

Conditional Hardness for Sensitivity Problems

Monika Henzinger^{*1}, Andrea Lincoln^{†2}, Stefan Neumann³, and Virginia Vassilevska Williams^{‡4}

- 1 University of Vienna, Faculty of Computer Science, Vienna, Austria
monika.henzinger@univie.ac.at
- 2 Stanford University, Computer Science Department, Stanford, USA
andreali@cs.stanford.edu
- 3 University of Vienna, Faculty of Computer Science, Vienna, Austria
stefan.neumann@univie.ac.at
- 4 Stanford University, Computer Science Department, Stanford, USA
virgi@cs.stanford.edu

Abstract

In recent years it has become popular to study dynamic problems in a sensitivity setting: Instead of allowing for an arbitrary sequence of updates, the sensitivity model only allows to apply batch updates of small size to the *original* input data. The sensitivity model is particularly appealing since recent strong conditional lower bounds ruled out fast algorithms for many dynamic problems, such as shortest paths, reachability, or subgraph connectivity.

In this paper we prove conditional lower bounds for these and additional problems in a sensitivity setting. For example, we show that under the Boolean Matrix Multiplication (BMM) conjecture combinatorial algorithms cannot compute the $(4/3 - \epsilon)$ -approximate diameter of an undirected unweighted dense graph with truly subcubic preprocessing time and truly subquadratic update/query time. This result is surprising since in the static setting it is not clear whether a reduction from BMM to diameter is possible. We further show under the BMM conjecture that many problems, such as reachability or approximate shortest paths, cannot be solved faster than by recomputation from scratch even after *only one or two* edge insertions. We extend our reduction from BMM to Diameter to give a reduction from All Pairs Shortest Paths to Diameter under one deletion in weighted graphs. This is intriguing, as in the static setting it is a big open problem whether Diameter is as hard as APSP. We further get a nearly tight lower bound for shortest paths after two edge deletions based on the APSP conjecture. We give more lower bounds under the Strong Exponential Time Hypothesis. Many of our lower bounds also hold for static oracle data structures where no sensitivity is required.

Finally, we give the first algorithm for the $(1 + \epsilon)$ -approximate radius, diameter, and eccentricity problems in directed or undirected unweighted graphs in case of single edges failures. The algorithm has a truly subcubic running time for graphs with a truly subquadratic number of edges; it is tight w.r.t. the conditional lower bounds we obtain.

1998 ACM Subject Classification F.2.2 Computations on discrete structures

Keywords and phrases sensitivity, conditional lower bounds, data structures, dynamic graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ITCS.2017.26

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

† Supported by a Stanford Graduate Fellowship.

‡ VVW and AL were supported by NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.



1 Introduction

A dynamic algorithm is an algorithm that is able to handle changes in the input data: It is given an input instance x and is required to maintain certain properties of x while x undergoes (possibly very many) updates. For example, an algorithm might maintain a graph, which undergoes edge insertions and deletions, and a query is supposed to return the diameter of the graph after the updates. Often dynamic algorithms are also referred to as data structures. During the last few years strong conditional lower bounds for many dynamic problems were derived (see, e.g., [36, 3, 27, 4, 21, 1, 32]), which rule out better algorithms than simple recomputation from scratch after each update or before each query.

Partially due to this, in recent years it has become popular to study dynamic problems in a more restricted setting that only allows for a *bounded* number of changes to the input instance (see, for example, [37, 23, 10, 19], and the references in Table 4). These algorithms are usually referred to as *sensitivity*¹ data structures. The hope is to obtain algorithms in the sensitivity setting which are faster than the conditional lower bounds for the general setting.

More formally, a *data structure with sensitivity d* for a problem P has the following properties: It obtains an instance p of P and is allowed polynomial preprocessing time on p . After the preprocessing, the data structure must provide the following operations:

(Batch) Update. Up to d changes are performed to the *initial* problem instance p , e.g., d edges are added to or removed from p .

Query. The user queries a specific property about the instance of the problem after the last update, e.g., the shortest path between two nodes avoiding the edges deleted in the last update.

The parameter d bounding the batch update size is referred to as the *sensitivity* of the data structure. Note that every batch update is performed on the *original* problem instance. Thus, in contrast to “classic” dynamic algorithms (without sensitivity), a query only reflects the changes made to p by the *last* batch update and *not* by previous batch updates. As the size of a batch update is constrained to at most d , each query is executed on a graph that differs from p by at most d edges. After a batch update an arbitrary number of queries may be performed.

Some data structures (usually called *oracles*) combine a query and an update into a single operation, i.e., the combined operation obtains an input tuple (Q, U) , where Q is a query and U is an update. A special case are *static oracles*, which have $U = \emptyset$. The conditional lower bounds we derive in this paper also hold in this setting, since oracles with an efficient combined operation can be used to solve sensitivity problems.

While some existing sensitivity data structures can preprocess the answers to all possible updates and queries during their preprocessing time, this is not possible in general (due to constraints in the preprocessing time and the fact that the number of possible updates/queries grows exponentially in the parameter d). Hence, we still consider a sensitivity data structure a dynamic (instead of static) algorithm.

¹ Sometimes sensitivity data structures are also called “fault-tolerant” or “emergency planning” algorithms. See Appendix A.2 for a discussion of terminology.

The Hypotheses

We state the hypotheses on which we base the conditional lower bounds in this paper. By now they are all considered standard in proving fine-grained reduction-based lower bounds. For a more detailed description of the hypotheses, see, e.g., Abboud and Williams [3] and the references therein. As usual we work in the word-RAM model of computation with word length of $O(\log n)$ bits. The hypotheses below concern the complexity of the Boolean Matrix Multiplication (BMM), Satisfiability of Boolean Formulas in Conjunctive Normal Form (CNF-SAT), All Pairs Shortest Paths (APSP), Triangle Detection and Online Boolean Matrix Vector Multiplication (OMv) problems. Other popular hypotheses from prior work consider other famous problems such as 3SUM and other sparsity regimes such as triangle detection in very sparse graphs (see, e.g. [3]).

► **Conjecture 1** (Impagliazzo, Paturi and Zane [29, 30]). *The Strong Exponential Time Hypothesis (SETH) states that for each $\varepsilon > 0$, there exists a $k \in \mathbb{N}$, such that k -SAT cannot be solved in time $O(2^{n(1-\varepsilon)}) \text{ poly}(n)$.*

► **Conjecture 2.** *The Boolean Matrix Multiplication (BMM) conjecture states that for all $\varepsilon > 0$, there exists no combinatorial algorithm that computes the product of two $n \times n$ matrices in expected time $O(n^{3-\varepsilon})$.*

Note that BMM can be solved in truly subcubic using fast matrix multiplication (FMM): the current fastest algorithms run in $O(n^{2.373})$ time [41, 25]. However, algorithms using FMM are not considered to be combinatorial. Formally, the term *combinatorial* algorithm is not well-defined and it is common to rule out the use of FMM or other “Strassen-like” methods in the design of such algorithms as most of them are not considered practical. True combinatorial algorithms are not only considered practical but also easily extendable. For instance, prior work on combinatorial BMM algorithms has almost always led to an algorithm for APSP with similar running time (e.g. [5] and [17]).

One of the simplest graph problems is that of detecting whether the graph contains a triangle, i.e., three nodes with all three edges between them. Itai and Rodeh [31] showed that any algorithm for BMM can solve Triangle detection in the same time. Conversely, Vassilevska Williams and Williams [42] showed that any truly subcubic combinatorial algorithm for Triangle Detection can be converted into a truly subcubic combinatorial algorithm for BMM. Hence, the BMM conjecture implies there is no truly subcubic *combinatorial* algorithm for Triangle Detection. We use this fact and the resulting Triangle Conjecture that there is no truly subcubic algorithm for Triangle Detection in our reductions based on BMM.

The following is a popular conjecture about the APSP problem.

► **Conjecture 3.** *The APSP conjecture states that given a graph G with n vertices, m edges, and edge weights in $\{1, \dots, n^c\}$ for some constant c , the All Pairs Shortest Paths problem (APSP) cannot be solved in $O(n^{3-\varepsilon})$ expected time for any $\varepsilon > 0$.*

Similar to the relationship between BMM and Triangle Detection, [42] showed that there is a triangle problem in weighted graphs, Negative Triangle, that is equivalent under subcubic reductions to APSP. We use that problem in our reductions.

Our final conjecture concerns Boolean matrix vector product.

► **Conjecture 4** (Henzinger et al. [27]). *Let B be a Boolean matrix of size $n \times n$. In the Online Matrix-vector (OMv) problem, n binary vectors v_1, \dots, v_n of size n appear online and an algorithm solving the problem must output the vector Bv_i before the next vector v_{i+1} arrives.*

The OMv conjecture states that for all $\varepsilon > 0$ and after any polynomial time preprocessing of B , it takes $\Omega(n^{3-\varepsilon})$ time to solve the OMv problem with error probability at most $1/3$.

Most of the conjectures are stated w.r.t. *expected* time, i.e., the conjectures rule out randomized algorithms. In case of dynamic algorithms using randomness, it is common to argue if an oblivious or a non-oblivious adversary is allowed. Previous literature on conditional lower bounds for dynamic algorithms did not explicitly state what kind of adversaries are allowed for their lower bounds. We give a quick discussion of this topic in Appendix A.3.

Our Results

In this paper we develop a better understanding of the possibilities and limitations of the sensitivity setting by providing conditional lower bounds for sensitivity problems. We show that under plausible assumptions for many dynamic graph problems even the update of only *one or two* edges cannot be solved faster than by re-computation from scratch. See Table 2 and Table 3 in the Appendix for a list of all our conditional lower bounds for sensitivity data structures, and our lower bounds for static oracles respectively. Table 1 gives explanations of the problems. The abbreviations used in the tables are explained in its captions. We next discuss our main results.

New reductions

We give several new reductions for data structures with small sensitivity.

- (1) We give a novel reduction from triangle detection and BMM to maintaining an approximation of the diameter of the graph and eccentricities of all vertices, under a single edge failure. This is particularly surprising because in the static case it is unknown how to reduce BMM to diameter computation. Using the BMM conjecture this results in lower bounds of $n^{3-o(1)}$ on the preprocessing time or of $n^{2-o(1)}$ update or query time for $(4/3 - \varepsilon)$ -approximate decremental diameter and eccentricity in unweighted graphs *with sensitivity 1*, i.e., when a single edge is deleted. Those results are tight w.r.t. the algorithm we present in Section 5.
- (2) A particular strength of BMM-based reductions is that they can very often be converted into APSP-based lower bounds for weighted variants of the problems. APSP-based lower bounds, in turn, no longer require the “combinatorial”-condition on the algorithms, making the lower bounds stronger. We show how our BMM-based lower bounds for approximate diameter with sensitivity 1 can be converted into an APSP-based lower bound for diameter with sensitivity 1 in weighted graphs. In particular, we show that unless APSP has a truly subcubic algorithm, any data structure that can support diameter queries for a single edge deletion must either have essentially cubic preprocessing time, or essentially quadratic query time. This lower bound is tight w.r.t. to a trivial algorithm using the data structure of [10]. The APSP to 1-sensitive Diameter lower bound is significant also because it is a big open problem whether in the static case Diameter and APSP are actually subcubically equivalent (see e.g. [2]).
- (3) We consider the problem of maintaining the distance between two fixed nodes s and t in an undirected weighted graph under edge failures. The case of a *single edge failure* can be solved in m edge, n node graphs with essentially optimal $O(m\alpha(n))$ preprocessing time and $O(1)$ query time with an algorithm of Nardelli et al. [33]. The case of two edge failures has been open for some time. We give compelling reasons for this by showing that under the APSP conjecture, maintaining the s - t distance in an unweighted graph under two edge failures requires either $n^{3-o(1)}$ preprocessing time or $n^{2-o(1)}$ query time. Notice that with no preprocessing time, just by using Dijkstra’s algorithm at query time, one can obtain $O(n^2)$ query time. Similarly, one can achieve $\tilde{O}(n^3)$ preprocessing time and $O(1)$

query time by applying the single edge failure algorithm of [33] n times at preprocessing, once for $G \setminus \{e\}$ for every e on the shortest st path. Thus our lower bound shows that under the APSP conjecture, the naive recomputation time is essentially optimal.

- (4) We show lower bounds with sensitivity d for deletions-only and insertions-only $(2 - \varepsilon)$ -approximate single source and $(5/3 - \varepsilon)$ -approximate st -shortest paths in undirected unweighted graphs, as well as for weighted bipartite matching problems under the OMv conjecture. The lower bounds show that with polynomial in n preprocessing either the update time must be super-polynomial in d or the query time must be $d^{1-o(1)}$.

