

# CSL-531: Dynamic Graph Algorithms

Mmukul Khedekar

January 14, 2026

Notes for the course *Dynamic Graph Algorithms* instructed  
by Dr. Shahbaz Khan at IIT Roorkee, Spring, 2026.

## Contents

<b>1</b>	<b>Jan 13, 2026</b>	<b>2</b>
1.1	Introduction to Dynamic Graph Algorithms . . . . .	2
1.2	Classification of DGAs . . . . .	3
1.3	Time Complexity Analysis . . . . .	3
1.4	Update-Query Tradeoff . . . . .	4
1.5	Time Complexity Bounds . . . . .	5
1.5.1	Upper Bound on $T_d(n)$ . . . . .	5
1.5.2	Lower Bound on $T_d(n)$ . . . . .	5

**1 Jan 13, 2026**

This course will be graded through the following components:

- **MTE:** 30%
- **CWS:** 30%
- **ETE:** 40%

Both the MTE and the ETE will be written examinations. The CWS component will be based on announced quizzes, tutorial session and a course project. The goal of the course is:

1. to be able to **Design** dynamic graph algorithms using standard design techniques and **prove** its correctness.
2. to be able to **Analyze** dynamic graph algorithms using standard analysis techniques and **prove** its tightness.
3. to be able to **Prove** the hardness of a given dynamic graph problem by **reducing** it to a standard hard problem.

The syllabus for this course outlines the following content to be taught.

1. **Introduction:** Terminology, definitions, models, classical problems.
2. **Amortization Techniques:** Doubling, monotonicity, bounded potential. *Case studies:* Partially dynamic MIS, BFS, incremental connectivity, reachability.
3. **Randomization Techniques:** Random sampling, hitting sets, probability tools. *Case studies:* Dynamic matching, decremental connectivity, fully dynamic MIS.
4. **Hierarchical Decomposition Techniques:** Decomposition, multilevel decomposition. *Case studies:* Dynamic matching, fully dynamic connectivity, shortest paths.
5. **Primal Dual Techniques:** Definition, applications in approximation algorithms. *Case studies:* Dynamic vertex cover, fractional matching, coloring.
6. **Fine Grained Complexity:** SETH, polynomial complexity conjectures, reductions. *Case studies:* Shortest paths, connectivity, reachability, matching, diameter.
7. **Miscellaneous Topics:** State of the art in DFS trees, maximal independent sets.

The class strength for this course is only 6 students! You need to perform well and meet the expectations of the instructor to fare well in this course.

## **1.1 Introduction to Dynamic Graph Algorithms**

**Definition 1.1.** **Dynamic Graphs** are graphs that *change* with time. A *change* in a graph refers to **graph updates**, that are *online* sequences of insertion or deletion of edges or vertices.

Therefore we can model a dynamic graph as a sequence of updates on a static graph. Formally, we can view this as

$$G_0 \xrightarrow{\Delta_1} G_1 \xrightarrow{\Delta_2} G_2 \xrightarrow{\Delta_3} \dots$$

where each  $\Delta_i$  is an update to the graph. Suppose we are interested in computing some quantity  $\mathcal{X}$  of the graph  $G$  for which there exists a known static graph algorithm  $\mathbf{X}$ . Upon each update  $\Delta_i$  to  $G$ , we can trivially compute  $\mathcal{X}$  for the new graph  $G'$  by updating  $G$  and running  $\mathbf{X}$  on it. The goal of a **dynamic graph algorithm** is to reuse previous computations to update the quantity  $\mathcal{X}$  *much faster* than a static graph algorithm.

Most graphs in real world are dynamic. The sizes of parameters for these graphs are significantly large and thus, recomputing solutions on such graphs from scratch is impractical. This motivates us to look for dynamic variants of static graph algorithms. However, we are only interested in dynamic graph algorithms that perform *better* than the best static graph algorithms that exist for the problem.

## 1.2 Classification of DGAs

Dynamic Graph Algorithms (DGAs) can be classified into

1. **Incremental** (only insertions)
2. **Decremental** (only deletions)
3. **Fully Dynamic** (both insertions and deletions)

It is worth mentioning that in case of weighted graphs, we might want to consider incrementing and decrementing the weights as graph updates too. However, any increment or decrement to the weight of an edge could be modelled as a deletion followed by an insertion. Therefore, we limit the definition of graph updates to

**Definition 1.2.** A dynamic graph  $G(V, E)$  supports the following **graph updates**

- Insertion of edges
- Deletion of edges
- Insertion of vertices
- Deletion of vertices

To analyse the performance of algorithms, we need to equip ourselves with some theory on time complexity analysis.

