Recent Advances in Fully Dynamic Graph Algorithms – A Quick Reference Guide

KATHRIN HANAUER, University of Vienna, Faculty of Computer Science, Austria
MONIKA HENZINGER, University of Vienna, Faculty of Computer Science, Austria
CHRISTIAN SCHULZ, Heidelberg University, Germany

In recent years, significant advances have been made in the design and analysis of fully dynamic algorithms. However, these theoretical results have received very little attention from the practical perspective. Few of the algorithms are implemented and tested on real datasets, and their practical potential is far from understood. Here, we present a quick reference guide to recent engineering and theory results in the area of fully dynamic graph algorithms.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Networks** → **Network dynamics**; • **Mathematics of computing** → **Graph algorithms**.

1 INTRODUCTION

A (fully) dynamic graph algorithm is a data structure that supports edge insertions, edge deletions, and answers certain queries that are specific to the problem under consideration. There has been a lot of research on dynamic algorithms for graph problems that are solvable in polynomial time by a static algorithm. The most studied dynamic problems are graph problems such as connectivity, reachability, shortest paths, or matching (see [124]). Typically, any dynamic algorithm that can handle edge insertions can be used as a static algorithm by starting with an empty graph and inserting all m edges of the static input graph step-by-step. A fundamental question that arises is which problems can be *fully dynamized*, which boils down to the question whether they admit a dynamic algorithm that supports updates in $O(T(m)/m)$ time, where $T(m)$ is the static running time. Thus, for static problems that can be solved in near-linear time, the research community is interested in near-constant time updates. By now, such results have been achieved for a wide range of problems [124], which resulted in a rich algorithmic toolbox spanning a wide range of techniques. However, while there is a large body of theoretical work on efficient dynamic graph algorithms, most of these algorithms were never implemented and empirically evaluated. For some classical dynamic algorithms, experimental studies have been performed, such as early works on (all pairs) shortest paths [80, 96] or transitive closure [164] and later contributions for fully dynamic graph clustering [84] and fully dynamic approximation of betweenness centrality [36]. However, for other fundamental dynamic graph problems, the theoretical algorithmic ideas have received very little attention from the practical perspective. In particular, very little work has been devoted to engineering such algorithms and providing efficient implementations in practice. Previous surveys on the topic [10, 260] are more than twenty years old and do not capture the state-of-the-field anymore. In this work, we aim to survey recent progress in theory as well as

Authors' addresses: Kathrin Hanauer, kathrin.hanauer@univie.ac.at, University of Vienna, Faculty of Computer Science, Währinger Str. 29, Vienna, Vienna, Austria, 1090; Monika Henzinger, monika.henzinger@univie.ac.at, University of Vienna, Faculty of Computer Science, Währinger Str. 29, Vienna, Vienna, Austria, 1090; Christian Schulz, christian.schulz@informatik.uni-heidelberg.de, Heidelberg University, Im Neuenheimer Feld 205, Heidelberg, Baden-Württemberg, Germany, 69120.

1084-6654/2022/8-ART $15.00
https://doi.org/10.1145/3555806

in the empirical evaluation of fully dynamic graph algorithms and summarize methodologies used to evaluate such algorithms. Moreover, we point to theoretical results that we think have a good potential for practical implementations. Hence, this paper should help an unfamiliar reader by providing most recent references for various problems in fully dynamic graph algorithms. Lastly, there is currently a lack of fully dynamic real-world graphs available online – most of the instances that can be found to date are insertions-only. Hence, together with this survey we will also start a new open-access graph repository that provides fully dynamic graph instances[1][2].

We want to point out that there are also various dynamic graph models which we cannot discuss in any depth for space limitations. These are insertions-only algorithms, deletions-only algorithms, offline dynamic algorithms, algorithms with vertex insertions and deletions, kinetic algorithms, temporal algorithms, algorithms with a limit on the number of allowed queries, algorithms for the sliding-windows model, and algorithms for sensitivity problems (also called emergency planning or fault-tolerant algorithms). We also exclude dynamic algorithms in other models of computation such as distributed algorithms and algorithms in the massively parallel computation (MPC) model. If the full graph is known at preprocessing time and vertices are "switched on and off", this is called the *subgraph model*, whereas *algorithms under failures* deal with the case that vertices or edges are only "switched off". We do not discuss these algorithms either.

Note that fully dynamic graph algorithms (according to our definition) are also sometimes called *algorithms for evolving graphs* or *for incremental graphs* or sometimes even *maintaining a graph online.*

2 PRELIMINARIES

Basic Definitions. Let $G = (V, E)$ be a (un)directed graph with vertex set V and edge set E. Throughout this paper, let $n = |V|$ and $m = |E| \in O(n^2)$. The *density* of G is $d = \frac{m}{n}$. In the directed case, an edge $(u, v) \in E$ has *tail* u and *head* v and u and v are said to be *adjacent*. (u, v) is said to be an *outgoing* edge or *out-edge* of u and an *incoming* edge or *in-edge* of v. The *outdegree* $\deg^+(v)$/*indegree* $\deg^-(v)$/*degree* $\deg(v)$ of a vertex v is its number of (out-/in-) edges. The *out-neighborhood* (*in-neighborhood*) of a vertex u is the set of all vertices v such that $(u, v) \in E$ ($(v, u) \in E$). In the undirected case, $N(v) := \{u : \{v, u\} \in E\}$ denotes the *neighbors* of v. The degree of a vertex v is $\deg(v) := |N(v)|$ here.

A graph is *weighted* if there is additionally a function $w : E \to \mathbb{R}$ that assigns each edge $e \in E$ some weight $w(e)$. The maximum weight of any edge in G is denoted as W and C refers to the ratio of the largest to the smallest edge weight. In general, the graph is unweighted unless explicitly mentioned otherwise.

We use $\tilde{O}(\cdot)$ to hide polylogarithmic factors.

Dynamic Graphs. Our focus in this paper are *fully dynamic graphs*, where the set of vertices is fixed, but edges can be added and removed. A fully dynamic graph can be seen as a sequence of graphs where two consecutive graphs differ by exactly one edge. As we only discuss fully dynamic graph algorithms in this survey, we drop the term "fully" when it does not lead to a confusion. For very few problems insertions and deletions of *vertices* instead of *edges* have been studied as well. If so, we will mention them briefly as well. In case of weighted graphs, an update operation might be supported by the dynamic graph algorithm, namely an update operation that changes the weight of a single edge. If this is the case, we state it explicitly.

Formally, we assume that a dynamic graph algorithm starts with a (possibly empty) initial graph and may perform some *preprocessing* on it. It is then presented with a sequence of *updates*, which may in each case either be an *edge insertion*, an *edge deletion*, or, in case of a weighted graph, a *weight change* of a single edge. Unless stated otherwise, (a) the initial graph is empty and during preprocessing the data structure is initialized in $O(n)$ time and (b) updates are given to the dynamic algorithm one-by-one and it is not able to look ahead in time.

[1]If you have access to fully dynamic instances, we are happy to provide them in our repository.
[2]https://DynGraphLab.github.io

The sequence of updates may be interspersed with an arbitrary number of *queries* for an algorithm's current solution. Depending on the problem setting and the algorithm, the result of a query may either be an entire solution or just a numeric or boolean value. A common approach for non-boolean queries is that the algorithm returns a numeric value, but the entire solution can be obtained optionally in time linear in the size of the solution. In some scenarios, the updates are grouped into *batches* of variable size B and an algorithm can process the updates in a batch in arbitrary order before the next batch or query arrives. Algorithms that are able to handle batches of updates are also called *batch-dynamic*. We use *operation* as an umbrella term for both updates and queries. If the running time for an operation depends on n or m (or W or C, if applicable), they always refer to the *current* graph unless denoted otherwise.

We do not consider *partially dynamic* algorithms here, i.e., algorithms that can either only deal with edge insertions (also called *incremental* algorithms) or only with edge deletions (also called *decremental* algorithms).

Parallel Algorithms. We consider both sequential and parallel algorithms, where in case of the latter, the operations are performed by a usually unspecified number of p processors operating in parallel. In this context, *work* refers to the sum of the running time of all processors, whereas the *depth* or *span* is the maximum running time of a single processor if p is infinite. The depth therefore corresponds to the length of a longest path of operations that need to be processed sequentially and imposes a lower bound on the smallest achievable (elapsed) running time.

Conditional Lower Bounds. There are lower bounds for fully dynamic graph algorithms based on various popular conjectures initiated by [3, 126, 193]. These lower bounds usually involve three parameters: the preprocessing time $p(m,n)$, the update time $u(m,n)$, and the query time $q(m,n)$. We will use the notation $(p(m,n), u(m,n), q(m,n))$ below to indicate that *no algorithm with preprocessing time at most* $p(m,n)$ *exists that requires update time at most* $u(m,n)$ and *query time at most* $q(m,n)$. Note that if the preprocessing time is larger than $p(m,n)$ or if the query time is larger than $q(m,n)$, then it might be possible to achieve an update time better than $u(m,n)$. In the same vein, if the preprocessing time is larger than $p(m,n)$ or if the update time is larger than $u(m,n)$, then it might be possible to achieve a query time better than $q(m,n)$. We will write $\text{poly}(\cdot)$ to denote any running time that is *polynomial* in the parameters.

Any conditional lower bound that is based on the OMv (Online Boolean Matrix-Vector Multiplication) conjecture [126] applies to both the (amortized or worst-case) running time of any fully dynamic algorithm *and also* to the worst-case running time of insertions-only and deletions-only algorithms. We will not mention this for each problem below and only state the lower bound, except in cases where as a result of the lower bound only algorithms for the insertions-only or deletions-only setting have been studied.

For many of the weighted graph problems the conditional lower bounds continue to hold even if the only update operations are edge weight changes, even if the edge weight can only be increased or decreased by a constant. More specifically, for any small $\epsilon > 0$ a conditional lower bound of $(\text{poly}(n), n^{1-\epsilon}, n^{2-\epsilon})$ can be achieved for weighted matching, maximum flow and shortest paths [129]. The same paper presents almost tight upper bounds for these problems. It also shows a (unconditional) lower bound of $\Omega(\log n)$ on the time per update or query operation for dynamically maintaining a minimum spanning tree. Due to these strong lower bounds we do not discuss edge weight changes in the theoretical results below any further.

3 FULLY DYNAMIC GRAPH ALGORITHMS

In this section, we describe recent efforts in fully dynamic graph algorithms. We start by describing fundamental problems that we think belong to a basic toolbox of fully dynamic graph algorithms: strongly connected components, minimum spanning trees, cycle detection/topological ordering, matching, core decomposition, subgraph detection, diameter, as well as independent sets. Later on, we discuss problems that are closer to the application

side. To this end we include fully dynamic algorithms for shortest paths, maximum flows, graph clustering, centrality measures, and graph partitioning.

3.1 (Strongly) Connected Components and BFS/DFS Trees

One of the most fundamental questions on graphs is whether two given vertices are connected by a path. In the undirected case, a path connecting two vertices u and w is a sequence of edges $\mathcal{P} = (\{u, v_0\}, \{v_0, v_1\}, \ldots, \{v_k, w\})$. A *connected component* is a maximal set of vertices that are pairwise connected by a path. A graph is *connected* if there is exactly one connected component, which is V. In a directed graph, we say that a vertex u can *reach* a vertex w if there is a directed path from u to w, i.e., a sequence of directed edges $\mathcal{P} = ((u, v_0), (v_0, v_1), \ldots, (v_k, w))$. A *strongly connected component* (SCC) is a maximal set of vertices that can reach each other pairwise. A directed graph is *strongly connected* if there is just one strongly connected component, which is V. The *transitive closure* of a graph G is a graph on the same vertex set with an edge $(u, w) \in V \times V$ if and only if u can reach w in G. Given an undirected graph, we can construct a directed graph from it by replacing each undirected edge $\{u, w\}$ by a pair of directed edges (u, w) and (w, u) and translate queries of connectedness into reachability queries on the directed graph.

A breadth-first search (BFS) or depth-first search (DFS) traversal of a directed or undirected graph defines a rooted tree that consists of the edges via which a new vertex was discovered. Apart from connectivity or reachability, BFS and DFS trees can be used to answer a variety of problems on graphs, such as testing bipartiteness, shortest paths in the unweighted setting, 2-edge connectivity, or biconnectivity.

3.1.1 Undirected Graphs (Connectivity).

Theory Results. Patrascu and Demaine [194] gave an (unconditional) lower bound of $\Omega(\log n)$ per operation for this problem, improving a bound of $\Omega(\log n / \log \log n)$ [133]. The first non-trivial dynamic algorithms for connectivity, and also for 2-edge connectivity, and 2-vertex connectivity [87, 88, 95, 131, 132] took time $\tilde{O}(\sqrt{n})$ per operation, including a query which is given two vertices and returns whether they are suitably connected. Henzinger and King [136] were the first to give a fully dynamic algorithm with polylogarithmic time per operation for this problem. Their algorithm is, however, randomized. Holm et al. [138] gave the first deterministic fully dynamic algorithm with polylogarithmic time per operation. The currently fastest fully dynamic connectivity algorithm takes $O(\log n(\log \log n)^2)$ amortized expected time per operation [143]. There also is a batch-dynamic parallel algorithm that answers k queries in $O(k \log(1 + n/k))$ expected work and $O(\log n)$ depth with $O(\log n \log(1 + n/B))$ expected amortized work per update and $O(\log^3 n)$ depth for an average batch size of B [6].

The fully dynamic connectivity problem can be reduced to the maintenance of a spanning forest, using, e.g., dynamic trees [7, 234] or Euler tour trees [134, 244] (see also Section 3.2), for the components. If the graph is a forest, updates and queries can be processed in amortized $O(\log n)$ time, whereas the theoretically fastest algorithms [152] to date for general graphs have polylogarithmic worst-case update time and $O(\log n / \log \log n)$ worst-case query time, the latter matching the lower bound [133, 178]. The key challenge on general graphs is to determine whether the deletion of an edge of the spanning forest disconnects the component or whether a replacement edge can be found. There are also fully dynamic algorithms for more refined notions of connectivity: Two-edge connectivity [136, 137] and two-vertex connectivity [137] can also be maintained in polylogarithmic time per operation. See [145] for a survey on that topic.

Experimental Results. Building on an earlier study by Alberts et al. [10], Iyer et al. [148] experimentally compared the Euler tour tree-based algorithms by Henzinger and King [134] and Holm et al. [137] to each other as well as several heuristics to achieve speedups in both candidates. The instances used in the evaluation were random graphs with random edge insertions and deletions, random graphs where a fixed set of edges appear and disappear dynamically, graphs consisting of cliques of equal size plus a set of inter-clique edges, where

only the latter are inserted and deleted, as well as specially crafted worst-case instances for the algorithms. The authors showed that the running time of both algorithms can be improved distinctly via heuristics; in particular a sampling approach to replace deleted tree edges has proven to be successful. The experimental running time of both algorithms was comparable, but with the heuristics, the algorithm by Holm et al. [137] performed better.

Baswana et al. [26] gave the first algorithm for maintaining an undirected DFS tree with $o(m)$ update time and showed a conditional lower bound of $\Omega(n)$ on the update time in case of vertex updates and, if the tree is maintained explicitly, an unconditional lower bound of $\Omega(n)$ under edge updates. Their algorithm has a preprocessing time of $O(m \log n)$, a worst-case update time of $O(\sqrt{mn} \log^{2.5} n)$, and uses $O(m \log^2 n)$ bits. Nakamura and Sadakane [182] improved the update time by polylog n factors and the space required to $O(m \log n)$. Recently, Baswana et al. [28] further reduced the update time down to $O(\sqrt{mn \log n})$. A parallel algorithm that uses m processors and $O(\text{polylog } n)$ update time was given by Khan [156]. To the best of our knowledge, experimental evaluations have only been conducted to date with algorithms designed for the *incremental* setting, but not for fully-dynamic algorithms.

No experimental studies on dynamically maintaining BFS trees are known to us.

3.1.2 Directed Graphs (Reachability, Strong Connectivity, Transitive Closure).

Theory Results. For directed graphs that are and remain acyclic, the same algorithms can be employed for reachability as for (undirected) connectivity in forests (see above). On general graphs, there is a conditional lower bound of $(\text{poly}(n), m^{1/2-\epsilon}, m^{1-\epsilon})$ for any small constant $\epsilon > 0$ based on the OMv conjecture. This bound even holds for the *s-t* reachability problem, where both s and t are fixed for all queries. The currently fastest algorithms for transitive closure are three Monte Carlo algorithms with one-sided error: Two by Sankowski [217] with $O(1)$ or $O(n^{0.58})$ worst-case query time and $O(n^2)$ or $O(n^{1.58})$ worst-case update time, respectively, and one by van den Brand, Nanongkai, and Saranurak [246] with $O(n^{1.407})$ worst-case update and worst-case query time. Both algorithms can only answer a reachability query, but if a vertex x can reach a vertex y, the algorithms cannot actually output a path from x to y. There exists a conditional lower bound based on a variant of the OMv conjecture that shows that these running times are optimal [246]. Moreover, there are two deterministic, combinatorial algorithms: Roditty's algorithm with constant query time and $O(n^2)$ amortized update time [209], as well as one by Roditty and Zwick [212] with an improved $O(m + n \log n)$ amortized update time at the expense of $O(n)$ worst-case query time. The former algorithm can also return a directed path from a given vertex x to a given vertex y if such a path exists in time linear in its length, and also supports the update of a vertex, not only of an edge in time $O(n^2)$.

Experimental Results for Single-Source Reachability. The single-source reachability problem, where a vertex s is fixed and reachability from s to any other vertex v is the subject of queries, was first studied experimentally by Hanauer et al. [119]. Here, two relatively straightforward algorithms, called SI and SES, showed outstanding performance in practice on both random graphs as well as on real-world instances in comparison to the static baseline algorithms breadth-first and depth-first search, which were up to this point the recommended choice to maintain reachability in a dynamic setting [98, 164]. SI maintains an arbitrary reachability tree which is re-constructed via a combined forward and backward breadth-first search traversal on edge deletions if necessary and is especially fast if insertions predominate, which can be handled in $O(n + m)$ time. By contrast, it may take up to $O(nm)$ time for a single edge removal. SES is an extension and simplification of Even-Shiloach trees [232], which originally only handle edge deletions. Its strength are hence instances with many deletions. As a plus, it is able to deliver not just any path as a witness for reachability, but even the shortest path (with respect to the number of edges). Furthermore, it internally maintains a BFS tree, which makes it viable also for numerous other applications, see above. Its worst-case update time is $O(n + m)$, and, like SI, it answers queries in constant time.

One key ingredient for the superior performance of both algorithms in practice are carefully chosen criteria for an abortion of the re-construction of their data structures and their re-building from scratch.

Experimental Results for Transitive Closure. Frigioni et al. [98] and later Krommidas and Zaroliagis [164] empirically studied the performance of an extensive number of algorithms for transitive closure, including those mentioned above. They also developed various extensions and variations and compared them not only to each other, but also to static, so-called "simple-minded" algorithms such as breadth-first and depth-first search. Their evaluation included random Erdős-Renyí graphs, specially constructed hard instances, as well as two instances based on real-world graphs. It showed that the "simple-minded" algorithms could outperform the dynamic ones distinctly and up to several factors, unless the query ratio was more than 65 % or the instances were dense random graphs.

In a follow-up experimental study by Hanauer et al. [118], the authors showed that the "simple-minded" algorithms can be outperformed by several orders of magnitude in practice again on both random and real-world dynamic graphs by maintaining multiple instances of their single-source reachability algorithms simultaneously. To query the transitive closure of a graph, a number of so-called "supportive vertices", for which both in- and out-reachability trees are maintained explicitly, can be picked either once or periodically anew and then be used to answer both positive and negative reachability queries between a number of pairs of vertices decisively in constant time. The fallback routine can be a simple static graph traversal and therefore be relatively expensive: With a random initial choice of supportive vertices and no periodic renewals, this approach has been shown to answer a great majority of reachability queries on both random and real-world instances in constant time already if the number of supportive vertices is very small, i.e., two or three.

These experimental studies clearly show the limitations of worst-case analysis: All implemented algorithms are fully dynamic with at least linear worst-case running time per operation and, thus, all perform "(very) poor" in the worst case. Still on all graphs used in the study the relatively simple new algorithms clearly outperformed the algorithms used in previous studies.

Experimental Results for DFS/BFS Trees. Yang et al. [257] were the first to give a fully dynamic algorithm for maintaining a DFS tree in a directed graph along with several optimizations to achieve speedups in practice. In an experimental evaluation on twelve real-world instances, they showed that the optimized version of their algorithm can handle edge insertions and deletions within few seconds on average for instances with millions of vertices. With regard to BFS trees, the already mentioned SES algorithm [119] is the only fully dynamic algorithm we are aware of that maintains a BFS tree on a directed graph.

3.2 Minimum Weight Spanning Trees

A minimum weight spanning tree (MST) of a connected graph is a subset of the edges such that all nodes are connected via the edges in the subset, the induced subgraph has no cycles and, lastly, has the minimum total weight among all possible subsets fulfilling the first two properties.

Theory Results. The lower bound of $\Omega(\log n)$ [194] on the time per operation for connectivity trivially extends to maintaining the weight of a minimum spanning tree. Holm et al. [138] gave the first fully dynamic algorithm with polylogarithmic time per operation for this problem. It was later slightly improved to $O(\log^4 n / \log\log n)$ time per operation [139].

Experimental Results. Amato et al. [144] presented the first experimental study of dynamic minimum spanning tree algorithms. In particular, the authors implemented different versions of Frederickson's algorithm [94] which uses partitions and topology trees. The algorithms have been adapted with sparsification techniques to improve their performance. The update running times of these algorithms range from $O(m^{2/3})$ to $O(m^{1/2})$. The authors

further presented a variant of Frederickson's algorithm that is significantly faster than all other implementations of this algorithm. However, the authors also proposed a simple adaption of a partially dynamic data structure of Kruskal's algorithm that was the fastest implementation on random inputs. Later, Cattaneo et al. [60, 61] presented an experimental study on several algorithms for the problem. The authors presented an efficient implementation of the algorithm of Holm et al. [138], proposed new simple algorithms for dynamic MST that are not as asymptotically efficient as the algorithm by Holm et al. but seem to be fast in practice, and lastly compared their algorithms with the results of Amato et al. [144]. The algorithm by Holm et al. uses a clever refinement of a technique by Henzinger and King [135] for developing fully dynamic algorithms starting from the deletions-only case. One outcome of their experiments is that simple algorithms outperform the theoretically more refined algorithms on random and worst-case networks. On the other hand, on k-clique inputs, i.e., graphs that contain k cliques of size c plus $2k$ randomly chosen inter-clique edges, the implementation of the algorithm by Holm et al. outperformed the simpler algorithms.