New upper bounds

We complement our lower bounds with an algorithm showing that some of our lower bounds are tight: In particular, we present a deterministic combinatorial algorithm that can compute a $(1 + \varepsilon)$ -approximation (for any $\varepsilon > 0$) for the eccentricity of any given vertex, the radius and the diameter of a directed or undirected unweighted graph after single edge failures. The preprocessing time of the data structure is $\tilde{O}(mn + n^{1.5}\sqrt{Dm/\varepsilon})$, where D is the diameter of the graph and m and n are the number of edges and vertices; the query time is constant. Since $D \leq n$, the data structure can be preprocessed in time $\tilde{O}(n^2\sqrt{m/\varepsilon})$. In particular, for sparse graphs with $m = \tilde{O}(n)$, it takes time $\tilde{O}(n^{2.5}\varepsilon^{-\frac{1}{2}})$ to build the data structure. Our lower bounds from BMM state that even getting a $(4/3 - \varepsilon)$ -approximation for diameter or eccentricity after a *single* edge deletion requires either $n^{3-o(1)}$ preprocessing time, or $n^{2-o(1)}$ query or update time. Hence, our algorithm's preprocessing time is tight (under the conjecture) since it has constant time queries.

Conditional Lower Bounds based on modifications of prior reductions

Some reductions in prior work [42, 3] only perform very few updates before a query is performed or they can be modified to do so. After the query, the updates are “undone” by rolling back to the initial instance of the input problem. Hence, some of their reductions also hold in a sensitivity setting. Specifically we achieve the following results in this way:

- (1) Based on the BMM conjecture we show that for reachability problems with st -queries already *two* edge insertions require $n^{3-o(1)}$ preprocessing time or $n^{2-o(1)}$ update or query time; for ss -queries we obtain the same bounds even for a *single* edge insertion. This lower bound is matched by an algorithm that recomputes at each step.
- (2) We present strong conditional lower bounds for static oracle data structures. We show that under the BMM conjecture, oracle data structures that answer about the reachability between any two queried vertices cannot have truly subcubic preprocessing time *and* truly subquadratic query time. This implies that combinatorial algorithms *either* essentially need to compute the transitive closure matrix of the graph during the preprocessing time *or* essentially need to traverse the graph at each query. We show the same lower bounds for static oracles that solve the $(5/3 - \varepsilon)$ -approximate ap -shortest paths problem in undirected unweighted graphs. This shows that we essentially cannot do better than solving APSP in the preprocessing or computing the distance in each query.
- (3) The subcubic equivalence between the replacement paths problem and APSP [42] immediately leads to a conditional lower bound for s - t distance queries with sensitivity 1 in *directed*, weighted graphs. Our lower bound for s - t distance queries with sensitivity 2 in undirected graphs is inspired by this reduction. The lower bound for sensitivity 1 is matched by the algorithm of Bernstein and Karger [10].

Similarly, a reduction from BMM to replacement paths in directed unweighted graphs from [42] shows that the $O(m\sqrt{n})$ time algorithm of Roditty and Zwick [38] is optimal among all combinatorial algorithms, for every choice of m as a function of n . It also immediately implies that under the BMM conjecture, combinatorial s - t distance 1-sensitivity oracles in unweighted graphs require either $mn^{0.5-o(1)}$ preprocessing time or $m/n^{0.5+o(1)}$ query time, for every choice of m as a function of n ; this is tight due to Roditty and Zwick’s algorithm. (The combinatorial restriction is important here as there is a faster $\tilde{O}(n^{2.373})$ time non-combinatorial algorithm for replacement paths [40] and hence for distance sensitivity oracles in directed unweighted graphs.)

- (4) We additionally provide new lower bounds under SETH: We show that assuming SETH the #SSR problem cannot be solved with truly subquadratic update and query times when any constant number of edge insertions is allowed; this matches the lower bound for the general dynamic setting. For the ST -reachability problem and the computation of $(4/3 - \varepsilon)$ -approximate diameter we show that under SETH truly sublinear update and query times are not possible even when only a constant number of edge insertions are supported. The sensitivity of the reductions is a constant $K(\varepsilon, t)$ that is determined by the preprocessing time $O(n^t)$ we allow and some properties of the sparsification lemma [30]. Notice that while the constant $K(\varepsilon, t)$ depends on the preprocessing time and the constant in the sparsification lemma, it does *not* depend on any property of the SAT instance in the reduction. See Section 4 for a thorough discussion of the parameter $K(\varepsilon, t)$. The lower bound for #SSR shows that we cannot do better than recomputation after each update.
- (5) Using a reduction from OMv we show lower bounds with sensitivity d for deletions-only or insertions-only st -reachability, strong connectivity in directed graphs. The lower bounds show that with polynomial in n preprocessing either the update time must be super-polynomial d or the query time must be $\Omega(d^{1-\varepsilon})$.

Related Work

In the last few years many conditional lower bounds were derived for dynamic algorithms. Abboud and Williams [3] gave such lower bounds under several different conjectures. New lower bounds were given by Henzinger et al. [27], who introduced the OMv conjecture, and by Abboud, Williams and Yu [4], who stated combined conjectures that hold as long as either the 3SUM conjecture *or* SETH *or* the APSP conjecture is correct. Dahlgaard [21] gave novel lower bounds for partially dynamic algorithms. Abboud and Dahlgaard [1] showed the first hardness results for dynamic algorithms on planar graphs and Kopelowitz, Pettie and Porat [32] gave stronger lower bounds from the 3SUM conjecture. However, none of the lower bounds mentioned in the above papers explicitly handled the sensitivity setting.

During the last decade there have been many new algorithms designed for the sensitivity setting. In Section A.5 we give a short discussion summarizing many existing algorithms.

2 Lower Bounds From Boolean Matrix Multiplication

The following theorem summarizes the lower bounds we derived from the BMM conjecture.

► **Theorem 5.** *Assuming the BMM conjecture, combinatorial algorithms cannot solve the following problems with preprocessing time $O(n^{3-\varepsilon})$, and update and query times $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$:*

1. *incremental st -reachability with sensitivity 2,*

2. *incremental ss-reachability with sensitivity 1,*
3. *static ap-reachability,*
4. *$(7/5 - \varepsilon)$ -approximate st shortest paths in undirected unweighted graphs with sensitivity 2,*
5. *$(3/2 - \varepsilon)$ -approximate ss shortest paths in undirected unweighted graphs with sensitivity 1,*
6. *static $(5/3 - \varepsilon)$ -approximate ap shortest paths*
7. *decremental $(4/3 - \varepsilon)$ -approx. diameter in undirected unweighted graphs with sensitivity 1,*
8. *decremental $(4/3 - \varepsilon)$ -approx. eccentricity in undirected unweighted graphs with sensitivity 1.*

Additionally, under the BMM conjecture, decremental st-shortest paths with sensitivity 1 in directed unweighted graphs with n vertices and $m \geq n$ edges require either $m^{1-o(1)}\sqrt{n}$ preprocessing time or $m^{1-o(1)}/\sqrt{n}$ query time for every function m of n .

A strength of the reductions from BMM is that they can usually be extended to provide APSP-based reductions for weighted problems without the restriction to combinatorial algorithms; we do this in Section 3. While we state our results in the theorem only for combinatorial algorithms under the BMM conjecture, we would like to point out that they also hold for any kind of algorithm under a popular version of the triangle detection conjecture for sparse graphs that states that finding a triangle in an m -edge graph requires $m^{1+\delta-o(1)}$ time for some $\delta > 0$. Our lower bounds then rule out algorithms with a preprocessing time of $O(m^{1+\delta-\varepsilon})$ and update and query times $O(m^{2\delta-\varepsilon})$ for any $\varepsilon > 0$.

Many of the bullets of the theorem follow from prior work via a few observations, which we discuss in Appendix A.6. Our results on decremental diameter and eccentricity, however, are completely novel. In fact, it was completely unclear before this work whether such results are possible. Impagliazzo et al. [16] define a strengthening of SETH under which there can be no deterministic fine-grained reduction from problems such as APSP and BMM to problems such as orthogonal vectors or diameter in sparse graphs. It is not clear whether a reduction from BMM to diameter in dense graphs is possible, as the same “quantifier issues” that arise in the sparse graph case arise in the dense graph case as well: Diameter is an $\exists\forall$ -type problem (i.e., do there exist two nodes such that all paths between them are long?), and BMM is equivalent to Triangle detection which is an \exists -type problem (i.e., do there exist three nodes that form a clique?).

Decremental Diameter

We give the reduction from BMM to decremental diameter in undirected unweighted graphs with sensitivity 1. Note that the lower bound also holds for eccentricity oracles: Instead of querying the diameter n times, we can query the eccentricity of a variable vertex n times.

Let $G = (V, E)$ be an undirected unweighted graph for Triangle Detection. We construct a graph G' as follows.

We create four copies of V denoted by V_1, V_2, V_3, V_4 , and for $i = 1, 2, 3$, we add edges between nodes $u_i \in V_i$ and $v_{i+1} \in V_{i+1}$ if $(u, v) \in E$. We create vertices a_v and b_v for each $v \in V$, and denote the set of all a_v by A and the set of all b_v by B . We connect the vertices in A to a clique and also those of B . For each $v \in V$, we add an edge (v_1, a_v) and an edge (a_v, b_v) . A node b_v is connected to all vertices in V_4 . We further introduce two additional vertices c, d , which are connected by an edge. We add edges between c and all nodes in V_2 and V_3 , and between d and all nodes in V_3 and V_4 . The node c has an edge to each vertex in A and the node d has an edge to each vertex in B . Notice that the resulting graph has $O(n)$ vertices and $O(n^2)$ edges. We visualized the graph in Figure 2 in the appendix.

Note that even without the edges from $B \times V_4$, no pair of nodes has distance larger than 3, except for pairs of nodes from $V_1 \times V_4$. If a node v participates in a triangle in G , then in

G' there is a path of length 3 from v_1 to v_4 without an edge from $B \times V_4$. Otherwise, there is no such path, i.e., the diameter increases to 4 after the deletion of (b_v, v_4) .

We perform one stage per vertex $v \in V$: Consider the copy $v_4 \in V_4$ of v . We remove the edge (b_v, v_4) and query the diameter of the graph. We claim that G has a triangle iff one of the queries returns diameter 3.

► **Lemma 6.** *For each vertex v in G , the diameter of $G' \setminus \{(b_v, v_4)\}$ is larger than 3 if and only if v does not participate in a triangle in G .*

Proof. Assume that G has a triangle $(v, u, w) \in V^3$ and consider the stage for v . Notice that only the shortest paths change that used edge (b_v, v_4) ; this is not the case for any $z \neq v$, because the path $z_1 \rightarrow a_z \rightarrow b_z \rightarrow z_4$ is not affected by the edge deletion. We only need to consider the path $v_1 \rightarrow a_v \rightarrow b_v \rightarrow v_4$. Since G has a triangle (v, u, w) , there exists the path $v_1 \rightarrow u_2 \rightarrow w_3 \rightarrow v_4$ of length 3 as desired. Hence, the diameter is 3.

Assume the query in the stage for vertex $v \in V$ returned diameter 3. Since we deleted the edge (b_v, v_4) , there is no path of length 3 from v_1 to v_4 via A and B . Hence, the new shortest path from v_1 to v_4 must have the form $v_1 \rightarrow u_2 \rightarrow w_3 \rightarrow v_4$. By construction of the graph, this implies that G has a triangle (v, u, w) . ◀

Altogether we perform n queries and n updates. Thus under the BMM conjecture any combinatorial algorithm requires $n^{3-o(1)}$ preprocessing time or $n^{2-o(1)}$ update or query time.

3 Sensitivity Lower Bounds from the APSP Conjecture

In this section we present new lower bounds based on the APSP conjecture. These lower bounds hold for arbitrary, not necessarily combinatorial, algorithms. We present our results in the following theorem and give the proofs in Appendix A.7.

► **Theorem 7.** *Assuming the APSP conjecture, no algorithms can solve the following problems with preprocessing time $O(n^{3-\varepsilon})$, and update and query times $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$:*

1. *Decremental st -shortest paths in directed weighted graphs with sensitivity 1,*
2. *decremental st -shortest paths in undirected weighted with sensitivity 2,*
3. *decremental diameter in undirected weighted graphs with sensitivity 1.*

Decremental st -shortest paths in directed weighted graphs with sensitivity 1

In 2010, Vassilevska Williams and Williams [42] showed that the so called Replacement Paths (RP) problem is subcubically equivalent to APSP. RP is defined as follows: given a directed weighted graph G and two nodes s and t , compute for every edge e in G , the distance between s and t in $G \setminus \{e\}$. Note that only the deletion of the at most $n - 1$ edges on the shortest path from s to t affect the distance from s to t . This has an immediate implication for 1-sensitivity oracles for st -shortest paths: The APSP conjecture would be violated by any 1-sensitivity oracle that uses $O(n^{3-\varepsilon})$ preprocessing time and can answer distance queries between two fixed nodes s and t with one edge deletion in time $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$.

