2014 IEEE Annual Symposium on Foundations of Computer Science

Popular conjectures imply strong lower bounds for dynamic problems

Amir Abboud
Computer Science Department
Stanford University
Palo Alto, CA, USA
abboud@cs.stanford.edu

Virginia Vassilevska Williams
Computer Science Department
Stanford University
Palo Alto, CA, USA
virgi@cs.stanford.edu

***Abstract*—We consider several well-studied problems in dynamic algorithms and prove that sufficient progress on any of them would imply a breakthrough on one of five major open problems in the theory of algorithms:**

1) **Is the 3SUM problem on n numbers in $O(n^{2-\varepsilon})$ time for some $\varepsilon > 0$?**
2) **Can one determine the satisfiability of a CNF formula on n variables and poly n clauses in $O((2-\varepsilon)^n \text{poly } n)$ time for some $\varepsilon > 0$?**
3) **Is the All Pairs Shortest Paths problem for graphs on n vertices in $O(n^{3-\varepsilon})$ time for some $\varepsilon > 0$?**
4) **Is there a linear time algorithm that detects whether a given graph contains a triangle?**
5) **Is there an $O(n^{3-\varepsilon})$ time combinatorial algorithm for $n \times n$ Boolean matrix multiplication?**

The problems we consider include dynamic versions of bipartite perfect matching, bipartite maximum weight matching, single source reachability, single source shortest paths, strong connectivity, subgraph connectivity, diameter approximation and some nongraph problems such as Pagh's problem defined in a recent paper by Pătraşcu[STOC 2010].

***Index Terms*—dynamic algorithms; all pairs shortest paths; 3SUM; lower bounds;**

I. Introduction

Dynamic algorithms are a natural extension of the typical notion of an algorithm: besides computing a function on an input x, the algorithm needs to be able to update the computed function value as x undergoes small changes, without redoing all of the computation. Dynamic algorithms have a multitude of applications, and their study has evolved into a vibrant research area. Among its many successes are efficient dynamic graph algorithms for graph connectivity [18], [35], [29], minimum spanning tree [14], [21], [19], graph matching [33], [4], [25], [16] and approximate shortest paths in undirected graphs [5], [6], [17]. Graph connectivity and minimum spanning tree for instance can be supported in only polylogarithmic time per edge update or query. Nevertheless, there are some dynamic problems that seem stubbornly difficult. For instance, consider maintaining a reachability tree from a fixed vertex under edge insertions or deletions, i.e. the so called dynamic single source reachability problem (ss-Reach). The best known dynamic ss-Reach algorithm [32] has update time $O(n^{1.495})$ in n-node graphs. This is only better than the trivial recomputation time for very dense graphs. Moreover, the result uses heavy machinery such as fast matrix multiplication, and is currently not practical. There are many such problems, including dynamic shortest paths, maximum matching, strongly connected components, and some non-graph problems such as Pagh's problem [28] supporting set intersection updates and membership queries. For many of these problems, the only known dynamic algorithms are to recompute the answer from scratch. (Although there has been some success when only insertions or only deletions are to be supported.)

For many of these seemingly hard problems we do not believe that a dynamic algorithm with an amortized polylogarithmic update time exists and, in fact, it seems plausible that these problems require update time that is polynomial in the number of vertices n or edges m. Unfortunately, the state of the art on unconditional (e.g. cell probe) lower bounds does not allow for any superpolylogarithmic lower bounds, and we do not have a handle on the true complexity of many (perhaps most) dynamic problems.

In this paper, we follow an approach that is similar in spirit to NP-hardness but more fine-grained. The approach strives to prove, via combinatorial *reductions*, that improving on a given upper bound for a computational problem X would yield breakthrough algorithms for many other famous and well-studied problems. Combining our reductions with widely believed conjectured lower bounds for these famous problems, we obtain a lower bound for the original problem X.

The famous (static) problems we consider in this work are CNF-SAT, triangle detection, Boolean Matrix Multiplication, All Pairs Shortest Paths (APSP) and 3-SUM. Using our approach we are able to provide many (conditional) lower bounds for fundamental dynamic problems most of which are *tight* up to sub-polynomial factors! (See the full version [1] for an exposition on these famous problems and their conjectured lower bounds.) Thus, *we are able to determine the exact (polynomial) complexity of many basic dynamic problems under certain widely believed conjectures.* As an example, we prove that the reachability tree from a single source cannot be maintained with an amortized $m^{1-\varepsilon}$ update time in m-edge graphs, for any $\varepsilon > 0$, without refuting the Strong Exponential Time Hypothesis, implying that the trivial algorithm for ss-Reach may be essentially optimal.

0272-5428 2014

DOI 10.1109/FOCS.2014.53

Our reductions start from a famous static problem (e.g. CNF-SAT) and generate an instance of a potentially very *different* dynamic problem (e.g. ss-Reach), along with a sequence of updates and queries. Note that it is typically easy to show that if a static problem requires $\Omega(m^c)$ time to solve, then an appropriate dynamic version of the *same* problem would require $\Omega(m^{c-1})$ time per update, by simply "creating" the instance. Such a simple approach, however, cannot obtain interesting lower bounds for basic problems like ss-Reach or strong connectivity, since the static case can be solved in linear time. More similar to our work are the reductions of Roditty and Zwick [31] from APSP to partially dynamic single source shortest paths (SSSP) in which the dynamic instance is non-trivially modified. Here we go much further by relating a variety of very different problems to provide a long list of lower bounds.

In some sense, most similar to our results are the surprising reductions of Pătraşcu [28] from 3-SUM on n numbers, a problem that is assumed to require $n^{2-o(1)}$ time. Assuming the hardness of 3-SUM, Pătraşcu is able to show that several dynamic problems (on input size m), such as transitive closure, require $\Omega(m^c)$ time updates for some small $c > 0$. Yet, as we explain below, following Pătraşcu's line of attack one cannot hope to obtain *tight* lower bounds for the problems he considers. In fact, in order to obtain a lower bound bound of $\Omega(m^c)$ for $c > 1/3$ different ideas seem to be required. Meanwhile, we give tight bounds for many problems.

Pătraşcu's reductions proceed as follows. He first reduces 3-SUM on n numbers to the problem of reporting $m = O(n^{1.5})$ triangles in a graph with $O(m)$ edges. Subsequently, he presents a simple reduction from the triangle reporting problem to an intermediate problem called the multiphase problem that can then be reduced to several dynamic problems. The obtained lower bound for the dynamic problems depends on the size of the required instance and the number of required updates and queries in these reductions. We extend this approach for many dynamic problems such as bipartite matching in the full version [1] of the paper and optimize the lower bounds from 3-SUM following Pătraşcu's approach. These lower bounds are unfortunately still not tight.

