COMS E6998-5: Fine-grained complexity (Spring'26) Spring 2026

Lecture 7: Dynamic Graph Problems and Online Matrix Mult

Instructor: Josh Alman Scribes: Zak Thayer and Eric Shao

1 3SUM Continued

One interesting property about 3SUM compared to the other conjectures we have examined is that we are not assuming the most straightforward brute-force algorithm is optimal; rather, we are conjecturing optimality of a slightly optimized $O(n^2)$ algorithm. Today we mention a variant of 3SUM where the most straightforward algorithm is already quadratic, and where a truly subquadratic improvement is equivalent to refuting the usual 3SUM conjecture.

1.1 Convolution 3SUM

Definition 1 (Convolution 3SUM)**.** *We are given an array Z of length n with entries in $\{-n^4, \ldots, n^4\}$ (any range $\pm n^{3+\delta}$ would suffice). The goal is to determine whether there exist indices i, j, k such that*

1. *$Z[i] + Z[j] = Z[k]$, and*
2. *$i + j = k$.*

Algorithm. There is a trivial $O(n^2)$ algorithm: iterate over all pairs (i, j), compute $k = i + j$, and check whether the equality $Z[i] + Z[j] = Z[k]$ holds (when k is in range).

Theorem 2. 3SUM *can be solved in truly subquadratic time if and only if Convolution* 3SUM *can be solved in truly subquadratic time.*

Remark. This equivalence was proven by Pătraşcu in [Pat10], with a simpler proof by Timothy Chan in [CH20]. The main takeaway for reductions is conceptual: 3SUM-hardness can also be used to argue that certain *straightforward* quadratic-time algorithms are optimal, and not only for problems where even achieving $O(n^2)$ requires some cleverness.

2 Dynamic Graph Algorithms

We now move on to a more advanced topic, and see new ways to apply our conjectures to trickier algorithmic settings. In the dynamic model, rather than receiving one input instance and computing an answer once, we maintain a graph under a sequence of updates while also answering queries along the way.

2.1 General model setup

We are given an initial graph G (and may preprocess it). After preprocessing, we receive:

- edge updates of the form $\textsc{insert}(u, v)$ or $\textsc{delete}(u, v)$;
- queries whose type depends on the problem. Examples include:
  - Can u reach v?
  - What is the shortest path between two nodes?
  - How many connected components are there?

The goal is to handle updates quickly and answer queries quickly.

2.2 Variants

If we only allow edge insertions, we call the problem *incremental*. If we only allow edge deletions, we call it *decremental*. If we allow both, we call it *fully dynamic*.

2.3 Time measures

We measure *update time* and *query time*. These can be worst-case (every operation satisfies the bound) or amortized (the average time over a long sequence of operations is bounded).

2.4 Motivation

Dynamic graph problems arise naturally when the underlying graph changes over time (e.g. shortest path queries in a road network with temporary closures or changing traffic). In such applications, worst-case time can matter. Dynamic algorithms also appear as subroutines inside larger static algorithms: for example, directed max flow repeatedly updates a residual graph and asks reachability/connectivity-type questions; interior point methods for linear programming repeatedly solve closely related linear systems (often modeled as maintaining some form of inverse) in [LS15]; and certain approaches to TSP use decremental min-cut style primitives such as in [CQ17]. In these uses, amortized time is often sufficient since we mainly care about the total time of the outer algorithm.

2.5 Dynamic (undirected) connectivity

In this problem we maintain an undirected graph G with two distinguished nodes s, t. Updates are fully dynamic edge insertions and deletions, and the query asks whether s and t are connected.

Straightforward algorithms. There is an $O(m)$ update time and $O(1)$ query time solution by recomputing connectivity after each update (e.g. run BFS/DFS from s and label components). Symmetrically, there is an $O(1)$ update time and $O(m)$ query time solution by doing nothing on updates and running BFS/DFS on each query.