Decremental st -shortest paths in undirected weighted with sensitivity 2

With this we are able to show that on *undirected* weighted graphs finding a shortest path between fixed s and t with 2 edge deletions cannot be done with truly sub-cubic preprocessing time and truly subquadratic query time assuming the APSP conjecture. This is surprising

because in the case of a *single edge failure* Nardelli et al. [33] show that shortest paths can be solved with an essentially optimal $O(m\alpha(n))$ preprocessing time and $O(1)$ query time. Thus, assuming the APSP conjecture we show a separation between 1 sensitivity and 2 sensitivity. Additionally, with sensitivity 2 and no preprocessing time $O(n^2)$ update time is achievable, and with $\tilde{O}(n^3)$ preprocessing time using Nardelli et al. we can get an $O(1)$ query time. Thus, we show these approaches are essentially tight assuming the APSP conjecture. The full reduction is in Appendix 3.

Decremental diameter in undirected weighted graphs with sensitivity 1

A nice property of BMM-based reductions is that they can very often be converted to APSP-based reductions to weighted versions of problems. Here we convert our BMM-based reduction for decremental 1-sensitive Diameter to a reduction from APSP into decremental 1-sensitive diameter in undirected weighted graphs. Note that, as in the BMM case we can get the same lower bounds for eccentricity.

4 SETH Lower Bounds with Constant Sensitivity

In this section, we prove conditional lower bounds with constant sensitivity from SETH. Before we give the reductions, we first argue about what their sensitivities are.

► **Theorem 8.** *Let $\varepsilon > 0$, $t \in \mathbb{N}$. The SETH implies that there exists no algorithm with preprocessing time $O(n^t)$, update time $u(n)$ and query time $q(n)$, such that $\max\{u(n), q(n)\} = O(n^{1-\varepsilon})$ for the following problems:*

1. *Incremental #SSR with constant sensitivity $K(\varepsilon, t)$,*
2. *$(4/3 - \varepsilon)$ -approximate incremental diameter with constant sensitivity $K(\varepsilon, t)$,*
3. *incremental ST-Reach with constant sensitivity $K(\varepsilon, t)$.*

We prove the theorem in Appendix A.8. The parameter $K(\varepsilon, t)$ is explained in the following paragraph*.

The Sensitivity of the Reductions

The conditional lower bounds we prove from SETH hold even for constant sensitivity; however, the derivation of these constants is somewhat unnatural. Nonetheless, we stress that our lower bounds hold for constant sensitivity and in particular for every algorithm with sensitivity $\omega(1)$.

To derive the sensitivity of our reductions, we use a similar approach as Proposition 1 in [3], but we need a few more details. We start by revisiting the sparsification lemma.

► **Lemma 9** (Sparsification Lemma, [30]). *For $\varepsilon > 0$ and $k \in \mathbb{N}$, there exists a constant $C = C(\varepsilon, k)$, such that any k -SAT formula F with \tilde{n} variables can be expressed as $F = \bigvee_{i=1}^l F_i$, where $l = O(2^{\varepsilon\tilde{n}})$ and each F_i is a k -SAT formula with at most $C\tilde{n}$ clauses. This disjunction can be computed in time $O(2^{\varepsilon\tilde{n}} \text{poly}(\tilde{n}))$.*

We set $C(\varepsilon)$ to the smallest $C(\varepsilon, k)$, over all k such that k -SAT cannot be solved faster than in $O^*(2^{(1-\varepsilon)\tilde{n}})$ time²; formally, $C(\varepsilon) = \min\{C(\varepsilon', k) : \varepsilon' < \varepsilon \text{ and } k \in \mathbb{N} \text{ with } k\text{-SAT} \notin O^*(2^{(1-\varepsilon')\tilde{n}})\}$. Note that $C(\varepsilon)$ is well-defined if we assume that SETH is true (see also

² The $O^*(\cdot)$ notation hides $\text{poly}(\tilde{n})$ factors.

Proposition 1 in [3]). Finally, for any $\varepsilon > 0$ and $t \in \mathbb{N}$, we set $K(\varepsilon, t) = C(\varepsilon) \cdot t/(1 - \varepsilon)$, which gives the sensitivity of our reductions.

In our reductions, $t \in \mathbb{N}$ is the exponent of the allowed preprocessing time and $\varepsilon > 0$ denotes the improvement in the exponent of the running time over the 2^n -time algorithm. We note that $K(\varepsilon, t)$ gives a tradeoff: For small t (i.e., less preprocessing time), the lower bounds hold for smaller sensitivities; a smaller choice of ε yields larger sensitivities.

In the reductions we will write K to denote $K(\varepsilon, t)$ and c to denote $C(\varepsilon, k)$ whenever it is clear from the context.

The Reductions

Our reductions are conceptually similar to the ones in [3], but the graph instances we construct are based on a novel idea to minimize the size of the batch updates we need to perform. Here we describe the construction of the graphs we use in the reductions and refer to Appendix A.8 for full proofs.

We give two graphs, H_δ and D_δ , for $\delta \in (0, 1)$. For the construction, let F be a SAT formula over a set V of \tilde{n} variables and $c \cdot \tilde{n}$ clauses. Let $U \subset V$ be a subset of $\delta\tilde{n}$ variables.

Construction of H_δ : For each partial assignment to the variables in U we introduce a node. The set of these nodes is denoted by \bar{U} . For each clause of F we introduce a node and denote the set of these nodes by C . We add an edge between a partial assignment $\bar{u} \in \bar{U}$ and a clause $c \in C$ if \bar{u} does not satisfy c . Observe that H_δ has $O(2^{\delta\tilde{n}})$ vertices and $O^*(2^{\delta\tilde{n}})$ edges.

Construction of D_δ : We partition the set of clauses C into $K = c/\delta$ groups of size $\delta\tilde{n}$ each and denote these groups by G_1, \dots, G_K . For all groups G_i , we introduce a vertex into the graph for each non-empty subset g of G_i . The edges to and from the nodes of D_δ will be introduced during reductions. Observe that for each group we introduce $O(2^{\delta\tilde{n}})$ vertices and D_δ has $O(K \cdot 2^{\delta\tilde{n}})$ vertices in total.

Our reductions have small sensitivity since we will only need to insert a single edge from H_δ to each group of clauses in D_δ . Hence, we only need to insert $K = O(1)$ edges in order to connect H_δ and D_δ at each stage in the reduction. However, we will need to argue how we can efficiently pick the correct sets in D_δ .

5 Diameter Upper Bound

In this section, we present deterministic algorithms, which can compute a $(1+\varepsilon)$ -approximation for the eccentricity, the radius and the diameter of directed and undirected unweighted graphs after single edge deletions. All of these algorithms run in time truly subcubic time for graphs with a truly subquadratic number of edges.

Bernstein and Karger [10] give an algorithm for the related problem of all-pairs shortest paths in a directed weighted graph $G = (V, E)$ in case of single edge deletions. Their oracle data structure requires $\tilde{O}(mn)$ preprocessing time. Given a triplet $(u, v, e) \in V^2 \times E$, the oracle can output the distance from u to v in $G \setminus e$ in $O(1)$ time.

For the diameter problem with single edge deletions, note that only deletions of the edges in the shortest paths trees can have an effect on the diameter. Using this property, a trivial algorithm to compute the exact diameter after the deletion of a single edge works as follows: Build the oracle data structure of Bernstein and Karger [10]. For each vertex v , consider its shortest paths tree T_v . Delete each tree-edge once and query the distance from v to u in $G \setminus e$ for all vertices u in the subtree of the deleted tree-edge. By keeping track of the maximum distances, the diameter of G after a single edge deletion is computed exactly. As

there are $n - 1$ edges in T_v , we spend $O(n^2)$ time for each vertex. Thus, the trivial algorithm requires $O(n^3)$ time.

In this section, we improve upon this result as follows.

► **Theorem 10.** *Let $G = (V, E)$ be a directed or undirected unweighted graph with n vertices and m edges, let $\varepsilon > 0$, and let D be the diameter of G . There exists a data structure that given a single edge $e \in E$ returns for $G \setminus e$ in constant time (1) the diameter, (2) the radius, and (3) the eccentricity of any vertex $v \in V$ within an approximation ratio of $1 + \varepsilon$. It takes $\tilde{O}(n^{1.5} \sqrt{Dm/\varepsilon} + mn)$ preprocessing time to build this data structure.*

The rest of this section is devoted to the proof of the theorem. We give the proof of the theorem for directed graphs and point out the same proof also works for undirected graphs. We first describe how we can answer queries for the eccentricity of a fixed vertex $v \in V$ after a single edge deletion. After this, we explain how to extend this algorithm to solve the diameter and the radius problems after single edge deletions, and analyse the correctness and running time of the algorithm.

The data structure preprocesses the answers to all queries. Then queries can be answered via table lookup in $O(1)$ time.

Preliminaries

Let $G = (V, E)$ be a directed unweighted graph. For two vertices $u, u' \in V$ we denote the distance of u and u' in G by $d_G(u, u')$. For an edge $e \in E$ and vertices $u, u' \in V$, we denote the distance in the graph $G \setminus e$ by $d_{G \setminus e}(u, u')$. Given a tree T with root v and a tree-edge $e \in T$, we denote the subtree of T that is created when e is removed from T and that does not contain v by T_e . We let d_e be the height of T_e . A node u in T has level i , if $d_G(v, u) = i$.

Let $F \in \mathbb{N}$ be some suitably chosen parameter (see the last paragraph* of this section). Then given a tree T with root v , we call a tree-edge e *high*, if both of its endpoints have level less than F from v ; we call all other edges *low*. We denote the set of all high edges by $T_{<}$, i.e., $T_{<} = \{e = (w, w') \in T : d_G(v, w) < F, d_G(v, w') < F\}$; the set of all low edges is given by $T_{>}$.

The Algorithm

Our data structure preprocesses the answers to all queries, and then queries can be answered via table lookup in $O(1)$ time. The preprocessing has three steps: First, in the initialization phase, we compute several subsets of vertices that are required in the next steps. Second, we compute the eccentricity of v after the deletion of a high edge exactly. We compute it exactly, since after the deletion of a tree-edge high up in the shortest path tree T_v of v , the nodes close to v in T_v might “fall down” a lot. This possibly affects all vertices in the corresponding subtrees and, hence, we need to be careful which changes occur after deleting a high edge. On the other side, the relative distance of nodes which are “far away” from v in G before any edge deletion cannot increase too much. Thus, we simply estimate their new distances in the third step. More precisely, in the third step we compute a $(1 + \varepsilon)$ -approximation of the eccentricity of v after the deletion of a low edge.

Step 1: Initialization. We build the data structure of Bernstein and Karger [10], in mn time which for each triplet (u, v, e) can answer queries of the form $d_{G \setminus e}(u, v)$ in $O(1)$ time.

We compute the shortest path tree $T = T_v$ of v in time $O(nm)$ and denote its depth by d_v . By traversing T bottom-up, we compute the height d_e of the subtree T_e for each

tree-edge e ; this takes time $O(n)$. We further construct the sets $T_{<}$ and $T_{>}$ of high and low tree-edges, respectively.

Fix $\varepsilon > 0$. We construct a set $S_v \subset V$ as follows: First add v to S_v . Then add each $u \in V$ which has the following two properties: (1) u is at level $i\varepsilon F$ for some integer³ $i > 0$ and (2) there exists a node u' in the subtree of u in T_v , such that u' has distance $\varepsilon F/2$ in T_v from u . Note that we can add the root, but every other node we add can be charged to the $\varepsilon F/2$ parent nodes that come before it. Thus, we can have at most $1 + \frac{2n}{\varepsilon F}$ nodes in S_v . Note that for every $z \in V$ there exists a $y \in S_v$, s.t. y is an ancestor of z in T_v and there exists a path from y to z in T_v of length at most εF .

Using a second bottom-up traversal of T , for each tree-edge $e \in T$, we compute the set $S_e = T_e \cap S_v$, i.e., the intersection of the vertices in T_e and those in S_v . This can be done in $O(n)$ time by, instead of storing S_e explicitly for each edge $e = (w, w')$, storing a reference to the set containing the closest children of w' which are in S_v ; then S_e can be constructed in $O(|S_e|)$ time by recursively following the references.

For each non-tree-edge e , we store d_v as the value for the eccentricity of v when e is deleted.

Step 2: Handling high edges. For each level $j = 1, \dots, F - 1$, we proceed as follows. We consider each tree-edge $e = (w, w') \in T_{<}$ with $d(v, w) < d(v, w') = j$, there are at most n of these. We build a graph G_e containing all nodes of T_e together with a additional directed path P of length $d_e + 4$ with startpoint r . The nodes in P are new vertices added to G_e . Each edge on P has weight 1, except the single edge incident to r , which has weight $d_G(v, w') - 1$. Additionally to the path, the graph contains as edges: (1) all edges from E , which have both endpoints in T_e , and (2) for each $e = (z, z')$ which has its startpoint $z \notin T_e$ and its endpoint $z' \in T_e$, an edge (z'', z') , where z'' is the node on P with distance $d_G(v, z)$ from r .

Observe that G_e has the property that all distances after the deletion of e are maintained *exactly*: By construction, the shortest path from r to $u \in T_e$ in G_e has exactly length $d_{G \setminus e}(v, u)$ (we prove this formally in Lemma 11).

After building G_e , we compute its depth starting from node r and store this value for edge e .