The loss in the approach comes from the fact that in the reductions, in order to get a subquadratic algorithm for 3-SUM, the dynamic problems are required to find m triangles in $O(m^{4/3-\varepsilon})$ time. However, the state-of-the-art algorithms even for finding a *single* triangle run in $O(m^{1.41})$ time [2]. Moreover, Pătraşcu's reduction from reporting triangles to the dynamic problems essentially reduces triangle reporting to triangle finding (in a special instance). This reduction, just as all known reductions from triangle reporting to detection, is lossy, and the final conditional lower bounds on the dynamic problems incur this loss. In a sense, the true barrier for the hardness of dynamic problems such as the ones Pătraşcu considers is triangle detection. We explore this barrier later in the paper and are able to tighten and extend the lower bounds.

The main contribution of this work is that we identify different barriers for different dynamic problems and find different famous hard problems that allow us to better explain these barriers.

a) SETH: Our first set of results is based on the observation that the hardness of improving on the trivial runtime of many basic dynamic problems can be "blamed" on the hardness of devising algorithms for CNF-SAT that are exponentially faster than the brute force 2^n solution.

Despite hundreds of papers on faster exponential algorithms for NP-Hard problems in recent years (see the survey by Woeginger for an exposition [42]), and despite the remarkable effort put into obtaining faster satisfiability algorithms, the best upper bounds for CNF-SAT on n variables and m clauses remain of the form $2^{n-o(n)}\text{poly}\ (m)$ (e.g. [20], [27], [34]). The Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi and Zane, which has received a lot of attention recently, states that better algorithms do not exist. (See the full version [1] for more background on SETH).

Conjecture 1 (SETH)**.** *For every $\varepsilon > 0$, there exists a k, such that SAT on k-CNF formulas on n variables cannot be solved in $O^*(2^{(1-\varepsilon)n})$ time.*

Our reductions from CNF-SAT can be classified into two kinds. In the first kind, given a CNF formula on n variables we construct a sparse graph on $N = 2^{n/2}\text{poly}\ (n)$ nodes and edges on which $O(N)$ updates and queries of certain dynamic problems, such as the problem of computing the number of nodes reachable from a single source (#SSR), would allow us to conclude whether the formula is satisfiable. Therefore, any dynamic algorithm for these problems with truly *sub-linear amortized* update and query times, i.e. $O(n^{1-\varepsilon})$ for some $\varepsilon > 0$, would refute SETH.

Intuitively speaking, *any problem that allows us to check whether a certain node $s \in V$ can reach every node in a certain set $T \subseteq V$ will have a lower bound from SETH.*

For #SSR in sparse n-node graphs, the lower bound of $\Omega(n^{1-\varepsilon})$ for all $\varepsilon > 0$ from above is tight up to $n^{o(1)}$ factors, since the trivial algorithm that computes the answer from scratch after every update runs in $O(n)$ time per update (the graph is sparse). We obtain similar *tight* linear lower bounds for other problems such as variants of maintaining the strongly connected components of a directed graph (SC2 and MaxSCC) and the connectivity problem in undirected graphs with node updates (ConnSub) studied by Chan et al. [7], [8]. (See Table I for definitions and the full version for background on the problems.)

Quite surprisingly, seemingly similar versions of these problems can be solved with *polylogarithmic* time updates, e.g. #SSR in undirected graphs or ConnSub with edge updates instead of node updates. Meanwhile, we prove that any "truly" faster than trivial algorithm for the above versions implies a breakthrough algorithm for CNF-SAT. Using an additional idea we show that SETH implies that even after an arbitrary *polynomial preprocessing* time ($O(n^c)$ for any constant c), one cannot maintain the updates and answer the queries in amortized sub-linear time.

Theorem I.1 (informal)**.** *SETH implies that #SSR, SC2, MaxSCC and ConnSub cannot be solved with $O(m^{1-\varepsilon})$ amortized update and query times on sparse graphs, for any $\varepsilon > 0$, even after an arbitrarily long polynomial time preprocessing.*

Interestingly, although our lower bound is tight when the graph is sparse, using an approach by Sankowski [32] we can solve #SSR with an $O(n^{1.575})$ update time using matrix multiplication. This is sub-linear in the number of edges when the graph is dense. We are not able to get a higher than $n^{1-o(1)}$ lower bound from SETH when the graph is dense. However, we will later show how a different conjecture allows us to get a better lower bound, also suggesting that matrix multiplication may be necessary to obtain improvements.

In the second kind of reductions from CNF-SAT, different ideas and different dynamic problems allow us to implement a similar construction where the number N of nodes, edges, queries and updates is only $2^{n/3}\text{poly}\,(n)$, thus implying a much higher, *quadratic*, lower bound on the amortized update and query times.

An example of a problem for which this kind of reduction applies is maintaining a $(4/3-\varepsilon)$-approximation of the graph diameter ($4/3$-Diam), for any $\varepsilon > 0$. The $n^{2-o(1)}$ update lower bound (under SETH) that we obtain for $4/3$-Diam in sparse graphs is quite striking: In the dynamic setting, $\tilde{O}(n^2)$ time per update [13] is sufficient to maintain even *all pairs* of distances in *dense* graphs, whereas our $n^{2-o(1)}$ lower bound is for maintaining a single distance in a *sparse* graph. In the static setting, one can solve even APSP in sparse graphs in $\tilde{O}(n^2)$ time, implying that the trivial dynamic algorithm that computes the exact diameter from scratch after every update is essentially optimal. Another basic dynamic problem with such a high $n^{2-o(1)}$ lower bound is ST-Reach in which we ask whether every node in the subset $S \subseteq V$ can reach every node in the subset $T \subseteq V$.

Theorem I.2 (informal)**.** *SETH implies that $4/3$-Diam and ST-Reach cannot be solved with $O(m^{2-\varepsilon})$ amortized update and query times, for any $\varepsilon > 0$, even after an arbitrarily long polynomial time preprocessing.*

b) Triangle Detection: Our second set of results is based on the observation that the hardness in solving many basic dynamic graph problems can be explained by the hardness of solving triangle detection. An example of such a problem is dynamic reachability between a single source-sink pair (s,t-Reach). Given an n node and m edge graph, the fastest algorithms for determining whether the graph contains a triangle run in time $O(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$ [23], [2] where $\omega < 2.373$ is the matrix multiplication exponent [36], [15], [10], [11].