Polylogarithmic algorithms. Sleator–Tarjan in [ST83] gave link-cut trees achieving $O(\log n)$ update and $O(\log n)$ query time when G remains a forest. Thorup in [Tho00] generalized this to arbitrary graphs with update time $O(\log n \cdot (\log \log n)^3)$ (and polylog query).

Lower bounds. In the cell-probe model, Pătraşcu–Demaine in [PD06] showed that one must have either $\Omega(\log n)$ update time or $\Omega(\log n)$ query time.

2.6 Dynamic (directed) reachability

In the directed setting, the corresponding problem is dynamic s–t reachability: updates are directed edge insertions/deletions, and the query asks whether there is a directed path from s to t.

The straightforward baselines again give $O(m)$ update time (and $O(1)$ query time) by recomputing reachability after each update, or $O(1)$ update time (and $O(m)$ query time) by answering each query with a DFS/BFS.

However, unlike undirected connectivity, the best known algorithms are much weaker. Sankowski in [San04] gave an algorithm with update and query time about $n^{1.57}$ (via algebraic techniques related to fast matrix multiplication). The best known lower bound remains only mildly superlogarithmic (roughly $\Omega(\log^{1.5} n)$ in the cell-probe model) by Larsen, Weinstein, and Yu in [LWY17], leaving a large gap between upper and lower bounds.

3 Warm-up Problem: Dynamic Diameter

As a warm-up, consider the dynamic diameter problem: we maintain an undirected graph, and the query asks for the diameter of the current graph.

Recall from earlier in the course that assuming SETH, static diameter cannot be solved in $O(n^{2-\varepsilon})$ time for any constant $\varepsilon > 0$, even on sparse graphs with $m = O(n)$ edges.

Theorem 3. *Assuming* SETH*, any dynamic algorithm for diameter requires either update time* $\Omega(n^{1-\varepsilon})$ *or query time* $\Omega(n^{2-\varepsilon})$ *for all constants* $\varepsilon > 0$*.*

Proof. A dynamic algorithm can be used to solve the static problem as follows: start from the empty graph, insert all m edges of the input (performing m updates), and then make one diameter query. More generally, if a static problem on m edges requires T time, then any dynamic algorithm must have either update time $\Omega(T/m)$ or query time $\Omega(T)$, since otherwise this simulation would solve the static problem faster than T. □

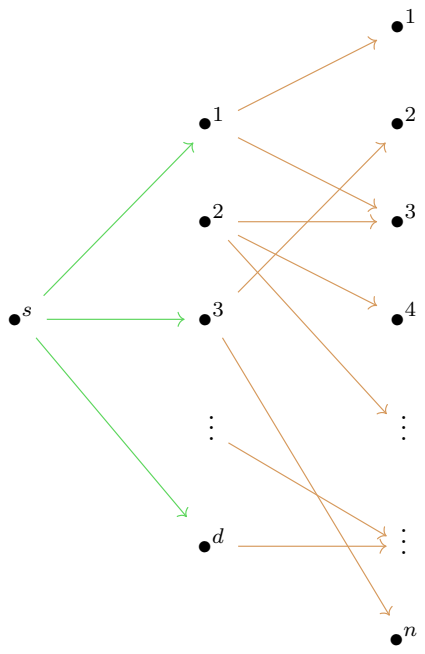
Unfortunately, this argument does not yield meaningful bounds for dynamic reachability, since static reachability can be solved in $O(m)$ time.

4 Number of Single-Source Reachability Lower Bound

We next consider a directed problem whose static version is easy, but whose dynamic version is still hard in the fine-grained sense.

#SSR. Maintain a directed graph with a distinguished source node s. The query asks: how many vertices are reachable from s?

Theorem 4. *Assuming* OVC*, any dynamic algorithm for #SSR requires* $\Omega(n^{1-\varepsilon})$ *update time or* $\Omega(n^{1-\varepsilon})$ *query time (for all constants* $\varepsilon > 0$*).*



Proof. We reduce OV to #SSR. Given $x_1, \ldots, x_n, y_1, \ldots, y_n \in \{0,1\}^d$, construct a three-layer directed graph. The first layer contains a single node s. The second layer contains nodes labeled $1, \ldots, d$ corresponding to the coordinates. The third layer contains nodes labeled $1, \ldots, n$ (corresponding to the y-vectors).

