# Selected Topics in Algorithms Simple (for real)

Gabriel Rovesti

April 19, 2025

## Contents

## 1 Course Overview

This is a research-oriented course on algorithms. The main goal is to bring students up to the research frontier and equip them to do original research in this area. The course covers several advanced topics, extending and complementing those usually covered in traditional graduate algorithms courses, with a focus on the latest research trends.

### 1.1 Course Format

This course is lecture-based and evaluated on the basis of a semester-long project. The course is structured as follows:

- First half: Regular lectures introducing advanced topics in algorithms
- Second half: Project management and technical discussions

### 1.2 Main Topics

The course covers the following main areas:

- Advanced data structures
- Online algorithms
- Algorithms with predictions
- Parallel algorithms

## 2 Advanced Data Structures

### 2.1 Project Proposals

The following are potential projects related to Advanced Data Structures:

1. [**Study**] Fibonacci heaps enable even faster algorithms for minimum spanning trees: in their original paper [21], Fredman and Tarjan give an algorithm with running time $O(m \log^* n)$, where $\log^*$ is the iterated logarithm function (i.e., the number of times we must take logarithms before the argument becomes smaller than 1). Study and present that algorithm.
2. [**Study/Implementation**] Kaplan et al. [27] revisited Fibonacci heaps proposing a one-tree version of this classic data structure in which each decrease-key requires only a single cut. Study and present the paper and/or implement this revisited version and compare your implementation with the original Fibonacci heaps.

3. **[Open Problem]** [27] leaves some open problems related to the analysis of their revisited Fibonacci heap. Try to solve one of those open problems.

4. **[Implementation]** A strict Fibonacci heap is a data structure with the same time bounds of Fibonacci heaps but in the worst case [10]. The authors also made available a Python implementation of their data structure: can you give a faster implementation?

5. **[Study]** Very recently, Haeupler et al. [25] developed a new heap data structure that makes Dijkstra's shortest-path algorithm universally optimal, namely running as fast as possible on every single graph layout. Study and present that paper.

6. **[Study]** Moving to another fundamental data structure, hash tables, very recently a new construction has been invented that achieves far better expected search complexities than were previously thought possible [19]. The main author is an undergraduate student! Study and present that paper.

7. **[Study]** A big news in the field of algorithms was the recent development of a near-linear algorithm for single-source shortest paths, working even in the presence of negative weights [8, 9]. Study and present that algorithm.

8. **[Implementation]** Very recently, Cassis et al. [11] implemented the algorithm of [9]: can you give a faster implementation?

## 2.2 Priority Queues and Heaps

A priority queue is a data structure supporting the following operations:

- **INSERT**($k$): Add a new element with key $k$
- **EXTRACT-MIN**: Return and remove an element with minimum key
- **DECREASE-KEY**: Replace the key value of some element with a new smaller value

Binary heaps provide an implementation with $O(\log n)$ time complexity for all operations. However, for applications like Dijkstra's algorithm and Prim's algorithm, this results in $O(m \log n)$ time complexity, which is suboptimal for dense graphs.

## 2.3 Fibonacci Heaps

Fibonacci heaps were developed by Fredman and Tarjan (1984) to improve Dijkstra's algorithm running time. They provide a more efficient implementation of priority queues with the following amortized time complexities:

- **INSERT**: $O(1)$
- **EXTRACT-MIN**: $O(\log n)$
- **DECREASE-KEY**: $O(1)$

### 2.3.1 Structure

A Fibonacci heap is a collection of trees with the following properties:

1. Vertices of the trees correspond to elements being stored in the queue
2. Roots of heap-ordered trees are arranged in a doubly linked list
3. We maintain a pointer to the root of a tree that contains the minimum key
4. The heap property is maintained: the key of every child is not smaller than the key of its parent

### 2.3.2 Operations

**INSERT** The insertion operation is very simple: create a new heap-ordered tree and add it to the collection of trees in the heap.

**DECREASE-KEY** When decreasing a key, if the new key is smaller than the parent's key, we cut the node from its parent and make it a new root in the linked list. To prevent trees from degenerating into lists, we introduce a marking mechanism:

- Mark nodes that have lost a child
- When cutting a marked node's child, also cut the marked node itself
- Unmarked nodes are marked when they lose a child

**EXTRACT-MIN** To extract the minimum:

1. Remove the minimum node
2. Make all its children roots of new trees in the collection
3. Consolidate trees by merging trees with the same rank (number of children)

### 2.3.3 Amortized Analysis

For the amortized analysis of Fibonacci heaps, we use the potential function method.