The known truly subcubic algorithms for matrix multiplication are largely impractical, and a lot of research has gone into obtaining a "combinatorial" algorithm for the problem: a simple, practical algorithm that does not suffer from the overhead of the current theoretically fast approaches. The fastest combinatorial algorithms for triangle detection, however, are far from fast: they run in $O(m^{3/2})$ [23] and $O(n^3/\log^{2.25} n)$ [3] time.

Below we introduce the "Strong Triangle conjecture", that essentially says that the current bounds for Triangle detection cannot be beaten. This conjecture allows us to get high lower bounds for several dynamic problems. The lower bounds are *tight* when restricted to combinatorial algorithms.

Conjecture 2 (Strong Triangle)**.** *In the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $\min\{n^{\omega-o(1)}, m^{2\omega/(\omega+1)-o(1)}\}$ time in expectation to detect whether an n node m edge graph contains a triangle. Moreover, any combinatorial algorithm requires $m^{3/2-o(1)}$ time.*

In the full version of the paper [1] we explain how the combinatorial part of the Strong Triangle conjecture is implied by a now standard conjecture that combinatorial algorithms for Boolean Matrix Multiplication (BMM) must take essentially cubic time. Thus, we can obtain tight lower bounds for many dynamic problems, also based on the hardness of BMM, suggesting that the use of fast matrix multiplication is necessary to obtain nontrivial algorithms.

Notice that if $\omega = 2$, then the best algorithm for triangle detection in m-edge graphs would run in $O(m^{4/3})$ time which is still far from linear. One of the weakest (and hence most likely to be true) conjectures in our paper is that Triangle detection does not admit linear time algorithms. We state a flexible version of this conjecture below, for any $\delta > 0$.

Conjecture 3 (δ-Triangle)**.** *In the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $m^{1+\delta-o(1)}$ time in expectation to detect whether an m edge graph contains a triangle.*

Our results based on the Strong Triangle conjecture are consequences of our results from the above conjecture. In particular, for sparse graphs, one can directly apply the results from the δ-Triangle conjecture with $\delta = (\omega-1)/(\omega+1)$.

Under the δ-Triangle conjecture (for any $\delta > 0$), we obtain $\Omega(m^{2\delta-o(1)})$ update lower bounds for many dynamic problems. If $\omega = 2$ and $\delta = 1/3$ (as in the Strong Triangle conjecture), our lower bounds would be $m^{2/3-o(1)}$. These bounds are significantly higher than the ones we can obtain from 3-SUM following Pătraşcu's approach (that at best achieve $\Omega(m^{1/3})$). Moreover, the δ-Triangle conjecture may be a more plausible assumption than the 3-SUM Conjecture (see the full version for a discussion).

Our reductions proceed as follows. We first show how using $O(n)$ updates and queries to s,t-Reach in a dynamic graph on $O(m)$ edges and $O(n)$ nodes one can detect whether an n node m edge graph contains a triangle. Then we show how s,t-Reach can be replaced by several other dynamic problems such as strong connectivity and bipartite matching. Finally, we add a high-degree-low-degree step to get a more efficient reduction when the graph is sparse. The intuitive moral of our reductions is that *s,t-Reach is possibly the simplest dynamic problem that can solve Triangle detection*, and any problem

that can capture it would suffer from similar limitations.

Below is an informal statement of some of our results.

Theorem I.3 (informal)**.** *Let $\gamma = (\omega - 1)/(\omega + 1) \in [1/3, 0.408]$ and let $\varepsilon > 0$ be any constant. The Strong Triangle conjecture implies that dynamic s,t-Reach, strong connectivity and bipartite matching cannot be solved with $O(m^{2\gamma-\varepsilon})$ update and query times even after $O(m^{1+\gamma-\varepsilon})$ preprocessing time, nor, if the algorithm is combinatorial, with $O(m^{1-\varepsilon})$ update and query time, even after $O(m^{3/2-\varepsilon})$ time preprocessing.*

Theorem I.3 shows that *any* non-trivial combinatorial algorithm for these dynamic problems would imply a surprising breakthrough for Triangle detection and hence also for Boolean matrix multiplication. Interestingly, for some of the above problems, fast matrix multiplication has been used to obtain faster update times for dense graphs, e.g. $O(n^{1.495})$ for bipartite matching [33]. Our results imply that update times faster than $O(n^{\omega-1}) = O(n^{1.373})$ for such problems would be difficult to obtain. Closing this gap between $n^{1.373}$ and $n^{1.495}$ is a very interesting open question.

c) APSP: Next we consider edge weighted graph problems and we show that the current upper bounds for dynamic maximum weight matching and s,t-shortest path (st-SP) are optimal, *even on partially dynamic graphs*, unless an unexpected truly subcubic algorithm for APSP exists.

Classical algorithms such as Dijkstra's or Floyd-Warshall's provide $O(n^3)$ running times for APSP in n-node graphs. Many $n^{o(1)}$ improvements over this cubic runtime were found, the current best is the recent $\frac{n^3}{2^{\Omega(\sqrt{(\log n)})}}$ by Williams [40]. Nevertheless, no truly subcubic time algorithm for APSP is known, and is conjectured to be impossible in many papers, e.g. [31], [41].

Conjecture 1 (APSP)**.** *There is a constant c, such that in the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $n^{3-o(1)}$ time in expectation to compute the distances between every pair of vertices in an n node graph with edge weights in $\{1, \ldots, n^c\}$.*

Roditty and Zwick [31] show how to reduce APSP to $O(n)$ updates and $O(n^2)$ queries of partially dynamic SSSP to obtain a tight lower bound for this problem, assuming the APSP conjecture above. We present a more efficient reduction to an even easier problem, reducing APSP to $O(n)$ updates and queries of partially dynamic st-SP, perhaps identifying the simplest dynamic problem that can efficiently solve APSP. This simplification allows us to conclude a similar tight lower bound for partially dynamic maximum weight matching in bipartite graphs (BWMatch), via a folklore reduction from st-SP to BWMatch.