Step 3: Handling low edges. For each tree-edge $e = (w, w') \in T_{>}$, we do the following: Let $S = S_e \cup \{w'\}$. As answer for a deleted edge e , we store $\max\{d_v, (1 + \varepsilon) \max_{y \in S} d_{G \setminus e}(v, y)\}$. To determine $d_{G \setminus e}(v, y)$, we use the data structure of [10].

Extension to $(1 + \varepsilon)$ -approximate Diameter and Radius. We repeat the previously described procedure for all $v \in V$ (but we build the data structure for Bernstein and Karger only once). To compute the diameter, we keep track of the maximum value we encounter for each deleted edge e . To compute the radius, we keep track of the minimum value we encounter for each deleted edge e .

Correctness

Observe that it is enough to show correctness for a fixed $v \in V$. We first prove the correctness of the algorithm after removing high edges.

► **Lemma 11.** *After the deletion of a high edge $e \in T_{<}$, we compute the eccentricity of v exactly.*

³ For readability we leave out the floors, however, we are considering the integer levels $\lfloor i\varepsilon F \rfloor$.

Proof. Consider any vertex $u \in T_e$ and consider the shortest path p from v to u in $G \setminus e$.

We can assume that p has exactly one edge (z, z') , s.t. $z \notin T_e$ and $z' \in T_e$: Assume that there is a path p' with two edges (x, x') and (y, y') , s.t. $x, y \notin T_e$, $x', y' \in T_e$ and y appears later on p' than x . Since $y \notin T_e$, there exists a path from v to y that does not use any vertex from T_e and that is of the same length as the subpath of p' from v to y in T , because T is a shortest-path tree with root v . Hence, we can choose a path to y without entering T_e .

Let z, z' be as before. Then $d_{G \setminus e}(v, u) = d_{G \setminus e}(v, z) + 1 + d_G(z', u)$. By construction of G_e , there exists a vertex on P in G_e with distance $d_{G \setminus e}(v, z)$ from r and which has an edge to z' . All paths which only traverse vertices from T_e are unaffected by the deletion of e . Hence, in G_e there exists a path of length $d_{G \setminus e}(v, u)$.

Also, there is no shorter path in G_e from r to u , because this would imply a shorter path in $G \setminus e$ by construction. \blacktriangleleft

Next we prove the correctness of the algorithm after the removal of low edges. Consider a tree-edge $e = (w, w') \in T_{>}$ with $F \geq d(v, w') > d(v, w)$. Let $S = S_e \cup \{w'\}$.

► **Lemma 12.** *For each node $z \in T_e$, there exists a vertex $y \in S$ s.t. $d_{G \setminus e}(y, z) \leq \varepsilon F$.*

Proof. Since $w' \in S$, the claim is true for all nodes $z \in T_e$ with $d(w', z) \leq \varepsilon F$. By construction of S_v , any (directed) tree path of length εF contains a node of S_v . For any node $z \in T_e$ with $d(w', z) > \varepsilon F$, there exists an ancestor u of z in T_e with $d(u, z) \leq \varepsilon F$. The path from u to z is a directed tree path and, thus, must contain a node in S_v . Thus, for each node in T_e there is a path of length at most εF from some node in S_v . \blacktriangleleft

► **Lemma 13.** *Consider two vertices $y, z \in T_e$ and assume there exists a path from y to z in $G \setminus e$. Then $d_{G \setminus e}(y, z) \leq X$ implies $d_{G \setminus e}(v, z) \leq d_{G \setminus e}(v, y) + X$.*

Proof. Concatenate the shortest paths from v to y in $G \setminus e$ and from y to z in $G \setminus E$, which both avoid e . This path cannot be shorter than the shortest path from v to z in $G \setminus e$. \blacktriangleleft

We define the maximum height achieved by the vertices of T_e in $G \setminus e$ by

$$n(v, e) = \max_{z \in T_e} d_{G \setminus e}(v, z).$$

Notice that the eccentricity of v in $G \setminus e$ is given by $\max\{d_v, n(v, e)\}$. Hence, by giving a $(1 + \varepsilon)$ -approximation of $n(v, e)$, we obtain a $(1 + \varepsilon)$ -approximation for the eccentricity of v in $G \setminus e$. In the remainder of this subsection, we show this guarantee on the approximation ratio of $n(v, e)$.

► **Lemma 14.** $n(v, e) \leq (1 + \varepsilon) \max_{y \in S} d_{G \setminus e}(v, y)$.

Proof. Let z' be any vertex in T_e such that $d_{G \setminus e}(v, z') = n(v, e)$. By Lemma 12 there exists a vertex $y' \in S$ with $d_{G \setminus e}(y', z') \leq \varepsilon F$. Then by Lemma 13,

$$\begin{aligned} n(v, e) &= d_{G \setminus e}(z', v) \\ &\leq d_{G \setminus e}(v, y') + \varepsilon F \\ &\leq \max_{y \in S} d_{G \setminus e}(v, y) + \varepsilon d_G(v, w') \\ &\leq (1 + \varepsilon) \max_{y \in S} d_{G \setminus e}(v, y), \end{aligned}$$

where in the second last step we used $F \leq d_G(v, w')$ and in the last step we used that $w' \in S$. \blacktriangleleft

► **Lemma 15.** $(1 + \varepsilon) \max_{y \in S} d_{G \setminus e}(v, y) \leq (1 + \varepsilon)n(v, e)$.

Proof. This follows from the definition of $n(v, e)$, since S is a subset of the vertices in T_e . \blacktriangleleft

Running Time Analysis.

Let us first consider the time spent on step 1, preprocessing. We build the data structure of Bernstein and Karger [10] in time $\tilde{O}(mn)$. For each node v , computing the shortest path tree of v takes time $\tilde{O}(m)$ and we spend time $O(n)$ computing the heights of the subtrees of T and computing the sets $T_{<}$, $T_{>}$, S_v . The sets S_e can as well be computed in $O(n)$ time by storing them only implicitly.

Now let us consider the time spent on step 2, the high edges. For the high edges e at level $j \leq F$, observe that the trees T_e are mutually disjoint. Hence, for a fixed level j , in time $\tilde{O}(m)$ we can compute the depths of *all* graphs G_e with e at level j . Since we have to do this for each level less than F , the total time for this step is $\tilde{O}(Fm)$.

Finally, let us consider the time spent on step 3, the low edges. For all low edges at level $j > F$, we query all nodes of S_v with height more than j . These are $O(\frac{n}{\varepsilon F})$ many such nodes. Thus, the total time we spend for all edges in T is $O(d_v \cdot \frac{n}{\varepsilon F})$.

To compute the diameter, we have to execute the above steps once for each $v \in V$, but we only need to build the data structure of Bernstein and Karger once. Hence, the total time is $\tilde{O}(mn + Fmn + nd_v \cdot \frac{n}{\varepsilon F})$. Denote the diameter of G by D . Then setting $F = \sqrt{\frac{Dn}{\varepsilon m}}$ yields a total running time of $\tilde{O}(n^{1.5} \sqrt{Dm/\varepsilon} + mn)$. Since $D \leq n$, this is $\tilde{O}(n^2 \sqrt{m/\varepsilon})$.

References

- 1 Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *FOCS*, 2016.
- 2 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *SODA*, pages 1681–1697, 2015.
- 3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE, 2014.
- 4 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *STOC*, pages 41–50, 2015.
- 5 V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. *Soviet Math. Dokl.*, 11:1209–1210, 1970.
- 6 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant reachability for directed graphs. In *DISC*, pages 528–543, 2015.
- 7 Surender Baswana, Keerti Choudhary, and Liam Roditty. Fault tolerant subgraph for single source reachability: Generic and optimal. In *STOC*, pages 509–518, 2016.
- 8 Surender Baswana and Neelesh Khanna. Approximate shortest paths avoiding a failed vertex: Near optimal data structures for undirected unweighted graphs. *Algorithmica*, 66(1):18–50, 2013.
- 9 Surender Baswana, Utkarsh Lath, and Anuradha S. Mehta. Single source distance oracle for planar digraphs avoiding a failed node or link. In *SODA*, pages 223–232, 2012.
- 10 Aaron Bernstein and David Karger. A nearly optimal oracle for avoiding failed vertices and edges. In *STOC*, pages 101–110, 2009.
- 11 Davide Bilò, Fabrizio Grandoni, Luciano Gualà, Stefano Leucci, and Guido Proietti. Improved purely additive fault-tolerant spanners. In *ESA*, 2015.
- 12 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Fault-tolerant approximate shortest-path trees. In *ESA*, 2014.
- 13 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Compact and fast sensitivity oracles for single-source distances. In *ESA*, 2016.

- 14 Davide Bilò, Luciano Gualà, Stefano Leucci, and Guido Proietti. Multiple-edge-fault-tolerant approximate shortest-path trees. In *STACS*, pages 18:1–18:14, 2016.
- 15 Gilad Braunschvig, Shiri Chechik, and David Peleg. Fault tolerant additive spanners. In *WG*, 2012.
- 16 Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *ITCS*, pages 261–270, 2016.
- 17 T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. In *STOC*, pages 590–598, 2007.
- 18 Shiri Chechik, Sarel Cohen, Amos Fiat, and Haim Kaplan. $1 + \varepsilon$ -approximate f -sensitive distance oracles. In *SODA*, 2017.
- 19 Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. f -sensitivity distance oracles and routing schemes. *Algorithmica*, 63(4):861–882, 2012.
- 20 Keerti Choudhary. An optimal dual fault tolerant reachability oracle. In *ICALP*, pages 130:1–130:13, 2016.
- 21 Søren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *ICALP*, pages 48:1–48:14, 2016.
- 22 Ran Duan and Seth Pettie. Dual-failure distance and connectivity oracles. In *SODA*, pages 506–515, 2009.
- 23 Ran Duan and Seth Pettie. Connectivity oracles for failure prone graphs. In *STOC*, pages 465–474, 2010.
- 24 Ran Duan and Seth Pettie. Connectivity oracles for graphs subject to vertex failures. In *SODA*, 2017.
- 25 François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, pages 296–303, 2014.
- 26 Fabrizio Grandoni and Virginia Vassilevska Williams. Improved distance sensitivity oracles via fast single-source replacement paths. In *FOCS*, pages 748–757, 2012.
- 27 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30, 2015.
- 28 Monika Henzinger and Stefan Neumann. Incremental and fully dynamic subgraph connectivity for emergency planning. In *ESA*, 2016.
- 29 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
- 30 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- 31 A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.
- 32 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *SODA*, pages 1272–1287, 2016.
- 33 E. Nardelli, G. Proietti, and P. Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.
- 34 Merav Parter. Dual failure resilient BFS structure. In *PODC*, pages 481–490, 2015.
- 35 Merav Parter and David Peleg. Fault tolerant approximate bfs structures. In *SODA*, pages 1073–1092, 2014.
- 36 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610, 2010.
- 37 Mihai Patrascu and Mikkel Thorup. Planning for fast connectivity updates. In *FOCS*, pages 263–271, 2007.

- 38 L. Roditty and U. Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. In *ICALP*, pages 249–260, 2005.
- 39 Liam Roditty and Uri Zwick. Replacement paths and k simple shortest paths in unweighted directed graphs. *ACM Trans. Algorithms*, 8(4):33, 2012.
- 40 Virginia Vassilevska Williams. Faster replacement paths. In *SODA*, pages 1337–1346, 2011.
- 41 Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.
- 42 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654, 2010.

A Appendix

A.1 Definitions of the Problems

We give definitions of the problems we consider in this paper in Table 1.

A.2 A Note on Terminology

The terminology used in the literature for dynamic data structures in the spirit of Section 1 is not consistent. The phrases which are used contain “fault-tolerant algorithms”, “algorithms with sensitivity”, “algorithms for emergency planning” and “algorithms for failure prone graphs”.

In the community of spanners and computational graph theory, it is common to speak about “fault-tolerant subgraphs”. In this area, this term is used consistently.

In the dynamic graph algorithms community, multiple phrases have been used to describe algorithms for the model proposed in Section 1. First, the field was introduced by [37] as algorithms for “emergency planning”. Later, the terminologies “sensitivity” and “failure prone graphs” were used (e.g., [19, 18, 23, 24]). When the number of failures in the graph was fixed (e.g. 1 or 2), then often this was stated explicitly (without further mentioning sensitivity or failure prone graphs). However, it appears that in the dynamic graph algorithms community the phrase “sensitivity” is the most widely used one.

A.3 A Note on Adversaries

Some of the conditional lower bounds we obtain are for *randomized* algorithms. Previous literature [3, 27, 32] also gave conditional lower bounds for randomized dynamic algorithms; however, it was not discussed under which kind of adversary the obtained lower bounds hold. This depends on the conjecture from which the lower bound was obtained. We observe that in reductions from the static triangle problem, the only randomness is over the input distribution of the static problem. Hence, for lower bounds from the triangle conjecture, we can assume an oblivious adversary. Furthermore, we assume the OMv conjecture in its strongest possible form, i.e. for oblivious adversaries. (In [27] the authors did not explicitly state which kind of adversary they assume for their conjecture.) Thus, all conditional lower bounds we obtain for randomized algorithms hold for oblivious adversaries. Note that a lower bound which holds for oblivious adversaries must always hold for non-oblivious ones.