**Definition 2.1.** *The potential function $\Phi$ maps a configuration $D$ of an evolving data structure into a number $\Phi(D)$.*

If operation $o_i$ has actual cost $c_i$, then its amortized cost is defined by:

$$a_i = c_i + \Delta\Phi_i = c_i + \Phi_i - \Phi_{i-1}$$

For Fibonacci heaps, we use the potential function:

$$\Phi(t) = t + 2m_t$$

where $t$ is the number of trees in the heap and $m_t$ is the number of marked nodes.

**Theorem 2.1.** *The amortized cost of EXTRACT-MIN is $O(\log n)$ and the amortized cost of DECREASE-KEY is $O(1)$.*

### 2.3.4 Application to Dijkstra and Prim's Algorithms

Using Fibonacci heaps improves the time complexity of these algorithms from $O(m \log n)$ to $O(m + n \log n)$, which is a significant improvement for dense graphs.

## 3 Online Algorithms

### 3.1 Project Proposals

The following are potential projects related to Online Algorithms:

1. [**Study**] Recently, researchers took a fresh look at the classic ski-rental problem by revisiting the performance of randomized algorithms [18]. Study and present that paper.

2. [**Open Problem**] With regard to [18]: can you simplify their analysis? Can you do the same kind of investigation for other online problems, such as paging?

3. [**Study**] In many modern computing systems the amount of available memory is not fixed but varies over time. Recently, Peserico [33] revisited the classic paging problem in this scenario, showing that several (but not all) well-known paging algorithms such as LRU still perform remarkably well. Study and present that paper.

4. **[Open Problem]** Investigate weighted paging, a classic generalization of paging where each page has its own fetching cost, in the model of [33].

5. **[Study]** Online bipartite matching is a classic problem in online algorithms, with tons of applications. Given a bipartite graph containing a perfect matching, vertices of one side arrive in an order selected by an adversary, and edges incident to a vertex are unknown until the vertex arrives. As a vertex arrives, we may irrevocably match it to a vertex of the other side or leave it unmatched forever. The goal is to maximize the size of the resulting matching. Study and present Ranking, the optimal algorithm for this problem [37].

6. **[Implementation]** In online matching on the line, $n$ points of a line are designated as servers, and $n$ requests arrive one by one at arbitrary points of the line. When a request arrives it must be immediately and irrevocably matched to a yet unmatched server. The cost of matching a request to a server is equal to their distance, and the goal is to minimize the total cost of matching all requests. Recently, a class of "hard" input instances was defined to prove a lower bound on the competitive ratio of any online algorithm [34]. Run some experiments to understand what is the expected cost incurred by the optimal offline algorithm on such a class; also, can you prove a lower bound on such a cost?

7. **[Open Problem]** Gairing and Klimm study the greedy algorithm for online metric matching with random arrival order [22]. Can you improve their upper and/or lower bound?

8. **[Open Problem]** Very recently, Harada and Itoh give a new deterministic algorithm for online metric b-matching that is $O(k)$-competitive, where $k$ is the number of server locations [26]. Make their algorithm and/or analysis simpler, and/or improve the constants in their upper bound.

9. **[Study/Implementation/Open Problem]** Gupta et al. [24] study the effects of recourse (that is, relaxing the assumption that the decisions of online algorithms be irrevocable) in online metric matching. Here the possibilities are to 1) study and present their algorithm, 2) implement it to see its performance in practice, and 3) prove or disprove their conjecture.

## 3.2 Introduction to Online Algorithms

Unlike traditional algorithms in the RAM model where all input is available at the beginning, online algorithms receive input piece by piece and must make irrevocable decisions before seeing the entire input.

**Definition 3.1.** *An online algorithm ALG is c-competitive if for every input instance $\sigma$:*

$$ALG(\sigma) \leq c \cdot OPT(\sigma) + b$$

*where $OPT(\sigma)$ is the value achieved by an optimal offline algorithm, and $b$ is a constant independent of the input size.*

For randomized algorithms, we consider the expected performance:

$$\mathbb{E}[ALG(\sigma)] \leq c \cdot OPT(\sigma) + b$$

## 3.3 The Ski Rental Problem

### 3.3.1 Problem Definition

You go skiing and don't know in advance how many days you will ski. Each morning, you must decide whether to rent skis for \$1 per day or buy skis for \$B. The goal is to minimize the total cost.