Tarjan and Werneck [239] performed experiments for several variants of dynamic trees data structure. The evaluated data structures have been used by Ribero and Toso [207], who focused on the case of changing weights, i.e., the edges of the graph are constant, but the edge weights can change dynamically. Ribero and Toso also proposed and used a new data structure for dynamic tree representation called DRD-trees. In their algorithm the dynamic tree data structure is used to speed up connectivity queries that check whether two vertices belong to different subtrees. More generally, Ribero and Toso compared different types of data structures to do this task. In particular, the authors used the dynamic tree data structures that have been evaluated by Tarjan and Werneck [239]. The experimental evaluation demonstrated that the new structure reduces the computation time observed for the algorithm of Cattaneo et al. [60], and at the same time yielded the fastest algorithms in the experiments.

3.3 Cycle Detection and Topological Ordering

A cycle in a (directed) graph $G = (V, E)$ is a non-empty path $\mathcal{P} = (v_1, \ldots, v_k = v_1)$ such that $(v_i, v_{i+1}) \in E$. A topological ordering of a directed graph is a linear ordering of its vertices from 1 to n such that for every directed edge (u, v) from vertex u to vertex v, u is ordered before v. In the static case, one can use a depth-first search (DFS) to compute a topological ordering of a directed acyclic graph or to check if a (un)directed graph contains a cycle.

Theory Results. Let $\epsilon > 0$ be any small constant. Based on the OMv conjecture [126] it is straightforward to construct a lower bound of $(\text{poly}(n), m^{1/2-\epsilon}, m^{1-\epsilon})$ for the (amortized or worst-case) running time of any fully dynamic algorithm that detects whether the graph contains any cycle. As any algorithm for topological ordering can be used to decide whether a graph contains a cycle, this lower bound also applies to any fully dynamic topological ordering algorithm. Via dynamic matrix inverse one can maintain fully dynamic directed cycle detection in $O(n^{1.407})$ [246], which is conditionally optimal based on a variant of the OMv conjecture.

Experimental Results. Pearce and Kelly [197, 198] were the first to evaluate algorithms for topological ordering in the presence of edge insertions and deletions. In their work, the authors compared three algorithms that can deal with the online topological ordering problem. More precisely, the authors implemented the algorithms by Marchetti-Spaccamela et al. [174] and Alpern et al. [12] as well as a newly developed algorithm. Their new algorithm is the one that performed best in their experiments. The algorithm maintains a node-to-index map, called $n2i$, that maps each vertex to a unique integer in $\{1 \ldots n\}$ and ensures that for any edge (u, v) in G, it holds $n2i[u] < n2i[v]$. When an insertion (u, v) invalidates the topological ordering, affected nodes are updated. The set of affected nodes are identified using a forward DFS from v and backward DFS from u. The two sets are then separately sorted into increasing topological order and afterwards a remapping to the available indices is

performed. The algorithm by Marchetti-Spaccamela et al. [174] is quite similar to the algorithm by Pearce and Kelly. However, it only maintains the forward set of affected nodes and obtains a correct solution by shifting the affected nodes up in the ordering (putting them after u). Alpern et al. [12] used a data structure to create new priorities between existing ones in constant worst-case time. The result by Pearce and Kelly has later been applied to online cycle detection and difference propagation in pointer analysis by Pearce et al. [199]. Furthermore, Pearce and Kelly [196] later extended their algorithm to be able to provide more efficient batch updates.

3.4 (Weighted) Matching

The matching problem is one of the most prominently studied combinatorial graph problems having a variety of practical applications. A matching $\mathcal{M}$ of a graph $G = (V, E)$ is a subset of edges such that no two elements of $\mathcal{M}$ have a common end point. Many applications require matchings with certain properties, like being maximal (no edge can be added to $\mathcal{M}$ without violating the matching property) or having maximum cardinality.

3.4.1 Cardinality Matching.

Theory Results. There is a conditional lower bound of $(\text{poly}(n), m^{1/2-\delta}, m^{1-\delta})$ (for any small constant $\delta > 0$) for the size of the maximum cardinality matching based on the OMv conjecture [126]. Of course, maintaining an actual maximum matching is only harder than maintaining the size of a maximum matching. Thus upper bounds have mostly focused on approximately maximum matching. However, also here we have to distinguish between (a) algorithms that maintain the *size* of an approximately maximum matching and (b) algorithms that maintain an *approximately maximum matching*. The algorithms below either assume that the algorithm is started on an empty graph or, in the case of the exact algorithms with non-empty initial graphs, they use $O(n^\omega)$ preprocessing time.

Note that the query time of all type-(a) algorithms is constant. As type-(b) algorithms maintain an actual matching the queries they support are more flexible: they can output the matching in time linear in its size, answer queries whether a given edge belongs to the matching in constant time, and output the size of the matching in constant time.

(a) Improving Sankowski's $O(n^{1.495})$ update time bound [219], van den Brand et al. [246] maintain the exact size of a maximum matching in $O(n^{1.407})$ update time. To maintain the approximate size of the maximum matching, dynamic algorithms use the duality of maximum matching and vertex cover and maintain instead a $(2 + \epsilon)$-approximate vertex cover. This line of work lead to a sequence of papers [42, 43, 45, 146], resulting finally in a deterministic $(2+\epsilon)$-approximation algorithm that maintains a hierarchical graph decomposition with $O(1/\epsilon^2)$ amortized update time [50]. The algorithm can be turned into an algorithm with worst-case $O(\log^3 n)$ time per update [46].

(b) One can trivially maintain a maximal matching in $O(n)$ update time by resolving all trivial augmenting paths, i.e., cycle-free paths that start and end on a unmatched vertex and where edges from $\mathcal{M}$ alternate with edges from $E \setminus \mathcal{M}$, of length one. As any maximal matching is a 2-approximation of a maximum matching, this leads to a 2-approximation algorithm. Onak and Rubinfeld [191] presented a randomized algorithm for maintaining an $O(1)$-approximate matching with $O(\log^2 n)$ expected amortized time per edge update. Baswana, Gupta, and Sen [27] gave an elegant algorithm that maintains a *maximal* matching with amortized update time $O(\log n)$. It is based on a hierarchical graph decomposition and was subsequently improved by Solomon to amortized constant expected update time [235].

For worst-case bounds, the best results are a $(1 + \epsilon)$-approximation in $O(\sqrt{m}/\epsilon)$ update time by Gupta and Peng [114] (see [188] for a 3/2-approximation in the same time), a $(3/2 + \epsilon)$-approximation in $O(m^{1/4}/\epsilon^{2.5})$ time by Bernstein and Stein [40], and a $(2 + \epsilon)$-approximation in $O(\text{polylog } n)$ time by Charikar and Solomon [64] and Arar et al. [17]. There exists also algorithms that achieve various tiny improvements over the approximation

factor of 2 [33, 213] with (small) polynomial update time. Recently, Bernstein et al. [39] improved the maximal matching algorithm of Baswana et al. [27] to $O(\log^5 n)$ worst-case time with high probability.

Experimental Results. Despite this variety of different algorithms, to the best of our knowledge, there have been only limited efforts so far to engineer and evaluate these algorithms on real-world instances. Henzinger et al. [125] initiated the empirical evaluation of algorithms for this problem in practice. To this end, the authors evaluated several dynamic maximal matching algorithms as well as an algorithm that is able to maintain the maximum matching. They implemented the algorithm by Baswana, Gupta and Sen [27], which performs edge updates in $O(\sqrt{n})$ time and maintains a 2-approximate maximum matching, the algorithm of Neiman and Solomon [188], which takes $O(\sqrt{m})$ time to maintain a 3/2-approximate maximum matching, as well as two novel dynamic algorithms, namely a random walk-based algorithm as well as a dynamic algorithm that searches for augmenting paths using a (depth-bounded) blossom algorithm. Their experiments indicate that an optimum matching can be maintained dynamically more than an order of magnitude faster than the naive algorithm that recomputes maximum matchings from scratch. Second, all non-optimum dynamic algorithms that have been considered in that work were able to maintain near-optimum matchings in practice while being multiple orders of magnitudes faster than the naive exact dynamic algorithm. The study concludes that in practice an extended random walk-based algorithms is the method of choice.

3.4.2 *Weighted Matching.*

Theory Results. For the *weighted* dynamic matching problem, Anand et al. [14] proposed an algorithm that can maintain an 4.911-approximate dynamic maximum weight matching that runs in amortized $O(\log n \log C)$ time where C is the ratio of the weight of the highest weight edge to the weight of the smallest weight edge. Furthermore, a sequence [1, 42, 44, 47, 49] of work on fully dynamic set cover resulted in $(1+\epsilon)$-approximate weighted dynamic matching algorithms, with $O(1/\epsilon^3 + (1/\epsilon^2)\log C)$ amortized and $O((1/\epsilon^3)\log^2(Cn))$ worst-case time per operation based on various hierarchical hypergraph decompositions. Gupta and Peng [114] maintain a $(1+\epsilon)$-approximation under edge insertions/deletions that runs in time $O(\sqrt{m}\epsilon^{-2-O(1/\epsilon)}\log W)$ time per update, if edge weights are in between 1 and W.

Their result is based on rerunning a static algorithm from time to time, a trimming routine that trims the graph to a smaller equivalent graph whenever possible and in the weighted case, a partition of the weights of the edges into intervals of geometrically increasing size. Stubbs and Williams [237] presented metatheorems for dynamic weighted matching. Here, the authors reduced the dynamic maximum weight matching problem to the dynamic maximum cardinality matching problem in which the graph is unweighted. The authors proved that using this reduction, if there is an α-approximation for maximum cardinality matching with update time T in an unweighted graph, then there is also a $(2+\epsilon)\alpha$-approximation for maximum weight matching with update time $O(\frac{T}{\epsilon^2}\log^2 W)$. Their basic idea is an extension of the algorithm of Crouch and Stubbs [70] who tackled the problem in the streaming model. Here, the reduction is to take matchings from weight-threshold based subgraphs of the dynamic graph, i.e., the algorithm maintains maximal matchings in $\log C$ subgraphs, where subgraph i contains all edges having weight at least $(1+\epsilon)^i$. The resulting matchings are then greedily merged together by considering the matched edges in descending order of i (heaviest edges first).

Experimental Results. Recently, the approach by Stubbs and Williams has been evaluated experimentally and has been compared against a new random walk-based approach [16] which gives a $(1+\epsilon)$ approximation w.h.p.. When inserting or deleting an edge, the random walk-based approach finds random simple paths (using random walks) and solves those paths using dynamic programming to improve the maintained matching. In practice, the random walk-based approach outperforms the approach by Stubbs and Williams significantly.

3.5 k-Core Decomposition

A k-core of a graph is a maximal connected subgraph in which all vertices have degree at least k. The core number of a vertex is the largest value k such that the vertex is still contained in the k-core. The k-core decomposition problem is to compute the core number of every node in the graph. A related problem is that of finding a *densest subgraph*, which is an induced subgraph that has maximum density.

Theory Results. It is well-known that a k-core decomposition can be computed in linear time for a static graph by repeatedly removing vertices of degree less than k from the graph. A 2-approximation for the densest subgraph problem can be obtained by replacing k with the minimum vertex degree in each step until it is empty and returning the induced subgraph on the set of vertices V_i that has the largest density, where $V_0 = V$ and V_i is the set of vertices after step i [63].

The problem of maintaining the k-core decomposition in a fully dynamic graph has not received much attention by the theoretical computer science community: Sun et al. [238] showed that the insertion and deletion of a single edge can change the core value of all vertices. They also gave a $(4 + \epsilon)$-approximate fully dynamic algorithm that maintains core values with polylogarithmic running time. The algorithm can be implemented in time $O(\log^2 n)$ in graphs using the algorithm of Bhattacharya et al. [48]. It dynamically maintains $O(\log_{(1+\epsilon)} n)$ many (α, β)-decompositions of the graph, one for each β-value that is a power of $(1 + \epsilon)$ between 1 and $(1 + \epsilon)n$.

For a subset $Z \subseteq V$ let $\deg_Z(v)$ denote the number of neighbors of v in Z. An (α, β)-*decomposition* of a graph $G = (V, E)$ [48] for $\alpha \geq 1, \beta \geq 0$ is a decomposition $Z_1, \ldots, Z_L$ of V into $L := 1 + \lceil (1 + \epsilon) \log n \rceil$ levels such that $Z_{i+1} \subseteq Z_i$ for all $1 \leq i < L$, $Z_1 = V$, and the following invariants are maintained: (1) All vertices v on level Z_i with $\deg_{Z_i}(v) > \alpha\beta$ belong to Z_{i+1} and (2) all vertices v on level Z_i with $\deg_{Z_i}(v) < \beta$ do not belong to Z_{i+1}. An (α, β)-decomposition is hence an approximate version of the procedure for the static setting described above, where in each step all vertices with degree less than β plus some with degree less than $\alpha\beta$ are removed. The technique can also be used to maintain a $2\alpha(1 + \epsilon)^2$-approximate densest subgraph [48].

There are no further lower bounds, neither conditional nor unconditional, and no faster algorithms known for maintaining an approximate k-core decomposition.

Experimental Results. Miorandi and De Pellegrini [179] proposed two methods to rank nodes according to their k-core number in fully dynamic networks and compare them experimentally. The focus of their work is to identify the most influential spreaders in complex dynamic networks. Li et al. [168] used a filtering method to only update nodes whose core number is affected by the network update. More precisely, the authors showed that nodes that need to be updated must be connected via a path to the endpoints of the inserted/removed edge and the core number must be equal to the smaller core number of the endpoints. Moreover, the authors presented efficient algorithms to identify such nodes as well as additional techniques to reduce the size of the nodes that need updates. Similarly, Sariyüce et al. [220] proposed the k-core algorithm TRAVERSAL and gave additional rules to prune the size of the subgraphs that are guaranteed to contain the vertices whose k-core number can have changed. Note that this algorithm can have a high variation in running time for the update operations depending on the size of the affected subgraphs. Zhang et al. [261] noted that due to this reason it can be impractical to process updates one by one and introduced the k-order concept which can reduce the cost of the update operations. A k-order is defined as follows: a node u is ordered before v in the k-order if u has a smaller core number than v or when the vertices have the same core number, if the linear time algorithm to compute the core decomposition would remove u before v. A recent result by Sun et al. [238] also contains experimental results. However, their main focus is on hypergraphs and there are no comparisons against the algorithms mentioned above.

Aridhi et al. [18] gave a distributed k-core decomposition algorithm in large dynamic graphs. The authors used a graph partitioning approach to distribute the workload and pruning techniques to find nodes that are

affected by the changes. Wang et al. [253] gave a parallel algorithm that appears to significantly outperform the TRAVERSAL algorithm. Jin et al. [149] presented a parallel approach based on matching to update core numbers in fully dynamic networks. Specifically, the authors showed that if a batch of inserted/deleted edges forms a matching, then the core number update step can be performed in parallel. However, the type of the edges has to be the same (i.e., only insertions, or only deletions) in each update. Hua et al. [141] noted that previous algorithms become inefficient for high superior degree vertices, i.e., vertices that have many neighbors that have a core number that is larger than its own core number. For example, the matching-based approach of Jin et al. [149] can only process one edge associated to a vertex in each iteration. Their new algorithm can handle multiple insertions/deletions per iteration.

It would be interesting to evaluate the algorithm of Sun et al. [238] which maintains a $(4+\epsilon)$-approximate core number, on graphs to see how far from the exact core numbers these estimates are and how its running time compares to the above approaches. Note that an (α, β)-decomposition actually gives a $(2\alpha+\epsilon)$ approximation and α has to be chosen to be slightly larger than 2 only to guarantee polylogarithmic updates. Thus, it would be interesting to also experiment with smaller values of α.

3.6 Subgraph and Motif Counting

Two graphs are isomorphic if there is a bijection between the vertex sets of the graphs that preserves adjacency. Given a graph pattern H, *induced subgraph counting*, also called *motif counting*, is concerned with the number of subgraphs of G that are isomorphic to H, divided by the number of automorphisms of H. By contrast, non-adjacencies in H need not be preserved in the *non-induced* case. For example, the non-induced count for so-called wedges (a path of length two) in a triangle is three, whereas the induced count for the same pattern is zero. A query for the induced or non-induced counting problem hence is answered by a single numeric value. Observe that induced and non-induced counts coincide if the pattern graph is a clique.

In the *detection* version of these problems, one is only interested in whether the count is nonzero and upon a query, an algorithm returns a Boolean value. In other scenarios, the subgraphs also need to be *enumerated*. Here, a query is expected to return a sequence of subgraphs and if running time is considered, the *enumeration delay*, i.e., the time between the enumeration of a subgraph and its successor in this sequence during answering a query, is of interest.

In the research that is currently available there is a subset of work that focuses on the special case of counting wedges, triangles, as well as various subgraph patterns on four vertices in dynamic networks. Unless denoted otherwise, graphs and patterns are undirected in the following, which is the more common setting.

Theory Results. There is a conditional lower bound of $(\text{poly}(n), m^{1/2-\epsilon}, m^{1-\epsilon})$ for any small constant $\epsilon > 0$ even for the fundamental problem of detecting whether an graph contains a triangle [126]. The same lower bound also extends to various four-vertex subgraphs, whereas there is a lower bound of $(\text{poly}(n), m^{1-\epsilon}, m^{2-\epsilon})$ for counting 4-cliques as well as *induced* connected four-vertex subgraphs [117]. Interestingly, assuming the OMv-conjecture the problem of dynamically counting 4-cycles remains hard also in the average case: Henzinger, Lincoln, and Saha [127] defined a dynamic random graph model based on Erdős-Rényi random graphs, where the adversary can determine whether to update or query, but in case of an update a random pair of vertices is picked uniformly at random and the corresponding edge is "flipped", i.e., inserted if it does not exist and deleted if it does. They gave a conditional lower bound of $(n^{3-\epsilon}, n^{1-\epsilon}, n^{2-\epsilon})$ for counting the number of 4-cycles in such a dynamic graph and a somewhat weaker polynomial bound for counting the number of triangles.

A fully dynamic algorithm with $O(\sqrt{m})$ update time and constant query time was recently given independently by Kara et al. [153, 154] for counting triangles. Their algorithm is also able to enumerate triangles with $O(1)$ delay. Subsequently, Lu and Tao [171] studied the trade-off between update time and approximation quality and presented a new data structure for exact triangle counting whose complexity depends on the arboricity of the

graph. The results by Kara et al. were extended to general k-clique counting by Dhulipala et al. [81]. Motivated by the fact that real-world graphs in certain applications often have small h-index h (i.e., there are at most h vertices of degree at least h), Eppstein and Spiro [90] showed that the undirected triangle count can be maintained in $O(h)$ update time and constant query time. Eppstein et al. [89] later extended this result to maintaining the counts of directed triangles in amortized $O(h)$ update time and of undirected four-vertex subgraphs in amortized $O(h^2)$ update time. Note that h can be as large as $O(\sqrt{m})$, resulting in an amortized time complexity of $O(m)$ per update for four-vertex subgraphs in general. Only very recently, Hanauer et al. [117] showed how to reduce this to amortized $O(m^{2/3})$ time per update for all four-vertex subgraphs except the 4-clique. The query time remains constant.

This is currently an active area of research.

Experimental Results on Triangle Counting. Pavan et al. [195] introduced neighborhood sampling to count and sample triangles in a one-pass streaming algorithm. In neighborhood sampling, first a random edge in the stream is sampled and in subsequent steps, edges that share an endpoint with the already sampled edges are sampled. The algorithm outperformed their implementations of the previous best algorithms for the problem, namely the algorithms by Jowhari and Ghodsi [151] and by Buriol et al. [57]. Note that the method does not appear to be able to handle edge deletions. Bulteau et al. [56] estimated the number of triangles in fully dynamic streamed graphs. Their method adapts 2-path sampling to work for dynamic graphs. The main idea of 2-path sampling is to sample a certain number of 2-paths and compute the ratio of 2-paths in the sample that are complete triangles. The total number of 2-paths in the graph is then multiplied with the ratio to obtain the total number of 2-paths in the graph. This approach fails, however, if one allows deletions. Thus, the contribution of the paper is a novel technique for sampling 2-paths. More precisely, the algorithm first streams the graph and sparsifies it. Afterwards, the sampling technique is applied on the sparsified graph. The core contribution of the authors is to show that the estimate obtained in the sparsified graph is similar to the number of triangles in the original graph. For graphs with constant transitivity coefficient, i.e., the ratio of 2-paths in G contained in a triangle to all 2-paths in G, the authors achieve constant processing time per edge. Makkar et al. [173] presented an exact and parallel approach using an inclusion-exclusion formulation for triangle counting in dynamic graphs. The algorithm is implemented in cuSTINGER [92] and runs on GPUs. The algorithm computes updates for batches of edge updates and also updates the number of triangles each vertex belongs to. The TRIÈST algorithm [236] estimates local and global triangles. An input parameter of the algorithm is the amount of available memory. The algorithm maintains a sample of the edges using reservoir sampling and random pairing to exploit the available memory as much as possible. The algorithm reduces the average estimation error by up to 90 % w.r.t. to the previous state-of-the-art. Han and Sethu [116] proposed a new sampling approach, called edge-sample-and-discard, which generates an unbiased estimate of the total number of triangles in a fully dynamic graph. The algorithm significantly reduces the estimation error compared to TRIÈST. The MASCOT algorithm [169, 170] focuses on local triangle counting, i.e., counting the triangles adjacent to every node. In their work, the authors provide an unbiased estimation of the number of local triangles.