We would like to point out another subtlety of our lower bounds: In reductions from the triangle detection conjecture, the running time of the algorithm is assumed to be a random variable, but the algorithm must always answer correctly. However, in reductions from OMv the running time of the algorithm is deterministic, but the probability of obtaining a correct answer must be at least $2/3$.

■ **Table 1** The problems we consider in this paper.

Problem		
Maintain	Update	Query
Reachability		
Directed graph	Edge insertions/deletions	Given two vertices u, v , can v be reached from u ?
#SSR		
Directed graph and a fixed source vertex s .	Edge insertions/deletions	How many vertices can be reached from s ?
Strong Connectivity (SC)		
Directed graph	Edge insertions/deletions	Is the graph strongly connected?
2 Strong Components (SC2)		
Directed graph	Edge insertions/deletions	Are there more than 2 SCCs?
2 vs k Strong Components (AppxSCC)		
Directed graph	Edge insertions/deletions	Is the number of SCCs 2 or more than k ?
Maximum SCC Size (MaxSCC)		
Directed graph	Edge insertions/deletions	What is the size of the largest SCC?
Subgraph Connectivity		
Fixed undirected graph, with some vertices on and some off.	Turn on/off vertex	Given two vertices u, v , are u and v connected by a path only traversing vertices that are on?
α -approximate Shortest Paths		
Directed or undirected (possibly weighted) graph	Edge insertions/deletions	Given two vertices u, v , return an α -approximation of the length of the shortest path from u to v .
α -approximate Eccentricity		
Undirected graph	Edge insertions/deletions	Given a vertex u , return an α -approximation of the eccentricity of v .
α -approximate Radius		
Undirected graph	Edge insertions/deletions	Return an α -approximation of the radius of the graph.
α -approximate Diameter		
Undirected graph	Edge insertions/deletions	Return an α -approximation of the diameter of the graph.
Bipartite Perfect Matching (BPMatch)		
Undirected bipartite graph	Edge insertions/deletions	Does the graph have a perfect matching?
Bipartite Maximum Weight Matching (BWMatch)		
Undirected bipartite graph with integer edge weights	Edge insertions/deletions	Return the weight of the maximum weight perfect matching.

A.4 Our Lower Bounds

In Table 2 we summarize our lower bounds for sensitivity data structures. Table 3 states our lower bounds for static oracle data structures.

A.5 Existing Sensitivity Data Structures

In Table 4 we summarize existing sensitivity data structures.

In the table, we also list algorithms for “fault-tolerant subgraphs” although they are not algorithms for the sensitivity setting in the classical sense. However, the fault-tolerant subgraphs are often much smaller than the input graphs and by traversing the fault-tolerant subgraph during queries, one can obtain better query times than by running the static algorithm on the original graph. Unfortunately, the construction time of these subgraphs is often very expensive, though still polynomial; the goal of these papers is to optimize the trade-offs between the size of the subgraphs and the approximation ratios achieved for the specific problem.

It is striking that (to the best of our knowledge) most of the existing algorithmic work was obtained for the case of *decremental* algorithms with a limited number of failures. While this is natural for the construction of fault-tolerant subgraphs, this is somewhat surprising from an algorithmic point of view. Our lower bounds might give an explanation of this phenomenon as they indicate that for many problems there is a natural bottleneck when it comes to the insertion of edges.

A.6 Triangle Detection Proofs

We provide full details of the reachability and shortest paths sensitivity results from Theorem 5.

Reachability

Let $G = (V, E)$ be an undirected unweighted graph for Triangle Detection. We create four copies of V denoted by V_1, V_2, V_3, V_4 , and for $i = 1, 2, 3$, we add edges between nodes $u_i \in V_i$ and $v_{i+1} \in V_{i+1}$ if $(u, v) \in E$.

For a fixed source $s \in V$ and sink $t \in V$, Abboud and Williams [3] give the following reduction: For each vertex $v \in V$, they insert the edges (s, v_1) and (v_4, t) , and query if there exists a path from s to t . They show that there exists a triangle in G iff one of the queries is answered positively. We observe that this reduction requires n batch updates of size 2 and n queries. Hence, it holds for sensitivity 2.

Now keep s fixed, but remove the sink t , and allow single-source reachability queries⁴. We perform a stage for each $v \in V$, in which we add the single edge (s, v_1) and query if there exists a path from s to v_4 . By the same reasoning as before, there exists a triangle in G iff one of the queries returns true. The reduction requires n updates of size 1 and n queries. Thus, it has sensitivity 1.

Finally, we remove the source node s and ask all-pairs reachability queries⁵. We perform a stage for each $v \in V$, which queries if there exists a path from v_1 to v_4 . There exists a triangle in G iff one of the queries returns true. This reduction has sensitivity 0, i.e., it uses

⁴ Given $v \in V$, a query returns true iff there exists a path from s to v .

⁵ Given two nodes $u, v \in V$, a query returns true iff there exists a path from u to v .

■ **Table 2** The conditional lower bounds we obtained for non-zero sensitivity. Problems for which there exists a tight upper bound are marked bold. Regarding the sensitivities, the lower bounds hold for any data structure that supports *at least* the sensitivity given in the table; d is a parameter that can be picked arbitrarily, and $K(\varepsilon, t)$ is a constant depending on properties of SAT and the allowed preprocessing time (see Section 4). Lower bounds for constant sensitivities hold in particular for any dynamic algorithm which allows for *any* larger fixed constant sensitivity or sensitivity $\omega(1)$. For the query type we use the following abbreviations: *st* – fixed source and sink, *ss* – single source, *ap* – all pairs, *ST* – a fixed set of sources and a fixed set of sinks. The rest of the abbreviations are as follows: *sh.* paths means shortest paths, *conn.* means connectivity, *SC* means strongly connected components, *SC2* means whether the number of strongly connected components is more than 2, *Reach.* means Reachability, *BPMatch* is bipartite matching, *BWMatch* is bipartite maximum weight matching, *ecc.* is eccentricity, *dir.* means directed, *und.* means undirected, *w.* means weighted, *unw.* means unweighted, *Conj.* means Conjecture.

Problem	Inc/Dec	Query Type	Lower Bounds			Sens.	Conj.	Cite
			$p(m, n)$	$u(m, n)$	$q(m, n)$			
Reachability Reach., SC, BPMatch	Inc	<i>ss</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	BMM	Theorem 5
		<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	2	BMM	[3]
(3/2 - ε)-sh. paths (und. unw.) (7/5 - ε)-sh. paths (und. unw.)	Inc	<i>ss</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	BMM	Theorem 5
		<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	2	BMM	Theorem 5
#SSR	Inc	<i>ss</i>	$\mathbf{n^t}$	$\mathbf{m^{1-\varepsilon}}$	$\mathbf{m^{1-\varepsilon}}$	$K(\varepsilon, t)$	SETH	Lemma 18
reachability, (4/3 - ε)-diameter for sparse graphs	Inc	<i>ST</i>	n^t	$n^{1-\varepsilon}$	$n^{1-\varepsilon}$	$K(\varepsilon, t)$	SETH	Lemmas 20 and 19
		-	$\text{poly}(n)$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\omega(\log n)$	SETH	[3]
SC2, AppxSCC, and MaxSCC	Inc	-	$\text{poly}(n)$	$\mathbf{m^{1-\varepsilon}}$	$\mathbf{m^{1-\varepsilon}}$	$\omega(\log n)$	SETH	[3]
subgraph conn. (\implies reachability, BPMatch, SC)	Inc	<i>st</i>	$\text{poly}(n)$	$\text{poly}(d)$	$\mathbf{d^{1-\varepsilon}}$	d	OMv	Theorem 21
			$n^{2-\varepsilon}$	$n^{1-\varepsilon}$	$d^{1-\varepsilon}$	d	3SUM	[32]
(2 - ε)-sh. paths (5/3 - ε)-sh. paths (\implies BWMatch)	Inc	<i>ss</i>	$\text{poly}(n)$	$\text{poly}(d)$	$d^{1-\varepsilon}$	d	OMv	Theorem 21
	Inc	<i>st</i>	$\text{poly}(n)$	$\text{poly}(d)$	$d^{1-\varepsilon}$	d	OMv	Theorem 21
diameter (4/3 - ε), und. unw. dir. & und. w.	Dec	-	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	BMM	Theorem 5
	Dec	-	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	APSP	Section 3
(4/3 - ε)-ecc.	Dec	-	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	BMM	Theorem 5
weighted-ecc.	Dec	-	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	APSP	Lemma 17
shortest paths dir. w. und. w.	Dec	<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	1	APSP	[42]
	Dec	<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	2	APSP	Section 3
reachability (\implies SC, BPMatch)	Dec	<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\Omega(\log n)$	BMM	[3]
			$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\Omega(\log n)$	BMM	[3]
shortest paths (undir. unw.)	Dec	<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\mathbf{n^{2-\varepsilon}}$	$\Omega(\log n)$	BMM	Theorem 5
subgraph conn. (\implies reachability, BPMatch, SC)	Dec	<i>st</i>	$\text{poly}(n)$	$\text{poly}(d)$	$\mathbf{d^{1-\varepsilon}}$	d	OMv	Theorem 21
			$n^{2-\varepsilon}$	$n^{1-\varepsilon}$	$d^{1-\varepsilon}$	d	3SUM	[32]
(2 - ε)-sh. paths (5/3 - ε)-sh. paths (\implies BWMatch)	Dec	<i>ss</i>	$\text{poly}(n)$	$\text{poly}(d)$	$d^{1-\varepsilon}$	d	OMv	Theorem 21
	Dec	<i>st</i>	$\text{poly}(n)$	$\text{poly}(d)$	$d^{1-\varepsilon}$	d	OMv	Theorem 21

■ **Table 3** The conditional lower bounds we obtained for static oracle data structures, i.e., data structures with zero sensitivity. Problems for which there exists a tight upper bound are marked bold. The query type “ap” denotes all pairs queries, Repl. means replacement, the rest of the abbreviations are as in Table 1.

Problem	Inc/Dec/ Static	Query Type	Lower Bounds			Conj.	Cite
			$p(m, n)$	$u(m, n)$	$q(m, n)$		
Reach.	static	<i>ap</i>	$\mathbf{n^{3-\varepsilon}}$	-	$\mathbf{n^{2-\varepsilon}}$	BMM	Theorem 5
(5/3 - ε)-sh. paths	static	<i>ap</i>	$\mathbf{n^{3-\varepsilon}}$	-	$\mathbf{n^{2-\varepsilon}}$	BMM	Theorem 5
Repl. paths (1 edge fault) (dir. w.)	static	<i>st</i>	$\mathbf{n^{3-\varepsilon}}$	-	$\mathbf{n^{2-\varepsilon}}$	APSP	Section A.7

no updates, and n queries. Hence, we have derived a very simple conditional lower bound for static reachability oracles.

These reductions prove the first three results of Theorem 5.

Shortest Paths

The above reduction for *st*-reachability can be easily altered to work for $(7/5 - \varepsilon)$ -approximate *st*-shortest paths in *undirected* unweighted graphs (for any $\varepsilon > 0$) with the same running time lower bounds: Just observe that the graphs in the reduction are bipartite. Thus, either there is a path from s to t of length 5 and there is a triangle in the original graph, or the shortest path between s and t has length at least 7. Thus distinguishing between length 7 and 5 solves the triangle problem.

To obtain a lower bound for $(3/2 - \varepsilon)$ -approximate *ss*-shortest paths, we take the construction for *ss*-reachability and again observe that the graph is bipartite so that if there is no path of length 4 between s and a node $v \in V$, then the shortest path between them must have length at least 6.

With the same bipartiteness observation, we obtain a conditional lower bound for $(5/3 - \varepsilon)$ -approximate static *ap*-shortest paths. In a stage for node $v \in V$, we query the shortest path from v_1 to v_4 . The query returns 3 if there exists a triangle in the original graph containing the vertex v and ≥ 5 otherwise. Thus, distinguishing between 3 and ≥ 5 suffices to solve the problem.

Finally, let us discuss how we obtain a lower bound for incremental shortest paths with sensitivity 1 in directed graphs. Vassilevska Williams and Williams [42] reduce BMM to the replacement paths problem in directed unweighted graphs: given a directed graph G on m edges and n nodes and two nodes s and t , compute for every $e \in E$, the distance between s and t in $G \setminus \{e\}$. [42] shows that if a combinatorial algorithm can solve the latter problem in $O(mn^{1/2-\varepsilon})$ time for any $\varepsilon > 0$ (for any choice of m as a function of n), then BMM has a truly subcubic combinatorial algorithm. This showed that the $O(m\sqrt{n})$ time algorithm of Roditty and Zwick [39] is tight.