### 3.3.2 Optimal Offline Algorithm

With complete knowledge of the number of skiing days $t$:

- If $t \geq B$, buy skis on day 1
- Otherwise, rent for all $t$ days

### 3.3.3 Deterministic Online Algorithm

Rent skis for the first $B - 1$ days, then buy skis on day $B$.

**Theorem 3.1.** *This algorithm is 2-competitive.*

*Proof.* We have two cases:

- Case 1: $t \leq B - 1$. The online cost is $t$, the optimal cost is $t$, so the ratio is $\frac{t}{t} = 1 \leq 2$.
- Case 2: $t \geq B$. The online cost is $(B - 1) + B = 2B - 1$, the optimal cost is $B$, so the ratio is $\frac{2B-1}{B} \leq 2$.

□

**Theorem 3.2.** *No deterministic algorithm can achieve a competitive ratio better than 2 for the ski rental problem.*

### 3.3.4 Randomized Online Algorithm

Algorithm: With probability $\frac{1}{2}$, rent for $B-1$ days then buy; with probability $\frac{1}{2}$, rent for $B$ days then buy.

**Theorem 3.3.** *This randomized algorithm is 1.875-competitive.*

## 3.4 Paging (Caching)

### 3.4.1 Problem Definition

We have a cache of size $k$ and a sequence of page requests. When a page is not in the cache, we must evict some page to make room for the requested page, causing a page fault. The goal is to minimize the total number of page faults.

### 3.4.2 Optimal Offline Algorithm

The Furthest-in-the-Future algorithm: Evict the page that will be requested furthest in the future.

### 3.4.3 Deterministic Online Algorithms

**Least Recently Used (LRU)** Evict the page that has not been used for the longest time.

**Theorem 3.4.** *LRU is $k$-competitive.*

*Proof.* We partition the request sequence into phases, where each phase contains at most $k$ distinct page requests.

- LRU incurs at most $k$ page faults in each phase
- Any algorithm (including OPT) incurs at least 1 page fault in each phase except possibly the last

If there are $m$ phases, then $LRU(\sigma) \leq k \cdot m$ and $OPT(\sigma) \geq m-1$. □

**Theorem 3.5.** *Any deterministic paging algorithm cannot have a competitive ratio better than $k$.*

### 3.4.4 Randomized Online Algorithms

**Marking Algorithm**

- Initially, all pages are unmarked
- When a page is requested:
  - If it's already in memory, mark it

  - If not:
    * If all pages in memory are marked, unmark them all
    * Evict a random unmarked page
    * Bring in the new page and mark it

**Theorem 3.6.** *The expected competitive ratio of the Marking algorithm is:*

- $2H_k$ *if there are more than* $k$ *distinct pages in the request sequence*
- $H_k$ *if there are exactly* $k$ *distinct pages*

*where* $H_k = 1 + \frac{1}{2} + \ldots + \frac{1}{k}$ *is the* $k$*-th harmonic number.*

## 4 Algorithms with Predictions

### 4.1 Project Proposals

The following are potential projects related to Algorithms with Predictions:

1. [**Writing**] Write lecture notes for algorithms with predictions. Besides what covered in class, these should contain a coverage of paging for possible classroom use.

2. [**Writing**] So far about a dozen papers, as well as a patent, investigated how to improve paging using ML predictions. Write a survey on paging with predictions highlighting the key ideas and techniques, experimental results, and the research questions that remain open.

3. [**Study/Implementation/Open Problem**] Dinitz et al. [17] initiate the study of algorithms with distributional predictions, where the prediction is a distribution instead of a single value. They initiate this line of research with binary search as a case study. Study and present their paper. Can you improve over their theoretical or experimental results? What about other problems (such as ski rental) in the setting of distributional predictions?

4. [**Study/Implementation/Open Problem**] The algorithms-with-predictions framework has been used extensively to design online algorithms with improved competitive ratios. Another possibility is to use predictions to improve the running time of offline algorithms: in fact, while in classic algorithmics we make the assumption that problems are solved from scratch, in real applications many problems are solved repeatedly on similar input instances, thus opening the opportunity to design better algorithms by leveraging similarities across problem instances. Researchers have recently developed such accelerated algorithms for several problems, such as shortest paths [12, 28], maximum flow [15], and

minimum cut [31], or to obtain better approximation algorithms [3]. Study and present one of those papers.