Experimental Results on Counting More Complex Patterns. The neighborhood sampling method of Pavan et al. [195] can also be used for more complex patterns, for example Pavan et al. also presented experiments for 4-cliques. Shiller et al. [224] presented the stream-based (insertions and deletions) algorithm StreaM for counting 4-vertex motifs in dynamic graphs. Ahmed et al. [8] presented a general purpose sampling framework for graph streams. The authors proposed a martingale formulation for non-induced subgraph count estimation and showed how to compute unbiased estimate of subgraph counts from a sample at any point during the stream. The estimates for triangle and wedge counting obtained are less than 1 % away from the true number of triangles/wedges. The algorithm outperformed their own implementation of TRIÈST and MASCOT. Mukherjee et al. [181] gave an exact counting algorithm for a given set of motifs in dynamic networks. Their focus is on

biological networks. The algorithm computes an initial embedding of each motif in the initial network. Then for each motif its embeddings are stored in a list. This list is then dynamically updated while the graph evolves.

Dhulipala et al. [81] recently gave parallel batch-dynamic algorithms for k-clique counting and enumeration. Their first algorithm is a batch-dynamic parallel algorithm for triangle counting that has amortized work $O(B\sqrt{B+m})$ and $O(\log^*(B+m))$ depth with high probability for a batch of B edge insertions or deletions. The algorithm is based on degree thresholding which divides the vertices into vertices with low- and high-degree. Given the classification of the vertex, different update routines are used. A multicore implementation of the triangle counting algorithm is given. Experiments indicate that the algorithms achieve 36.54 to 74.73-times parallel speedups on a machine with 72 cores. Lastly, the authors developed a simple batch-dynamic algorithm for enumerating k-cliques, which has expected $O(B(m+B)\alpha^{k-4})$ work and $O(\log^{k-2} n)$ depth with high probability, for graphs with arboricity α.

To summarize for this problem the empirical work is far ahead of the theoretical work and it would be interesting to better understand the theoretical complexity of subgraph and motif counting.

3.7 Eccentricity and Diameter

The *eccentricity* of a vertex is the greatest distance between the vertex and any other vertex in an undirected graph. Here, the distance between two vertices in a graph refers to the number of edges in a shortest path between those two vertices. For any vertex v note that the depth of the BFS tree rooted at v equals its eccentricity. Based on this definition, the *diameter* of a graph is defined as the maximum eccentricity over all vertices in the graph. Let x and y be two vertices whose distance $d(x,y)$ equals the graph diameter. Note that the eccentricity $ecc(v)$ of any vertex v is never larger and at most a factor 2 smaller than the diameter, as $2 \cdot ecc(v) \geq 2 \cdot \max(d(x,v), d(y,v)) \geq d(x,v) + d(v,y) \geq d(x,y)$. The *radius* is the minimum eccentricity of all vertices. Let x be the vertex with minimum eccentricity, and let y be a vertex such that $d(x,y) = ecc(x)$. Note that the eccentricity of any vertex v is never smaller and at most a factor 2 larger than the radius, as $d(x,v) \leq d(x,y) = ecc(x)$ and, thus, $ecc(v) \leq d(x,y) + ecc(x) \leq 2ecc(x)$. Thus, through recomputation from scratch it is straightforward to compute a 2-approximation for diameter and radius in linear time.

Theory Results. Anacona et al. [15] recently showed that under the strong exponential time hypothesis (SETH) there can be no $(2-\epsilon)$-approximate fully dynamic approximation algorithm for any of these problems with $O(m^{1-\delta})$ update or query time for any $\delta > 0$. There also exist non-trivial (and sub-n^2 time) fully dynamic algorithms for $(1.5+\epsilon)$ approximate diameter (and also for radius and eccentricities) [246]. In that paper, the authors also construct a non-trivial algorithm for exact diameter.

We are not aware of any experimental study for fully dynamic diameter.

3.8 Independent Set and Vertex Cover

Given a graph $G = (V, E)$, an *independent set* is a set $S \subseteq V$ such that no vertices in S are adjacent to one another. The *maximum independent set problem* is to compute an independent set of maximum cardinality, called a *maximum independent set* (MIS). The *minimum vertex cover* problem is equivalent to the maximum independent set problem: S is a minimum vertex cover C in G iff $V \setminus S$ is a maximum independent set $V \setminus C$ in G. Thus, an algorithm that solves one of these problems can be used to solve the other. Note, however, that this does not hold for approximation algorithms: If C' is an α-approximation of a minimum vertex cover, then $V \setminus C'$ is not necessarily an α-approximation of a maximum independent set. Another related problem is the *maximal* independent set problem. A set S is a maximal independent set if it is an independent set such that for any vertex $v \in V \setminus S$, $S \cup \{v\}$ is not independent.

Theory Results. As computing the size of an MIS is NP-hard, all dynamic algorithms of independent set study the maximal independent set problem. Note, however, that unlike for matching a maximal independent set does not give an approximate solution for the MIS problem, as shown by a star graph. In a sequence of papers [20, 21, 32, 65, 113] the running time for the maximal independent set problem was reduced to $O(\log^4 n)$ expected worst-case update time. All these algorithms actually maintain a maximal independent set. A query can either return the size of that set in constant time or output the whole set in time linear in its size.

Experimental Results. While quite a large amount of engineering work has been devoted to the computation of independent sets/vertex covers in static graphs, the amount of engineering work for the dynamic independent set problem is very limited. Zheng et al. [263] presented a heuristic fully dynamic algorithm and proposed a lazy search algorithm to improve the size of the maintained independent set. A year later, Zheng et al. [262] improved the result such that the algorithm is less sensitive to the quality of the initial solution used for the evolving MIS. In their algorithm, the authors used two well known data reduction rules, degree one and degree two vertex reduction, that are frequently used in the static case. Moreover, the authors can handle batch updates. Bhore et al. [51] focused on the special case of MIS for independent rectangles which is frequently used in map labelling applications. The authors presented a deterministic algorithm for maintaining a MIS of a dynamic set of uniform rectangles with amortized sub-logarithmic update time. Moreover, the authors evaluated their approach using extensive experiments.

3.9 Shortest Paths

One of the most studied problems on weighted dynamic networks is the maintenance of shortest path information between pairs of vertices, where edge weights represent distances and the length of a path is the sum of the weights of the edges on the path. In the case of *uniform edge weights* all weights are assumed to equal 1. Both the *single-source* as well as the in the *all-pairs shortest path problem* (*APSP*) we are interested in the shortest path between two arbitrary vertices s and t that are query parameters. For the *single-source shortest path problem* (*SSSP*), the source vertex s is fixed beforehand and the dynamic graph algorithm is only required to answer distance queries between s and an (arbitrary) vertex t which is specified by the query operation. In the *s-t shortest path problem* (*STSP*) both s and t are fixed beforehand and the data structure is only required to return the distance between s and t as answer to a query. There are two types of queries, namely a *distance query*, which returns the length of a shortest path and a *path reporting query*, which returns the shortest path itself.

The problem can be cast on both undirected or directed graphs: In case of the latter, we are asking for a shortest path from s to t rather than a shortest path between s and t. By replacing each edge in an undirected graph by a pair of directed, anti-parallel edges, any algorithm for the directed case can also be used in the undirected setting. In some settings, the range of possible edge weights is restricted to positive integers or to positive or non-negative real values. Note that the length of a shortest path is undefined in the presence of negative-length cycles.

Journey Planning. Algorithms for shortest path computations also play an important role in graph-based journey planning approaches. Here, the input is a timetable consisting of connections in, e.g., a public transportation system, which can then be modelled as a directed graph, and queries ask, e.g., for the earliest arrival time or the minimum number of vehicle changes if one wants to travel from a given station to another and depart not earlier than at a specified time. Delays result in updates to the directed graph, which can be edge insertions, edge deletions, and weight changes. Different graph models have been proposed, such as the time-expanded model, the time-dependent model, the reduced time-expanded model [203], and the dynamic timetable model [68]. In the time-dependent model, nodes represent stations and each edge represents a connection between stations, where the corresponding edge weight is time-dependent and encodes the timetable information. In the other

models, departure or arrival times are encoded within the nodes, and a single update to the timetable may result in an entire set of changes to the topology of the directed graph as well as edge weights.

3.9.1 Theory Results. Let $\delta > 0$ be any small constant. There is a conditional lower bound of $(\text{poly}(n, \log W), m^{1/2-\delta}, m^{1-\delta})$ based on the OMv conjecture, even for s-t shortest paths with uniform edge weights[126]. This lower bound applies also to any algorithm that gives a better than 5/3-approximation. For planar graphs the product of query and update time is $\Omega(n^{1-\delta})$ based on the APSP conjecture [2]. As even the partially dynamic versions have shown to be at least as hard as the static all-pairs shortest paths problem [2, 210], one cannot hope for a *combinatorial* fully dynamic all-pairs shortest paths algorithm with $O(n^{3-\delta})$ preprocessing time, $O(n^{2-\delta})$ amortized update time, and constant query time.

The state-of-the-art algorithms come close to this, even for the APSP problem.

For directed, weighted graphs, Demetrescu and Italiano [79] achieved an *amortized* update time of $O(n^2 \log^3 n)$, which was later improved to $O(n^2(\log n + \log^2((n+m)/n)))$ by Thorup [240]. Both of these algorithms actually allow vertex insertions and deletion, not just edge updates, but can only answer distance queries. In undirected graphs with uniform weight and with preprocessing time $O(n^2 \text{ poly}(\log n))$ a worst-case update time of $O(n^{1.9})$ can be achieved, however, with a $O(n^{1.529})$ time per distance query and $O(n^{1.9})$ per path reporting query[35].

There is also a fully dynamic $2^{O(k^2)}$-approximation algorithm that takes time $\tilde{O}(\sqrt{m}n^{1/k})$ per update and $O(k^2)$ per distance query for any positive integer k [5] and an $n^{o(1)}$-approximation with $n^{o(1)}$ update time [93]. These two algorithms only allow edge insertions and deletions. The second one can answer distance queries in constant time and path reporting queries in time linear in the number of edges on the approximate shortest path whose length it reports.

With respect to *worst-case* update times, the currently fastest algorithms are randomized with $\tilde{O}(n^{2+2/3})$ update time per vertex update [4, 115]. Moreover, Probst Gutenberg and Wulff-Nilsen [115] presented a deterministic algorithm with $\tilde{O}(n^{2+5/7})$ update time, thereby improving a 15 years old result by Thorup [241]. Van den Brand and Nanongkai [245] showed that Monte Carlo-randomized $(1+\epsilon)$-approximation algorithms exist with $\tilde{O}(n^{1.823}/\epsilon^2)$ worst-case update time for the fully dynamic single-source shortest path problem and $\tilde{O}(n^{2.045}/\epsilon^2)$ for all-pairs shortest paths, in each case with positive real edge weights and constant query time. However, both of these algorithm can only answer distance queries and not path-reporting queries.

Slightly faster exact and approximative algorithms exist in part for the "special cases" of uniform edge weights [4, 115, 210, 218, 245, 246] and/or *undirected* graphs [211, 245] (every edge has a reverse edge of the same weight). More details on shortest paths algorithms including fully dynamic algorithms are given in the survey of Madkour et al. [172].

3.9.2 Experimental Results for Single-Source Shortest Paths. The first experimental study for fully dynamic single-source shortest paths on *directed* graphs with positive real edge weights was conducted by Frigioni et al. [96], who evaluated Dijkstra's seminal static algorithm [83] against a fully dynamic algorithm by Ramalingam and Reps [206] (RR) as well as one by Frigioni et al. [97] (FMN). RR is based on Dijkstra's static algorithm and maintains a spanning subgraph consisting of edges that belong to at least one shortest s-t path for some vertex t. After an edge insertion, the spanning subgraph is updated starting from the edge's head until all affected vertices have been processed. In case of an edge deletion, the affected vertices are identified as a first step, followed by an update of their distances. The resulting worst-case update time is $O(x_\delta + n_\delta \log n_\delta) \subseteq O(m + n \log n)$, where n_δ corresponds to the number of vertices affected by the update, i.e., whose distance from s changes and x_δ equals n_δ plus the number of edges incident to an affected vertex. Similarly, Frigioni et al. [97] analyzed the update complexity of their algorithm FMN with respect to the change in the solution and showed a worst-case running time of $O(|U_\delta|\sqrt{m}\log n)$, where U_δ is the set of vertices where either the distance from s must be updated or their parent in the shortest paths tree. The algorithm assigns each edge (u, v) a forward (backward) level, which

corresponds to the difference between the (sum of) v's (u's) distance from s and the edge weight, as well as an owner, which is either u or v, and used to bound the running time. Incident outgoing and incoming edges of a vertex that it does not own are kept in a priority queue each, with the priority corresponding to the edge's level. In case of a distance update at a vertex, only those edges are scanned that are either owned by the vertex or have a priority that indicates a shorter path. Edge insertion and deletion routines are based on Dijkstra's algorithm and handled similar as in RR, but using level and ownership information. The experiments were run on three types of input instances: randomly generated ones, instances crafted specifically for the tested algorithms, and random updates on autonomous systems networks. The static Dijkstra algorithm is made dynamic in that it is re-run from scratch each time its shortest paths tree is affected by an update. The evaluation showed that the dynamic algorithms can speed up the update time by 95 % over the static algorithm. Furthermore, RR turned out to be faster in practice than FMN except on autonomous systems instances, where the spanning subgraph was large due to many alternative shortest paths.

In a follow-up work, Demetrescu et al. [76, 78] extended this study to dynamic graphs with arbitrary edge weight, allowing in particular also for negative weights. In addition to the above mentioned algorithm by Ramalingam and Reps [206] in a slightly lighter version (RRL) and the one by Frigioni et al. [97] (FMN), their study also includes a simplified variant of the latter which waives edge ownership (DFMN), as well as a rather straightforward dynamic algorithm (DF) that in case of a weight increase on an edge (u, v) first marks all vertices in the shortest paths subtree rooted at v and then finds for each of these vertices an alternative path from s using only unmarked vertices. The new weight of these vertices can be at most this distance or the old distance plus the amount of weight increase on (u, v). Therefore, the minimum is taken as a distance estimate for the second step, where the procedure is as in Dijkstra's algorithm. In case of a weight decrease on an edge (u, v) the first step is omitted. As Dijkstra's algorithm is employed as a subroutine, the worst-case running time of DF for a weight change is $O(m + n \log n)$. For updates, all algorithms use a technique introduced by Edmonds and Karp [86] to transform the weight $w(u, v)$ of each edge (u, v) to a non-negative one by replacing it with the reduced weight $w(u, v) - (d(v) - d(u))$, where $d(\cdot)$ denotes the distance from s. This preserves shortest paths and allows Dijkstra's algorithm to be used during the update process. The authors compared these dynamic algorithms to re-running the static algorithm by Bellman and Ford on each update from scratch on various randomly generated dynamic instances with mixed incremental and decremental updates on the edge weights, always avoiding negative-length cycles. Their study showed that DF is the fastest in practice on most instances, however, in certain circumstances RR and DFMN are faster, whereas FMN turned out to be too slow in practice due to its complicated data structures. The authors observed a runtime dependency on the interval size of the edge weights; RR was the fastest if this interval was small, except for very sparse graphs. DFMN on the other hand was shown to perform better than DF in presence of zero-length cycles, whereas RR is incapable of handling such instances. It is interesting to note here that the differences in running time are only due to the updates that increase distances, as all three candidates used the same routine for operations that decrease distances. The static algorithm was slower than the dynamic algorithms by several orders of magnitude.

Buriol et al. [58] presented a technique that reduces the number of elements that need to be processed in a heap after an update for various dynamic shortest paths algorithms by excluding vertices whose distance changes by exactly the same amount as the weight change and handling them separately. They showed how this improvement can be incorporated into RR [206], a variant similar to RRL [206], the algorithm by King and Thorup [159] (KT), and DF [76] and achieves speedups of up to 1.79 for random weight changes and up to 5.11 for unit weight changes. Narváez et al. [183] proposed a framework to dynamize static shortest path algorithms such as Dijkstra's or Bellman-Ford [34]. In a follow-up work [184], they developed a new algorithm that fits in this framework and is based on the linear programming formulation of shortest paths and its dual, which yields the problem in a so-called *ball-and-string model*. The authors experimentally showed that their algorithm needs

fewer comparisons per vertex when processing an update than the algorithms from their earlier work, as it can reuse intact substructures of the old shortest path tree.

Misra and Oommen [180] presented algorithms for single-source shortest paths that are based on learning automata and designed to find "statistical" shortest paths in a stochastic graph with stochastically changing edge weights. The algorithms are extensions of RR [206] and FMN [97] and shown to be superior to the original versions of RR and FMN by several orders of magnitude once they have converged. Chan and Yang [62] studied the problem of dynamically updating a single-source shortest path tree under multiple concurrent edge weight updates. They amended the algorithm by Narváez et al. [184] (MBS), for which they showed that it may misbehave in certain circumstances and suggested two further algorithms: MFP is an optimized version of an algorithm by Ramalingam and Reps [205] (DynamicSWSF-FP), which can handle multiple updates at once. The second algorithm is a generalization of the dynamic Dijkstra algorithm proposed by Narváez et al. [183]. In a detailed evaluation, they showed that an algorithm obtained by combining the incremental phase of MBS and the decremental phase of their dynamization of Dijkstra's algorithm performed best on road networks, whereas the dynamized Dijkstra's algorithm was best on random networks.

An extensive experimental study on single-source shortest path algorithms was conducted by Bauer and Wagner [30]. They suggested several tuned variants of DynamicSWSF-FP [205] and evaluated them against FMN [97], different algorithms from the framework by Narváez et al. [183], as well as RR [206] on a diverse set of instances. The algorithms from the Narváez framework showed similar performance in case of single-edge updates and were the fastest on road networks and generated grid-like graphs. By contrast, the tuned variants of DynamicSWSF-FP behaved less consistent. RR was superior on Internet networks, whereas FMN was the slowest, especially on sparse instances. Interestingly, the authors showed that for batch updates with a set of randomly chosen edges, the algorithms behave similar as for single-edge updates, as there was almost no interference. The picture changed slightly for simulated node failures and strongly for simulated traffic jams. RR and a tuned variant of DynamicSWSF-FP showed the best performance for simulated node failures, and two tuned variants of DynamicSWSF-FP dominated in case of simulated traffic jams. Notably, the algorithms from the Narváez framework were faster here if instead of in batches, the updates were processed one-by-one.

In follow-up works, D'Andrea et al. [71] evaluated several batch-dynamic algorithms for single-source shortest paths, where the batches are homogeneous, i.e., all updates are either incremental or decremental. Their study contains RR [206], a tuned variant of DynamicSWSF-FP [205] described by Bauer and Wagner [30] (TSWSF), as well as a new algorithm DDFLP, which is designed specifically to handle homogeneous batches and uses similar techniques as FMN [97]. The instance set comprised road and Internet networks as well as randomly generated graphs according to the Erdős-Rényi model (uniform degree distribution) and the Barabási-Albert model (power-law degree distribution). Batch updates were obtained from simulated node failure and recovery, simulated traffic jam and recovery, as well as randomly selected edges for which the weights were either increased or decreased randomly. The evaluation confirmed the results by Bauer and Wagner [30] and showed that DDFLP and TSWSF are best in case of update scenarios like node failures or traffic jams and otherwise TSWSF and RR, where RR is preferable to TSWSF if the interference among the updates is low and vice versa. DDFLP generally benefited from dense instances.

Singh and Khare [233] presented the first batch-dynamic parallel algorithm for single-source shortest paths for GPUs and showed in experiments that it outperforms the (sequential) tuned DynamicSWSF-FP algorithm [30] by a factor of up to 20 if the distances of up to 10 % of the nodes are affected.

3.9.3 Experimental Results for All-Pairs Shortest Paths. The first fully dynamic algorithm for all-pairs shortest paths in graphs with positive integer weights less than a constant W was presented by King [158] (Ki) and later also evaluated experimentally. It has an amortized update time of $O(n^{2.5}\sqrt{W \log n})$. For each vertex v, Ki maintains two shortest paths trees up to a distance d: one outbound with v as source and one inbound with v as

target. A so-called stitching algorithm is used to stitch together longer paths from shortest paths of distance at most d. To achieve the above mentioned running time, d is set to $\sqrt{nW \log n}$. The space requirement is $O(n^3)$ originally, but can be reduced to $\tilde{O}(n^2\sqrt{nW})$ [159].

For non-negative, real-valued edge weights, Demetrescu and Italiano [79] proposed the above-mentioned algorithm (DI) with an amortized update time of $\tilde{O}(n^2)$. The algorithm uses the concept of *locally shortest paths*, which are paths such that each proper subpath is a shortest path, but not necessarily the entire path, and *historical shortest paths*, which are paths that have once been shortest paths and whose edges have not received any weight updates since then. The combination of both yields so-called locally historical paths, which are maintained by the algorithm. To keep their number small, the original sequence of updates is transformed into an equivalent, but slightly longer *smoothed sequence*. In case of a weight update, the algorithm discards all maintained paths containing the updated edge and then computes new locally historical paths using a routine similar to Dijkstra's algorithm. Both Ki and DI have constant query time and were evaluated experimentally in a study by Demetrescu and Italiano [80] against RRL [78] on random instances, graphs with a single bottleneck edge, which serves as a bridge between two equally-sized complete bipartite graphs and only its weight is subject to updates, as well as real-world instances obtained from the US road networks and autonomous systems networks. Apart from RRL, the study also comprises the Dijkstra's static algorithm. Both these algorithms are designed for single-source shortest paths and were hence run once per vertex. All algorithms were implemented with small deviations from their respective theoretical description to speed them up in practice. The study showed that RRL and the algorithm based on locally historical paths (LHP) can outperform the static algorithm by a factor of up to 10 000, whereas the algorithm by King only achieved a speedup factor of around 10. RRL turned out to be especially fast if the solution changes only slightly, but by contrast exhibited the worst performance on the bottleneck instances unless the graphs were sparse. In comparison, LHP was slightly slower on sparse instances, but could beat RRL as the density increased. The authors also point out differences in performance that depend mainly on the memory architecture of the machines used for benchmarking, where RRL could better cope with small caches or memory bandwidth due to its reduced space requirements and better locality in the memory access pattern, whereas LHP benefited from larger caches and more bandwidth.