Theorem I.4 (informal)**.** *The APSP conjecture implies that st-SP and BWMatch cannot be solved with amortized $O(n^{2-\varepsilon})$ update and query times in decremental or incremental graphs.*

Theorem I.4 is quite surprising since the shortest s,t-path can be computed from scratch in $\tilde{O}(n^2)$ time and yet we show that (under the above conjecture) quadratic time cannot be avoided over a sequence of only edge deletions (or only insertions) even in an amortized sense! For comparison, observe that incremental single source reachability can be solved with $O(1)$ amortized update and query time. Similarly, dynamic BWMatch can be easily solved with $\tilde{O}(n^2)$ worst case update time by checking for augmenting paths after every update, and unexpectedly, we show that this is optimal even for amortized, partially dynamic algorithms.

d) More results: The majority of our results appear in Table II, and the problems we study are defined in Table I. Our results apply for many well-studied problems beyond the ones we have mentioned so far. Notably, our reductions also apply to non-graph problems. An interesting example is the problem we call $\emptyset$-PP (a variant of which appeared in [28] as Pagh's problem). In this problem, one is to maintain a collection of k subsets $X_1, \ldots, X_k$ over a universe of size n, where the updates are, given two indices i, j, add the subset $X_{ij} = X_i \cap X_j$ to the collection. A query points to a subset in the current collection and asks whether it is empty. The motivation for this problem comes from conjunctive queries in databases and information retrieval [9], [37], [26]. For instance, each set X_i might correspond to the set of documents containing a word W_i and one might be interested in finding documents containing multiple words of interest. We prove that SETH implies that even when the number of subsets $k = O(\log n)$ is quite small and when we are allowed to preprocess the initial collection in arbitrarily long polynomial time, it is impossible to support the updates in amortized $O(n^{1-\varepsilon})$ time, for any $\varepsilon > 0$. Notice that the trivial update time is $O(n)$.

More careful reductions allow us to also prove statements about maintaining an approximate matching dynamically. In particular, we show that a certain natural approach is unlikely to yield very efficient updates for c-approximate dynamic matching for a fixed constant c. For details, see the full version.

e) Meaning of the results: One way to look at our results is that we prove conditional lower bounds for dynamic problems. There is another way to look at them however: our results present new approaches to attacking famous open problems. Perhaps a new technique in dynamic algorithms may be the key to truly subcubic APSP, or to breaking SETH.

Due to the nature of our techniques, all of our reductions imply results not only for fully dynamic algorithms, but also for *partially dynamic* ones. In particular, most of the lower bounds in Table II also hold when only insertions or only deletions are to be performed, and when the update and query times are *worst case*. Also due to the nature of our techniques, our results also hold for algorithms that have *lookahead* access, i.e. they know the updates and queries in advance. This is interesting since many dynamic problems have more efficient algorithms in the presence of lookahead. In our reductions, the number of updates and queries we make to the dynamic algorithm is roughly linear (or quadratic) in the input size, and therefore the lower bounds hold even when amortizing over a sequence of that number of updates and queries.

f) Rules of thumb: We can conclude the following rules of thumb for future work on conditional lower bounds for dynamic problems. Let P be a dynamic problem.

- If P looks like graph reachability, attempt reducing from Triangle detection, BMM or 3-SUM.
- If P can detect a pair of nodes s, t such that t is not reachable from s, then attempt reducing from CNF-SAT.
- If P can maintain some shortest path, then attempt reducing from APSP.

II. Summary of Techniques

Detailed proofs are given in the full version of the paper. In the rest of this version, we summarize the main ideas in our reductions.

g) Lower bounds based on the SETH: By a careful use of the sparsification lemma of Impagliazzo, Paturi and Zane [22], one can show that, to refute SETH, it is enough to solve SAT on CNF formulas with n variables and $O(n)$ clauses in $O^*(2^{(1-\delta)n})$ time for $\delta > 0$. Given such a formula F, we construct graphs as follows.

To explain the first kind of reductions from CNF-SAT, let us focus on the reduction to dynamic #SSR. The other reductions require different ideas to implement but proceed along the same lines. We start by splitting the variables V into two sets U and $V \setminus U$ of size $n/2$ each. We create a set A on $N = 2^{n/2}$ nodes, each corresponding to a partial assignment to the variables in U. We also create a set C on $O(n)$ nodes, one corresponding to each clause, and an additional node s which will be our source. So far the reduction resembles most reductions from CNF-SAT to polynomial time solvable problems. The first such reduction was devised by Williams in his study of the orthogonal pairs and subset query problems [38], [39], [30].

Suppose now that we add a directed edge from a clause $c \in C$ to each partial assignment $a \in A$ if and only if a does not satisfy c. In our reduction, we will add and remove edges from the node s to the nodes in C, while the edges from part C to part A will remain unchanged. We will have N stages, one stage for each partial assignment b to the variables in $V \setminus U$. We start the stage by connecting the node s with edges to all the clauses $c \in C$ such that the partial assignment b does not satisfy c. Then, the main observation is that b can be completed to a satisfying assignment to our formula F if and only if there is a node $a \in A$ that s cannot reach. To see this, observe that s can reach a if and only if there is a clause $c \in C$ that is not satisfied by neither a nor b. Therefore, the second part of our stage is a query asking about the number of nodes reachable from the source s. If k is the number of clauses not satisfied by our current b then the answer to the query will be less than $k + 2^{n/2}$ if and only if b can be completed to a satisfying assignment. If the satisfiability of F was not confirmed by the query, we finish the stage by removing all the edges from s to C and move on to the next partial assignment.

The formula F is satisfiable if and only if at least one of the N stages confirms the satisfiability, which concludes the correctness of our reduction. Note that the graph we create has $O(2^{n/2}) = O(N)$ nodes and $m = O(n2^{n/2}) = O(N \log N)$ edges, while we do $O(n2^{n/2}) = O(N \log N)$ updates and N queries. Hence any dynamic algorithm with $O(N^{2-\varepsilon})$ preprocessing time, and $O(m^{1-\varepsilon})$ update and query time would violate the SETH.

Now, suppose that we could achieve $O(m^{1-\varepsilon})$ update and query time after $O(N^t)$ preprocessing for some big constant t. Then we could still contradict the SETH by modifying the above construction. Instead of splitting the variables into two parts on $n/2$ variables each, we split them into U of size δn and $V \setminus U$ of size $(1-\delta)n$ for some constant $\delta < 1/t$. Then we apply exactly the same construction as above where A is the set of $2^{\delta n}$ partial assignments to U, while the number of stages is the number of partial assignments to $V \setminus U$ which is $2^{(1-\delta)n}$. The number of vertices and edges of the graph is now $O(n2^{\delta n}) \leq O(2^{n(1-\gamma)/t})$ for some $\gamma > 0$. Hence the $O(N^t)$ preprocessing time only takes $O(2^{n(1-\gamma)})$ time. The number of updates we do is $O(n2^{(1-\delta)n})$ but since the graph is much smaller we get that $O(m^{1-\varepsilon})$ time updates and queries imply a runtime of

$$2^{\delta n(1-\varepsilon)} \cdot 2^{(1-\delta)n} = 2^{n(1-\varepsilon\delta)}$$

(excluding polynomial factors) for solving the SAT instance. Hence we again violate the SETH.