5. **[Study/Implementation/Open Problem]** Data structures can also benefit from applying ML predictions, giving better performance than their classical counterparts that do not take advantage of patterns in the input data. Study and present [7], which shows how to improve heaps.

6. **[Study/Implementation/Open Problem]** Pick your favorite problem and 1) study the relevant paper(s) from the list in Further Material, 2) do an experimental evaluation of algorithms with predictions for that problem (such as [13] for paging), or 3) solve an open problem suggested by any of the papers from the list in Further Material.

## 4.2 Introduction

Algorithms with predictions (also known as Learning-Augmented Algorithms) combine classical algorithms with machine learning. The idea is to use predictions about the input to improve performance, while providing guarantees based on the accuracy of these predictions.

Desirable properties of algorithms with predictions:

- **Consistency**: When predictions are accurate, the algorithm should perform better than classical algorithms
- **Robustness**: Even with inaccurate predictions, the algorithm should not perform much worse than classical algorithms

## 4.3 Predicted Binary Search

### 4.3.1 Problem Definition

Given a sorted array $A$ with $n$ elements and a query element $q$, find the position of $q$ in $A$ or report that it is not in $A$.

### 4.3.2 Classical Algorithm

Binary search with $O(\log n)$ comparisons.

### 4.3.3 Algorithm with Prediction

Let $h(q)$ be a predictor for the position of $q$ in $A$. The algorithm:

1. If $q = A[h(q)]$, return $h(q)$
2. Else if $q > A[h(q)]$, probe $A[h(q)+1], A[h(q)+2], A[h(q)+4], \ldots$ until finding the right boundary, then binary search

3. Else probe $A[h(q)-1], A[h(q)-2], A[h(q)-4], \dots$ until finding the left boundary, then binary search

**Theorem 4.1.** *Let $t(q)$ be the true position of $q$ in $A$ and $\eta = |h(q) - t(q)|$ be the error of the predictor. The algorithm requires $O(\log \eta + \log \log n)$ comparisons.*

## 4.4 Ski Rental with Predictions

### 4.4.1 Prediction Model

Let $t$ be the true number of skiing days and $y$ be the predicted number. The error is $\eta = |y - t|$.

### 4.4.2 Cautious Algorithm

Let $\lambda \in (0, 1)$ be a tunable parameter. Algorithm:

- If $y < B$, then buy on day $\lceil \lambda B \rceil$
- Else buy on day $B/\lambda$

**Theorem 4.2.** *The competitive ratio of this algorithm is at most* $\min\{1 + \lambda, 1 + \frac{\eta}{OPT} \cdot \frac{1}{1-\lambda}\}$.

This algorithm balances consistency and robustness:

- With perfect predictions ($\eta = 0$), the competitive ratio approaches 1
- With arbitrarily bad predictions, the competitive ratio is bounded by $1 + \lambda$

# 5 Parallel Algorithms

## 5.1 Project Proposals

The following are potential projects related to Parallel Algorithms:

1. **[Study]** The fastest known deterministic MPC algorithm to determine the connected components of a graph in the sub-linear space regime is due to Coy and Czumaj [14]. Study and present that paper.

2. **[Study]** The fastest known MPC algorithm to compute a minimum spanning tree of a graph in the sub-linear space regime is a simple $O(\log n)$-round algorithm, and there is a matching conditional lower bound. Thus, researchers investigated special cases where this logarithmic barrier could be broken, such as when the vertices correspond to points in a metric space [4] or in a Euclidean space [2, 23]. Study and present one (or more) of these papers.

3. **[Study]** Behnezhad et al. [6] showed that a very simple algorithm for maximal matching runs very fast in the MPC model, most notably when the space per machine is just $O(n)$. Study and present that paper.

4. **[Study]** A recent line of research is trying to understand the capabilities of transformers to solve graph problems, and in particular which architectural properties (such as depth and width) transformers need to do that. This can be done by establishing a connection (through simulations) between MPC and transformers [35, 36, 38]. Study and present one of those papers.

5. **[Open Problem]** Counting the number of triangles in a graph can be done in $O(1)$ MPC rounds in some specific classes of graphs [29]. The paper concludes with some interesting open problems: attack one of them.

6. **[Study]** Working in the classical PRAM model for parallel computing, Fischer et al. [20] recently designed a new parallel algorithm for negative-weight single-source shortest paths. Study and present that paper.