To speed up shortest paths computations experimentally, Wagner et al. [251] introduced a concept for pruning the search space by *geometric containers*. Here, each edge (u, v) is associated with a set of vertices called container, which is a superset of all vertices w whose shortest u-w path starts with (u, v). The authors assume that each vertex is mapped to a point in two-dimensional Euclidean space and based on this, suggest different types of geometric objects as containers, such as disks, ellipses, sectors or boxes. All types of container only require constant additional space per edge. The experimental evaluation on static instances obtained from road and railway networks showed that using the bounding box as container reduces the query time the most in comparison to running the Dijkstra algorithm without pruning, as the search space could be reduced to 5 % to 10 %. This could be preserved for dynamic instances obtained from railway networks if containers were grown and shrunk in response to an update, with a speedup factor of 2 to 3 over a recomputation of the containers from scratch. For bidirectional search, reverse containers need to be maintained additionally, which about doubled the absolute update time.

Delling and Wagner [75] adapted the static ALT algorithm [100] to the dynamic setting. ALT is a variant of bidirectional A^* search that uses a small subset of vertices called *landmarks*, for which distances from and to all other vertices are precomputed, and the triangle inequality to direct the search for a shortest path towards the target more efficiently. The authors distinguish between an eager and a lazy dynamic version of ALT, where the eager one updates all shortest path trees of the landmarks immediately after an update. The lazy variant instead keeps the preprocessed information as long as it still guarantees correctness, which holds as long as the weight of an edge is at least its initial weight, however at the expense of a potentially larger search space. The choice of

landmarks remains fixed. The experimental study on large road networks showed that queries in the lazy version are almost as fast as in the eager version for short distances or if no edges representing motorways are affected, but slower by several factors for longer distances, larger changes to the weight of motorway edges, or after many updates.

Schultes and Sanders [228] combined and generalized different techniques that have been successfully used in the static setting, such as separators, highway hierarchies, and transit node routing in a multi-level approach termed *highway-node routing*: For the set of vertices V_i on each level i, $V_i \subseteq V_{i-1}$, and the overlay graph G_i is defined on V_i with an edge $(s,t) \in V_i \times V_i$ iff there is a shortest s-t path in G_{i-1} that contains no vertices in V_i except for s and t. Queries are carried out by a modified Dijkstra search on this graph hierarchy. The authors extended this approach also to the dynamic setting and consider two scenarios: a server scenario, where in case of edge weight changes the sets of highway nodes V_i are kept and the graphs G_i are updated, and a mobile scenario, where only those vertices that are potentially affected are determined and the query routine needs to be aware of possibly outdated information during a search. In an experimental evaluation on a very large road network with dynamically changing travel times as weights it is shown that the dynamic highway-node routing outperformed recomputation from scratch as well as dynamic ALT search with 16 landmarks clearly with respect to preprocessing, update, and query time as well as space overhead.

For real-time shortest path computations on networks with fixed topology, but varying metric, Delling et al. [74] suggested a three-stage approach: In the first, preprocessing step, a metric-independent, moderate amount of auxiliary data is obtained from the network's topology. It is followed by a customization step, which is run for each metric and produces few additional data. Whereas the first phase is run only once and can therefore use more computation time, the second phase must complete within seconds in real-life scenarios. Shortest path queries form the third phase and must be fast enough for actual applications. For the first, metric-independent stage, the authors describe an approach based on graph partitioning, where the number of *boundary edges*, i.e., edges between different partitions, is to be minimized. For the second stage, they compute an overlay network consisting of shortest paths between all pairs of *boundary nodes*, i.e. nodes that are incident to at least one boundary edge. An s-t query is then answered by running a bidirectional Dijkstra algorithm on the graph obtained by combining the overlay graph with the subgraphs induced by the partitions containing s and t, respectively. The authors also considered various options for speedups, such as a sparsification of the overlay network, incorporating goal-directed search techniques, and multiple levels of overlays. An experimental evaluation on road networks with travel distances and travel times as metrics showed that their approach allows for real-time queries and needs only few seconds for the metric-dependent customization phase.

Arc flags belong in the category of goal-directed techniques to speed up shortest path computations and have been successfully used in the static setting [29]. To this end, the set of vertices is partitioned into a number of regions. Each edge receives a label consisting of a flag for each region, which tells whether there is a shortest path starting with this edge and ending in the region. The technique is related to geometric containers and uses the arc flags to prune a (bidirectional) Dijkstra search. Berettini et al. [41] were the first to consider arc flags in a dynamic setting, however only for the case of weight increases. Their main idea is to maintain a threshold for each edge and region that gives the increase in weight required for the edge to lie on a shortest path. On a weight increase, the thresholds are updated and used to determine when to change an arc flag. Although this potentially reduces the quality of the arc flags with each update, the experimental evaluation showed that the increase in query time is very small as long as the update sequence is short. With respect to the update time, a significant speedup could be achieved over recomputing arc flags from scratch.

To refresh arc flags more exactly and in a fully dynamical setting, D'Angelo et al. [72] introduced a data structure called *road signs*. Road signs complement arc flags and store for each edge e and region R the set of boundary nodes contained in any shortest path starting with e and ending in R. In case of a weight increase, the algorithm first identifies all affected nodes whose shortest path to a boundary node changed and then updates all

road signs for all outgoing edges of an affected node. In case of a weight decrease on edge (u, v), the authors observed that all shortest paths containing (u, v) remain unchanged. However, shortest paths starting with other outgoing or incoming edges of u might require updates, as well as other paths containing an incoming edge of u. In an experimental study on road networks, the authors compared their algorithm against one that recomputes arc flags from scratch as well as the algorithm by Berettini et al. [41] (BDD). To mimic traffic jams and similar occurrences, the weight of a randomly chosen edge increases and then decreases by the same amount, however not necessarily in subsequent updates. The evaluation showed that updating both road signs and arc flags is by several factors faster than recomputing arc flags from scratch. On instances with weight increases only, the authors showed that their new algorithm outperforms BDD distinctly both for updates and queries.

A further speedup technique for shortest path queries are *2-hop cover labelings*, where the label $L(v)$ of each node v is a carefully chosen set of nodes U_v along with the distance between v and u for each $u \in U$. For each pair of vertices s and t, the shortest s-t path can be obtained by intersecting U_s and U_t and taking the minimum over all combinations of s-x and x-t paths for all nodes $x \in U_s \cap U_t$. In the static setting, a 2-hop cover labeling can be computed based on a breadth-first search that is run once for every vertex ("naive landmark labeling"). Akiba et al. [9] introduced *pruned landmark labeling* (PLL), which constitutes a more refined approach and uses pruned breadth-first searches instead. The authors developed an incremental algorithm for PLL, which was complemented by D'Angelo et al. [73] to a fully dynamic algorithm. The experimental evaluation showed that the algorithm achieves speedups of several orders of magnitude over a recomputation from scratch, while at the same time preserving the quality of the labeling, which makes this speedup technique suitable for practical use in dynamic scenarios.

Hayashi et al. [120] proposed a method to support shortest paths queries on unweighted networks with billions of edges by combining a bidirectional breadth-first search, which is optimized for the structure of small-world networks, with landmarks. To this end, the authors choose high-degree vertices and store shortest path trees as well as those of a subset of their neighbors in a so-called "bit-parallel" form. This increases the number of landmarks, which in turn generally speeds up the search and in particular for high-degree vertices, and at the same time keeps the memory requirements comparatively small. After an edge insertion or deletion, the bit-parallel shortest paths trees are updated accordingly. The experimental evaluation on twelve real-world instances having between 1.5 million and 3.7 billion edges showed that the new algorithm was able to process queries on average in less than 8 ms and even considerably less on many instances. The average edge insertion and deletion times were less than 1.3 ms and 8.1 s, respectively, after an initialization time of less than 1 h. The incremental algorithm by Akiba et al. [9], which was included in the study, was distinctly faster on queries, but on some instances several factors slower on insertions. However, it failed to complete the preprocessing step within 10 h or required more than 128 GB of memory on half of all instances.

3.9.4 Experimental Results for Journey Planning. Cionini et al. [68] engineered different update and query algorithms for the reduced time-expanded and the dynamic timetable model and evaluated them on public transportation timetables of different sizes. They showed that update times are negligible in practice and that their new heuristic query algorithms combined with ALT outperform other approaches and is at least competitive with array-based models. For the reduced time-expanded model, the algorithm outperforms one of the fastest array-based algorithms, however at the expense of a larger memory footprint in comparison to the dynamic timetable model. The query time is in the order of milliseconds even for large timetables with several millions of connections. Giannakopoulou et al. [99] extended the dynamic timetable model to the multi-modal journey planning problem, which can deal with more than one transport mode. In an experimental study, they showed that their algorithms answer different kinds of queries effectively and are competitive to state-of-the-art approaches.

3.10 Maximum Flows and Minimum Cuts

An instance of the maximum flow/minimum cut problem consists of an edge-weighted directed graph $G = (V, E, w)$ along with two distinguished vertices s and t. The edge weights w are positive and commonly referred to as *capacities*. An (s-t) *flow* f is a non-negative weight function on the edges such that $f(e) \leq w(e)$ for all $e \in E$ (capacity constraints) and except for s and t, the total flow on the incoming edges of each vertex must equal the total flow on the outgoing edges (conservation constraints). The *excess* of a vertex v is the total flow on its incoming edges minus that on its outgoing edges, which must be zero for all vertices except s and t. The value of a flow f then is the excess of t. The task is to find a flow of maximum value.

An (s-t) *cut* is a partition (S, T) of the set of vertices where $s \in S$ and $t \in T$. Its value is the sum of the capacities of all edges (u, v) such that $u \in S$ and $v \in T$. The well-known max-flow min-cut theorem states that the maximum value of a flow equals the minimum value of a cut.

The *global minimum cut* problem for an undirected edge-weighted graph asks us to divide its set of nodes into two blocks while minimizing the weight sum of the cut edges.

Theory Results. In the dynamic setting, there is a conditional lower bound of $(\text{poly}(n, \log W), m^{1/2-\delta}, m^{1-\delta})$ (for any small constant $\delta > 0$) for the value of the maximum s-t flow even in unweighted (i.e., $w \equiv 1$), undirected graphs based on the OMv conjecture [126]. The fastest static algorithm whose running time does not depend on the size of the largest edge weight computes an optimal solution in $O(nm)$ time [192]. Recently, Chen et al. [67] gave an $O(\log n \log\log n)$-approximate fully dynamic algorithm that maintains the maximum flow value in time $\tilde{O}(n^{2/3+o(1)})$ per update and Goranci et al. [105] gave a $n^{o(1)}$-approximate fully dynamic algorithm to maintain the maximum flow value in time $n^{o(1)}$ worst-case update time and $O(\log^{1/6} n)$ query time. In the unweighted setting, Jin and Sun [150] gave a data structure that can be constructed for any fixed positive integer $c = (\log n)^{o(1)}$ and that answers for any pair (s, t) of vertices that are parameters of the query in time $n^{o(1)}$ whether s and t are c-edge connected.

For *global minimum cuts* in the unweighted setting Thorup and Karger [243] presented a $\sqrt{2 + o(1)}$-approximation algorithm to maintain a solution that takes polylogarithmic time per update and query and Thorup [242] designed a $(1 + \epsilon)$-approximate algorithm to maintain a minimum cut in $\tilde{O}(\sqrt{n})$ time per update and query.

Experimental Results for Maximum Flow/Minimum Cut. Kumar and Gupta [165] extended the preflow-push approach [102] to solve maximum flow in static graphs to the dynamic setting. A preflow is a flow under a relaxed conservation constraint in that the excess of all vertices except s must be non-negative. Vertices with positive excess are called *active*. Preflow-push algorithms, also called push-relabel algorithms, use this relaxed variant of a flow during the construction of a maximum flow along with distance labels on the vertices. Generally speaking, they push flow out of active vertices towards vertices with smaller distance (to t) and terminate with a valid flow (i.e., observing conservation constraints). In case of an edge insertion or deletion, Kumar and Gupta first identify affected vertices via forward and backward breadth-first search while observing and updating distance labels and then follow the scheme of a basic preflow-push algorithm, however restricted to the set of affected vertices. The authors evaluated their algorithm only for the incremental setting on a set of randomly generated instances against the static preflow-push algorithm in [102] and found that their algorithm is able to reduce the number of push and relabel operations significantly as long as the instances are sparse and the number of affected vertices remains small.

Many important fields of application for the maximum flow/minimum cut problem stem from computer vision. In this area, the static algorithm of Boykov and Kolmogorov [53] (BK) is widely used due to its good performance in practice on computer vision instances and despite its pseudo-polynomial worst-case running time of $O(nm \cdot \text{Opt})$, with Opt being the value of a maximum flow/minimum cut. Interestingly, however, a study

by Verma and Batra [250] shows that its practical superiority only holds for sparse instances. BK follows the Ford-Fulkerson method of augmenting flow along s-t paths, but uses two search trees grown from s and t, respectively, to find such paths. Kohli and Torr [160, 161] extended BK to the fully dynamic setting by updating capacities and flow upon changes and discuss an optimization that tries to recycle the search trees. They experimentally compared their algorithm to repeated executions of the static algorithm on dynamic instances obtained from video sequences and achieve a substantial speedup. They also observed that reusing the search trees leads to longer s-t paths, which affects the update time negatively as the instances undergo many changes.

Goldberg et al. [101] developed EIBFS, a generalization of their earlier algorithm IBFS, that by contrast also extends to the dynamic setting in a straightforward manner. IBFS in turn is a modification of BK that ensures that the two trees grown from s and t are height-minimal (i.e., BFS trees) and is closely related to the concept of blocking flows. The running time of EIBFS in the static setting and thus the initialization in the dynamic setting, is $O(mn \log(n^2/m))$ with dynamic trees or $O(mn^2)$ without. The algorithm works with a so-called pseudoflow, which observes capacity constraints, but may violate conservation constraints. It maintains two vertex-disjoint forests S and T, where the roots are exactly those vertices with a surplus of incoming flow and those with a surplus of outgoing flow, respectively, and originally only contain s and t. The steps of the algorithm consist in growth steps, where S or T are grown level-wise, augmentation steps, which occur if a link between the forests has been established and flow is pushed to a vertex in the other forest and further on to the root, and adoption steps, where vertices in T with surplus incoming flow or vertices in S with surplus outgoing flow are either adopted by a new parent in the same forest or become a root in the other forest. In case of an update in the dynamic setting, the invariants of the forests are restored and flow is pushed where possible, followed by alternating augmentation and adoption steps if necessary. The authors also mention that resetting the forests every $O(m)$ work such that they contain only vertices with a surplus outgoing or incoming flow seemed to be beneficial in practice. In their experimental evaluation of EIBFS against the algorithm by Kohli and Torr as well as an altered version thereof and a more naive dynamization of IBFS, they showed for different dynamic real-world instances from the field of computer vision that EIBFS is the fastest on eight out of fourteen instances and relatively robust: In contrast to its competitors, it always takes at most roughly twice the time of the fastest algorithm on an instance. Notably, no algorithm is dominated by another one across all instances.

Zhu et al. [264] described a dynamic update strategy based on augmenting and de-augmenting paths as well as cancelling cyclic flows. The latter serves as a preparatory step and only reroutes flow in the network without increasing or decreasing the total s-t flow and is only necessary in a decremental update operation. They experimentally evaluated the effectiveness of their algorithm for online semi-supervised learning, where real-world big data is classified via minimum cuts, and showed that their algorithm outperforms state-of-the-art stream classification algorithms. A very similar algorithm was proposed by Greco et al. [111]. The authors compared it experimentally against EIBFS and the dynamic BK algorithm by Kohli and Torr as well as a number of the currently fastest static algorithms. Their experiments were conducted on a set of instances from computer vision where equally many edges are randomly chosen to be inserted and deleted, respectively. They showed that their algorithm is with one exception always the fastest on average in performing edge insertions if compared to the average update time of the competitors, and on half of all instances also in case of edge deletions. On the remaining instances, the average update time of EIBFS dominated.

Experimental Results for Global Minimum Cuts. For the global minimum cut problem, Henzinger et al. [128] implemented an algorithm for large dynamic graphs under both edge insertions and deletions. For edge insertions, the algorithm uses the approach of Henzinger [130] and Goranci et al. [104], which maintain a compact data structure of all minimum cuts in a graph and invalidate only the minimum cuts that are affected by an edge insertion. For edge deletions, the algorithms use the push-relabel algorithm of Goldberg and Tarjan [103] to

certify whether the previous minimum cut is still a minimum cut. The algorithm outperformed static approaches by up to five orders of magnitude on large graphs.

3.11 Graph Clustering

Graph clustering is the problem of detecting tightly connected regions of a graph. More precisely, a clustering C is a partition of the set of vertices, i.e., a set of disjoint *clusters* of vertices $V_1,\ldots,V_k$ such that $V_1 \cup \cdots \cup V_k = V$. However, k is usually not given in advance and some objective function that models intra-cluster density versus inter-cluster sparsity, is optimized. It is common knowledge that there is neither a single best strategy nor objective function for graph clustering, which justifies a plethora of existing approaches. Moreover, most quality indices for graph clusterings have turned out to be NP-hard to optimize and are rather resilient to effective approximations, see [22, 54], e.g.,, allowing only heuristic approaches for optimization. There has been a wide-range of algorithms for static graph clustering, the majority are based on the paradigm of intra-cluster density versus inter-cluster sparsity. For dynamic graphs, there has been a recent survey on the topic of community detection [214]. The survey covers features and challenges of dynamic community detection and classifies published approaches. Here we focus on engineering results and extend their survey in that regard with additional references as well as results that appeared in the meantime. A large amount of algorithms in the area optimize for modularity which has been proposed by [189]. The core idea for modularity is to take coverage, i.e., the fraction of edges covered by clusters, minus the expected value of the same quantity in a network with the same community divisions, but random connections between the vertices. The commonly used formula is as follows: $\text{mod}(C) := \frac{m(C)}{m} - \frac{1}{4m^2} \sum_{C \in C} \left(\sum_{v \in C} \deg(v)\right)^2$.

Experimental Results. Miller and Eliassi-Rad [177] adapted a dynamic extension of Latent Dirichlet Allocation for dynamic graph clustering. Latent Dirichlet Allocation has been originally proposed for modeling text documents, i.e., the algorithm assumes that a given set of documents can be classified into k topics. This approach has been transferred to graphs [122] and was adapted by the authors for dynamic networks. Aynaud and Guillaume [24] tracked communities between successive snapshots of the input network. They first noted that using standard community detection algorithms results in stability issues, i.e., little modifications of the network can result in wildly different clusterings. Hence, the authors propose a modification of the Louvain method to obtain stable clusterings. This is done by modifying the initialization routine of the Louvain method. By default, the Louvain method starts with each node being in its own clustering. In the modified version of Aynaud and Guillaume, the algorithm keeps the clustering of the previous time step and uses this as a starting point for the Louvain method which results in much more stable clusterings. Bansal et al. [25] also reused the communities from previous time steps. However, their approach is based on greedy agglomeration where two communities are merged at each step to optimize the modularity objective function. The authors improved the efficiency of dynamic graph clustering algorithms by limiting recomputation to regions of the network and merging processes that have been affected by insertion and deletion operations. Görke et al. [106] showed that the structure of minimum s-t-cuts in a graph allows for efficient updates of clusterings. The algorithm builds on partially updating a specific part of a minimum-cut tree and is able to maintain a clustering fulfilling a provable quality guarantee, i.e., the clusterings computed by the algorithm are guaranteed to yield a certain expansion. To the best of our knowledge, this is the only dynamic graph clustering algorithm that provides such a guarantee. Later, Görke et al. [108, 109] formally introduced the concept of smoothness to compare consecutive clusterings and provided a portfolio of different update strategies for different types of local and global algorithms. Moreover, their fastest algorithm is guaranteed to run in time $\Theta(\log n)$ per update. Their experimental evaluation indicates that dynamically maintaining a clustering of a dynamic random network saves time and at the same time also yields higher modularity than recomputation from scratch. Alvari et al. [13] proposed a dynamic game theory method to tackle the community detection problem in dynamic social networks. Roughly speaking, the authors

model the process of community detection as an iterative game performed in a dynamic multiagent environment where each node is an agent who wants to maximize its total utility. In each iteration, an agent can decide to join, switch, leave, or stay in a community. The new utility is then computed by the best outcome of these operations. The authors use neighborhood similarity to measure structural similarity and optimize for modularity. The experimental evaluation is limited to two graphs. Zakrzweska and Bader [259] presented two algorithms that update communities. Their first algorithm is similar to the dynamic greedy agglomeration algorithm by Görke et al. [108]. The second algorithm is a modification of the first one that runs faster. This first is achieved by more stringent backtracking of merges than Görke et al. [108], i.e., merges are only undone if the merge has significantly changed the modularity score of the clustering. Moreover, the authors used a faster agglomeration scheme during update operations that uses information about previous merges to speed up contractions. Recently, Zhuang et al. [265] proposed the DynaMo algorithm which also is a dynamic algorithm for modularity maximization, however the algorithm processes network changes in batches.

3.12 Centralities

We will describe three popular measures to find central nodes in networks in the fully dynamic setting: Katz centrality, betweenness centrality and closeness centrality.

Katz centrality is a centrality metric that measure the relation between vertices by counting weighted walks between them. Formally, the Katz centrality of a vertex i is given as $\mathbf{e}_i^T \sum_{k=1}^{\infty} \alpha^{k-1} A^k \mathbf{1}$ where A is the adjacency matrix of the graph under consideration, A^k is the kth potence of the matrix A (using matrix multiplication), $\mathbf{e}_i$ is the ith canonical basis vector, $\mathbf{e}_i^T$ is the transposed version of $\mathbf{e}_i$, α is an input parameter, and $\mathbf{1}$ is the vector that contains one at each component.