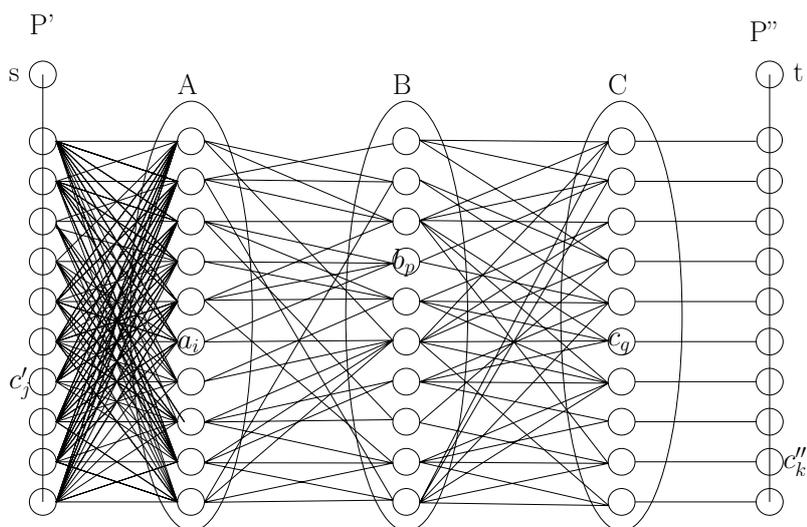
Here we observe that the [42] reduction immediately implies a 1-sensitivity oracle lower bound, as any 1-sensitivity oracle for *st* distances must be able to answer the replacement paths problem by querying the less than n nodes on the shortest *s-t* path: either the preprocessing time is at least $mn^{0.5-o(1)}$ or the query time is at least $m/n^{0.5+o(1)}$. For dense graphs this gives a lower bound of either $n^{2.5-o(1)}$ preprocessing or $n^{1.5-o(1)}$ query time. The lower bound is again tight via Roditty and Zwick’s algorithm [39] for replacement paths.

Table 4 Upper Bounds. We omit polylog factors in the stated running times and spaces usages. We use the following abbreviations: “ap” means “all pairs”, “ss” means “single source”, “st” denotes problems with a fixed source and a fixed sink. Oracles combine update and query into a single operation. For algorithms with an additive approximation guarantee, we included the optimal result \mathcal{O} ; all other approximation algorithms achieve multiplicative approximation guarantees. See table 5 for APSP upper bounds.

Problem	Approx.	Space	Upper Bounds	Sensi- tivity	Ref.	Remark
		$p(m, n)$	$u(m, n)$	$q(m, n)$		
Dec ap-Connectivity		n	$\text{poly}(n)$	d	[37]	
Dec ap-SubgraphConn		$d^{1-2/c}mn^{1/c}$	$d^{1-2/c}mn^{1/c}$	d^{4+2c}	[23]	Any $c \in \mathbb{N}$ can be picked; space simplified.
Inc ap-SubgraphConn		dm	mn	d^3	[24]	Deterministic.
		m	mn	d^2	[24]	Randomized.
Fully Dynamic ap-SubgraphConn		n^2	n^3	d^2	[28]	
		n^2m	n^3m	d^4	[28]	Uses [24] as a blackbox.
Dec ss-Reachability (implies SCC, dominator tree)		n	m	1	[6]	Subgraph.
		n	n	1	[9]	Oracle. Planar graph.
		n	$\text{poly}(n)$	1	[20]	Allows for vertex failures.
		$2^d n$	$2^d mn$	1	[7]	Subgraph. Reasonable for $d = o(\log(m/n))$.
Dec ss-SP undirected unweighted	3	n	m	1	[8]	Oracle.
	$1 + \varepsilon$	n/ε^3	1	1	[8]	Oracle.
directed unweighted undirected weighted	exact	$n^{5/3}$	$\text{poly}(n)$	1	[34]	More robust BFS tree.
	exact	n^2	n^ω	1	[26]	Algorithm and oracle.
	$1 + \varepsilon$	$m + n/\varepsilon$	mn	$1/\varepsilon$	[13]	Oracle.
	2	m	mn	1	[13]	Oracle.
	$2\mathcal{O} + 1$	dn	dm	d^2	[14]	Oracle.

■ **Table 5** Upper Bounds for APSP. We omit polylog factors in the stated running times and spaces usages. For algorithms with an additive approximation guarantee, we included the optimal result \mathcal{O} ; all other approximation algorithms achieve multiplicative approximation guarantees.

Problem	Approx.	Space	Upper Bounds		Sensi- tivity	Ref.	Remark
			$p(m, n)$	$u(m, n)$	$q(m, n)$		
Dec APSP unweighted undirected	$\mathcal{O} + 2$	$n^{5/3}$	$\text{poly}(n)$	1	$n^{5/3}$	[11]	Additive spanner.
	$\mathcal{O} + 4$	$n^{3/2}$	$\text{poly}(n)$	1	$n^{3/2}$	[11]	Additive spanner.
	$\mathcal{O} + 10$	$n^{7/5}$	$\text{poly}(n)$	1	$n^{7/5}$	[11]	Additive spanner.
	$\mathcal{O} + 14$	$n^{4/3}$	$\text{poly}(n)$	1	$n^{4/3}$	[11]	Additive spanner.
	$(2k - 1)(1 + \varepsilon)$	$kn^{1+(k\varepsilon^4)^{-1}}$			k	[8]	Oracle. Any $k > 1$ and $\varepsilon > 0$.
non-negative weights, undirected	3	n	$\text{poly}(n)$	1	n	[35]	Spanner.
	$3(d+1)\mathcal{O} + (d+1)\log n$	dn	$\text{poly}(n)$	1	dn	[35]	Spanner.
	$1 + \varepsilon$	n/ε^2	$\text{poly}(n)$	1	n/ε^2	[12]	Spanner. Vertex and edge deletions.
weighted, undirected	$(8k + 2)(d + 1)$	$dkn^{1+1/k}$	$\text{poly}(n)$		d	[19]	Oracle. Any $k \in \mathbb{N}$.
	$1 + \varepsilon$	$dn^2(\log n/\varepsilon)^d$	$dn^{5 \log(n/\varepsilon)^d}$		d^5	[18]	Oracle.
weighted, directed	exact		$Mn^{2.88}$		$n^{0.7}$	[26]	Oracle. Weights: $\{-M, \dots, M\}$. Simplified running times.
	exact	n^2	mn		1	[10]	Oracle.
	exact	n^2	$\text{poly}(n)$		1	[22]	Oracle.
	$\mathcal{O} + d$	$dn^{4/3}$	$\text{poly}(n)$	1	$n^{4/3}$	[15]	Additive spanner.



■ **Figure 1** The graph G .

A.7 All Pairs Shortest Paths Proofs

We prove the statements of Theorem 7 in Lemma 16 and Lemma 17.

► **Lemma 16.** *Assuming the APSP conjecture, decremental st -shortest paths in undirected weighted graphs with sensitivity 2 cannot be solved with preprocessing time $O(n^{3-\varepsilon})$, and update and query times $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$.*

Proof. We use a reduction similar to the reduction from APSP to RP from [42], but to deal with the undirected edges we add more weights and we add additional nodes to the graph. As in [42], we start by taking an instance of APSP and turning it into a tripartite graph for the negative triangle detection problem⁶; denote resulting graph H' .

If H' has no negative edge weights, we are done (there are no negative triangles). If H' has negative edge weights, let $M = \min\{w(e) \mid e \in E_{H'}\}$ and add $-M + 1$ to all edges (thus making all edges have positive weights). Now we want to detect if there is a triangle with (positive) weight less than $-3M + 3$. Denote the new graph by H and denote the three tripartite groups A , B and C ; each set A , B and C has n nodes.

We construct a graph G in which n shortest paths queries with two edge deletions determine if there exist any triangles with weight less than $-3M + 3$ in H . Let $W = 4 \max\{w(e) \mid e \in E_H\}$ and observe that W is larger than the maximum possible difference in the weight of two triangles. We will use this weight to enforce that we must take certain paths.

We add two vertices s and t to G . We add a path P' of length n to s , where each edge on the path has weight 0; the first node after s on P' is denoted c'_1 , the next node on P' is denoted c'_2 , and the i 'th node on P' is denoted c'_i . Next, we add a path P'' of length n to t , where each edge on the path has weight 0; the first node after t on P'' is denoted c''_n , second node on P'' is denoted c''_{n-1} and the i 'th node away from t on P'' is denoted c''_{n-i+1} . We add the nodes in A , B and C from H to G and keep all edges from $A \times B$ and from $B \times C$,

⁶ In the negative triangle detection problem we are given an edge-weighted graph $G = (V, E)$ with possibly negative edge-weights from \mathbb{Z} , and we must determine if G contains a triangle consisting of vertices u, v, x such that $w(u, v) + w(v, x) + w(x, u) < 0$.

however, we delete all edges from $A \times C$. We increase the weight of all edges from A to B and of all edges from B to C by $6nW$. We add edges between all nodes in A and all nodes on the path P' ; specifically, for all $a_i \in A$ and for all $j \in [1, n]$, we add an edge (a_i, c'_j) of weight $(7n - j)W + w((a_i, c_j))$. We further add edges from C to the path P'' ; specifically, we add an edge from $c_i \in C$ to c''_i of weight $(6n + i)W$. The resulting graph is given in Figure 1.

Note that all edges in the graph G either have weight 0 or their weight is from the range $[6nW, 7nW + W/4]$. All edges of weight 0 are on the paths P' and P'' ; all paths from s to t must contain at least one edge from P' to A , one from A to B , one from B to C and finally one edge from C to P'' . Each of the non-path-edges has weight from the range $[6nW, 7nW + W/4]$ and we must take at least four of them in total. If we backtrack (and go from A back to P' or from B back to A , etc) then we must take at least six non-zero edges; hence, it is never optimal to backtrack since $(7nW + W/4)4 < 66nW$.

We explain which n queries answer the negative triangle question in H . For each $i = 1, \dots, n$, we delete the edge (c'_i, c'_{i+1}) from path P' and the edge (c''_i, c''_{i-1}) from path P'' , and we query the shortest path from s to t . Note that with these edges deleted to take only four “heavy” edges, one must leave from a c'_j where $j \leq i$ and enter a c''_k where $k \geq i$. The length from s to c'_j and from c''_k to t is zero. So a shortest path from s to t has length

$$(7n - j)W + w(a_p, c_j) + w(a_p, b_q) + 6nW + w(b_q, c_k) + 6nW + (6n + k)W.$$

Note that because W is large and we want to minimize the length of the shortest path, we want to maximize j and minimize k . Due to the deleted edges, the maximum plausible value of j is i and the minimum plausible value of k is i . In that case, the path length is

$$(7n - i)W + w(a_p, c_i) + w(a_p, b_q) + 6nW + w(b_q, c_i) + 6nW + (6n + i)W.$$

This simplifies to $25nW + w(a_p, c_i) + w(a_p, b_q) + w(b_q, c_i)$. If the length of the shortest st -path is less than $25nW - 3M + 3$, then there exists a negative triangle in the graph H' containing c_i . Otherwise, there is no such triangle. ◀

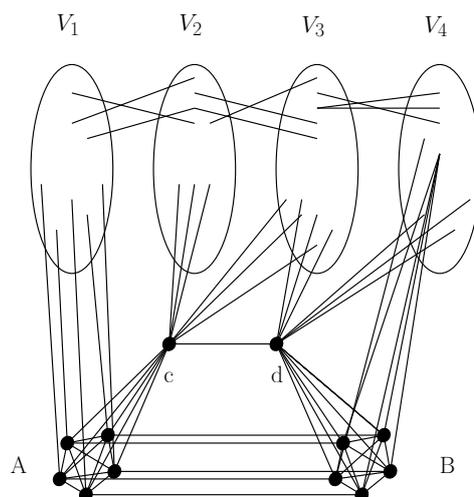
► **Lemma 17.** *Assuming the APSP conjecture, decremental diameter in undirected weighted graphs with sensitivity 1 cannot be solved with preprocessing time $O(n^{3-\varepsilon})$, and update and query times $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$.*

Proof. As in [42] we start by taking an instance of APSP and turning it into a weighted tripartite graph for the negative triangle detection problem; denote the resulting graph for the negative triangle detection problem H' .

Let M be a positive integer such that all edges in H' have weights between $-M$ and M . We increase all edge weights in H by $5M$ (thus making all edges have weight at least $4M$). We denote this new graph H and call the three tripartite groups X , Y and Z . Note that X , Y and Z each have n nodes. Now we want to detect if there is a triangle with weight less than $15M$ in H .

We construct the graph G depicted in Figure 2 as follows. We create sets V_1 and V_4 containing copies of the nodes in X , the set V_2 containing copies of the nodes from Y , and the set V_3 containing copies of the nodes from Z . We additionally create two groups of vertices, A and B , containing copies of X . Finally, we add two vertices c and d to G . For convenience, we denote the different copies of $x_i \in X$ as follows: The copy in V_1 as x_i^1 , the copy in V_4 as x_i^4 , the copy in A as x_i^A , and the copy in B as x_i^B .

We introduce edges between V_1 and V_2 if the corresponding nodes in X and Y have an edge; the edges between V_2 and V_3 are determined by the edges between Y and Z ; the edges between V_3 and V_4 are determined by the corresponding edges between nodes of X and Z .