## 5.2 Massively Parallel Computation (MPC) Model

The MPC model is an abstraction of popular parallel computing frameworks like MapReduce, Hadoop, Spark, and Flume.

### 5.2.1 Model Definition

- A cluster of machines all connected to each other
- Input of size $N$
- Each machine has memory of size $S = O(N^{1-\varepsilon})$ for some constant $\varepsilon \in (0, 1)$
- Computation proceeds in synchronous rounds:
  - Beginning: Read all messages received in the previous round
  - Middle: Local computation on local data
  - End: Send messages to other machines
- In every round, each machine can send or receive $O(N^{1-\varepsilon})$ data words
- Goal: Minimize the number of rounds to solve the problem

### 5.2.2 Regimes for Graph Problems

For a graph with $n$ vertices and $m$ edges:

- Sub-linear regime: $S = O(n^{1-\varepsilon})$ for some constant $\varepsilon \in (0, 1)$
- Near-linear regime: $S = \tilde{O}(n)$ (suppressing polylog factors)
- Super-linear regime: $S = O(n^{1+\varepsilon})$ for some constant $\varepsilon \in (0, 1)$

## 5.3 Minimum Spanning Tree in the Super-Linear Regime

We focus on connected graphs with $m = \tilde{O}(n)$ for some constant $c \geq 1$ and available memory $S = O(n^{\varepsilon})$ with $\varepsilon > c$.

### 5.3.1 Algorithm

**Algorithm 1** MPC-MST($G = (V, E)$)

```
1: if $|E| \leq n^{\varepsilon}$ then
2:     Compute $T = \text{MST}(E)$ locally
3:     return $T$
4: else
5:     $\ell = |E|/n^{\varepsilon}$                ▷ Number of machines to use in this round
6:     Partition $E$ into $E_1, E_2, \ldots, E_\ell$ using a hash function $h : E \to \{1, 2, \ldots, \ell\}$
7:     In parallel, compute $T_i = \text{MST}(E_i \cup V)$ for each $i \in \{1, 2, \ldots, \ell\}$
8:     return MPC-MST($\cup_{i=1}^{\ell} T_i$)
9: end if
```

### 5.3.2 Analysis

**Lemma 5.1.** *With high probability, the memory constraint of each machine is never violated.*

*Proof.* By Chernoff bound and union bound. □

**Lemma 5.2.** *The algorithm correctly computes an MST of the input graph.*

*Proof.* Based on the cycle property of MSTs: if $e$ is an edge not in the MST $T$, and $C$ is the cycle formed by adding $e$ to $T$, then $e$ has weight at least as large as every other edge in $C$. □

**Theorem 5.3.** *The algorithm runs in $O(\log \log n)$ rounds with high probability.*

*Proof.* In every round, the input size is reduced by a factor of $\lambda = n^{\varepsilon}$. Therefore, after $O(\log_\lambda m) = O(\log_\lambda n) = O(1/\varepsilon \cdot \log \log n) = O(\log \log n)$ rounds, the input is small enough to fit onto a single machine. □

### 5.4 Maximal Matching in MPC

#### 5.4.1 Algorithm

1. Each machine $i$ samples each local edge independently with probability $p = 1/\sqrt{m}$ and sends the sampled edges to machine 1
2. Machine 1 finds a maximal matching $M$ of the sampled edges locally and adds $M$ to the final output
3. Remove the matched vertices in $M$ from the graph and recurse on the remaining edges until the number of remaining edges can fit into a single machine

**Lemma 5.4.** *With high probability, the number of sampled edges fits into a single machine.*

**Lemma 5.5.** *After the $i$-th round, the remaining graph has $O(m(1-1/\sqrt{m})^i)$ edges with high probability.*

**Theorem 5.6.** *The MPC Maximal Matching algorithm runs in $O(\log\log n)$ rounds with high probability.*

## 6 Additional Project Topics

The following are project proposals on additional topics beyond the main course content:

1. [**Machine Learning Algorithms**] Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. Study and present one of the following two papers on hierarchical clustering algorithms: [30, 16].
2. [**Sublinear Algorithms**] A sublinear algorithm is an algorithm whose running time grows slower than the size of the input, and thus only gives an approximate or probably correct answer. Study and present [32], which gives a simple sublinear algorithm to approximate the weight of a minimum spanning tree.
3. [**Fine-Grained Complexity**] Fine-grained complexity theory allows to rule out faster algorithms by proving conditional lower bounds via fine-grained reductions from certain key conjectures. Study and present one of the following two papers on fine-grained complexity [1, 5].
4. [**Quantum Computing**] Quantum computing has the potential to efficiently solve problems that would be extremely difficult if not impossible for classical computers. Tech companies and governments

have been funneling billions of dollars into quantum computers for years; the global race to build fully functional quantum computers was kicked off by Shor's factoring algorithm. Study and present it.