Given a graph and a vertex v in the graph, the *betweenness centrality* measure is defined to be $c(v) = \sum_{u,w,u\neq w} \frac{\sigma_{u,w}(v)}{\sigma_{u,w}}$, where $\sigma_{u,w}$ is the number of shortest paths between u and w and $\sigma_{u,w}(v)$ is the number of shortest paths between u and w that include v. There is also the (less often used) notion of edge betweeness $\tilde{c}(e) = \sum_{u,w,u\neq w} \frac{\tilde{\sigma}_{u,w}(e)}{\sigma_{u,w}}$ where $\tilde{\sigma}_{u,w}(e)$ denotes the number of shortest paths from u to w in G that go though e. In both cases, a shortest path is a path with minimum amount of edges.

Given a graph and a vertex v, the *harmonic closeness centrality measure* is defined as $\text{clo}(v) = \sum_{u\in V, u\neq v} \frac{1}{d(u,v)}$ where $d(u,v)$ is the distance between u and v. Here, distance $d(u,v)$ refers to the number of edges of a shortest path between u and v. Roughly speaking, it is the sum of the reciprocal length of the shortest path between the node and all other nodes in the graph. Bavela's definition of *closeness centrality* is similarly $\frac{|V|-1}{\sum_{u\in V} d(v,u)}$.

Theory Results. The only two theoretical fully dynamic results that we are aware of are due to Pontecorvi and Ramachandran [201], who achieve amortized $O(\nu^{*2} \cdot \log^2 n)$ update time for betweenness centrality in directed networks where ν^* bounds the number of distinct edges that lie on shortest paths through any single vertex, and a result due to van den Brand and Nanongkai [245], who present a $(1+\epsilon)$-approximate fully-dynamic algorithm for closeness centrality with $O(n^{1.823})$ update time in undirected networks. This is an obvious area for future work.

Experimental Results on Katz Centrality. Nathan and Bader [187] were the first to look at the problem in a dynamic setting. At that time, static algorithms mostly used linear algebra-based techniques to compute Katz scores. The dynamic version of their algorithm for undirected graphs incrementally updates the scores by exploiting properties of iterative solvers, i.e. Jacobi iterations. Their algorithm achieved speedups of over two orders of magnitude over the simple algorithms that perform static recomputation every time the graph changes. Later, they improved their algorithm [186] to handle updates by using an alternative, agglomerative method of calculating Katz scores. While their static algorithm is already several orders of magnitude faster than typical

linear algebra approaches, their dynamic algorithm is also faster than pure static recomputation every time the graph changes. A drawback of the algorithms by Nathan and Bader is that they are unable to reproduce the exact Katz ranking after dynamic updates. Van der Grinten et al. [247] fixed this problem by presenting a dynamic algorithm that iteratively improves upper and lower bounds on the centrality scores. The computed scores are approximate, but the bounds guarantee the correct ranking. The dynamic algorithm improves over the static recomputation of the Katz rankings as long as the size of the batches in the update sequence is smaller than 10 000. Moreover, the authors are able to deal with undirected and with directed dynamic networks.

Experimental Results on Betweenness Centrality. Statically computing betweenness centrality involves solving the all-pairs shortest path problem. Dynamically maintaining betweenness centrality is challenging as the insertion or deletion of a single edge can lead to changes of many shortest paths in the graph. The QUBE algorithm [167] was the first to provide a non-trivial update routine for undirected graphs. The key idea is to perform the betweenness computation on a reduced set of vertices, i.e., the algorithm first finds vertices whose centrality index might have changed. Betweenness centrality is then only computed on the first set of vertices. However, QUBE is limited to the insertion and deletion of non-bridge edges. Lee et al. [166] extended the QUBE algorithm [167] to be able to insert and delete non-bridge edges. Moreover, the authors reduced the number of shortest paths that need to be recomputed and thus gained additional speedups over QUBE. Kourtellis et al. [162, 163] contributed an algorithm that maintains both vertex and edge betweenness centrality (for directed as well as undirected networks). Their algorithm needs less space than the algorithm by Green et al. [112] as it avoids storing predecessor lists. Their method can be parallelized and runs on top of parallel data processing engines such as Hadoop. Bergamini et al. [38] presented an incremental *approximation* algorithm for the problem which is based on the first theory result that is asymptotically faster than recomputing everything from scratch due to Nasre et al. [185]. The algorithm works for directed as well as for undirected networks. As a building block of their algorithm, the authors used an asymptotically faster algorithm for the dynamic single-source shortest path problem and additionally sample shortest paths. Experiments indicate that the algorithm can be up to four orders of magnitude faster compared to restarting the static approximation algorithm by Riondato and Kornaropoulos [208]. In the same year, the authors extended their algorithm to become a fully dynamic approximation algorithm for the problem [36, 37]. In addition to dynamic single-source shortest paths, the authors also employed an approximation of the vertex diameter that is needed to compute the number of shortest paths that need to be sampled as a function of a given error guarantee that should be achieved. Hayashi et al. [121] provided a fully dynamic approximation algorithm for directed networks that is also based on sampling. In contrast to Bergamini et al. [36–38], which samples shortest paths between randomly selected vertices, the authors save all the paths between each sampled pair of vertices. Moreover, the shortest paths are represented in a data structure called hypergraph sketch. To further reduce the running time when handling unreachable pairs, the authors maintain a reachability index. Gil-Pons [200] focused on exact betweenness in incremental directed graphs. The author presented a space-efficient algorithm with linear space complexity. Lastly, Chehreghani et al. [66] focused on the special case in which the betweenness of a single node has to be maintained under updates (vertex/edge insertion/deletion) in directed graphs.

Experimental Results on Closeness Centrality. Kas et al. [155] were the first to give a *fully dynamic* algorithm for the problem (directed/undirected). As computing closeness centrality depends on the all-pairs shortest path problem, the authors extended an existing dynamic all-pairs shortest path algorithm [204] for their problem. As the algorithm stores pairwise distances between nodes it has quadratic memory requirement. Sariyuce et al. [221] provided an algorithm that can handle insertions and deletions in undirected graphs. In contrast to Kas et al. [155], the authors used static single-source shortest paths from each vertex. The algorithm does not need to store pairwise distances and hence requires only a linear amount of memory. Moreover, the authors observed that in scale-free networks the diameter grows proportional to the logarithm of the number of nodes, i.e., the diameter

is typically small. When the graph is modified with minor updates, the diameter also tends to stay small. This can be used to limit the number of vertices that need to be updated. In particular, the authors showed that recomputation of closeness can be skipped for vertices s such that $|d(s,u) - d(s,v)| = 1$ where u, v are the endpoints of the newly inserted edge. Lastly, the authors used data reduction rules to filter vertices, i.e., real-life networks can contain nodes that have the same or similar neighborhood structure that can be merged. Later, Sariyuce et al. [222, 223] proposed a distributed memory-parallel algorithm for the problem in undirected dynamic graphs. Yen et al. [258] proposed the fully dynamic algorithm CENDY which can reduce the number of internal updates to a few single-source shortest path computations necessary by using breadth-first searches. The authors introduce the notion of augmented rooted BFS trees, which are rooted BFS trees plus two types of edges that indicate if two vertices are on the same level of the BFS tree or if they are on different levels of the BFS tree. The main idea is that given an augmented rooted BFS tree of an unweighted network, edges that are inserted or deleted within the same level of the tree do not change the distances from the root to all other vertices. Putman et al. [202] provided a faster algorithm for fully dynamic harmonic closeness in directed networks. The authors also used a filtering method to heavily reduce the number of computations for each incremental update. The filtering method is an extension of level-based filtering to directed and weighted networks. The dynamic algorithm by Shao et al. [231] maintains closeness centrality by efficiently detecting all affected shortest paths based on articulation points. The main observation is that a graph can be divided into a series of biconnected components which are connected by articulation points – the distances between two arbitrary vertices in the graph can be expressed as multiple distances between different biconnected components.

Bisenius et al. [52] contributed an algorithm to maintain top-k harmonic closeness in fully dynamic graphs for undirected and directed networks. The algorithm is not required to compute closeness centrality for the initial graph and the memory footprint of their algorithm is linear. Their algorithm also tries to skip recomputations of vertices that are unaffected by the modifications of the graph by running breadth-first searches. Crescenzi et al. [69] gave a fully dynamic *approximation* algorithm for top-k harmonic closeness for undirected as well as directed neworks. The algorithm is based on sampling paths and a backward dynamic breadth-first search algorithm.

3.13 Graph Partitioning

Typically the graph partitioning problem asks for a partition of a graph into k blocks of about equal size such that there are few edges between them. More formally, we are given a graph $G = (V, E)$ with vertex weights $c : V \to \mathbb{R}_{>0}$ and edge weights $\omega : E \to \mathbb{R}_{>0}$. We extend c and ω to sets in the natural way, i.e., $c(U) := \sum_{v \in U} c(v)$ and $\omega(F) := \sum_{e \in F} \omega(e)$. The graph partitioning problems is looking for disjoint *blocks* of vertices $V_1,\ldots,V_k$ that partition V, i.e., $V_1 \cup \cdots \cup V_k = V$. A *balancing constraint* demands that all blocks have weight $c(V_i) \leq (1+\epsilon)\lceil \frac{c(V)}{k} \rceil$ for some imbalance parameter ϵ. The most used objective is to minimize the total *cut* which is defined as $\omega(E \cap \bigcup_{i<j} V_i \times V_j)$. The problem is known to be NP-hard and no constant-factor approximation algorithms exist. Thus heuristic algorithms are mostly used in practice. Dynamic graph partitioning algorithms are also known in the community as *repartitioning algorithms*. As the problem is typically not solved to optimality in practice, repartitioning involves a tradeoff between the *quality*, i.e., the number of edges in different sets of the partitioning, and the amount of vertices that need to change their block as they cause communication when physically moved between the processors as the partition is adopted. The latter is especially important when graph partitioning is used in adaptive numerical simulations. In these simulations, the main goal is to partition a model of computation and communication in which nodes model computation and edges model communication. The blocks of the partition are then fixed to a specific processing element. When the dynamic graph partitioning algorithm decides to change the blocks due to changes in the graph topology, nodes that are moved to a different block create communication in the simulation system as the underlying data needs to be moved between the corresponding processors.

Experimental Results. Hendrikson et al. [123] tackled the repartitioning problem by introducing k virtual vertices. Each of the virtual vertices is connected to all nodes of its corresponding block. The edges get a weight α which is proportional to the migration cost of a vertex and the vertex weights of the virtual vertices are set to zero. Then an updated partition can be computed using a static partitioning algorithm since the model accounts for migration costs and edge cut size at the same time.

Schloegel et al. [225] presented heuristics to control the tradeoff between edge-cut size and vertex migration costs. The most simple algorithm is to compute a completely new partition and then determine a mapping between the blocks of the old and the new partition that minimizes migration. The more sophisticated algorithm of [225] is a multilevel algorithm based on a simple process, i.e., nodes are moved from blocks that contain too many vertices to blocks that contain not enough vertices. However, this approach often yields partitions that cut a large number of edges. The result has been improved later by combining the two approaches in the parallel partitioning tool ParMetis [226]. Schloegel et al. [227] later extended their algorithm to be able to handle multiple balance constraints. Hu and Blake [140] noted that diffusion processes can suffer from slow convergence and improved the performance of diffusion through the use of Chebyshev polynomials. More precisely, the diffusion process in their paper is a directed diffusion that computes a diffusion solution by solving a so-called head conduction equation while minimizing the data movement. Walshaw et al. [252] integrated a repartitioning algorithm into their parallel (meanwhile uncontinued) tool Jostle. The algorithm is a directed diffusion process based on the solver proposed by Hu and Blake [140]. Rotaru and Nägeli [215] extended previous diffusion-based algorithms to be able to handle heterogeneous systems. These approaches, however, have certain weaknesses: For example, in numerical applications the maximum number of boundary nodes of a block is often a better estimate of the occurring communication in the simulation than the number of cut edges. Meyerhenke and Gehweiler [175, 176] explored a disturbed diffusion process that is able to overcome some of the issues of the previous approaches. To do so, Meyerhenke adapted DIBAP, a previously developed algorithm that aims at computing well-shaped partitions. A diffusion process is called disturbed if its convergence state does not result in a balanced distribution. These processes can be helpful to find densely connected regions in the graph.

There has been also work that tackles slightly different problem formulations. Kiefer et al. [157] noted that performance in applications usually does not scale linearly with the amount of work per block due to contention on different compute components. Their algorithm uses a simplified penalized resource consumption model. Roughly speaking, the authors introduced a penalized block weight and modified the graph partitioning problem accordingly. More precisely, a positive, monotonically increasing penalty function p is used to penalize the weight of a block based on the partition cardinality. Vaquero et al. [249] looked at the problem for distributed graph processing systems. Their approach is based on iterative vertex migration based on label propagation. More precisely, a vertex has a list of candidate blocks where the highest number of its neighbors are located. However, initial partitions are computed using hashing which does not yield high quality partitions since it completely ignores the structure of the graph. The authors did not compare their work against other state-of-the art repartitioning algorithms, so it is unclear how well the algorithm performs compared to other algorithms. Xu et al. [256] and Nicoara et al. [190] also presented dynamic algorithms specifically designed for graph processing systems. Other approaches have focused on the edge partitioning problem [91, 142, 216] or the special case of road networks [55].

4 DYNAMIC GRAPH SYSTEMS

The methodology of the previous two sections is to engineer algorithms for specific dynamic graph problems. In contrast to this, there are also approaches that try to engineer dynamic graph systems that can be applied to a wide range of dynamic graph problems. Alberts et al. [11] started this effort and presented a software library of dynamic graph algorithms. The library is written in C++ and is an extension of the well known LEDA library of

efficient data types and algorithms. The library contains algorithms for connectivity, spanning trees, single-source shortest paths and transitive closure.
A decade later Weigert et al. [254] presented a system that is able to deal with dynamic distributed graphs, i.e., in settings in which a graph is too large for the memory of a single machine and, thus, needs to be distributed over multiple machines. A user can implement a query function to implement graph queries. Based on their experiments, the system appears to scale well to large distributed graphs. Ediger et al. [85] engineered STINGER which is an acronym for Spatio-Temporal Interaction Networks and Graphs Extensible Representation. STINGER provides fast insertions, deletions, and updates on semantic graphs that have a skewed degree distribution. The authors showed in their experiments that the system can handle 3 million updates per second on a scale-free graph with 537 million edges on a Cray XMT machine. The authors already implemented a variety of algorithms on STINGER including community detection, k-core extraction, and many more. Later, Feng et al. [92] presented DISTINGER which has the same goals as STINGER, but focuses on the distributed memory case, i.e. the authors presented a distributed graph representation. Vaquero et al. [248] presented a dynamic graph processing system that uses adaptive partitioning to update the graph distribution over the processors over time. This speeds up queries as a better graph distribution significantly reduces communication overhead. Experiments showed that the repartitioning heuristic (also explained in Section 3.13) improves computation performance in their system up to 50 % for an algorithm that computes the estimated diameter in a graph. Sengupta et al. [229] introduced a dynamic graph analytics framework called GraphIn. Part of GraphIn is a new programming model based on the gather-apply-scatter programming paradigm that allows users to implement a wide range of graph algorithms that run in parallel. Compared to STINGER, the authors reported a 6.6-fold speedup. Iwabuchi et al. [147] presented an even larger speedup over STINGER. Their dynamic graph data store is, like STINGER, designed for scale-free networks. The system uses compact hash tables with high data locality. In their experiments, their system called DegAwareRHH, is a factor 206.5 faster than STINGER.

Another line of research focuses on graph analytic frameworks and data structures for GPUs. Green and Bader [92] presented cuSTINGER, which is a GPU extension of STINGER and targets NVIDIA's CUDA supported GPUs. One drawback of cuSTINGER is that the system has to perform restarts after a large number of edge updates. Busato et al. [59] fixed this issue in their system, called Hornet, and, thus, outperform cuSTINGER. Moreover, Hornet uses a factor of 5 to 10 less memory than cuSTINGER. In contrast to previous approaches, faimGraph due to Winter et al. [255] is able to deal with a changing number of vertices. Awad et al. [23] noted that the experiments performed by Busato et al. are missing true dynamism that is expected in real world scenarios and proposed a dynamic graph structure that uses one hash table per vertex to store adjacency lists. The system achieves speedups between 5.8 to 14.8 compared to Hornet and 3.4 to 5.4 compared to faimGraph for batched edge insertions (and slightly smaller speedups for batched edge deletions). The algorithm also supports vertex deletions, as does faimGraph.

5 METHODOLOGY

Currently there is a limited amount of real-world *fully* dynamic networks publicly available. There are repositories that feature a lot of real-world insertions only instances such as SNAP[3] and KONECT[4]. However, since the fully dynamic instances are rarely available at the moment, we start a new graph repository that provides fully dynamic graph instances[5]. Currently, there is also very limited work on dynamic graph generators. A generator for clustered dynamic random networks has been proposed by Görke et al. [107]. Another approach is due to Sengupta [230] to generate networks for dynamic overlapping communities in networks. A generative model

[3]https://snap.stanford.edu/
[4]http://konect.cc/
[5]https://DynGraphLab.github.io

for dynamic networks with community structure can be found in [31]. This is a widely open topic for future work, both in terms of oblivious adversaries as well as adaptive adversaries. To still be able to evaluate fully dynamic algorithms in practice, research uses a wide range of models at the moment to turn static networks into dynamic ones. We give a brief overview over the most important ones. In *undo-based* approaches, edges of a static network are inserted in some order until all edges are inserted. In the end, x % of the last insertions are undone. The intuition here is that one wants undo changes that happened to a network and to recreate a previous state of the data structure. In *window-based* approaches, edges are inserted and have a predefined lifetime. That means an edge is deleted after a given number d of new edges have been inserted. In *remove and add* based approaches, a small fraction of random edges from a static network is removed and later on reinserted. In practice, researchers use a single edge as well as whole batches of edges. In *morphing-based* approaches, one takes two related networks and creates a sequence of edge updates such that the second network obtained after the update sequence has been applied to the first network.

6 FINAL REMARKS

Traditionally, algorithms are designed using simple models. In turn, performance guarantees are provable for all possible inputs. For researchers working in algorithm theory, however, implementing an algorithm is only part of the application development. This causes a growing gap between theory and practice. For dynamic algorithms, this works in both ways. On the one hand, there is a large body of theoretical work on efficient dynamic graph algorithms that received very little attention from practitioners (e.g., spanning trees or diameter). On the other hand, there is a range of problems with a large amount of practical work, yet theoretical foundations seem to be (somewhat) missing (e.g., centralities, graph partitioning/clustering). For some problems theoretical and practical work exists to a large extend, but appear to be disconnected (e.g., motif counting, maximum flows). Lastly, there are also a range of problems in which ideas from theory have been picked up and evaluated by practitioners (e.g., reachability, matching, shortest paths).

Acknowledgements.
This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024. Moreover, we have been partially supported by DFG grant SCHU 2567/1-2.

REFERENCES

[1] Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. 2019. Dynamic set cover: improved algorithms and lower bounds. In *Proc. of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, Moses Charikar and Edith Cohen (Eds.). ACM, 114–125. https://doi.org/10.1145/3313276.3316376

[2] Amir Abboud and Søren Dahlgaard. 2016. Popular conjectures as a barrier for dynamic planar graph algorithms. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 477–486.

[3] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 434–443. https://doi.org/10.1109/FOCS.2014.53

[4] Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. 2017. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, Philip N. Klein (Ed.). SIAM, 440–452. https://doi.org/10.1137/1.9781611974782.28

[5] Ittai Abraham, Shiri Chechik, and Kunal Talwar. 2014. Fully Dynamic All-Pairs Shortest Paths: Breaking the $O(n)$ Barrier. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain (LIPIcs, Vol. 28)*, Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 1–16. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2014.1

[6] Umut A. Acar, Daniel Anderson, Guy E. Blelloch, and Laxman Dhulipala. 2019. Parallel Batch-Dynamic Graph Connectivity. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, Christian Scheideler and Petra Berenbrink (Eds.). ACM, 381–392. https://doi.org/10.1145/3323165.3323196

[7] Umut A. Acar, Daniel Anderson, Guy E. Blelloch, Laxman Dhulipala, and Sam Westrick. 2020. Parallel Batch-Dynamic Trees via Change Propagation. In *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference) (LIPIcs, Vol. 173)*, Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1–2:23. https://doi.org/10.4230/LIPIcs.ESA.2020.2

[8] Nesreen K. Ahmed, Nick G. Duffield, Theodore L. Willke, and Ryan A. Rossi. 2017. On Sampling from Massive Graph Streams. *Proc. VLDB Endow.* 10, 11 (2017), 1430–1441. https://doi.org/10.14778/3137628.3137651

[9] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2014. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, Chin-Wan Chung, Andrei Z. Broder, Kyuseok Shim, and Torsten Suel (Eds.). ACM, 237–248. https://doi.org/10.1145/2566486.2568007

[10] David Alberts, Giuseppe Cattaneo, and Giuseppe F. Italiano. 1997. An Empirical Study of Dynamic Graph Algorithms. *ACM J. Exp. Algorithmics* 2 (1997), 5. https://doi.org/10.1145/264216.264223

[11] David Alberts, Giuseppe Cattaneo, Giuseppe F Italiano, Umberto Nanni, and Christos Zaroliagis. 1998. A software library of dynamic graph algorithms. In *Proc. Workshop on Algorithms and Experiments*. Citeseer, 129–136.