■ **Figure 2** The graph G' .

All edges we added to the graph have the same weight as their corresponding copies in H . Note that there are no edges between V_1 and V_3 , nor between V_1 and V_4 , nor between V_2 and V_4 .

For all $i = 1, \dots, n$, we add edges of weight $4M$ between x_i^1 and x_i^A , and between x_i^A and x_i^B . Additionally, for all $i = 1, \dots, n$ and all $j = 1, \dots, n$, we add an edge between x_i^B and x_j^A of weight $4M$. For all $v \in V_2 \cup V_3 \cup A$, we add an edge of weight $4M$ between c and v , and for all $v \in V_3 \cup V_4 \cup B$, we add an edge of weight $4M$ between d and v . We add edges with weight $4M$ to connect all vertices in A into a clique, and we do the same for B . Finally, we add an edge of weight $4M$ between c and d .

Note that all edges in this graph have weights between $4M$ and $6M$. Further note that in G all pairs of nodes have a path of length at most $12M$ between them.

Our n queries are picked as follows: For all $i = 1, \dots, n$, we delete the edge between x_i^B and x_i^A and query the diameter. Observe that the only path lengths that could become greater than $12M$ are between the nodes in V_1 and the node x_i^A . However, for all x_j^1 where $j \neq i$, the path $x_j^1 \rightarrow x_j^A \rightarrow x_j^B \rightarrow x_i^A$ still has all of its edges, and there exists a path of length $12M$ for these vertices.

Thus the only pair of vertices for the path length might increase is x_i^1 to x_i^A . One must use at most three edges to obtain a path of length at most $15M$ (since any path with four hops has length at least $16M$ because every edge in G has weight at least $4M$). Hence, any path of length less than $15M$ between x_i^1 and x_i^A must go from x_i^1 to V_2 to V_3 to x_i^A . Thus, if a path of length less than $15M$ exists between x_i^1 and x_i^A after the edge deletion between x_i^B and x_i^A , then there is a negative triangle in H' containing node x_i .

Thus, we can detect if any negative triangle exists in the tripartite graph H' by asking these n queries, determining for all $x_i \in X$ if a negative triangle containing x_i exists. ◀

Note that if for all $i = 1, \dots, n$ we delete edge (x_i^B, x_i^A) , then the eccentricity of x_i^1 is less than $15M$ if and only if the diameter of the graph is less than $15M$. Thus, this serves as lower bound for eccentricity as well.

A.8 SETH Proofs

► **Lemma 18.** *Let $\varepsilon > 0$, $t \in \mathbb{N}$. SETH implies that there exists no algorithm for incremental #SSR with sensitivity $K(\varepsilon, t)$, which has preprocessing time $O(n^t)$, update time $u(n)$ and query time $q(n)$, such that $\max\{u(n), q(n)\} = O(n^{1-\varepsilon})$.*

Proof. Set $\delta = (1 - \varepsilon)/t$. Assume that the k -CNFSAT formula F has $c \cdot \tilde{n}$ clauses for some constant c . Partition the clauses into $K = c/\delta$ groups of size $\delta\tilde{n}$ and denote these groups by G_1, \dots, G_K . Further let U be a subset of the \tilde{n} variables of F of size $\delta\tilde{n}$, and let \bar{U} denote the set of all partial assignments to the variables in the U .

We construct a graph G as the union of D_δ and H_δ . It consists of \bar{U} and the set C of clauses. We direct the edges in H_δ from C to \bar{U} , and add a directed edge from a node $d \in D_\delta$ to $c \in C$ iff $c \in d$. We further add a single node s to the graph. The resulting graph has $n = O(2^{\delta\tilde{n}})$ nodes and $O(2^{\delta\tilde{n}}\tilde{n})$ edges.

We proceed in stages with one stage for each partial assignment to the variables in $V \setminus U$. At a stage for a partial assignment ϕ to the variables in $V \setminus U$, we proceed as follows: For each group $G_i \subset C$, we add an edge from s to the largest non-empty subset d_i of G_i which only contains clauses that are not satisfied by ϕ , i.e., to the set $d_i = \{c \in G_i : \phi \not\models c\}$; if d_i is empty, then we do not introduce an edge. Let D' be the set of nodes d_i that received an edge from s and let $d(s) = |D'|$. Note that $d(s) \leq K$, since we introduce at most one edge for each of the K groups. Further, let B denote the number of clauses in C reachable from the sets $d_i \in D'$, i.e., $B = \sum_{i=1}^K |d_i|$. We query if the number of nodes reachable from s is less than $d(s) + B + 2^{\delta\tilde{n}}$. If the answer to the query is true, then we return that F is satisfiable, otherwise, we proceed to the next partial assignment to the variables in $V \setminus U$.

We prove the correctness of the reduction: Assume that F is satisfiable. Then there exist partial assignments ϕ and ϕ' to the variables in $V \setminus U$ and U , such that $\phi \cdot \phi'$ satisfies F . Hence, for each subset of clauses $d \subset C$ we have that each clause $c \in d$ is satisfied by ϕ or by ϕ' . Thus, the node $u \in \bar{U}$ corresponding to ϕ' cannot be reachable from s and there must be less than $d(s) + B + 2^{\delta\tilde{n}}$ nodes reachable from s . Now assume that at a stage for the partial assignment ϕ the result to the query is true, i.e., less than $d(s) + B + 2^{\delta\tilde{n}}$ nodes are reachable from s ; namely $d(s)$ at distance 1, B at distance 2, and less than $2^{\delta\tilde{n}}$ at distance 3. In this case, there must be a node $u \in \bar{U}$ which is not reachable from s : In D_δ there are exactly $d(s)$ nodes reachable and in C there are exactly B nodes reachable by construction of the graph and definition of $d(s)$ and B . This implies that for the partial assignment ϕ' corresponding to u , each clause $c \in C$ must be satisfied by ϕ or ϕ' . Hence, F is satisfiable.

Note that determining the sets $d_i \in D'$ can be done in time $O(\delta\tilde{n}^2)$ per group G_i as for each clause we can check in time $O(\tilde{n})$ whether it is satisfied by ϕ . Thus the set D' and the value B can be computed in total time $O(cn)$ and the total time for all stages is $O(2^{1-\delta\tilde{n}}(\tilde{n}^2 + Ku(n) + Kq(n)))$. If both $u(n)$ and $q(n)$ are $O(n^{1-\varepsilon}) = O(2^{\delta\tilde{n}(1-\varepsilon)})$, then SAT can be solved in time $O^*(2^{\tilde{n}(1-\varepsilon\delta)})$. ◀

► **Lemma 19.** *Let $\varepsilon > 0$, $t \in \mathbb{N}$. SETH implies that there exists no $(4/3 - \varepsilon)$ -approximation algorithm for incremental diameter with sensitivity $K(\varepsilon, t)$, which has preprocessing time $O(n^t)$, update time $u(n)$ and query time $q(n)$, such that $\max\{u(n), q(n)\} = O(n^{1-\varepsilon})$.*

Proof. Set $\delta = (1 - \varepsilon)/t$. During the proof we will assume that the k -CNFSAT formula F has $c \cdot \tilde{n}$ clauses for some constant c . We partition these clauses into $K = c/\delta$ groups of size $\delta\tilde{n}$ and denote these groups by G_1, \dots, G_K .

We construct a graph as follows: For $U \subset V$ of size $\delta\tilde{n}$ we create the graph H_δ with the set of all partial assignments to the variables in U denoted by \bar{U} and the set of clauses C .

We also add the graph D_δ containing all the subsets of the groups G_i . Furthermore, we introduce four additional nodes x, y, z and t . Observe that the graph has $O(2^{\delta\tilde{n}})$ vertices.

We add an edge from x to each node in \bar{U} . We connect y to all nodes in C and to all nodes in D_δ . We add further edges between a clause c and a set $g \in D_\delta$ if $c \in g$. We also add the following edges to E : $\{x, y\}$, $\{y, z\}$ and $\{z, t\}$. Hence, we have $O(2^{\delta\tilde{n}}\tilde{n})$ edges in total.

If during the construction of the graph we encounter that a clause is satisfied by all partial assignments in \bar{U} , then we remove this clause. Also, if there exists a partial assignment from \bar{U} which satisfies all clauses, we return that the formula is satisfiable. Thus, we can assume that each node in \bar{U} must have an edge to a node in C and vice versa.

We proceed in stages with one stage for each partial assignment to the variables in $V \setminus U$. Denote the partial assignment of the current stage by ϕ . For each G_i , we add an edge between t and the subset of G_i that contains all clauses of G_i which are not satisfied by ϕ , i.e., we add an edge between t and $\{c \in G_i : \phi \not\models c\}$ for all $i = 1, \dots, K$. Hence, in each stage we have $O(K) = O(1)$ updates. We query the diameter of the resulting graph. The diameter is 3, if the formula F is not satisfiable, and it is 4, otherwise. After that we remove the edges that were added in the update.

We prove the correctness of our construction: Observe that via x and y all nodes from $\bar{U} \cup C \cup G$ have a distance of at most 3. From z we can reach all vertices of G and C via y within two steps and all nodes of \bar{U} within three steps via y and x . From t we can reach all nodes of $\{x, y, z\} \cup C \cup G$ within three steps using the path $t \rightarrow z \rightarrow y \rightarrow v$, where $v \in C \cup D_\delta \cup \{x\}$. Hence, all nodes in $\{x, y, z, t\} \cup C \cup G$ have a maximum distance of 3. From $\bar{u} \in \bar{U}$ we can reach t in four steps with the path $\bar{u} \rightarrow x \rightarrow y \rightarrow z \rightarrow t$.

Assume that for \bar{u} there exists a path $\bar{u} \rightarrow c \rightarrow g \rightarrow t$, then by construction $\bar{u} \not\models c$ and $\phi \not\models c$, since $c \in g$. Hence, F is not satisfied by $\bar{u} \cdot \phi$. Thus, if the diameter is 3, then F is not satisfiable. On the other hand, if F is not satisfiable, then for each pair of partial assignments \bar{u} and ϕ , there must be a clause c which both partial assignments do not satisfy. Hence, there must be a path of the form $\bar{u} \rightarrow c \rightarrow g \rightarrow t$ for some g with $c \in g$ and g has an edge to t . Thus, if F is not satisfiable, then the graph has diameter 3.

The sets d_i can be computed as in the previous proof. \blacktriangleleft

► Lemma 20. *Let $\varepsilon > 0$, $t \in \mathbb{N}$. SETH implies that there exists no algorithm for incremental ST-Reach with sensitivity $K(\varepsilon, t)$, which has preprocessing time $O(n^t)$, update time $u(n)$ and query time $q(n)$, such that $\max\{u(n), q(n)\} = O(n^{1-\varepsilon})$.*

Proof. We reuse the graph from the proof of Lemma 19. We update it by removing the vertices x, y, z . We further set $S = \bar{U}$ and $T = \{t\}$.

We proceed in stages with one stage for each partial assignment to the variables in $V \setminus U$. Denote the partial assignment of the current stage by ϕ . For each G_i , we add an edge between t and the subset of G_i that contains all clauses of G_i which are not satisfied by ϕ , i.e. we add an edge between t and $\{c \in G_i : \phi \not\models c\}$ for all $i = 1, \dots, K$. Hence, in each stage we have $O(K) = O(1)$ updates. We query for ST-Reachability. If the answer is true, then F is not satisfiable, otherwise, it is.

We prove the correctness of our construction: If F is not satisfiable, then for each pair of partial assignments \bar{u} and ϕ , there must be a clause c which both partial assignments do not satisfy. Hence, there must be a path of the form $\bar{u} \rightarrow c \rightarrow g \rightarrow t$ for some g with $c \in g$ and g has an edge to t . Thus, if F is not satisfiable, then all nodes in S will be able to reach t . On the other hand, assume that for \bar{u} there exists a path $\bar{u} \rightarrow c \rightarrow g \rightarrow t$, then by construction $\bar{u} \not\models c$ and $\phi \not\models c$, since $c \in g$. Hence, F is not satisfied by $\bar{u} \cdot \phi$. Thus, if all nodes from S can reach t , then F is not satisfiable.

The sets d_i can be computed as in the first proof of the section. \blacktriangleleft

A.9 Conditional Lower Bounds For Variable Sensitivity

In this section we prove conditional lower bounds for algorithms where the sensitivity is not fixed, but given a parameter d . Before we give our results, we shortly argue why this setting is relevant.

First, several results were obtained in the setting with sensitivity d . Some of these results are by Patrascu and Thorup [37] for decremental reachability, by Duan and Pettie [23, 24] and by Henzinger and Neumann [28] for subgraph connectivity and by Chechik et al. [19, 18] for decremental all pairs shortest paths. Our lower bounds show that the results in [23, 24] and the incremental algorithm in [28] are tight.

Second, when d is not fixed and we can prove a meaningful lower bound, this will help us understand whether updates or queries are more sensitive to changes of the problem instance.

