Algorithms". In: *Theory of Computing* 8.1 (2012). Announced at FOCS'09, pp. 69–94 (cit. on pp. 8, 9, 17, 18).

[Che12] Shiri Chechik. "Improved Distance Oracles and Spanners for Vertex-Labeled Graphs". In: *European Symposium on Algorithms (ESA)*. 2012, pp. 325–336 (cit. on pp. 4, 11, 20, 24).

[CPR11] Timothy M. Chan, Mihai Patrascu, and Liam Roditty. "Dynamic Connectivity: Connecting to Networks and Geometry". In: *SIAM Journal on Computing* 40.2 (2011). Announced at FOCS'08, pp. 333–349 (cit. on pp. 6, 20, 40).

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. "All-Pairs Almost Shortest Paths". In: *SIAM Journal on Computing* 29.5 (2000). Announced at FOCS'96, pp. 1740–1759 (cit. on pp. 3, 7, 8, 34).

[DP10] Ran Duan and Seth Pettie. "Connectivity Oracles for Failure Prone Graphs". In: *Symposium on Theory of Computing (STOC)*. 2010, pp. 465–474 (cit. on pp. 11, 20, 26).

[Dua10] Ran Duan. "New Data Structures for Subgraph Connectivity". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2010, pp. 201–212 (cit. on pp. 6, 20, 40).

[ES81] Shimon Even and Yossi Shiloach. "An On-Line Edge-Deletion Problem". In: *Journal of the ACM* 28.1 (1981), pp. 1–4 (cit. on pp. 7, 35).

[FHW12] Silvio Frischknecht, Stephan Holzer, and Roger Wattenhofer. "Networks Cannot Compute Their Diameter in Sublinear Time". In: *Symposium on Discrete Algorithms (SODA)*. 2012, pp. 1150–1162 (cit. on p. 29).

[FI00] Daniele Frigioni and Giuseppe F. Italiano. "Dynamically Switching Vertices in Planar Graphs". In: *Algorithmica* 28.1 (2000), pp. 76–103 (cit. on p. 6).

[FKN+14] Jittat Fakcharoenphol, Tanee Kumpijit, Danupon Nanongkai, Thatchaphol Saranurak, and Pattara Sukprasert. "Fully-Dynamic Minimum Cut on Plane Graphs in Poylogarithmic Time". Manuscript. 2014 (cit. on pp. 4, 6, 39).

[Gal12] François Le Gall. "Faster Algorithms for Rectangular Matrix Multiplication". In: *Symposium on Foundations of Computer Science (FOCS)*. 2012, pp. 514–523 (cit. on p. 8).

[Gal14] François Le Gall. "Powers of Tensors and Fast Matrix Multiplication". In: *International Symposium on Symbolic and Algebraic Computation (ISSAC)*. 2014, pp. 296–303 (cit. on p. 8).

[HKM+12] Bernhard Haeupler, Telikepalli Kavitha, Rogers Mathew, Siddhartha Sen, and Robert Endre Tarjan. "Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance". In: *ACM Transactions on Algorithms* 8.1 (2012). Announced at ICALP'08, p. 3 (cit. on p. 40).

[HKN13] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization". In: *Symposium on Foundations of Computer Science (FOCS)*. 2013, pp. 538–547 (cit. on pp. 3, 8, 35).

[HKN14a] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "Decremental Single-Source Shortest Paths on Undirected Graphs in Near-Linear Total Update Time". In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 146–155 (cit. on pp. 7, 35, 40).

[HKN14b] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "Sublinear-Time Decremental Algorithms for Single-Source Reachability and Shortest Paths on Directed Graphs". In: *Symposium on Theory of Computing (STOC)*. 2014, pp. 674–683 (cit. on p. 40).

[HKN15] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "Improved Algorithms for Decremental Single-Source Reachability on Directed Graphs". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2015, pp. 725–736 (cit. on p. 40).

[HLW+11] Danny Hermelin, Avivit Levy, Oren Weimann, and Raphael Yuster. "Distance Oracles for Vertex-Labeled Graphs". In: *International Colloquium on Automata, Languages and Programming (ICALP)*. 2011, pp. 490–501 (cit. on pp. 4, 20, 24).

[Ita86] Giuseppe F. Italiano. "Amortized Efficiency of a Path Retrieval Data Structure". In: *Theoretical Computer Science* 48.3 (1986), pp. 273–281 (cit. on p. 40).

[KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. "Dynamic graph connectivity in polylogarithmic worst case time". In: *Symposium on Discrete Algorithms (SODA)*. 2013, pp. 1131–1142 (cit. on pp. 6, 7, 20, 40).

[KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. "Higher Lower Bounds from the 3SUM Conjecture". In: *Symposium on Discrete Algorithms (SODA)*. 2016, to appear (cit. on pp. 4, 21, 34, 40).

[Lac13] Jakub Łącki. "Improved Deterministic Algorithms for Decremental Reachability and Strongly Connected Components". In: *ACM Transactions on Algorithms* 9.3 (2013). Announced at SODA'11, p. 27 (cit. on p. 35).

[LOP+15] Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. "The Power of Dynamic Distance Oracles: Efficient Dynamic Algorithms for the Steiner Tree". In: *Symposium on Theory of Computing (STOC)*. 2015, pp. 11–20 (cit. on pp. 4, 6, 11, 20, 25).

[LZ09] Edo Liberty and Steven W. Zucker. "The Mailman algorithm: A note on matrix-vector multiplication". In: *Information Processing Letters* 109.3 (2009), pp. 179–182 (cit. on p. 3).

[Mot55] Theodore S. Motzkin. "Evaluation of polynomials and evaluation of rational functions". In: *Bulletin of the AMS* 61.163 (1955), p. 10 (cit. on p. 3).