5. [**Misc**] Pick and study a recent paper on your favorite topic/problem from SODA (the top conference for research in algorithms) or HALG (a conference for presenting the highlights of recent developments in algorithms).

## 7 Course Projects

The course evaluation is based on a project that requires students to dive deeper into one of the topics covered. The project can take several forms:

- **Study and Presentation**: Present in detail a recent research paper from a top conference/journal.
- **Writing**: Create a survey or lecture notes on a few related papers.
- **Implementation**: Implement some recent algorithm, or possibly multiple algorithms to compare, trying to find new heuristics that might provide practical speedup.
- **Open Problem**: Attack and try to solve an open problem in the field.

All projects conclude with a presentation, which can be scheduled during one of the 5 official exam dates. There is no strict deadline, but earlier completion is encouraged. Students are allowed and encouraged to work in teams, but the scope of the project should match the team size.

## References

[1] A. Abboud, F. Grandoni, and V. Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP, and diameter. ACM Trans. Algorithms, 19(1):3:1–3:30, 2023.

[2] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In Proceedings of the 46th Annual ACM SIGACT Symposium on Theory of Computing (STOC), pages 574–583, 2014.

[3] A. Antoniadis, M. Eliáš, A. Polak, and M. Venzin. Approximation algorithms for combinatorial optimization with predictions. CoRR, abs/2411.16600, 2024.

[4] A. Azarmehr, S. Behnezhad, R. Jayaram, J. Lacki, V. Mirrokni, and P. Zhong. Massively parallel minimum spanning tree in general metric spaces. In Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 143–174, 2025.

[5] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). SIAM J. Comput., 47(3):1087–1097, 2018.

[6] S. Behnezhad, M. Hajiaghayi, and D. G. Harris. Exponentially faster massively parallel maximal matching. J. ACM, 70(5):34:1–34:18, 2023.

[7] Z. Benomar and C. Coester. Learning-augmented priority queues. In Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS), 2024.

[8] A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In Proceedings of the 63rd IEEE Annual Symposium on Foundations of Computer Science (FOCS), pages 600–611, 2022.

[9] K. Bringmann, A. Cassis, and N. Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! In Proceedings of the 64th IEEE Annual Symposium on Foundations of Computer Science (FOCS), pages 515–538, 2023.

[10] G. S. Brodal, G. Lagogiannis, and R. E. Tarjan. Strict Fibonacci heaps. ACM Trans. Algorithms, 21(2), 2025.

[11] A. Cassis, A. Karrenbauer, A. Nusser, and P. L. Rinaldi. Algorithm engineering of SSSP with negative edge weights. CoRR, abs/2502.11999, 2025.

[12] J. Y. Chen, S. Silwal, A. Vakilian, and F. Zhang. Faster fundamental graph algorithms via learned predictions. In International Conference on Machine Learning (ICML), volume 162 of Proceedings of Machine Learning Research, pages 3583–3602. PMLR, 2022.

[13] J. Chledowski, A. Polak, B. Szabucki, and K. T. Zolna. Robust learning-augmented caching: An experimental study. In Proceedings of the 38th International Conference on Machine Learning (ICML), volume 139 of Proceedings of Machine Learning Research, pages 1920–1930. PMLR, 2021.

[14] S. Coy and A. Czumaj. Deterministic massively parallel connectivity. SIAM J. Comput., 52(5):1269–1318, 2023.

[15] S. Davies, B. Moseley, S. Vassilvitskii, and Y. Wang. Predictive flows for faster Ford-Fulkerson. In International Conference on Machine Learning (ICML), volume 202 of Proceedings of Machine Learning Research, pages 7231–7248. PMLR, 2023.

[16] L. Dhulipala, D. Eisenstat, J. Lacki, V. S. Mirrokni, and J. Shi. Hierarchical agglomerative graph clustering in nearly-linear time. In Proceedings of the 38th International Conference on Machine Learning (ICML), volume 139 of Proceedings of Machine Learning Research, pages 2676–2686, 2021.