To explain the second kind of reduction from CNF-SAT, we focus on dynamic ST-Reach. The key observation is that dynamic ST-Reach allows us to efficiently check the existence of a *triple* of partial assignments that satisfy all our clauses, allowing us to split the variables V into three sets U_1, U_2, U_3 and eventually get a higher lower bound.

Assume for simplicity of presentation that the variables are split evenly among the sets. For each of the $N = 2^{n/3}$ partial assignments to the variables in U_1 we create a node a in part S of our graph, and for each of the N partial assignments to the variables in U_2 we create a node b in part T of our graph. We also add two sets of nodes C_1 and C_2 of size $O(n)$. For every clause c in our formula we add nodes c_1 to C_1 and c_2 to C_2. A node a in S will be connected with a directed edge to a clause node c_1 if and only if a does not satisfy c, and a node b in T will have an incoming directed edge from c_2 if and only if b does not satisfy c. These edges from S to the C_1 nodes and from C_2 to T will not change during our reduction, and we will be adding and removing edges from C_1 to C_2.

We will have N stages, one stage for each partial assignment d to the variables in U_3. We start the stage by adding edges $c_1 \to c_2$ for all the clauses $c \in C$ such that the partial assignment d does not satisfy c. Then, the main observation is that d can be completed to a satisfying assignment to our formula F if and only if there are two nodes $a \in S$ and $b \in T$ such that a cannot reach b. To see this, observe that a can reach b if and only if there is a clause $c \in C$ that is not satisfied by any of a, b and d. Therefore, the answer to an ST-Reach query tells us whether d can be completed to a satisfying assignment. If the satisfiability of F was not confirmed by the query, we finish the stage by removing all the edges from C_1 to C_2 and move on to the next partial assignment.

Problem		
Maintain	Update	Query
(s,t)-Subgraph Connectivity (st-SubConn)		
A fixed undirected graph, a subset S of its vertices and fixed vertices s,t	Insert/remove a node into/from S	Are s and t connected in the subgraph induced by the nodes in S?
Bipartite Perfect Matching (BPMatch)		
An undirected bipartite graph	Edge insertions/deletions	Does the graph have a perfect matching?
Bipartite Maximum Weight Matching (BWMatch)		
An undirected bipartite graph with integer edge weights	Edge insertions/deletions	What is the weight of the maximum weight matching?
Bipartite matching without length k augmenting paths (k-BPM)		
An undirected bipartite graph	Edge insertions/deletions	What is the size of a matching that does not admit length k augmenting paths?
Single Source Reachability (SS-Reach)		
A directed graph and a fixed vertex s	Edge insertions/deletions	Given a vertex t, is t reachable from s?
(s,t)-Reachability (st-Reach)		
A directed graph and fixed vertices s,t	Edge insertions/deletions	Is t reachable from s?
(s,t)-shortest path (st-SP)		
An undirected weighted graph and fixed vertices s,t	Edge insertions/deletions	What is the distance between s and t?
Strong Connectivity (SC)		
A directed graph	Edge insertions/deletions	Is the graph strongly connected?
2 Strong Components (SC2)		
A directed graph	Edge insertions/deletions	Are there more than 2 strongly connected components?
2 vs k Strong Components (AppxSC)		
A directed graph	Edge insertions/deletions	Is the number of SCCs 2 or more than k?
Maximum SCC size (MaxSCC)		
A directed graph	Edge insertions/deletions	What is the size of the largest SCC?
Single Source Reachability Count (# SSR)		
A directed graph with a fixed source s	Edge insertions/deletions	Given ℓ, is the number of nodes reachable from $s < \ell$?
Connected Subgraph (ConnSub)		
A fixed undirected graph and a vertex subset S	Insert/remove a node into/from S	Is the subgraph induced by S connected?
(S,T)-Reachability (ST-Reach)		
A directed graph and fixed node subsets S and T	Edge insertions/deletions	Are there some $s \in S$, $t \in T$ s.t. t is unreachable from s?
$(4/3-\varepsilon)$-Approximate Diameter ($4/3$-Diam)		
An undirected graph	Edge insertions/deletions	Is the diameter 3 or 4?
Chan's Subset Union Problem (SubUnion)		
A subset S of a fixed collection $X = \{X_1, \ldots, X_n\}$ of subsets over a universe U, with $\sum_i \lvert X_i \rvert = m$	Insert/remove a set X_i into/from S	Is $\cup_{X_i \in S} X_i = U$?
Pagh's Problem (PP)		
A collection X of subsets $X_1, \ldots, X_k \subseteq [n]$	Given i,j, insert $X_i \cap X_j$ into X	Given index i and $u \in U$, is $u \in X_i$?
Pagh's Problem with Emptiness Queries ($\emptyset$-PP)		
A collection X of subsets $X_1, \ldots, X_k \subseteq [n]$	Given i,j, insert $X_i \cap X_j$ into X	Given index i, is $X_i = \emptyset$?

TABLE I
THE PROBLEMS WE CONSIDER.