[12] Bowen Alpern, Roger Hoover, Barry K. Rosen, Peter F. Sweeney, and F. Kenneth Zadeck. 1990. Incremental Evaluation of Computational Circuits. In *Proc. of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA*, David S. Johnson (Ed.). SIAM, 32–42. http://dl.acm.org/citation.cfm?id=320176.320180

[13] Hamidreza Alvari, Alireza Hajibagheri, and Gita Reese Sukthankar. 2014. Community detection in dynamic social networks: A game-theoretic approach. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2014, Beijing, China, August 17-20, 2014*, Xindong Wu, Martin Ester, and Guandong Xu (Eds.). IEEE Computer Society, 101–107. https://doi.org/10.1109/ASONAM.2014.6921567

[14] Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. 2012. Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India (LIPIcs, Vol. 18)*, Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 257–266. https://doi.org/10.4230/LIPIcs.FSTTCS.2012.257

[15] Bertie Ancona, Monika Henzinger, Liam Roditty, Virginia Vassilevska Williams, and Nicole Wein. 2019. Algorithms and Hardness for Diameter in Dynamic Graphs. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece (LIPIcs, Vol. 132)*, Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:14. https://doi.org/10.4230/LIPIcs.ICALP.2019.13

[16] Eugenio Angriman, Henning Meyerhenke, Christian Schulz, and Bora Uçar. 2021. Fully-dynamic Weighted Matching Approximation in Practice. In *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2021, Virtual Conference, July 19-21, 2021*, Michael Bender, John Gilbert, Bruce Hendrickson, and Blair D. Sullivan (Eds.). SIAM, 32–44. https://doi.org/10.1137/1.9781611976830.4

[17] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. 2018. Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*. 7:1–7:16. https://doi.org/10.4230/LIPIcs.ICALP.2018.7

[18] Sabeur Aridhi, Martin Brugnara, Alberto Montresor, and Yannis Velegrakis. 2016. Distributed k-core decomposition and maintenance in large dynamic graphs. In *Proc. of the 10th ACM International Conference on Distributed and Event-based Systems*. 161–168. https://doi.org/10.1145/2933267.2933299

[19] Lavy Libman Ariel Orda, Nidhi Hegde (Ed.). 2010. *8th International Symposium on Modeling and Optimization in Mobile, Ad-Hoc and Wireless Networks (WiOpt 2010), May 31 - June 4, 2010, University of Avignon, Avignon, France*. IEEE. https://ieeexplore.ieee.org/xpl/conhome/5509122/proceeding

[20] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. 2018. Fully dynamic maximal independent set with sublinear update time. In *Proc. of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, Ilias Diakonikolas, David Kempe, and Monika Henzinger (Eds.). ACM, 815–826. https://doi.org/10.1145/3188745.3188922

[21] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. 2019. Fully Dynamic Maximal Independent Set with Sublinear in n Update Time. In *Proc. of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1919–1936. https://doi.org/10.1137/1.9781611975482.116

[22] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. 2012. *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties*. Springer Science & Business Media.

[23] Muhammad A. Awad, Saman Ashkiani, Serban D. Porumbescu, and John D. Owens. 2020. Dynamic Graphs on the GPU. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, May 18-22, 2020*. IEEE, 739–748. https://doi.org/10.1109/IPDPS47924.2020.00081

[24] Thomas Aynaud and Jean-Loup Guillaume. 2010. Static community detection algorithms for evolving networks, See [19], 513–519. http://ieeexplore.ieee.org/document/5520221/

[25] Shweta Bansal, Sanjukta Bhowmick, and Prashant Paymal. 2010. Fast Community Detection for Dynamic Complex Networks. In *Complex Networks - Second International Workshop, CompleNet 2010, Rio de Janeiro, Brazil, October 13-15, 2010, Revised Selected Papers (Communications in Computer and Information Science, Vol. 116)*, Luciano da F. Costa, Alexandre G. Evsukoff, Giuseppe Mangioni, and Ronaldo Menezes (Eds.). Springer, 196–207. https://doi.org/10.1007/978-3-642-25501-4_20

[26] Surender Baswana, Shreejit Ray Chaudhury, Keerti Choudhary, and Shahbaz Khan. 2019. Dynamic DFS in Undirected Graphs: Breaking the $O(m)$ Barrier. *SIAM J. Comput.* 48, 4 (2019), 1335–1363. https://doi.org/10.1137/17M114306X

[27] Surender Baswana, Manoj Gupta, and Sandeep Sen. 2015. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time. *SIAM J. Comput.* 44, 1 (2015), 88–113. https://doi.org/10.1137/16M1106158

[28] Surender Baswana, Shiv Kumar Gupta, and Ayush Tulsyan. 2019. Fault Tolerant and Fully Dynamic DFS in Undirected Graphs: Simple Yet Efficient. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany (LIPIcs, Vol. 138)*, Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 65:1–65:16. https://doi.org/10.4230/LIPIcs.MFCS.2019.65

[29] Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. 2010. Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. *ACM J. Exp. Algorithmics* 15 (2010). https://doi.org/10.1145/1671970.1671976

[30] Reinhard Bauer and Dorothea Wagner. 2009. Batch Dynamic Single-Source Shortest-Path Algorithms: An Experimental Study. In *Experimental Algorithms, 8th International Symposium, SEA 2009, Dortmund, Germany, June 4-6, 2009. Proc. (Lecture Notes in Computer Science, Vol. 5526)*, Jan Vahrenhold (Ed.). Springer, 51–62. https://doi.org/10.1007/978-3-642-02011-7_7

[31] F. Becker. 2020. *Generative Model for Dynamic Networks with Community Structures.* Master's Thesis. Heidelberg University.

[32] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. 2019. Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time, See [266], 382–405. https://doi.org/10.1109/FOCS.2019.00032

[33] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. 2020. Fully Dynamic Matching: Beating 2-Approximation in Δ^{ϵ} Update Time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla (Ed.). SIAM, 2492–2508. https://doi.org/10.1137/1.9781611975994.152

[34] Richard Bellman. 1958. On a routing problem. *Quarterly of applied mathematics* 16, 1 (1958), 87–90.

[35] Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. 2021. New Techniques and Fine-Grained Hardness for Dynamic Near-Additive Spanners. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, Dániel Marx (Ed.). SIAM, 1836–1855. https://doi.org/10.1137/1.9781611976465.110

[36] Elisabetta Bergamini and Henning Meyerhenke. 2015. Fully-Dynamic Approximation of Betweenness Centrality. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proc. (Lecture Notes in Computer Science, Vol. 9294)*, Nikhil Bansal and Irene Finocchi (Eds.). Springer, 155–166. https://doi.org/10.1007/978-3-662-48350-3_14

[37] Elisabetta Bergamini and Henning Meyerhenke. 2016. Approximating Betweenness Centrality in Fully Dynamic Networks. *Internet Math.* 12, 5 (2016), 281–314. https://doi.org/10.1080/15427951.2016.1177802

[38] Elisabetta Bergamini, Henning Meyerhenke, and Christian Staudt. 2015. Approximating Betweenness Centrality in Large Evolving Networks. In *Proc. of the Seventeenth Workshop on Algorithm Engineering and Experiments, ALENEX 2015, San Diego, CA, USA, January 5, 2015*, Ulrik Brandes and David Eppstein (Eds.). SIAM, 133–146. https://doi.org/10.1137/1.9781611973754.12

[39] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. 2019. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proc. of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1899–1918. https://doi.org/10.1137/1.9781611975482.115

[40] Aaron Bernstein and Cliff Stein. 2016. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proc. of the 27th Symposium on Discrete Algorithms SODA*. SIAM, 692–711. https://doi.org/10.1137/1.9781611974331.ch50

[41] Emanuele Berrettini, Gianlorenzo D'Angelo, and Daniel Delling. 2009. Arc-Flags in Dynamic Graphs. In *ATMOS 2009 - 9th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, IT University of Copenhagen, Denmark, September 10, 2009 (OASICS, Vol. 12)*, Jens Clausen and Gabriele Di Stefano (Eds.). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. http://drops.dagstuhl.de/opus/volltexte/2009/2149

[42] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. 2020. Deterministic Dynamic Matching in O(1) Update Time. *Algorithmica* 82, 4 (2020), 1057–1080. https://doi.org/10.1007/s00453-019-00630-4

[43] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. 2018. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. *SIAM J. Comput.* 47, 3 (2018), 859–887. https://doi.org/10.1137/140998925

[44] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. 2018. Dynamic algorithms via the primal-dual method. *Inf. Comput.* 261 (2018), 219–239. https://doi.org/10.1016/j.ic.2018.02.005

[45] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2016. New deterministic approximation algorithms for fully dynamic matching. In *Proc. of the 48th Annual Symposium on Theory of Computing*. ACM, 398–411. https://doi.org/10.1145/2897518.2897568

[46] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2017. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover in $O(\log^3 n)$ Worst Case Update Time. In *Proc. of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms SODA*, Philip N. Klein (Ed.). SIAM, 470–489. https://doi.org/10.1137/1.9781611973730.54

[47] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. 2019. A New Deterministic Algorithm for Dynamic Set Cover. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 406–423. https://doi.org/10.1109/FOCS.2019.00033

[48] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. 2015. Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams. In *Proc. of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, Rocco A. Servedio and Ronitt Rubinfeld (Eds.). ACM, 173–182. https://doi.org/10.1145/2746539.2746592

[49] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. 2020. An Improved Algorithm for Dynamic Set Cover. *CoRR* abs/2002.11171 (2020). arXiv:2002.11171 https://arxiv.org/abs/2002.11171

[50] Sayan Bhattacharya and Janardhan Kulkarni. 2019. Deterministically Maintaining a $(2+\varepsilon)$-Approximate Minimum Vertex Cover in $O(1/\varepsilon^2)$ Amortized Update Time. In *Proc. of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1872–1885. https://doi.org/10.1137/1.9781611975482.113

[51] Sujoy Bhore, Guangping Li, and Martin Nöllenburg. 2020. An Algorithmic Study of Fully Dynamic Independent Sets for Map Labeling, See [110], 19:1–19:24. https://doi.org/10.4230/LIPIcs.ESA.2020.19

[52] Patrick Bisenius, Elisabetta Bergamini, Eugenio Angriman, and Henning Meyerhenke. 2018. Computing Top-k Closeness Centrality in Fully-dynamic Graphs. In *Proc. of the Twentieth Workshop on Algorithm Engineering and Experiments, ALENEX 2018, New Orleans, LA, USA, January 7-8, 2018*, Rasmus Pagh and Suresh Venkatasubramanian (Eds.). SIAM, 21–35. https://doi.org/10.1137/1.9781611975055.3

[53] Yuri Boykov and Vladimir Kolmogorov. 2004. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 9 (2004), 1124–1137. https://doi.org/10.1109/TPAMI.2004.60

[54] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. 2008. On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering* 20, 2 (2008), 172–188. https://doi.org/10.1109/TKDE.2007.190689

[55] Valentin Buchhold, Daniel Delling, Dennis Schieferdecker, and Michael Wegner. 2020. Fast and Stable Repartitioning of Road Networks. In *18th International Symposium on Experimental Algorithms (SEA 2020) (LIPIcs, Vol. 160)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 26:1–26:15. https://doi.org/10.4230/LIPIcs.SEA.2020.26

[56] Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. 2016. Triangle Counting in Dynamic Graph Streams. *Algorithmica* 76, 1 (2016), 259–278. https://doi.org/10.1007/s00453-015-0036-4

[57] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. 2006. Counting triangles in data streams. In *Proc. of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, Stijn Vansummeren (Ed.). ACM, 253–262. https://doi.org/10.1145/1142351.1142388

[58] Luciana S. Buriol, Mauricio G. C. Resende, and Mikkel Thorup. 2008. Speeding Up Dynamic Shortest-Path Algorithms. *INFORMS J. Comput.* 20, 2 (2008), 191–204. https://doi.org/10.1287/ijoc.1070.0231

[59] Federico Busato, Oded Green, Nicola Bombieri, and David A. Bader. 2018. Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs. In *2018 IEEE High Performance Extreme Computing Conference, HPEC 2018, Waltham, MA, USA, September 25-27, 2018*. IEEE, 1–7. https://doi.org/10.1109/HPEC.2018.8547541

[60] Giuseppe Cattaneo, Pompeo Faruolo, Umberto Ferraro Petrillo, and Giuseppe F. Italiano. 2002. Maintaining Dynamic Minimum Spanning Trees: An Experimental Study. In *Algorithm Engineering and Experiments, 4th International Workshop, ALENEX 2002, San Francisco, CA, USA, January 4-5, 2002, Revised Papers (Lecture Notes in Computer Science, Vol. 2409)*, David M. Mount and Clifford Stein (Eds.). Springer, 111–125. https://doi.org/10.1007/3-540-45643-0_9

[61] Giuseppe Cattaneo, Pompeo Faruolo, Umberto Ferraro Petrillo, and Giuseppe F. Italiano. 2010. Maintaining dynamic minimum spanning trees: An experimental study. *Discret. Appl. Math.* 158, 5 (2010), 404–425. https://doi.org/10.1016/j.dam.2009.10.005

[62] Edward P. F. Chan and Yaya Yang. 2009. Shortest Path Tree Computation in Dynamic Graphs. *IEEE Trans. Computers* 58, 4 (2009), 541–557. https://doi.org/10.1109/TC.2008.198

[63] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1913)*, Klaus Jansen and Samir Khuller (Eds.). Springer, 84–95. https://doi.org/10.1007/3-540-44436-X_10

[64] Moses Charikar and Shay Solomon. 2018. Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018 (LIPIcs, Vol. 107)*, Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 33:1–33:14. https://doi.org/10.4230/LIPIcs.ICALP.2018.33

[65] Shiri Chechik and Tianyi Zhang. 2019. Fully Dynamic Maximal Independent Set in Expected Poly-Log Update Time, See [266], 370–381. https://doi.org/10.1109/FOCS.2019.00031

[66] Mostafa Haghir Chehreghani, Albert Bifet, and Talel Abdessalem. 2018. DyBED: An Efficient Algorithm for Updating Betweenness Centrality in Directed Dynamic Graphs. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, Naoki Abe, Huan Liu, Calton Pu, Xiaohua Hu, Nesreen K. Ahmed, Mu Qiao, Yang Song, Donald Kossmann, Bing Liu, Kisung Lee, Jiliang Tang, Jingrui He, and Jeffrey S. Saltz (Eds.). IEEE, 2114–2123. https://doi.org/10.1109/BigData.2018.8622452

[67] Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. 2020. Fast Dynamic Cuts, Distances and Effective Resistances via Vertex Sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. IEEE, 1135–1146. https://doi.org/10.1109/FOCS46700.2020.00109

[68] Alessio Cionini, Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos D. Zaroliagis. 2017. Engineering graph-based models for dynamic timetable information systems. *J. Discrete Algorithms* 46-47 (2017), 40–58. https://doi.org/10.1016/j.jda.2017.09.001

[69] Pierluigi Crescenzi, Clémence Magnien, and Andrea Marino. 2020. Finding Top-k Nodes for Temporal Closeness in Large Temporal Graphs. *Algorithms* 13, 9 (2020), 211. https://doi.org/10.3390/a13090211

[70] Michael Crouch and Daniel S. Stubbs. 2014. Improved Streaming Algorithms for Weighted Matching, via Unweighted Matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain (LIPIcs, Vol. 28)*, Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 96–104. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2014.96

[71] Annalisa D'Andrea, Mattia D'Emidio, Daniele Frigioni, Stefano Leucci, and Guido Proietti. 2015. Dynamic Maintenance of a Shortest-Path Tree on Homogeneous Batches of Updates: New Algorithms and Experiments. *ACM J. Exp. Algorithmics* 20 (2015), 1.5:1.1–1.5:1.33. https://doi.org/10.1145/2786022

[72] Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. 2014. Fully dynamic update of arc-flags. *Networks* 63, 3 (2014), 243–259. https://doi.org/10.1002/net.21542

[73] Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. 2019. Fully Dynamic 2-Hop Cover Labeling. *ACM J. Exp. Algorithmics* 24, 1 (2019), 1.6:1–1.6:36. https://doi.org/10.1145/3299901

[74] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato Fonseca F. Werneck. 2011. Customizable Route Planning. In *Experimental Algorithms - 10th International Symposium, SEA 2011, Kolimpari, Chania, Crete, Greece, May 5-7, 2011. Proc. (Lecture Notes in Computer Science, Vol. 6630)*, Panos M. Pardalos and Steffen Rebennack (Eds.). Springer, 376–387. https://doi.org/10.1007/978-3-642-20662-7_32

[75] Daniel Delling and Dorothea Wagner. 2007. Landmark-Based Routing in Dynamic Graphs, See [77], 52–65. https://doi.org/10.1007/978-3-540-72845-0_5

[76] Camil Demetrescu. 2001. *Fully Dynamic Algorithms for Path Problems on Directed Graphs*. Ph.D. Dissertation. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.8921

[77] Camil Demetrescu (Ed.). 2007. *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proc.* Lecture Notes in Computer Science, Vol. 4525. Springer. https://doi.org/10.1007/978-3-540-72845-0

[78] Camil Demetrescu, Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. 2000. Maintaining Shortest Paths in Digraphs with Arbitrary Arc Weights: An Experimental Study. In *Algorithm Engineering, 4th International Workshop, WAE 2000, Saarbrücken, Germany, September 5-8, 2000, Proc. (Lecture Notes in Computer Science, Vol. 1982)*, Stefan Näher and Dorothea Wagner (Eds.). Springer, 218–229. https://doi.org/10.1007/3-540-44691-5_19

[79] Camil Demetrescu and Giuseppe F. Italiano. 2004. A new approach to dynamic all pairs shortest paths. *J. ACM* 51, 6 (2004), 968–992. https://doi.org/10.1145/1039488.1039492

[80] Camil Demetrescu and Giuseppe F. Italiano. 2006. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Trans. Algorithms* 2, 4 (2006), 578–601. https://doi.org/10.1145/1198513.1198519

[81] Laxman Dhulipala, Quanquan C. Liu, and Julian Shun. 2020. Parallel Batch-Dynamic k-Clique Counting. *CoRR* abs/2003.13585 (2020). arXiv:2003.13585 https://arxiv.org/abs/2003.13585

[82] Jana Diesner, Elena Ferrari, and Guandong Xu (Eds.). 2017. *Proc. of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, Sydney, Australia, July 31 - August 03, 2017*. ACM. https://doi.org/10.1145/3110025

[83] Edsger W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1 (1959), 269–271. https://doi.org/10.1007/BF01386390

[84] Christof Doll, Tanja Hartmann, and Dorothea Wagner. 2011. Fully-Dynamic Hierarchical Graph Clustering Using Cut Trees. In *12th Intl. Symp. on Algorithms and Data Structures, WADS'11 (LNCS, Vol. 6844)*. 338–349. https://doi.org/10.1007/978-3-642-22300-6_29

[85] David Ediger, Robert McColl, E. Jason Riedy, and David A. Bader. 2012. STINGER: High performance data structure for streaming graphs. In *IEEE Conference on High Performance Extreme Computing, HPEC 2012, Waltham, MA, USA, September 10-12, 2012*. IEEE, 1–5. https://doi.org/10.1109/HPEC.2012.6408680

[86] Jack R. Edmonds and Richard M. Karp. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* 19, 2 (1972), 248–264. https://doi.org/10.1145/321694.321699

[87] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. 1996. Separator Based Sparsification. I. Planary Testing and Minimum Spanning Trees. *J. Comput. Syst. Sci.* 52, 1 (1996), 3–27. https://doi.org/10.1006/jcss.1996.0002

[88] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. 1998. Separator-Based Sparsification II: Edge and Vertex Connectivity. *SIAM J. Comput.* 28, 1 (1998), 341–381. https://doi.org/10.1137/S0097539794269072

[89] David Eppstein, Michael T. Goodrich, Darren Strash, and Lowell Trott. 2012. Extended dynamic subgraph statistics using h-index parameterized data structures. *Theor. Comput. Sci.* 447 (2012), 44–52. https://doi.org/10.1016/j.tcs.2011.11.034

[90] David Eppstein and Emma S. Spiro. 2012. The h-Index of a Graph and its Application to Dynamic Subgraph Statistics. *J. Graph Algorithms Appl.* 16, 2 (2012), 543–567. https://doi.org/10.7155/jgaa.00273

[91] Wenfei Fan, Muyang Liu, Chao Tian, Ruiqi Xu, and Jingren Zhou. 2020. Incrementalization of Graph Partitioning Algorithms. *Proc. VLDB Endow.* 13, 8 (2020), 1261–1274. https://doi.org/10.14778/3389133.3389142

[92] Guoyao Feng, Xiao Meng, and Khaled Ammar. 2015. DISTINGER: A distributed graph data structure for massive dynamic graph processing. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 1814–1822. https://doi.org/10.1109/BigData.2015.7363954

[93] Sebastian Forster, Gramoz Goranci, and Monika Henzinger. 2020. Dynamic Maintenance of Low-Stretch Probabilistic Tree Embeddings with Applications. *CoRR* abs/2004.10319 (2020). arXiv:2004.10319 https://arxiv.org/abs/2004.10319

[94] Greg N. Frederickson. 1985. Data Structures for On-Line Updating of Minimum Spanning Trees, with Applications. *SIAM J. Comput.* 14, 4 (1985), 781–798. https://doi.org/10.1137/0214055

[95] Greg N. Frederickson. 1997. Ambivalent Data Structures for Dynamic 2-Edge-Connectivity and k Smallest Spanning Trees. *SIAM J. Comput.* 26, 2 (1997), 484–538. https://doi.org/10.1137/S0097539792226825

[96] Daniele Frigioni, Mario Ioffreda, Umberto Nanni, and Giulio Pasqualone. 1998. Experimental Analysis of Dynamic Algorithms for the Single-Source Shortest-Path Problem. *ACM J. Exp. Algorithmics* 3 (1998), 5. https://doi.org/10.1145/297096.297147

[97] Daniele Frigioni, Alberto Marchetti-Spaccamela, and Umberto Nanni. 1996. Fully Dynamic Output Bounded Single Source Shortest Path Problem (Extended Abstract). In *Proc. of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, 28-30 January 1996, Atlanta, Georgia, USA*, Éva Tardos (Ed.). ACM/SIAM, 212–221. http://dl.acm.org/citation.cfm?id=313852.313926

[98] Daniele Frigioni, Tobias Miller, Umberto Nanni, and Christos D. Zaroliagis. 2001. An Experimental Study of Dynamic Algorithms for Transitive Closure. *ACM J. Exp. Algorithmics* 6 (2001), 9. https://doi.org/10.1145/945394.945403

[99] Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos D. Zaroliagis. 2019. Multimodal Dynamic Journey-Planning. *Algorithms* 12, 10 (2019), 213. https://doi.org/10.3390/a12100213

[100] Andrew V. Goldberg and Chris Harrelson. 2005. Computing the shortest path: *A* search meets graph theory. In *Proc. of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2005, Vancouver, British Columbia, Canada, January 23-25, 2005*. SIAM, 156–165. http://dl.acm.org/citation.cfm?id=1070432.1070455

[101] Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Pushmeet Kohli, Robert Endre Tarjan, and Renato F. Werneck. 2015. Faster and More Dynamic Maximum Flow by Incremental Breadth-First Search. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proc. (Lecture Notes in Computer Science, Vol. 9294)*, Nikhil Bansal and Irene Finocchi (Eds.). Springer, 619–630. https://doi.org/10.1007/978-3-662-48350-3_52

[102] Andrew V. Goldberg and Robert Endre Tarjan. 1988. A new approach to the maximum-flow problem. *J. ACM* 35, 4 (1988), 921–940. https://doi.org/10.1145/48014.61051

[103] Andrew V. Goldberg and Robert E. Tarjan. 1988. A new approach to the maximum-flow problem. *J. ACM* 35, 4 (1988), 921–940.