First, add edges encoding the y-vectors: for each $i \in [n]$ and each $j \in [d]$ with $y_i[j] = 1$, add the directed edge $j \to i$ from the second to the third layer (from the coordinate node j to the vector node i), which are the orange edges in the diagram above.

Now iterate over the x-vectors. For a fixed x, insert edges $s \to j$ from first layer to second layer for every coordinate j with $x[j] = 1$, which are the green edges in the diagram above. Then query the number of nodes reachable from s, and finally delete the inserted green edges before moving to the next x.

Let $k = \|x\|_1$. The node s always reaches itself and the k selected coordinate nodes. Moreover, a y-node i is reachable iff there exists a coordinate j with $x[j] = 1$ and $y_i[j] = 1$, i.e. iff $\langle x, y_i \rangle \neq 0$. Therefore s reaches all n y-nodes iff x is not orthogonal to any y_i. In particular, x is not orthogonal to any y_i iff the reachability count equals $1 + k + n$, and otherwise the count is strictly smaller. This lets us solve OVC by running one query per x.

The reduction uses n queries. The initial construction inserts $O(nd)$ edges for the y-layer, and each x inserts/deletes up to d edges, for a total of $O(nd)$ additional updates. If update and query time were both $O(n^{1-\varepsilon})$, then the total running time would be $O(nd \cdot n^{1-\varepsilon}) = O(n^{2-\varepsilon'})$ (for suitable $\varepsilon' > 0$ when $d = n^{o(1)}$), contradicting OVC. □

Remark (incremental-only trick). Although the reduction above uses deletions to "reset" the graph between different x's, one can often obtain an incremental worst-case lower bound by explicitly undoing the changes made to the data structure (record which memory/cells changed during the temporary insertions, then restore them), rather than calling delete operations.

5 Online Matrix-Vector Product Problem

To obtain stronger lower bounds for dynamic directed reachability, we introduce a conjecture designed to capture the difficulty of online dynamic settings.

5.1 Boolean Matrix Multiplication

Given $A, B \in \{0,1\}^{n \times n}$, the Boolean product C is defined by

$$C[i,j] = \bigvee_{k=1}^{n} \left(A[i,k] \wedge B[k,j] \right).$$

This is closely related to OVC: the Boolean product of an $n \times d$ matrix and a $d \times n$ matrix tests (non-)orthogonality across all pairs.

Algorithm. We can compute Boolean matrix multiplication in $O(n^\omega)$ time by computing the ordinary integer matrix product $A \cdot B$ and then thresholding: set $C[i,j] = 1$ iff $(A \cdot B)[i,j] \geq 1$.

5.2 Online Matrix-Vector Multiplication (OMv)

Definition 5 (OMv)**.** *We are given a fixed matrix $M \in \{0,1\}^{n \times n}$ and an online sequence of vectors $v_1, \ldots, v_n \in \{0,1\}^n$. After seeing v_i, we must output Mv_i (over the Boolean semiring) before receiving v_{i+1}.*

Remark. If all vectors were given at once, we could batch them into an $n \times n$ matrix and answer all products in $O(n^\omega)$ time. In the online setting, Strassen-like global recombination ideas do not seem to apply in an obvious way.

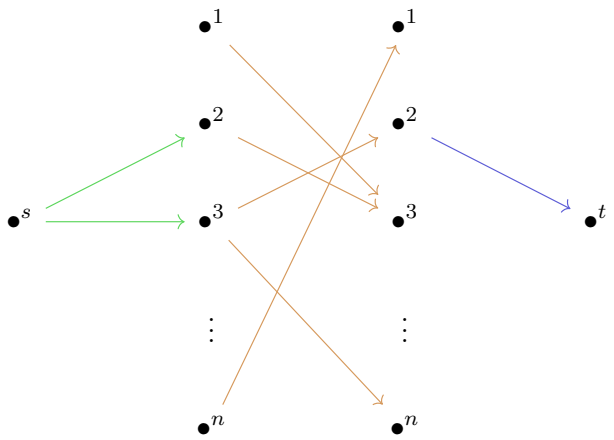
Conjecture 1 (OMv Conjecture)**.** *There is no algorithm for OMv with total running time $O(n^{3-\varepsilon})$ for any constant $\varepsilon > 0$, even allowing polynomial preprocessing.*