	Best Upper Bounds		Our Lower Bounds				
#SSR, SC2, AppxSC							
	Trivial	Full Version	SETH	Comb.-Δ	Strong-Δ	δ-Δ	3SUM
Update	m	$n^{1.575}$	$\boldsymbol{n}$ (*)	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	m^{α}
Query	1	1	$\boldsymbol{n}$ (*)	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	$m^{2/3-\alpha}$
ConnSub, SubUnion							
	Trivial	Full Version	SETH	Comb.-Δ	Strong-Δ	δ-Δ	3SUM
Update	m	$n^{1.575}$	$\boldsymbol{n}$ (*)	-	-	-	-
Query	1	1	$\boldsymbol{n}$ (*)	-	-	-	-
ST-Reach, 4/3-Diam							
	Trivial	[13], [12]	SETH	Comb.-Δ	Strong-Δ	δ-Δ	3SUM
Update	mn	n^2	$\boldsymbol{n^2}$ (*)	m	$m^{0.81}$	$m^{2\delta}$	m^{α}
Query	1	1	$\boldsymbol{n^2}$ (*)	m	$m^{0.81}$	$m^{2\delta}$	$m^{2/3-\alpha}$
st-Reach, BPMatch, 17-BPM							
	Trivial (*)	[32], [33]	SETH	Comb.-Δ	Strong-Δ	δ-Δ	3SUM
Update	m	$n^{1.495}$	-	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	m^{α}
Query	1	1	-	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	$m^{2/3-\alpha}$
SC							
	Trivial	Full Version	SETH	Comb.-Δ	Strong-Δ	δ-Δ	3SUM
Update	m	$n^{1.575}$	-	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	m^{α}
Query	1	1	-	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	$m^{2/3-\alpha}$
st-SubConn							
	[8]		SETH	Comb.-Δ	Strong-Δ	δ-Δ	3SUM
Update	$m^{2/3}$		-	$m^{1/2}$	$m^{0.41}$	m^{δ}	m^{α}
Query	1		-	$\boldsymbol{m}$	$m^{0.81}$	$m^{2\delta}$	$m^{2/3-\alpha}$
Decremental or Incremental BWMatch or st-SP							
	Trivial (**)		APSP (total update time)				
Update	m		$\boldsymbol{n^2}$				
Query	1		$\boldsymbol{n^2}$				

TABLE II

THE TABLE INCLUDES THE CURRENT BEST UPPER BOUNDS FOR THE LISTED PROBLEMS, TOGETHER WITH BOUNDS FOR WHICH A LISTED CONJECTURE WOULD BE FALSIFIED. AN n^c UPPER BOUND IN THE TABLE INDICATES THE EXISTENCE OF AN $\tilde{O}(n^c)$ ALGORITHM, WHILE AN n^c LOWER BOUND INDICATES THAT AN $O(n^{c-\varepsilon})$ ALGORITHM, FOR ANY $\varepsilon > 0$, FALSIFIES THE LISTED CONJECTURE. THE UPPER BOUNDS MARKED BY FULL VERSION ARE SIMPLE CONSEQUENCES OF SANKOWSKI'S ALGORITHM FOR TRANSITIVE CLOSURE [32] AND ARE OBTAINED IN THE FULL VERSION OF THE PAPER. THE 3SUM BOUNDS HOLD FOR ANY α IN $[1/6, 1/3]$. (*) THE LOWER BOUND HOLDS EVEN WHEN $m = \tilde{O}(n)$. (**) FOR DYNAMIC MATCHING ONE CAN SUPPORT UPDATES IN $O(m)$ TIME BY LOOKING FOR AUGMENTING PATHS. ALL LOWER BOUNDS CAN BE AMORTIZED AND EXPECTED IN THE FULLY DYNAMIC CASE, AND HOLD IN THE CASE OF PARTIALLY DYNAMIC ALGORITHMS AS WORST CASE LOWER BOUNDS, EXCEPT THE APSP BOUNDS WHICH ARE AMORTIZED IN ANY CASE.

As before, the formula F is satisfiable if and only if at least one of the N stages confirms the satisfiability, which concludes the correctness of our reduction. Note that the graph we create is smaller this time, with $O(2^{n/3}) = O(N)$ nodes and $m = O(n2^{n/3}) = O(N \log N)$ edges, and we do $O(n2^{n/3}) = O(N \log N)$ updates and N queries, which is fewer than before. Hence any dynamic algorithm with $O(N^{3-\varepsilon})$ preprocessing time, and $O(m^{2-\varepsilon})$ update and query time would violate the SETH.

h) Lower bounds from Triangle Detection and BMM: Here we give an overview of our reductions from the δ-Triangle conjecture from the introduction, and a related conjecture on the complexity of BMM. The BMM conjecture (formally defined in the full version) roughly states that any combinatorial algorithm for $n \times n$ Boolean matrix multiplication requires $n^{3-o(1)}$ time.

To obtain lower bounds based on the BMM conjecture, we first obtain lower bounds from the δ-Triangle conjecture that hold for arbitrary δ and an arbitrary number of edges m, and then apply them for $m = n^2$ and a carefully chosen δ to obtain the lower bounds from BMM. For instance if the δ-Triangle conjecture for any constant δ implies that problem P cannot have a dynamic algorithm with $m^{1+\delta}$ preprocessing time, m^{δ} update time and $m^{2\delta}$ query time, then we get that the BMM conjecture implies that P cannot have a dynamic algorithm with $n^{2+2\delta}$ preprocessing time, $n^{2\delta}$ update time and $n^{4\delta}$ query time. Then picking $\delta = (1-\varepsilon)/2$, we get a lower bound for all $\varepsilon > 0$ of preprocessing time $n^{3-\varepsilon}$, update time $n^{1-\varepsilon}$ and query time $n^{2-\varepsilon}$.

Our reductions from Triangle Detection typically begin with the following construction. Given a graph $G = (V, E)$ on m edges and n vertices, we create 4 copies of V, A, B, C, D, and for each edge $(u, v) \in E$ we add the directed edges $(u_A, v_B), (u_B, v_C), (u_C, v_D)$ where u_X is the copy of u in $X \in \{A, B, C, D\}$. Now G contains a triangle if and only if for some u, there is a path from u_A to u_D. Since the new graph has $O(m)$ edges and $O(n)$ vertices, it suffices to simulate the n reachability queries (u_A, u_D) with dynamic algorithms for the problem at hand.

For st-Reach for instance, we add two additional nodes s

and t to the above graph and we proceed in stages, one for each node $u \in V$. In each stage, we add edges (s, u_A) and (u_D, t), and ask whether t is reachable from s. This will be the case iff u appears in a triangle in G. If the answer to the query is no, we remove the edges incident to s and t and move on to the next stage. The number of queries and updates is $O(n)$ overall, and hence any dynamic algorithm with $O(m^{1+\delta})$ preprocessing time, and $O(m^{2\delta})$ update and query time would imply an $O(m^{1+\delta} + nm^{2\delta})$ time triangle algorithm. We then apply a high-degree low-degree argument as in [2] to show that this also implies an $O(m^{1+\delta})$ time triangle algorithm.