[104] Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. 2018. Incremental exact min-cut in polylogarithmic amortized update time. *ACM Transactions on Algorithms (TALG)* 14, 2 (2018), 1–21.

[105] Gramoz Goranci, Harald Räcke, Thatchaphol Saranurak, and Zihan Tan. 2020. The Expander Hierarchy and its Applications to Dynamic Graph Algorithms. *CoRR* abs/2005.02369 (2020). arXiv:2005.02369 https://arxiv.org/abs/2005.02369

[106] Robert Görke, Tanja Hartmann, and Dorothea Wagner. 2012. Dynamic Graph Clustering Using Minimum-Cut Trees. *J. Graph Algorithms Appl.* 16, 2 (2012), 411–446. https://doi.org/10.7155/jgaa.00269

[107] Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. 2012. An Efficient Generator for Clustered Dynamic Random Networks. In *Design and Analysis of Algorithms - First Mediterranean Conference on Algorithms, MedAlg 2012, Kibbutz Ein Gedi, Israel, December 3-5, 2012. Proc. (Lecture Notes in Computer Science, Vol. 7659)*, Guy Even and Dror Rawitz (Eds.). Springer, 219–233. https://doi.org/10.1007/978-3-642-34862-4_16

[108] Robert Görke, Pascal Maillard, Andrea Schumm, Christian Staudt, and Dorothea Wagner. 2013. Dynamic graph clustering combining modularity and smoothness. *ACM J. Exp. Algorithmics* 18 (2013). https://doi.org/10.1145/2444016.2444021

[109] Robert Görke, Pascal Maillard, Christian Staudt, and Dorothea Wagner. 2010. Modularity-Driven Clustering of Dynamic Graphs. In *Experimental Algorithms, 9th International Symposium, SEA 2010, Ischia Island, Naples, Italy, May 20-22, 2010. Proc. (Lecture Notes in Computer Science, Vol. 6049)*, Paola Festa (Ed.). Springer, 436–448. https://doi.org/10.1007/978-3-642-13193-6_37

[110] Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders (Eds.). 2020. *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*. LIPIcs, Vol. 173. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[111] Sergio Greco, Cristian Molinaro, Chiara Pulice, and Ximena Quintana. 2017. Incremental maximum flow computation on evolving networks. In *Proc. of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*, Ahmed Seffah, Birgit Penzenstadler, Carina Alves, and Xin Peng (Eds.). ACM, 1061–1067. https://doi.org/10.1145/3019612.3019816

[112] Oded Green, Robert McColl, and David A. Bader. 2012. A Fast Algorithm for Streaming Betweenness Centrality. In *2012 International Conference on Privacy, Security, Risk and Trust, PASSAT 2012, and 2012 International Confernece on Social Computing, SocialCom 2012, Amsterdam, Netherlands, September 3-5, 2012*. IEEE Computer Society, 11–20. https://doi.org/10.1109/SocialCom-PASSAT.2012.37

[113] Manoj Gupta and Shahbaz Khan. to appear, 2021. Simple dynamic algorithms for Maximal Independent Set and other problems. In *4th Symposium on Simplicity in Algorithms, SOSA@SODA 2021*. arXiv:1804.01823 http://arxiv.org/abs/1804.01823

[114] Manoj Gupta and Richard Peng. 2013. Fully Dynamic $(1+\epsilon)$-Approximate Matchings. In *54th Symposium on Foundations of Computer Science, FOCS*. IEEE Computer Society, 548–557. https://doi.org/10.1109/FOCS.2013.65

[115] Maximilian Probst Gutenberg and Christian Wulff-Nilsen. 2020. Fully-Dynamic All-Pairs Shortest Paths: Improved Worst-Case Time and Space Bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla (Ed.). SIAM, 2562–2574. https://doi.org/10.1137/1.9781611975994.156

[116] Guyue Han and Harish Sethu. 2017. Edge Sample and Discard: A New Algorithm for Counting Triangles in Large Dynamic Graphs, See [82], 44–49. https://doi.org/10.1145/3110025.3110061

[117] Kathrin Hanauer, Monika Henzinger, and Qi Cheng Hua. 2022. Fully Dynamic Four-Vertex Subgraph Counting. In *1st Symposium on Algorithmic Foundations of Dynamic Networks, SAND 2022, March 28-30, 2022, Virtual Conference (LIPIcs, Vol. 221)*, James Aspnes and Othon Michail (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 18:1–18:17. https://doi.org/10.4230/LIPIcs.SAND.2022.18

[118] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. 2020. Faster Fully Dynamic Transitive Closure in Practice. In *18th International Symposium on Experimental Algorithms, SEA 2020, June 16-18, 2020, Catania, Italy (LIPIcs, Vol. 160)*, Simone Faro and Domenico Cantone (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:14. https://doi.org/10.4230/LIPIcs.SEA.2020.14

[119] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. 2020. Fully Dynamic Single-Source Reachability in Practice: An Experimental Study. In *Proc. of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020*, Guy E. Blelloch and Irene Finocchi (Eds.). SIAM, 106–119. https://doi.org/10.1137/1.9781611976007.9

[120] Takanori Hayashi, Takuya Akiba, and Ken-ichi Kawarabayashi. 2016. Fully Dynamic Shortest-Path Distance Query Acceleration on Massive Networks. In *Proc. of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang Zhou, Yi Chang, Yunyao Li, and Parikshit Sondhi (Eds.). ACM, 1533–1542. https://doi.org/10.1145/2983323.2983731

[121] Takanori Hayashi, Takuya Akiba, and Yuichi Yoshida. 2015. Fully Dynamic Betweenness Centrality Maintenance on Massive Networks. *Proc. VLDB Endow.* 9, 2 (2015), 48–59. https://doi.org/10.14778/2850578.2850580

[122] Keith Henderson and Tina Eliassi-Rad. 2009. Applying latent dirichlet allocation to group discovery in large graphs. In *Proc. of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*, Sung Y. Shin and Sascha Ossowski (Eds.). ACM, 1456–1461. https://doi.org/10.1145/1529282.1529607

[123] Bruce Hendrickson, Robert W. Leland, and Rafael Van Driessche. 1996. Enhancing Data Locality by Using Terminal Propagation. In *29th Annual Hawaii International Conference on System Sciences (HICSS-29), January 3-6, 1996, Maui, Hawaii, USA*. IEEE Computer Society, 565–574. https://doi.org/10.1109/HICSS.1996.495507

[124] Monika Henzinger. 2018. The State of the Art in Dynamic Graph Algorithms. In *44th Intl. Conf. on Current Trends in Theory and Practice of Computer Science, SOFSEM'18 (LNCS, Vol. 10706)*. Springer, 40–44. https://doi.org/10.1007/978-3-319-73117-9_3

[125] Monika Henzinger, Shahbaz Khan, Richard Paul, and Christian Schulz. 2020. Dynamic Matching Algorithms in Practice, See [110], 58:1–58:20. https://doi.org/10.4230/LIPIcs.ESA.2020.58

[126] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proc. of the forty-seventh annual ACM symposium on Theory of computing*. 21–30. https://doi.org/10.1145/2746539.2746609

[127] Monika Henzinger, Andrea Lincoln, and Barna Saha. 2022. The Complexity of Average-Case Dynamic Subgraph Counting. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, Joseph (Seffi) Naor and Niv Buchbinder (Eds.). SIAM, 459–498. https://doi.org/10.1137/1.9781611977073.23

[128] Monika Henzinger, Alexander Noe, and Christian Schulz. 2022. Practical Fully Dynamic Minimum Cut Algorithms. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*, Cynthia A. Phillips and Bettina Speckmann (Eds.). SIAM, 13–26. https://doi.org/10.1137/1.9781611977042.2

[129] Monika Henzinger, Ami Paz, and Stefan Schmid. 2021. On the Complexity of Weight-Dynamic Network Algorithms. In *IFIP Networking Conference, IFIP Networking 2021, Espoo and Helsinki, Finland, June 21-24, 2021*, Zheng Yan, Gareth Tyson, and Dimitrios Koutsonikolas (Eds.). IEEE, 1–9. https://doi.org/10.23919/IFIPNetworking52078.2021.9472803

[130] Monika Rauch Henzinger. 1995. Approximating minimum cuts under insertions. In *International Colloquium on Automata, Languages, and Programming*. Springer, 280–291.

[131] Monika Rauch Henzinger. 1995. Fully Dynamic Biconnectivity in Graphs. *Algorithmica* 13, 6 (1995), 503–538. https://doi.org/10.1007/BF01189067

[132] Monika Rauch Henzinger. 2000. Improved Data Structures for Fully Dynamic Biconnectivity. *SIAM J. Comput.* 29, 6 (2000), 1761–1815. https://doi.org/10.1137/S0097539794263907

[133] Monika Rauch Henzinger and Michael L. Fredman. 1998. Lower Bounds for Fully Dynamic Connectivity Problems in Graphs. *Algorithmica* 22, 3 (1998), 351–362. https://doi.org/10.1007/PL00009228

[134] Monika Rauch Henzinger and Valerie King. 1995. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proc. of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, Frank Thomson Leighton and Allan Borodin (Eds.). ACM, 519–527. https://doi.org/10.1145/225058.225269

[135] Monika R Henzinger and Valerie King. 1997. Maintaining minimum spanning trees in dynamic graphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 594–604.

[136] Monika Rauch Henzinger and Valerie King. 1999. Randomized Fully Dynamic Graph Algorithms with Polylogarithmic Time per Operation. *J. ACM* 46, 4 (1999), 502–516. https://doi.org/10.1145/320211.320215

[137] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 1998. Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. In *Proc. of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, Jeffrey Scott Vitter (Ed.). ACM, 79–89. https://doi.org/10.1145/276698.276715

[138] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM* 48, 4 (2001), 723–760. https://doi.org/10.1145/502090.502095

[139] Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. 2015. Faster Fully-Dynamic Minimum Spanning Forest. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proc. (Lecture Notes in Computer Science, Vol. 9294)*, Nikhil Bansal and Irene Finocchi (Eds.). Springer, 742–753. https://doi.org/10.1007/978-3-662-48350-3_62

[140] Y.F. Hu and R.J. Blake. 1999. An improved diffusion algorithm for dynamic load balancing. *Parallel Comput.* 25, 4 (1999), 417 – 444. https://doi.org/10.1016/S0167-8191(99)00002-2

[141] Qiang-Sheng Hua, Yuliang Shi, Dongxiao Yu, Hai Jin, Jiguo Yu, Zhipeng Cai, Xiuzhen Cheng, and Hanhua Chen. 2020. Faster Parallel Core Maintenance Algorithms in Dynamic Graphs. *IEEE Trans. Parallel Distributed Syst.* 31, 6 (2020), 1287–1300. https://doi.org/10.1109/TPDS.2019.2960226

[142] Jiewen Huang and Daniel Abadi. 2016. LEOPARD: Lightweight Edge-Oriented Partitioning and Replication for Dynamic Graphs. *Proc. VLDB Endow.* 9, 7 (2016), 540–551. https://doi.org/10.14778/2904483.2904486

[143] Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. 2017. Fully Dynamic Connectivity in $O(\log n(\log\log n)^2)$ Amortized Expected Time. In *Proc. of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, Philip N. Klein (Ed.). SIAM, 510–520. https://doi.org/10.1137/1.9781611974782.32

[144] Giuseppe Amato II, Giuseppe Cattaneo, and Giuseppe F. Italiano. 1997. Experimental Analysis of Dynamic Minimum Spanning Tree Algorithms (Extended Abstract). In *Proc. of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, 5-7 January 1997, New Orleans, Louisiana, USA*, Michael E. Saks (Ed.). ACM/SIAM, 314–323. http://dl.acm.org/citation.cfm?id=314161.314314

[145] Giuseppe F. Italiano. 2016. Fully Dynamic Higher Connectivity. In *Encyclopedia of Algorithms*. 797–800. https://doi.org/10.1007/978-1-4939-2864-4_154

[146] Zoran Ivkovic and Errol L. Lloyd. 1993. Fully Dynamic Maintenance of Vertex Cover. In *19th International Workshop Graph-Theoretic Concepts in Computer Science (LNCS, Vol. 790)*. 99–111.

[147] Keita Iwabuchi, Scott Sallinen, Roger Pearce, Brian Van Essen, Maya Gokhale, and Satoshi Matsuoka. 2016. Towards a distributed large-scale dynamic graph data store. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 892–901. https://doi.org/10.1109/IPDPSW.2016.189

[148] Raj Iyer, David R. Karger, Hariharan Rahul, and Mikkel Thorup. 2001. An Experimental Study of Polylogarithmic, Fully Dynamic, Connectivity Algorithms. *ACM J. Exp. Algorithmics* 6 (2001), 4. https://doi.org/10.1145/945394.945398

[149] Hai Jin, Na Wang, Dongxiao Yu, Qiang-Sheng Hua, Xuanhua Shi, and Xia Xie. 2018. Core Maintenance in Dynamic Graphs: A Parallel Approach Based on Matching. *IEEE Trans. Parallel Distributed Syst.* 29, 11 (2018), 2416–2428. https://doi.org/10.1109/TPDS.2018.2835441

[150] Wenyu Jin and Xiaorui Sun. 2020. Fully Dynamic c-Edge Connectivity in Subpolynomial Time. *CoRR* abs/2004.07650 (2020). arXiv:2004.07650 https://arxiv.org/abs/2004.07650

[151] Hossein Jowhari and Mohammad Ghodsi. 2005. New Streaming Algorithms for Counting Triangles in Graphs. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proc. (Lecture Notes in Computer Science, Vol. 3595)*, Lusheng Wang (Ed.). Springer, 710–716. https://doi.org/10.1007/11533719_72

[152] Bruce M. Kapron, Valerie King, and Ben Mountjoy. 2013. Dynamic graph connectivity in polylogarithmic worst case time. In *Proc. of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, Sanjeev Khanna (Ed.). SIAM, 1131–1142. https://doi.org/10.1137/1.9781611973105.81

[153] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2019. Counting Triangles under Updates in Worst-Case Optimal Time. In *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal (LIPIcs, Vol. 127)*, Pablo Barceló and Marco Calautti (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 4:1–4:18. https://doi.org/10.4230/LIPIcs.ICDT.2019.4

[154] Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. 2020. Maintaining Triangle Queries under Updates. *ACM Trans. Database Syst.* 45, 3 (2020), 11:1–11:46. https://doi.org/10.1145/3396375

[155] Miray Kas, Kathleen M. Carley, and L. Richard Carley. 2013. Incremental closeness centrality for dynamically changing social networks. In *Advances in Social Networks Analysis and Mining 2013, ASONAM '13, Niagara, ON, Canada - August 25 - 29, 2013*, Jon G. Rokne and Christos Faloutsos (Eds.). ACM, 1250–1258. https://doi.org/10.1145/2492517.2500270

[156] Shahbaz Khan. 2019. Near Optimal Parallel Algorithms for Dynamic DFS in Undirected Graphs. *ACM Trans. Parallel Comput.* 6, 3 (2019), 18:1–18:33. https://doi.org/10.1145/3364212

[157] Tim Kiefer, Dirk Habich, and Wolfgang Lehner. 2016. Penalized Graph Partitioning for Static and Dynamic Load Balancing. In *Euro-Par 2016: Parallel Processing - 22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proc. (Lecture Notes in Computer Science, Vol. 9833)*, Pierre-François Dutot and Denis Trystram (Eds.). Springer, 146–158. https://doi.org/10.1007/978-3-319-43659-3_11

[158] Valerie King. 1999. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. IEEE Computer Society, 81–91. https://doi.org/10.1109/SFFCS.1999.814580

[159] Valerie King and Mikkel Thorup. 2001. A Space Saving Trick for Directed Dynamic Transitive Closure and Shortest Path Algorithms. In *Computing and Combinatorics, 7th Annual International Conference, COCOON 2001, Guilin, China, August 20-23, 2001, Proc. (Lecture Notes in Computer Science, Vol. 2108)*, Jie Wang (Ed.). Springer, 268–277. https://doi.org/10.1007/3-540-44679-6_30

[160] Pushmeet Kohli and Philip H. S. Torr. 2007. Dynamic Graph Cuts for Efficient Inference in Markov Random Fields. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 12 (2007), 2079–2088. https://doi.org/10.1109/TPAMI.2007.1128

[161] Pushmeet Kohli and Philip H. S. Torr. 2010. Dynamic Graph Cuts and Their Applications in Computer Vision. In *Computer Vision: Detection, Recognition and Reconstruction*, Roberto Cipolla, Sebastiano Battiato, and Giovanni Maria Farinella (Eds.). Studies in Computational Intelligence, Vol. 285. Springer, 51–108. https://doi.org/10.1007/978-3-642-12848-6_3

[162] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi. 2015. Scalable Online Betweenness Centrality in Evolving Graphs. *IEEE Trans. Knowl. Data Eng.* 27, 9 (2015), 2494–2506. https://doi.org/10.1109/TKDE.2015.2419666

[163] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi. 2016. Scalable online betweenness centrality in evolving graphs. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society, 1580–1581. https://doi.org/10.1109/ICDE.2016.7498421

[164] Ioannis Krommidas and Christos D. Zaroliagis. 2008. An experimental study of algorithms for fully dynamic transitive closure. *ACM J. Exp. Algorithmics* 12 (2008), 1.6:1–1.6:22. https://doi.org/10.1145/1227161.1370597

[165] S. Kumar and P. Gupta. 2003. An Incremental Algorithm for the Maximum Flow Problem. *J. Math. Model. Algorithms* 2, 1 (2003), 1–16. https://doi.org/10.1023/A:1023607406540

[166] Min-Joong Lee, Sunghee Choi, and Chin-Wan Chung. 2016. Efficient algorithms for updating betweenness centrality in fully dynamic graphs. *Inf. Sci.* 326 (2016), 278–296. https://doi.org/10.1016/j.ins.2015.07.053

[167] Min-Joong Lee, Jungmin Lee, Jaimie Yejean Park, Ryan Hyun Choi, and Chin-Wan Chung. 2012. QUBE: a quick algorithm for updating betweenness centrality. In *Proc. of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab (Eds.). ACM, 351–360. https://doi.org/10.1145/2187836.2187884

[168] Rong-Hua Li, Jeffrey Xu Yu, and Rui Mao. 2014. Efficient Core Maintenance in Large Dynamic Graphs. *IEEE Trans. Knowl. Data Eng.* 26, 10 (2014), 2453–2465. https://doi.org/10.1109/TKDE.2013.158

[169] Yongsub Lim, Minsoo Jung, and U Kang. 2018. Memory-Efficient and Accurate Sampling for Counting Local Triangles in Graph Streams: From Simple to Multigraphs. *ACM Trans. Knowl. Discov. Data* 12, 1 (2018), 4:1–4:28. https://doi.org/10.1145/3022186

[170] Yongsub Lim and U Kang. 2015. MASCOT: Memory-efficient and Accurate Sampling for Counting Local Triangles in Graph Streams. In *Proc. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, Longbing Cao, Chengqi Zhang, Thorsten Joachims, Geoffrey I. Webb, Dragos D. Margineantu, and Graham Williams (Eds.). ACM, 685–694. https://doi.org/10.1145/2783258.2783285

[171] Shangqi Lu and Yufei Tao. 2021. Towards Optimal Dynamic Indexes for Approximate (and Exact) Triangle Counting. In *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus (LIPIcs, Vol. 186)*, Ke Yi and Zhewei Wei (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:23. https://doi.org/10.4230/LIPIcs.ICDT.2021.6

[172] Amgad Madkour, Walid G Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. 2017. A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044* (2017).