Remark (cell-probe). Larsen–Williams gave improved upper bounds for OMv, including a truly subcubic algorithm in the cell-probe model in [LW17]. In the cell-probe model, computation is free and the only cost is probing a memory cell (each holding $\Theta(\log n)$ bits). This suggests that proving the OMv conjecture may be subtle, but we will assume it for the purpose of deriving conditional lower bounds.

6 OMv Hardness of Dynamic s–t Reachability

We now show how a sufficiently fast dynamic algorithm for directed s–t reachability would refute OMv.

Theorem 6. *If dynamic directed s–t reachability admits $O(n^{1-\varepsilon})$ update time and $O(n^{1-\varepsilon})$ query time, then OMv can be solved in total time $O(n^{3-\varepsilon'})$ for some constant $\varepsilon' > 0$.*



Proof. Given an OMv instance with matrix M, construct a 4-layer directed graph: the first layer contains only a source node s, the second layer contains nodes labeled $1, \ldots, n$, the third layer contains nodes labeled $1, \ldots, n$, and the last layer contains only a sink node t.

Between the two middle layers, add edges encoding the matrix: add an edge $i \to j$ iff $M[i, j] = 1$, which are the orange edges in the diagram above.

Now, we process an online vector $v \in \{0, 1\}^n$. When we receive such a vector, insert edges $s \to i$ for all i with $v[i] = 1$, which are the green edges in the diagram above.

To determine the jth output bit of Mv, temporarily insert the edge $j \to t$ from the third-layer node j to t shown by the blue edge, query whether t is reachable from s, and then delete the blue edge $j \to t$. There is a path $s \to i \to j \to t$ iff $v[i] = 1$ and $M[i, j] = 1$ for some i, which is exactly the condition $(Mv)[j] = 1$ in Boolean matrix-vector multiplication. Repeating this for all j recovers Mv. Finally, delete all green edges $s \to i$ and proceed to the next vector.

The initial matrix insertion uses $O(n^2)$ updates. For each of n vectors, we insert/delete up to n edges out of s, and for each j we insert/delete one edge $j \to t$ and make one reachability query. Thus we use $O(n^2)$ queries and $O(n^2)$ additional updates overall. If update and query time were both $O(n^{1-\varepsilon})$, the total time would be truly subcubic, contradicting the OMv conjecture. □

Stronger statement. There is a refined reduction showing that even $O(n^{1-\varepsilon})$ update time together with $O(n^{2-\varepsilon})$ query time for dynamic s–t reachability would refute OMv (informally, by reducing the number of "expensive" reachability queries needed) proven by Henzinger et al in [HKNS15].

References

[CH20] Timothy M. Chan and Qizheng He. Reducing 3sum to convolution-3sum. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020.

[CQ17] Chandra Chekuri and Kent Quanrud. Approximating the held-karp bound for metric TSP in nearly-linear time. *CoRR*, abs/1702.04307, 2017.

[HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector mul-

tiplication conjecture. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30, 2015.

[LS15] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. *CoRR*, abs/1503.01752, 2015.

[LW17] Kasper Green Larsen and Ryan Williams. Faster online matrix-vector multiplication. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2182–2189. SIAM, 2017.

[LWY17] Kasper Green Larsen, Omri Weinstein, and Huacheng Yu. Crossing the logarithmic barrier for dynamic boolean data structure lower bounds. *CoRR*, abs/1703.03575, 2017.

[Pat10] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610. ACM, 2010.

[PD06] Mihai Patrascu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006.

[San04] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 509–517. IEEE Computer Society, 2004.

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[Tho00] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350. ACM, 2000.