To obtain the lower bounds for Strong Connectivity and Bipartite Perfect Matching, we prove general reductions from st-Reach to SC and BPMatch that show that if the latter two problems can be solved with preprocessing time $p(m, n)$, update time $u(m, n)$ and query time $q(m, n)$, then st-Reach can be solved with preprocessing time $p(O(m), O(n))$, update time $u(O(m), O(n))$ and query time $q(O(m), O(n))$. We show a separate reduction from Triangle Detection to st-SubConn (similar to the one to st-Reach) that performs m updates and n queries, giving an $m^{\delta - o(1)}$ lower bound on the update time and $m^{2\delta - o(1)}$ on the query time.

Our lower bound for PP is more involved than the rest of the lower bounds based on the δ-Triangle conjecture. We will explain the main ideas. Given an n-node, m-edge graph, first let us look for triangles containing a node of high degree $\geq \Delta$. We begin by creating for every node j of high degree a set X_j containing node i iff j is not a neighbor of i. The number of such sets is $O(m/\Delta)$ and constructing them takes $O(mn/\Delta)$ time. Now, for each node a, using $d(a)$ updates, we create the intersection Y_a of all sets X_j for the neighbors j of a. Then, for every edge (a, b), we query whether $b \in Y_a$. Notice that $b \in Y_a$ if and only if b is not a neighbor of any of the neighbors j of a. Thus, if any one of the m queries returns "no", we have detected a triangle.

Suppose now that no triangle with a node of high degree is found. Then, all nodes of any triangle have degree $< \Delta$. We can attempt to do exactly the same reduction as above. The only problem is that the number of sets X_j that we would have to create could be n, and thus just creating the sets would take $O(n^2)$ time. This is sufficient for a reduction from triangle in dense graphs, however it is too costly for a reduction from sparse graphs. Fortunately, we can avoid the high cost. Before we create the sets X_j, we pick a universal hash function h and hash all nodes with it into a universe of size $O(\Delta^2)$. We are guaranteed that with constant probability, if we take two nodes a and b of low degree, then $N(a) \cup N(b)$ won't contain any two nodes hashing to the same element. Thus, we can simulate the search for a triangle with an edge (a, b) where both a and b have low degree, just as before, except that we create a set for each *hash value* v, $X_v = \{j \mid \forall c \in N(j),\ h(c) \neq v\}$. The creation time is now $O(n\Delta^2)$, and everything else works out with constant probability. We can obtain correctness with high probability by using $O(\log n)$ hash functions. Picking $\Delta = m^{1/3}$, we obtain an extra term $m^{2/3}n$ in our reduction which is negligible if we are trying to contradict the δ-Triangle conjecture for $\delta > 1/3$.

i) Lower bounds on partially dynamic algorithms: Notice that our reductions almost always look like this (with the exception of PP and $\emptyset$-PP). They proceed in stages, and each stage i has the following form: I_i insertions are performed, then some number of queries Q_i are asked. Finally the I_i insertions are undone.

We can simulate this type of a reduction with an incremental algorithm as follows. During each stage, we perform the I_i insertions and Q_i queries, and while we do them, we record the sequence of all changes to the data structure that the insertions (and queries) cause. This makes our reduction no longer black box (it was black box for fully dynamic algorithms). It also increases the space usage to be on the order of the time that it takes to perform the I_i insertions. However, once we have recorded all the changes, we can undo them in reverse order in roughly the same time as they originally took, and bring the data structure to the same state that it was before the beginning of the stage. We obtain lower bounds on the preprocessing, update and query time of incremental algorithms. However, since we undo changes, the lower bounds only hold for worst case runtimes.

Simulating the above algorithms with decremental algorithms is more challenging since it would seem that we need to simulate I_i insertions with roughly I_i deletions, and this is not always possible. We develop some techniques that work for many of our reductions. For instance, we are able to simulate the following with only $O(n)$ deletions (and undeletions) over all n stages: in each stage i a node s has an edge to only the ith node from a set of size n. This is useful for our proof that efficient worst-case decremental st-Reach implies faster triangle algorithms.

j) Lower bounds based on APSP: To show our lower bounds from APSP to incremental/decremental st-SP and BWM, we first reduce st-SP to BWM, thus showing that we only have to concentrate on st-SP. Then, we carefully combine the ideas behind Roditty and Zwick's [31] original reduction with Vassilevska Williams and Williams' [41] proof that negative triangle detection is equivalent to APSP. In particular, we show that the number of shortest paths queries can be reduced to n (from n^2) since we only need to simulate determining whether there is a path on 3 edges from each vertex back to itself.

k) Lower bounds from 3*SUM:* Pătraşcu [28] showed that 3-SUM on n numbers can be reduced to the problem of listing $O(n^2/R)$ triangles in a certain tripartite graph on partitions A, B, C where $|A| = |B| = \sqrt{n}R$, $|C| = n$, $|E(A, B)| = O(nR)$ and $|E(A, C)| + |E(B, C)| = O(n^{1.5})$, for any $R = n^{\frac{1}{2}+\delta}$ and $0 < \delta < \frac{1}{2}$, in truly subquadratic time. Then, he reduced this triangle listing problem to "the multiphase problem", which in turn can be reduced to several dynamic problems. We examine Pătraşcu's reduction in more detail and show that by directly reducing the triangle listing problem to dynamic problems like st-SubConn we can overcome some inefficiencies incurred by "the multiphase

problem" and get improved lower bounds.

A first approach is to use the known reductions from triangle listing to triangle finding [41], [24] to directly apply our hardness results based on triangle finding. However, using the currently best reductions, even a linear time algorithm for triangle finding would not be able to get us a faster than $m^{4/3}$ time algorithm for listing m triangles which is what we need in order to get subquadratic 3SUM.

Instead, we reason about Pătraşcu's construction directly. First, we observe that to falsify the 3SUM conjecture, it is enough to list in subquadratic time all pairs of nodes $(a, b) \in A \times B$ that participate in a triangle. To do this, note that in Pătraşcu's construction, every node of A has at most $O(n/R)$ neighbors in C. Thus, once the $\leq O(n^2/R)$ pairs of nodes that appear in triangles are known, one can go through each one pair (a, b), and check each of the at most $O(n/R)$ neighbors $c \in C$ of a, to find all triangles going through (a, b). Thus 3SUM would be in $O(n^2/R \cdot n/R) = O(n^3/R^2)$ time which is truly subquadratic when $R = n^{\frac{1}{2}+\delta}$ for $\delta > 0$.