[17] M. Dinitz, S. Im, T. Lavastida, B. Moseley, A. Niaparast, and S. Vassilvitskii. Binary search with distributional predictions. In Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS), 2024.

[18] M. Dinitz, S. Im, T. Lavastida, B. Moseley, and S. Vassilvitskii. Controlling tail risk in online ski-rental. In Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 4247–4263, 2024.

[19] M. Farach-Colton, A. Krapivin, and W. Kuszmaul. Optimal bounds for open addressing without reordering. In Proceedings of the 65th IEEE Annual Symposium on Foundations of Computer Science (FOCS), pages 594–605, 2024.

[20] N. Fischer, B. Haeupler, R. Latypov, A. Roeyskoe, and A. L. Sulser. A simple parallel algorithm with near-linear work for negative-weight single-source shortest path. In Proceedings of the 2025 Symposium on Simplicity in Algorithms (SOSA), pages 216–225, 2025.

[21] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM, 34(3):596–615, 1987.

[22] M. Gairing and M. Klimm. Greedy metric minimum online matchings with random arrivals. Oper. Res. Lett., 47(2):88–91, 2019.

[23] J. Gan, A. Wirth, and Z. Zhang. O(1)-round MPC algorithms for multidimensional grid graph connectivity, euclidean MST and DBSCAN. In Proceedings of the 28th International Conference on Database Theory (ICDT), volume 328, pages 7:1–7:20, 2025.

[24] V. Gupta, R. Krishnaswamy, and S. Sandeep. Permutation strikes back: the power of recourse in online metric matching. In Proceedings of the 23rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), pages 40:1–40:20, 2020.

[25] B. Haeupler, R. Hladík, V. Rozhon, R. E. Tarjan, and J. Tetek. Universal optimality of Dijkstra via beyond-worst-case heaps. In Proceedings of the 65th IEEE Annual Symposium on Foundations of Computer Science (FOCS), pages 2099–2130, 2024.

[26] T. Harada and T. Itoh. A nearly optimal deterministic algorithm for online transportation problem. CoRR, abs/2406.03778, 2024.

[27] H. Kaplan, R. E. Tarjan, and U. Zwick. Fibonacci heaps revisited. CoRR, abs/1407.5750, 2014.

[28] S. Lattanzi, O. Svensson, and S. Vassilvitskii. Speeding up Bellman Ford via minimum violation permutations. In International Conference on Machine Learning (ICML), volume 202 of Proceedings of Machine Learning Research, pages 18584–18598. PMLR, 2023.

[29] Q. C. Liu and C. Seshadhri. Brief announcement: Improved massively parallel triangle counting in O(1) rounds. In Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing (PODC), pages 519–522, 2024.

[30] B. Moseley and J. R. Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. J. Mach. Learn. Res., 24:1:1–1:36, 2023.

[31] B. Moseley, H. Niaparast, and K. Singh. Faster global minimum cut with predictions. CoRR, abs/2503.05004, 2025.

[32] G. Patlin and J. van den Brand. Sublinear-time algorithm for MST-weight revisited. In Proceedings of the 2025 Symposium on Simplicity in Algorithms (SOSA), pages 46–53, 2025.

[33] E. Peserico. Paging with dynamic memory capacity. In Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS), pages 56:1–56:18, 2019.

[34] E. Peserico and M. Scquizzato. Matching on the line admits no $o(\sqrt{\log n})$-competitive algorithm. ACM Trans. Algorithms, 19(3), 2023.

[35] C. Sanford, D. Hsu, and M. Telgarsky. Transformers, parallel computation, and logarithmic depth. In Proceedings of the 41st International Conference on Machine Learning (ICML), 2024.

[36] C. Sanford, B. Fatemi, E. Hall, A. Tsitsulin, M. Kazemi, J. Halcrow, B. Perozzi, and V. Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. In Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS), 2024.

[37] V. V. Vazirani. Online bipartite matching and Adwords (invited talk). In Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science (MFCS), volume 241 of LIPIcs, pages 5:1–5:11, 2022.

[38] G. Yehudai, C. Sanford, M. Bechler-Speicher, O. Fischer, R. Gilad-Bachrach, and A. Globerson. Depth-width tradeoffs in algorithmic reasoning of graph tasks with transformers. CoRR, abs/2503.01805, 2025.