[Mąd11] Aleksander Mądry. "From Graphs to Matrices, and Back: New Techniques for Graph Algorithms". PhD thesis. Massachusetts Institute of Technology, 2011 (cit. on p. 24).

[Pan01] Victor Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001. ISBN: 0-8176-4240-4 (cit. on p. 3).

[Pat10] Mihai Patrascu. "Towards Polynomial Lower Bounds for Dynamic Problems". In: *Symposium on Theory of Computing (STOC)*. 2010, pp. 603–610 (cit. on pp. 4, 6, 9, 22, 27, 38–40).

[RH97] Monika Rauch Henzinger. "A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity". In: *Journal of Algorithms* 24.1 (1997), pp. 194–220 (cit. on p. 40).

[RT13] Liam Roditty and Roei Tov. "Approximating the Girth". In: *ACM Transactions on Algorithms* 9.2 (2013). Announced at SODA'11, 15:1–15:13 (cit. on pp. 3, 8).

[RZ11] Liam Roditty and Uri Zwick. "On Dynamic Shortest Paths Problems". In: *Algorithmica* 61.2 (2011). Announced at ESA'04, pp. 389–401 (cit. on pp. 7, 34).

[RZ12] Liam Roditty and Uri Zwick. "Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs". In: *SIAM Journal on Computing* 41.3 (2012). Announced at FOCS'04, pp. 670–683 (cit. on pp. 7, 35).

[San04] Piotr Sankowski. "Dynamic Transitive Closure via Dynamic Matrix Inverse". In: *Symposium on Foundations of Computer Science (FOCS)*. 2004, pp. 509–517 (cit. on pp. 4, 7, 8).

[San05a] Piotr Sankowski. "Shortest Paths in Matrix Multiplication Time". In: *Algorithms - ESA 2005, 13th Annual European Symposium, Proceedings*. 2005, pp. 770–778 (cit. on p. 7).

[San05b] Piotr Sankowski. "Subquadratic Algorithm for Dynamic Shortest Distances". In: *International Computing and Combinatorics Conference (COCOON)*. 2005, pp. 461–470 (cit. on p. 7).

[San07] Piotr Sankowski. "Faster Dynamic Matchings and Vertex Connectivity". In: *Symposium on Discrete Algorithms (SODA)*. 2007, pp. 118–126 (cit. on p. 8).

[Sav74] John E. Savage. "An Algorithm for the Computation of Linear Forms". In: *SIAM Journal on Computing* 3.2 (1974), pp. 150–158 (cit. on p. 3).

[Sto10] Andrew James Stothers. "On the Complexity of Matrix Multiplication". PhD thesis. The University of Edinburgh, 2010 (cit. on p. 9).

[Str69] Volker Strassen. "Gaussian Elimination is not Optimal." In: *Numerische Mathematik* 13 (1969), pp. 354–356 (cit. on p. 8).

[SU86] Nicola Santoro and Jorge Urrutia. "An Improved Algorithm for Boolean Matrix Multiplication". In: *Computing* 36.4 (1986), pp. 375–382 (cit. on p. 3).

[Tho00] Mikkel Thorup. "Dynamic Graph Algorithms with Applications". In: *Scandinavian Workshop on Algorithm Theory (SWAT)*. 2000, pp. 1–9 (cit. on p. 39).

[Tho07] Mikkel Thorup. "Fully-Dynamic Min-Cut". In: *Combinatorica* 27.1 (2007). Announced at STOC'01, pp. 91–127 (cit. on pp. 39, 40).

[VW12] Virginia Vassilevska Williams. "Multiplying Matrices Faster Than Coppersmith-Winograd". In: *Symposium on Theory of Computing (STOC)*. 2012 (cit. on p. 9).

[VWW10] Virginia Vassilevska Williams and Ryan Williams. "Subcubic Equivalences between Path, Matrix and Triangle Problems". In: *Symposium on Foundations of Computer Science (FOCS)*. 2010, pp. 645–654 (cit. on pp. 3, 8, 15).

[Wil07] Ryan Williams. "Matrix-Vector Multiplication in Sub-Quadratic Time: (Some Preprocessing Required)". In: *SODA*. 2007, pp. 995–1001 (cit. on pp. 1, 3, 17).

Appendix

A Conjectures (from [AVW14])

Conjecture A.1 (No truly subquadratic 3SUM (3SUM)). *In the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $n^{2-o(1)}$ time in expectation to determine whether a set $S \subset \{-n^3, \dots, n^3\}$ of $|S| = n$ integers contains three distinct elements $a, b, c \in S$ with $a + b = c$.*

Conjecture A.2 (No truly subcubic APSP (APSP)). *There is a constant c, such that in the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $n^{3-o(1)}$ time in expectation to compute the distances between every pair of vertices in an n node graph with edge weights in $\{1, ..., n^c\}$.*

Conjecture A.3 (Strong Exponential Time Hypothesis (SETH)). *For every $\epsilon > 0$, there exists a k, such that SAT on k-CNF formulas on n variables cannot be solved in $O^*(2^{(1-\epsilon)n})$ time, where $O^*(\cdot)$ hides polynomial factor.*

Conjecture A.4 (No almost linear time triangle (Triangle)). *There is a constant $\delta > 0$, such that in the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $m^{1+\delta-o(1)}$ time in expectation to detect whether an m edge graph contains a triangle.*

Conjecture A.5 (No truly subcubic combinatorial BMM (BMM)). *In the Word RAM model with words of $O(\log n)$ bits, any* combinatorial *algorithm requires $n^{3-o(1)}$ time in expectation to compute the Boolean product of two $n \times n$ matrices.*