Third, when d is fixed to a constant, the problems might become easier in the sense that constant or polylogarithmic update and query times can be achieved. For example, for APSP with single edge failures one can achieve query and update times $O(1)$ (see [10]); for APSP with two edge failures one can achieve query and update times $O(\log n)$ (see [22]). In these cases we cannot prove any non-trivial conditional lower bounds for them. However, with an additional parameter d we can derive conditional lower bounds which are polynomial in the parameter d .

Our results are summarized in the following theorem.

► **Theorem 21.** *Under the OMv conjecture for any $\varepsilon > 0$, there exists no algorithm with preprocessing time $\text{poly}(n)$, update time $\text{poly}(d)$ and query time $\Omega(d^{1-\varepsilon})$ for the following problems:*

1. *Decremental/incremental st -SubConn in undirected graphs with sensitivity d*
2. *decremental/incremental st -reach in directed graphs with sensitivity d ,*
3. *decremental/incremental BP-Match in undirected bipartite graphs with sensitivity d , and*
4. *decremental/incremental SC in directed graphs with sensitivity d .*
5. *$(2 - \varepsilon)$ -approximate ss -shortest paths with sensitivity d in undirected unweighted graphs,*
6. *$(5/3 - \varepsilon)$ -approximate st -shortest paths with sensitivity d in undirected unweighted graphs,*
7. *BW-Matching with sensitivity d .*

Conditional Lower Bounds for Directed Graphs

We observe that some existing reductions can be used to obtain conditional lower bounds for sensitivity problems. In this section, we summarize the results that can be obtained this way.

We call a reduction from a dynamic problem A to another dynamic problem B *sensitivity-preserving* if in the reduction a single update in problem A propagates to problem B as at most one update. We observe that the reductions in [3] from st -subgraph-connectivity to st -reachability (Lemma 6.1) and from st -reachability to BP-Match (Lemma 6.2) and SC (Lemma 6.4) are sensitivity-preserving. Henzinger et al. [27] give conditional lower bounds for the st -subgraph-connectivity problem with sensitivity d . The previous observations about sensitivity preserving reductions imply that we get the same lower bounds for st -reach, BP-Match and SC with sensitivity d . This implies the first four points of Theorem 21.

The construction of the lower bound in [27] required $d = m^\delta$ for some $\delta \in (0, 1/2]$. This appears somewhat artificial, since in practice one would rather expect situations with much smaller values for d , e.g., $d = O(1)$ or $d = \text{poly} \log(n)$. However, the lower bound is still interesting because it applies to all algorithms that allow setting $d = m^\delta$. For example, the sensitive subgraph connectivity algorithms of Duan and Pettie [23, 24] is tight w.r.t. to the above lower bound.

Conditional Lower Bounds for Undirected Graphs

In this subsection, we prove the last three points of Theorem 21. We give a reduction from the γ -uMv-problem, which was introduced by Henzinger et al. [27]. The γ -uMv-problem is as follows: Let $\gamma > 0$. An algorithm for the γ -uMv problem is given an $n_1 \times n_2$ binary matrix M with $n_1 = \lfloor n_2^\gamma \rfloor$, that can be preprocessed. Then two vectors u and v appear and the algorithm must output the result of the Boolean vector-matrix-vector-product $u^t M v$.

Henzinger et al. [27, Corollary 2.8] show that under the OMv conjecture for all $\varepsilon > 0$, no algorithm exists for the γ -uMv-problem that has preprocessing time $\text{poly}(n_1, n_2)$, computation time $O(n_1^{1-\varepsilon} n_2 + n_1 n_2^{1-\varepsilon})$, and error probability at most $1/3$. We give a reduction from the γ -uMv-problem to $(2 - \varepsilon)$ -approximate ss -shortest-paths with sensitivity d in the following lemma. This lemma and the proof of Corollary 3.12 in [27] imply the result in Theorem 21 for ss -shortest-paths.

► **Lemma 22.** *Let $\delta \in (0, 1/2]$. Given an algorithm \mathcal{A} for incremental/decremental $(2 - \varepsilon)$ - ss -shortest paths with sensitivity d , 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 $O(m^{1-\delta})$ vertices, then making a single batch update of size $O(d)$ and $O(m^{1-\delta})$ queries, where m is such that $m^{1-\delta} = n_1$ and $d = m^\delta = n_2$.*

Proof. We prove the lower bound for the incremental problem.

Let M be a $n_1 \times n_2$ binary matrix for $(\frac{\delta}{1-\delta})$ -uMv. We construct a bipartite graph G_M from the matrix M : Set $G_M = ((L \cup R), E)$, where $L = \{l_1, \dots, l_{n_1}\}$, $R = \{r_1, \dots, r_{n_2}\}$ and the edges are given by $E = \{(l_i, r_j) : M_{ij} = 1\}$. We add an additional vertex s to G_M and attach a path of length 3 to s , the vertex on the path with distance 3 from s has edges to all vertices in L . Observe that G_M has $O(n_1 n_2) = O(m)$ edges and $n_1 + n_2 = \Theta(m^\delta + m^{1-\delta}) = \Theta(m^{1-\delta})$ vertices.

When the vectors u and v arrive, we add an edge (s, r_j) for each $v_j = 1$ in a single batch of $O(d)$ updates. After that for each $u_i = 1$, we query the shortest path from s to l_i . In total, we perform $O(m^{1-\delta})$ queries and only use a single batch update consisting of $O(d)$ insertions. We claim that one of the queries returns less than 4 iff $u^t M v = 1$.

First assume that $u^t M v = 1$. Then there exist indices i, j such that $u_i = M_{ij} = v_j = 1$. Hence, there must be a path $l_i \rightarrow r_j \rightarrow s$ of length 2 in G_M and since $l_i = 1$ we ask the query for l_i . Hence, any $(2 - \varepsilon)$ -approximation algorithm must return less than 4 in the query for l_i . Now assume that a query for vertex l_i returns less than 4. Since any path from a vertex in L to s must have length $2k$ for some $k \in \mathbb{N}$, there exists a path $l_i \rightarrow r_j \rightarrow s$. Since query for l_i , we have $u_i = 1$. Due to the edge $l_i \rightarrow r_j$, $M_{ij} = 1$, and due to edge $r_j \rightarrow s$, $v_j = 1$. Thus, $u_i M_{ij} v_j = 1$ and $u^t M v = 1$.

The proof for the decremental problem works by initially adding all edges from s to R to the graph and then removing edges corresponding to the 0-entries of v . ◀

To obtain the result of Theorem 21 for st -shortest-paths, observe that the above reduction can be changed to work for this problem: We add additional vertices s, t to the original bipartite graph and connect s and t by a path of length 5 (i.e., introducing 3 additional vertices). Then a similar proof to the above shows that any algorithm for st -reachability that can distinguish between a shortest path of length at most 3 or at least 5 can be used to decide if $u^t M v = 1$. The result for BW-Match follows from the reduction in [3].

B No (globally) fixed constant sensitivity with polynomial preprocessing time

In this section, we will first discuss how SETH reductions depend on their sensitivity. In particular, we will see that even though some of our previous reductions had constant sensitivities, these constants are not bounded globally. Afterwards we will prove that if we allow for polynomial preprocessing time, then there cannot be a globally fixed upper bound on the sensitivities.

B.1 A note on SETH reductions

Let us recall that the sensitivities of the reductions in Section 4 had the form $K = c/\delta$, where cn was the number of clauses in the k -CNFSAT formula and $\delta < 1$ was a parameter indicating the size of our initial graph. We will first discuss the dependency of K on δ and after that on c .

Firstly, let us discuss how the SETH reductions depend on the parameter δ . Let F be a k -CNFSAT formula with n variables and $O(n)$ clauses. Notice that if the input graph that we construct in the reduction had size $N = O(2^{n/2})$, then the preprocessing time of an algorithm refuting SETH would have to be $O(N^{2-\epsilon})$. In order to allow for arbitrary polynomial preprocessing times of the algorithm, the reductions in [3] (and also the ones from the section before) were parameterized such that the initial graphs have size $O(2^{\delta n})$ (disregarding $\text{poly}(n)$ factors). Then for an algorithm with preprocessing time $O(N^t)$ we can pick $\delta < 1/t$ and hence the preprocessing takes time $O(2^{\delta nt}) = O(2^{(1-\gamma)n})$ for some $\gamma > 0$. Thus, despite the “large” preprocessing time we could still refute the SETH. However, the sensitivities $K = c/\delta$ are not bounded if we consider δ as a parameter, i.e. $K \rightarrow \infty$ as $\delta \rightarrow 0$. If we consider δ as the inverse of the power of the preprocessing time $O(N^t)$, i.e. $\delta = 1/t$, then this can be interpreted as “large preprocessing time” corresponds to “large sensitivity” (since $\delta \rightarrow 0$ iff $t \rightarrow \infty$).

Now one might want to fix δ (and thus bound the preprocessing time) in order to get (globally) fixed constant sensitivities. Unfortunately, this approach is also not feasible; in the SETH reductions for proving an algorithm that solves k -CNFSAT in time $2^{(1-\epsilon)n}$, there is another parameter c which denotes the number of clauses of the k -CNFSAT instance after the application of the sparsification lemma by Impagliazzo, Paturi and Zane [30]. Looking at the proof of the lemma one can see that $c = c(k, \epsilon)$ and that $c \rightarrow \infty$ as $k \rightarrow \infty$, as well as $c \rightarrow \infty$ as $\epsilon \rightarrow 0$. Hence, if we use the sparsification lemma and fix δ , then we still cannot refute the SETH.

B.2 Upper bound in case of polynomial preprocessing

The former reasoning heavily relies on the constructions in our proofs and does not rule the possibility of a reduction with (globally) fixed constant sensitivity from SETH. However, the following theorem and the corollary after it will show that if we allow for polynomial preprocessing time, then there cannot be a polynomial time upper bound on update and query time under any conjecture.

► **Theorem 23.** *Let P be a fully dynamic graph problem⁷ with sensitivity K on a graph*

⁷ We only argue about dynamic graph problems to simplify our language. The theorem holds for all dynamic problems with the given properties.

$G = (V, E)$ with n vertices and m edges where edges are added and removed. Assume that for the static version of P there exists an algorithm running in time $\text{poly}(n)$.

Then there exists an algorithm with $p(n) = n^t$ for some $t = t(K)$, $u(n) = O(K)$ and $q(n) = \log(n)$. The algorithm uses space $O(n^t)$.

Proof. The basic idea of the algorithm is to preprocess the results of all possible queries that might be encountered during P . Since we have only sensitivity K , we will only have polynomially many graphs during the running time of the algorithm. We can save all of these results in a balanced tree of height $O(\log n)$ and during a query we just traverse the tree in logarithmic time.

Denote the initial input graph by G_0 .

Since we know that after each update has size at most K and after the update we roll back to the initial graph, we can count how many graphs can be created while running P . Particularly, notice that a graph $G_1 = (V, E_1)$ can be created while running P if and only if $S = E_0 \Delta E_1$ has $|S| \leq K$.⁸ Then in total the number of graphs which P will have to answer queries on can be bounded by computing how many such sets S exist:

$$\sum_{i=0}^K \binom{n^2}{i} \leq K \max_{i=0, \dots, K} \binom{n^2}{i} = K \text{poly}(n) = n^t,$$

for some t (where in the second step we used that K is constant).

Now consider the naïve algorithm which just preprocesses all trees and stores the results: We enumerate all n^t possible trees and run the static algorithm on them. This can be done in time $O(\text{poly}(n))$. We store the results of the static algorithm in a balanced binary tree with $O(n^t)$ leaves which is of height $O(\log n)$. The traversal of the tree can be done, e.g., in the following way: We fix some order \prec on $V \times V$; this implicitly gives an order \prec on the set $\{S \subseteq V \times V\}$. For a graph G_1 we compute $S = E_1 \Delta E_0$ and traverse according to S .

During updates the algorithm maintains an array of size K which contains the edges that are to be removed or added ordered by \prec . This can be done in time $O(K) = O(1)$.

During a query the algorithm will traverse the binary tree from the preprocessing according to \prec and the updates that were saved during the updates. This takes time $O(\log n)$.

Hence, we have found algorithm with $p(n) = \text{poly}(n)$ and $u(n) = O(1)$ and $q(n) = O(\log n)$. ◀

▶ **Corollary 24.** *If for a problem with the properties from Theorem 23 there exists a reduction from conjecture \mathcal{C} to P with $p(n) = \text{poly}(n)$ and $\max\{u(n), q(n)\} = \Omega(n^{\gamma-\varepsilon})$ for any $\gamma > 0$ and all $\varepsilon \in (0, \gamma)$. Then \mathcal{C} is false.*

Proof. We use Theorem 23 to obtain an algorithm which is better than the lower bound given in the assumption of the corollary. Hence, we obtain a contradiction to \mathcal{C} . ◀

Notice that Corollary 24 implies that in order to obtain meaningful lower bounds for dynamic problems with a certain sensitivity, we must either bound the preprocessing time of the algorithm or bound the space usage of the algorithm or allow the sensitivity to become arbitrarily large.

⁸ $A \Delta B$ denotes the symmetric difference of A and B , i.e. $A \Delta B = A \setminus B \cup B \setminus A$.