## 1.3 Time Complexity Analysis

Some key notations on time complexities.

**Definition 1.3.** For functions  $f$  and  $g$ , we have

1.  $f(n) = \mathcal{O}(g(n))$ , if there exists positive real numbers  $M$  and  $n_0$  such that,

$$|f(n)| \leq Mg(n), \quad \forall n \geq n_0$$

2.  $f(n) = o(g(n))$ , if for every positive real number  $M$ , there exists  $n_0$  such that,

$$|f(n)| \leq Mg(n), \quad \forall n \geq n_0$$

3.  $f(n) = \Omega(g(n))$ , if there exists positive real numbers  $M$  and  $n_0$  such that,

$$|f(n)| \geq Mg(n), \quad \forall n \geq n_0$$

4.  $f(n) = \omega(g(n))$ , if for every positive real number  $M$ , there exists  $n_0$  such that,

$$|f(n)| \geq Mg(n), \quad \forall n \geq n_0$$

5.  $f(n) = \Theta(g(n))$ , if there exists positive real numbers  $M_1, M_2$  and  $n_0$  such that,

$$M_1g(n) \leq |f(n)| \leq M_2g(n), \quad \forall n \geq n_0$$

A common way to establish a tight bound while analysing an algorithm is to find a worst-case example, which helps us establish a lower bound. In dynamic graphs, we commonly deal with the following.

1. **Preprocessing Time**: Initial time to preprocess the graph.
2. **Update Time**: Time taken after each update to reconstruct the data structure.
3. **Query Time**: Time taken to answer query on the updated structure.

## 1.4 Update-Query Tradeoff

There is often a trade-off between how fast updates are processed with how fast queries can be answered. Suppose a static algorithm has time complexity  $\mathcal{O}(T)$ . The extremes of the update-query tradeoff occur when we adopt the following approaches.

- **Eager Approach**

- $\mathcal{O}(T)$  update time.
- $\mathcal{O}(1)$  query time.

- **Lazy Approach**

- $\mathcal{O}(1)$  update time.
- $\mathcal{O}(T)$  query time.

The above approaches are feasible when queries are more frequent, and when updates are more frequent, respectively. Usually when we design an algorithm, we aim to find a sweet spot between these extremes depending on the constraints and requirements of the problem. At times, even when the worst-case bound is proven to be tight, improving the amortized bound is still considered an improvement.

### Example 1.1

DSU (**Disjoin Set Union**) is a popular example of a dynamic graph algorithm. It supports two operations, **Join** and **Find**, both of which achieve an amortized time

complexity of  $\mathcal{O}(\alpha(n))$ . Here,  $\alpha(n)$  is the **inverse ackermann function**. Suppose we consider an incremental dynamic graph model that has  $m$  edge updates. One might think that the total time complexity of this algorithm would be  $\mathcal{O}(m\alpha(n))$ . However, we can only have  $n - 1$  edges in the final graph. Therefore, the total time complexity must be  $\mathcal{O}(n\alpha(n))$  and the amortized time complexity achieved per update is

$$\mathcal{O}\left(\frac{n\alpha(n)}{m}\right)$$

## 1.5 Time Complexity Bounds

We previously gave an informal discussion about the ideal performance a dynamic algorithm. Now, we consider various scenarios and analyse this in detail.

Consider a problem  $\mathcal{P}$  having a best static algorithm  $T_s(n)$ . Now consider a dynamic algorithm  $T_d(n)$  for the problem  $\mathcal{P}$ .

### 1.5.1 Upper Bound on $T_d(n)$

Since we cannot have  $T_d(n)$  perform worse than  $T_s(n)$ , hence we have

$$T_d(n) = \mathcal{O}(T_s(n))$$

### 1.5.2 Lower Bound on $T_d(n)$

1. If  $T_d(n)$  is an incremental algorithm, then consider a graph with size  $\mathcal{O}(n)$  constructed with  $m$  incremental updates. Applying  $T_d(n)$  at every incremental update and summing up, we get

$$mT_d(n) = \Omega(T_s(n)) \implies T_d(n) = \Omega\left(\frac{T_s(n)}{m}\right)$$

2. If  $T_d(n)$  is a decremental algorithm,
3. If  $T_d(n)$  is fully dynamic,

**Question 1.2.** Does there exist a dynamic algorithm for an **NP-complete** problem that lies in **P**?