Thus, to obtain lower bounds for our dynamic problems, we show how to list the pairs of nodes in $A \times B$ that appear in triangles using a small number of queries and updates. We first reduce st-Reach to st-SubConn, thus also showing that st-SubConn is at least as hard as SC and BPMatch. Then we focus on st-SubConn. Given Pătraşcu's graph for some choice of R, we create an instance H of st-SubConn. H is a copy of G in which all the edges between parts A, B are removed. Thus H has only $O(n^{1.5})$ edges for any choice of R. We also add a node s that is connected to all the nodes in A and a node t that is connected to all the nodes in B. Initially, s, t and all nodes in C are activated, while the nodes in A, B are deactivated.

We preprocess this graph in $p(n^{1.5})$ time which is subquadratic if $p(m) = O(m^{\frac{4}{3}-\varepsilon})$. Then, we have a stage for each of the $O(nR)$ edges in $A \times B$ in G. In the stage for (a, b), we activate the nodes a, b in H and query if s, t are connected. s and t are connected iff there is a node in C that is a neighbor of both a, b, i.e. (a, b) participates in a triangle. Then we deactivate a and b and move on to the next edge. This way, we can list all the pairs that are in triangles with $O(nR)$ updates and queries to st-SubConn, which would be in subquadratic time if $R = n^{1+\varepsilon/2}$ and the update and query times are $u(m), q(m) = O(m^{\frac{1}{3}-\varepsilon})$.

This type of approach is insufficient to prove a tradeoff between the query and update time, however. To obtain such a tradeoff, we need to be able to reduce the search for triangle edges to st-SubConn where the number of queries is very different from the number of updates. To achieve this, on the same underlying graph as before, we use st-SubConn to binary search for the nodes in B that participate in a triangle with a given node a (instead of simply trying each neighbor of a as we did above). This allows us to reduce the number of queries in the reduction to $\tilde{O}(n^2/R)$, while keeping the number of updates $\tilde{O}(nR)$. This lets us pick a larger R and trade-off the lower bounds for the query and the update times.

In the binary search for a fixed $a \in A$, we use the queries to check whether there is a node b in a certain contiguous subset of B (interval) that participates in a triangle with a. This can be done by activating all neighbors of a in the interval at once, and asking the s, t connectivity query. We start the search with an interval that contains all of B. If we discover that an interval I contains a node b that participates in a triangle with a, we proceed to search within both subintervals of I of half the size. (Thus, we only search in an interval if its parent interval returned "yes".) Since no b that appears in a triangle with a appears in more than $O(\log n)$ B-intervals, the number of queries to st-SubConn is only bigger than the number of triangles by a logarithmic factor, and is thus $\tilde{O}(n^2/R)$. The number of updates is no more than $O(d_B(a) \log n)$ for each a where $d_B(a)$ is the number of neighbors of a in B. Hence the total number of updates is $\tilde{O}(nR)$.

Acknowledgments

We would like to thank Liam Roditty, Ryan Williams, and Uri Zwick for valuable discussions. In particular, we are grateful to Ryan Williams for suggesting the title of this manuscript, and to Liam Roditty for suggesting to us to consider the subgraph connectivity problem. This research was supported by a Stanford School of Engineering Hoover Fellowship, NSF Grant CCF-1417238 and BSF Grant BSF:2012338.

References

[1] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. *CoRR*, abs/1402.0054, 2014.

[2] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[3] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. FOCS*, pages 745–754, 2009.

[4] S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(logn)$ update time. In *FOCS*, pages 383–392, 2011.

[5] A. Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *FOCS*, pages 693–702, 2009.

[6] A. Bernstein and L. Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *SODA*, pages 1355–1365, 2011.

[7] T. M. Chan. Dynamic subgraph connectivity with geometric applications. *SIAM J. Comput.*, 36(3):681–694, 2006.

[8] T. M. Chan, M. Pătraşcu, and L. Roditty. Dynamic connectivity: Connecting to networks and geometry. In *FOCS*, pages 95–104, 2008.

[9] H. Cohen and E. Porat. Fast set intersection and two-patterns matching. *Theor. Comput. Sci.*, 411(40-42):3795–3800, 2010.

[10] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.

[11] A. Davie and A. J. Stothers. Improved bound for complexity of matrix multiplication. *Proceedings of the Royal Society of Edinburgh, Section: A Mathematics*, 143:351–369, 4 2013.

[12] C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: Breaking through the $o(n^2)$ barrier. In *Proc. FOCS*, volume 41, pages 381–389, 2000.

[13] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM (JACM)*, 51(6):968–992, 2004.

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

[15] F. L. Gall. Powers of tensors and fast matrix multiplication. *CoRR*, abs/1401.7714, 2014.

[16] M. Gupta and R. Peng. Fully dynamic $(1 + \varepsilon)$-approximate matchings. In *FOCS*, 2013.

[17] M. Henzinger, S. Krinninger, and D. Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the O(mn) barrier and derandomization. In *Proc. FOCS*, 2013.

[18] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.
[19] M. R. Henzinger and V. King. Maintaining minimum spanning forests in dynamic graphs. *SIAM J. Comput.*, 31(2):364–374, 2001.
[20] E. A. Hirsch. Two new upper bounds for SAT. In *Proc. SODA*, pages 521–530, 1998.
[21] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.
[22] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
[23] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.
[24] Z. Jafargholi and E. Viola. 3sum, 3xor, triangles. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:9, 2013.
[25] O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *STOC*, pages 745–754, 2013.
[26] R. Pagh. Personal communication.
[27] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.
[28] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *STOC*, pages 603–610, 2010.
[29] M. Pătraşcu and M. Thorup. Planning for fast connectivity updates. In *FOCS*, pages 263–271, 2007.
[30] M. Pătraşcu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. SODA*, pages 1065–1075, 2010.
[31] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *ESA*, pages 580–591, 2004.
[32] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. FOCS*, volume 45, pages 509–517, 2004.
[33] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *Proc. SODA*, pages 118–126, 2007.
[34] U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proc. FOCS*, pages 410–414, 1999.
[35] M. Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC*, pages 343–350, 2000.
[36] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. STOC*, pages 887–898, 2012.
[37] D. E. Willard. An algorithm for handling many relational calculus queries efficiently. *J. Comput. Syst. Sci.*, 65(2):295–331, 2002.
[38] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Proc. ICALP*, pages 1227–1237, 2004.
[39] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2–3):357–365, 2005.
[40] R. Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. STOC*, 2014.
[41] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.
[42] G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. In *Proc. IWPEC, LNCS 3162*, pages 281–290, 2004.