[173] Devavret Makkar, David A. Bader, and Oded Green. 2017. Exact and Parallel Triangle Counting in Dynamic Graphs. In *24th IEEE International Conference on High Performance Computing, HiPC 2017, Jaipur, India, December 18-21, 2017*. IEEE Computer Society, 2–12. https://doi.org/10.1109/HiPC.2017.00011

[174] Alberto Marchetti-Spaccamela, Umberto Nanni, and Hans Rohnert. 1996. Maintaining a Topological Order Under Edge Insertions. *Inf. Process. Lett.* 59, 1 (1996), 53–58. https://doi.org/10.1016/0020-0190(96)00075-0

[175] Henning Meyerhenke. 2009. Dynamic load balancing for parallel numerical simulations based on repartitioning with disturbed diffusion. In *15th International Conference on Parallel and Distributed Systems*. IEEE, 150–157. https://doi.org/10.1109/ICPADS.2009.114

[176] Henning Meyerhenke and Joachim Gehweiler. 2010. On Dynamic Graph Partitioning and Graph Clustering using Diffusion. In *Algorithm Engineering, 27.06. - 02.07.2010 (Dagstuhl Seminar Proc., Vol. 10261)*, Giuseppe F. Italiano, David S. Johnson, Petra Mutzel, and Peter Sanders (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany. http://drops.dagstuhl.de/opus/volltexte/2010/2798/

[177] Kurt T Miller and Tina Eliassi-Rad. 2009. Continuous time group discovery in dynamic graphs. In *Notes of the 2009 NIPS Workshop on Analyzing Networks and Learning with Graphs, Whistler, BC, Canada.*

[178] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. 1994. Complexity Models for Incremental Computation. *Theor. Comput. Sci.* 130, 1 (1994), 203–236. https://doi.org/10.1016/0304-3975(94)90159-7

[179] Daniele Miorandi and Francesco De Pellegrini. 2010. K-shell decomposition for dynamic complex networks, See [19], 488–496. http://ieeexplore.ieee.org/document/5520231/

[180] Sudip Misra and B. John Oommen. 2005. Dynamic algorithms for the shortest path routing problem: Learning automata-based solutions. *IEEE Trans. Syst. Man Cybern. Part B* 35, 6 (2005), 1179–1192. https://doi.org/10.1109/TSMCB.2005.850180

[181] Kingshuk Mukherjee, Md Mahmudul Hasan, Christina Boucher, and Tamer Kahveci. 2018. Counting motifs in dynamic networks. *BMC Syst. Biol.* 12, 1 (2018), 1–12. https://doi.org/10.1186/s12918-018-0533-6

[182] Kengo Nakamura and Kunihiko Sadakane. 2019. Space-Efficient Fully Dynamic DFS in Undirected Graphs. *Algorithms* 12, 3 (2019), 52. https://doi.org/10.3390/a12030052

[183] Paolo Narváez, Kai-Yeung Siu, and Hong-Yi Tzeng. 2000. New dynamic algorithms for shortest path tree computation. *IEEE/ACM Trans. Netw.* 8, 6 (2000), 734–746. https://doi.org/10.1109/90.893870

[184] Paolo Narváez, Kai-Yeung Siu, and Hong-Yi Tzeng. 2001. New dynamic SPT algorithm based on a ball-and-string model. *IEEE/ACM Trans. Netw.* 9, 6 (2001), 706–718. https://doi.org/10.1109/90.974525

[185] Meghana Nasre, Matteo Pontecorvi, and Vijaya Ramachandran. 2014. Betweenness Centrality - Incremental and Faster. In *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8635)*, Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik (Eds.). Springer, 577–588. https://doi.org/10.1007/978-3-662-44465-8_49

[186] Eisha Nathan and David A. Bader. 2017. Approximating Personalized Katz Centrality in Dynamic Graphs. In *Parallel Processing and Applied Mathematics - 12th International Conference, PPAM 2017, Lublin, Poland, September 10-13, 2017, Revised Selected Papers, Part I (Lecture Notes in Computer Science, Vol. 10777)*, Roman Wyrzykowski, Jack J. Dongarra, Ewa Deelman, and Konrad Karczewski (Eds.). Springer, 290–302. https://doi.org/10.1007/978-3-319-78024-5_26

[187] Eisha Nathan and David A. Bader. 2017. A Dynamic Algorithm for Updating Katz Centrality in Graphs, See [82], 149–154. https://doi.org/10.1145/3110025.3110034

[188] Ofer Neiman and Shay Solomon. 2016. Simple Deterministic Algorithms for Fully Dynamic Maximal Matching. *ACM Trans. Algorithms* 12, 1 (2016), 7:1–7:15. https://doi.org/10.1145/2700206

[189] M. EJ Newman and M. Girvan. 2004. Finding and Evaluating Community Structure in Networks. *Physical review E* 69, 2 (2004), 026113. https://doi.org/10.1103/PhysRevE.69.026113

[190] Daniel Nicoara, Shahin Kamali, Khuzaima Daudjee, and Lei Chen. 2015. Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases.. In *EDBT*. 25–36. https://doi.org/10.5441/002/edbt.2015.04

[191] Krzysztof Onak and Ronitt Rubinfeld. 2010. Maintaining a large matching and a small vertex cover. In *STOC*. 457–464. https://doi.org/10.1145/1806689.1806753

[192] James B. Orlin. 2013. Max flows in $O(nm)$ time, or better. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM, 765–774. https://doi.org/10.1145/2488608.2488705

[193] Mihai Patrascu. 2010. Towards polynomial lower bounds for dynamic problems. In *Proc. of the 42nd ACM Symposium on Theory of Computing, STOC*. ACM, 603–610. https://doi.org/10.1145/1806689.1806772

[194] Mihai Patrascu and Erik D. Demaine. 2006. Logarithmic Lower Bounds in the Cell-Probe Model. *SIAM J. Comput.* 35, 4 (2006), 932–963. https://doi.org/10.1137/S0097539705447256

[195] A. Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and Sampling Triangles from a Graph Stream. *Proc. VLDB Endow.* 6, 14 (2013), 1870–1881. https://doi.org/10.14778/2556549.2556569

[196] D. Pearce and P. Kelly. 2010. A batch algorithm for maintaining a topological order. In *ACSC*. https://doi.org/10.5555/1862199.1862208

[197] David J. Pearce and Paul H. J. Kelly. 2004. A Dynamic Algorithm for Topologically Sorting Directed Acyclic Graphs. In *Experimental and Efficient Algorithms, Third International Workshop, WEA 2004, Angra dos Reis, Brazil, May 25-28, 2004, Proc. (Lecture Notes in Computer*

Science, Vol. 3059), Celso C. Ribeiro and Simone L. Martins (Eds.). Springer, 383–398. https://doi.org/10.1007/978-3-540-24838-5_29

[198] David J. Pearce and Paul H. J. Kelly. 2006. A dynamic topological sort algorithm for directed acyclic graphs. *ACM J. Exp. Algorithmics* 11 (2006). https://doi.org/10.1145/1187436.1210590

[199] David J. Pearce, Paul H. J. Kelly, and Chris Hankin. 2004. Online Cycle Detection and Difference Propagation: Applications to Pointer Analysis. *Softw. Qual. J.* 12, 4 (2004), 311–337. https://doi.org/10.1023/B:SQJO.0000039791.93071.a2

[200] Reynaldo Gil Pons. 2018. Space Efficient Incremental Betweenness Algorithm for Directed Graphs. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications - 23rd Iberoamerican Congress, CIARP 2018, Madrid, Spain, November 19-22, 2018, Proc. (Lecture Notes in Computer Science, Vol. 11401)*, Rubén Vera-Rodríguez, Julian Fiérrez, and Aythami Morales (Eds.). Springer, 262–270. https://doi.org/10.1007/978-3-030-13469-3_31

[201] Matteo Pontecorvi and Vijaya Ramachandran. 2015. Fully Dynamic Betweenness Centrality. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9472)*, Khaled M. Elbassioni and Kazuhisa Makino (Eds.). Springer, 331–342. https://doi.org/10.1007/978-3-662-48971-0_29

[202] K. (Lynn) Putman, Hanjo D. Boekhout, and Frank W. Takes. 2019. Fast incremental computation of harmonic closeness centrality in directed weighted networks. In *ASONAM '19: International Conference on Advances in Social Networks Analysis and Mining, Vancouver, British Columbia, Canada, 27-30 August, 2019*, Francesca Spezzano, Wei Chen, and Xiaokui Xiao (Eds.). ACM, 1018–1025. https://doi.org/10.1145/3341161.3344829

[203] Evangelia Pyrga, Frank Schulz, Dorothea Wagner, and Christos D. Zaroliagis. 2007. Efficient models for timetable information in public transportation systems. *ACM J. Exp. Algorithmics* 12 (2007), 2.4:1–2.4:39. https://doi.org/10.1145/1227161.1227166

[204] G Ramalingam and Thomas Reps. 1991. *On the computational complexity of incremental algorithms.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

[205] G. Ramalingam and Thomas W. Reps. 1996. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *J. Algorithms* 21, 2 (1996), 267–305. https://doi.org/10.1006/jagm.1996.0046

[206] G. Ramalingam and Thomas W. Reps. 1996. On the Computational Complexity of Dynamic Graph Problems. *Theor. Comput. Sci.* 158, 1&2 (1996), 233–277. https://doi.org/10.1016/0304-3975(95)00079-8

[207] Celso C. Ribeiro and Rodrigo F. Toso. 2007. Experimental Analysis of Algorithms for Updating Minimum Spanning Trees on Graphs Subject to Changes on Edge Weights, See [77], 393–405. https://doi.org/10.1007/978-3-540-72845-0_30

[208] Matteo Riondato and Evgenios M. Kornaropoulos. 2014. Fast approximation of betweenness centrality through sampling. In *Seventh ACM International Conference on Web Search and Data Mining, WSDM 2014, New York, NY, USA, February 24-28, 2014*, Ben Carterette, Fernando Diaz, Carlos Castillo, and Donald Metzler (Eds.). ACM, 413–422. https://doi.org/10.1145/2556195.2556224

[209] Liam Roditty. 2008. A faster and simpler fully dynamic transitive closure. *ACM Trans. Algorithms* 4, 1 (2008), 6:1–6:16. https://doi.org/10.1145/1328911.1328917

[210] Liam Roditty and Uri Zwick. 2011. On Dynamic Shortest Paths Problems. *Algorithmica* 61, 2 (2011), 389–401. https://doi.org/10.1007/s00453-010-9401-5

[211] Liam Roditty and Uri Zwick. 2012. Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. *SIAM J. Comput.* 41, 3 (2012), 670–683. https://doi.org/10.1137/090776573

[212] Liam Roditty and Uri Zwick. 2016. A Fully Dynamic Reachability Algorithm for Directed Graphs with an Almost Linear Update Time. *SIAM J. Comput.* 45, 3 (2016), 712–733. https://doi.org/10.1137/13093618X

[213] Mohammad Roghani, Amin Saberi, and David Wajc. 2022. Beating the Folklore Algorithm for Dynamic Matching. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA (LIPIcs, Vol. 215)*, Mark Braverman (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 111:1–111:23. https://doi.org/10.4230/LIPIcs.ITCS.2022.111

[214] Giulio Rossetti and Rémy Cazabet. 2018. Community Discovery in Dynamic Networks: A Survey. *ACM Comput. Surv.* 51, 2 (2018), 35:1–35:37. https://doi.org/10.1145/3172867

[215] Tiberiu Rotaru and Hans-Heinrich Nägeli. 2004. Dynamic load balancing by diffusion in heterogeneous systems. *J. Parallel and Distrib. Comput.* 64, 4 (2004), 481–497. https://doi.org/10.1016/j.jpdc.2004.02.001

[216] Chayma Sakouhi, Sabeur Aridhi, Alessio Guerrieri, Salma Sassi, and Alberto Montresor. 2016. DynamicDFEP: A Distributed Edge Partitioning Approach for Large Dynamic Graphs. In *Proc. of the 20th International Database Engineering & Applications Symposium, IDEAS 2016, Montreal, QC, Canada, July 11-13, 2016*, Evan Desai, Bipin C. Desai, Motomichi Toyama, and Jorge Bernardino (Eds.). ACM, 142–147. https://doi.org/10.1145/2938503.2938506

[217] Piotr Sankowski. 2004. 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, Proc.* IEEE Computer Society, 509–517. https://doi.org/10.1109/FOCS.2004.25

[218] Piotr Sankowski. 2005. Subquadratic Algorithm for Dynamic Shortest Distances. In *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3595)*, Lusheng Wang (Ed.). Springer, 461–470. https://doi.org/10.1007/11533719_47

[219] Piotr Sankowski. 2007. Faster dynamic matchings and vertex connectivity. In *SODA*. 118–126. https://doi.org/10.1145/1283383.1283397

[220] Ahmet Erdem Sariyüce, Bugra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2016. Incremental k-core decomposition: algorithms and evaluation. *VLDB J.* 25, 3 (2016), 425–447. https://doi.org/10.1007/s00778-016-0423-8

[221] Ahmet Erdem Sariyüce, Kamer Kaya, Erik Saule, and Ümit V. Çatalyürek. 2013. Incremental algorithms for closeness centrality. In *Proc. of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, Xiaohua Hu, Tsau Young Lin, Vijay V. Raghavan, Benjamin W. Wah, Ricardo Baeza-Yates, Geoffrey C. Fox, Cyrus Shahabi, Matthew Smith, Qiang Yang, Rayid Ghani, Wei Fan, Ronny Lempel, and Raghunath Nambiar (Eds.). IEEE Computer Society, 487–492. https://doi.org/10.1109/BigData.2013.6691611

[222] Ahmet Erdem Sariyüce, Erik Saule, Kamer Kaya, and Ümit V. Çatalyürek. 2013. STREAMER: A distributed framework for incremental closeness centrality computation. In *2013 IEEE International Conference on Cluster Computing, CLUSTER 2013, Indianapolis, IN, USA, September 23-27, 2013*. IEEE Computer Society, 1–8. https://doi.org/10.1109/CLUSTER.2013.6702680

[223] Ahmet Erdem Sariyüce, Erik Saule, Kamer Kaya, and Ümit V. Çatalyürek. 2015. Incremental closeness centrality in distributed memory. *Parallel Comput.* 47 (2015), 3–18. https://doi.org/10.1016/j.parco.2015.01.003

[224] Benjamin Schiller, Sven Jager, Kay Hamacher, and Thorsten Strufe. 2015. StreaM - A Stream-Based Algorithm for Counting Motifs in Dynamic Graphs. In *Algorithms for Computational Biology - Second International Conference, AlCoB 2015, Mexico City, Mexico, August 4-5, 2015, Proc. (Lecture Notes in Computer Science, Vol. 9199)*, Adrian-Horia Dediu, Francisco Hernández Quiroz, Carlos Martín-Vide, and David A. Rosenblueth (Eds.). Springer, 53–67. https://doi.org/10.1007/978-3-319-21233-3_5

[225] Kirk Schloegel, George Karypis, and Vipin Kumar. 1997. Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes. *J. Parallel Distributed Comput.* 47, 2 (1997), 109–124. https://doi.org/10.1006/jpdc.1997.1410

[226] Kirk Schloegel, George Karypis, and Vipin Kumar. 2000. A Unified Algorithm for Load-balancing Adaptive Scientific Simulations. In *Proc. Supercomputing 2000, November 4-10, 2000, Dallas, Texas, USA. IEEE Computer Society, CD-ROM*, Jed Donnelley (Ed.). IEEE Computer Society, 59. https://doi.org/10.1109/SC.2000.10035

[227] Kirk Schloegel, George Karypis, and Vipin Kumar. 2002. Parallel static and dynamic multi-constraint graph partitioning. *Concurr. Comput. Pract. Exp.* 14, 3 (2002), 219–240. https://doi.org/10.1002/cpe.605

[228] Dominik Schultes and Peter Sanders. 2007. Dynamic Highway-Node Routing, See [77], 66–79. https://doi.org/10.1007/978-3-540-72845-0_6

[229] Dipanjan Sengupta, Narayanan Sundaram, Xia Zhu, Theodore L Willke, Jeffrey Young, Matthew Wolf, and Karsten Schwan. 2016. Graphin: An online high performance incremental graph processing framework. In *European Conference on Parallel Processing*. Springer, 319–333. https://doi.org/10.1007/978-3-319-43659-3_24

[230] Neha Sengupta, Michael Hamann, and Dorothea Wagner. 2017. Benchmark Generator for Dynamic Overlapping Communities in Networks. In *2017 IEEE International Conference on Data Mining, ICDM 2017, New Orleans, LA, USA, November 18-21, 2017*, Vijay Raghavan, Srinivas Aluru, George Karypis, Lucio Miele, and Xindong Wu (Eds.). IEEE Computer Society, 415–424. https://doi.org/10.1109/ICDM.2017.51

[231] Zhenzhen Shao, Na Guo, Yu Gu, Zhigang Wang, Fangfang Li, and Ge Yu. 2020. Efficient Closeness Centrality Computation for Dynamic Graphs. In *Database Systems for Advanced Applications - 25th International Conference, DASFAA 2020, Jeju, South Korea, September 24-27, 2020, Proc., Part II (Lecture Notes in Computer Science, Vol. 12113)*, Yunmook Nah, Bin Cui, Sang-Won Lee, Jeffrey Xu Yu, Yang-Sae Moon, and Steven Euijong Whang (Eds.). Springer, 534–550. https://doi.org/10.1007/978-3-030-59416-9_32

[232] Y. Shiloach and S. Even. 1981. An On-Line Edge-Deletion Problem. *J. ACM* 28, 1 (1981), 1–4. https://doi.org/10.1145/322234.322235

[233] Dhirendra Singh and Nilay Khare. 2019. Parallel batch dynamic single source shortest path algorithm and its implementation on GPU based machine. *Int. Arab J. Inf. Technol.* 16, 2 (2019), 217–225. http://iajit.org/index.php?option=com_content&task=blogcategory&id=137&Itemid=469

[234] Daniel Dominic Sleator and Robert Endre Tarjan. 1981. A Data Structure for Dynamic Trees. In *Proc. of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*. ACM, 114–122. https://doi.org/10.1145/800076.802464

[235] Shay Solomon. 2016. Fully Dynamic Maximal Matching in Constant Update Time. In *57th Symposium on Foundations of Computer Science FOCS*. 325–334. https://doi.org/10.1109/FOCS.2016.43

[236] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. TRIÈST: Counting Local and Global Triangles in Fully Dynamic Streams with Fixed Memory Size. *ACM Trans. Knowl. Discov. Data* 11, 4 (2017), 43:1–43:50. https://doi.org/10.1145/3059194

[237] Daniel Stubbs and Virginia Vassilevska Williams. 2017. Metatheorems for Dynamic Weighted Matching. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA (LIPIcs, Vol. 67)*, Christos H. Papadimitriou (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 58:1–58:14. https://doi.org/10.4230/LIPIcs.ITCS.2017.58

[238] Bintao Sun, T.-H. Hubert Chan, and Mauro Sozio. 2020. Fully Dynamic Approximate K-Core Decomposition in Hypergraphs. *ACM Trans. Knowl. Discov. Data* 14, 4, Article 39 (May 2020), 21 pages. https://doi.org/10.1145/3385416

[239] Robert Endre Tarjan and Renato Fonseca F. Werneck. 2009. Dynamic trees in practice. *ACM J. Exp. Algorithmics* 14 (2009). https://doi.org/10.1145/1498698.1594231

[240] Mikkel Thorup. 2004. Fully-Dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles. In *Algorithm Theory - SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Humlebaek, Denmark, July 8-10, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3111)*, Torben Hagerup and Jyrki Katajainen (Eds.). Springer, 384–396. https://doi.org/10.1007/978-3-540-27810-8_33

[241] Mikkel Thorup. 2005. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, Harold N. Gabow and Ronald Fagin (Eds.). ACM, 112–119. https://doi.org/10.1145/1060590.1060607

[242] Mikkel Thorup. 2007. Fully-Dynamic Min-Cut. *Comb.* 27, 1 (2007), 91–127. https://doi.org/10.1007/s00493-007-0045-2

[243] Mikkel Thorup and David R. Karger. 2000. Dynamic Graph Algorithms with Applications. In *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings (Lecture Notes in Computer Science, Vol. 1851)*, Magnús M. Halldórsson (Ed.). Springer, 1–9. https://doi.org/10.1007/3-540-44985-X_1

[244] Thomas Tseng, Laxman Dhulipala, and Guy E. Blelloch. 2019. Batch-Parallel Euler Tour Trees. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments, ALENEX 2019, San Diego, CA, USA, January 7-8, 2019*, Stephen G. Kobourov and Henning Meyerhenke (Eds.). SIAM, 92–106. https://doi.org/10.1137/1.9781611975499.8

[245] Jan van den Brand and Danupon Nanongkai. 2019. Dynamic Approximate Shortest Paths and Beyond: Subquadratic and Worst-Case Update Time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 436–455. https://doi.org/10.1109/FOCS.2019.00035

[246] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. 2019. Dynamic Matrix Inverse: Improved Algorithms and Matching Conditional Lower Bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 456–480. https://doi.org/10.1109/FOCS.2019.00036

[247] Alexander van der Grinten, Elisabetta Bergamini, Oded Green, David A. Bader, and Henning Meyerhenke. 2018. Scalable Katz Ranking Computation in Large Static and Dynamic Graphs. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland (LIPIcs, Vol. 112)*, Yossi Azar, Hannah Bast, and Grzegorz Herman (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:14. https://doi.org/10.4230/LIPIcs.ESA.2018.42

[248] Luis M. Vaquero, Félix Cuadrado, Dionysios Logothetis, and Claudio Martella. 2013. xDGP: A Dynamic Graph Processing System with Adaptive Partitioning. *CoRR* abs/1309.1049 (2013). arXiv:1309.1049 http://arxiv.org/abs/1309.1049

[249] Luis M. Vaquero, Félix Cuadrado, Dionysios Logothetis, and Claudio Martella. 2014. Adaptive Partitioning for Large-Scale Dynamic Graphs. In *IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, Madrid, Spain, June 30 - July 3, 2014*. IEEE Computer Society, 144–153. https://doi.org/10.1109/ICDCS.2014.23

[250] Tanmay Verma and Dhruv Batra. 2012. MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems. In *British Machine Vision Conference, BMVC 2012, Surrey, UK, September 3-7, 2012*, Richard Bowden, John P. Collomosse, and Krystian Mikolajczyk (Eds.). BMVA Press, 1–12. https://doi.org/10.5244/C.26.61

[251] Dorothea Wagner, Thomas Willhalm, and Christos D. Zaroliagis. 2005. Geometric containers for efficient shortest-path computation. *ACM J. Exp. Algorithmics* 10 (2005). https://doi.org/10.1145/1064546.1103378

[252] Chris Walshaw, Mark Cross, and Martin G. Everett. 1997. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *J. Parallel Distributed Comput.* 47, 2 (1997), 102–108. https://doi.org/10.1006/jpdc.1997.1407

[253] Na Wang, Dongxiao Yu, Hai Jin, Chen Qian, Xia Xie, and Qiang-Sheng Hua. 2017. Parallel Algorithm for Core Maintenance in Dynamic Graphs. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*, Kisung Lee and Ling Liu (Eds.). IEEE Computer Society, 2366–2371. https://doi.org/10.1109/ICDCS.2017.288

[254] Stefan Weigert, Matti Hiltunen, and Christof Fetzer. 2011. Mining large distributed log data in near real time. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*. 1–8. https://doi.org/10.1145/2038633.2038638

[255] Martin Winter, Daniel Mlakar, Rhaleb Zayer, Hans-Peter Seidel, and Markus Steinberger. 2018. faimGraph: high performance management of fully-dynamic graphs under tight memory constraints on the GPU. In *Proc. of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*. IEEE / ACM, 60:1–60:13. http://dl.acm.org/citation.cfm?id=3291736

[256] Ning Xu, Lei Chen, and Bin Cui. 2014. LogGP: a log-based dynamic graph partitioning method. *Proc. of the VLDB Endowment* 7, 14 (2014), 1917–1928. https://doi.org/10.14778/2733085.2733097

[257] Bohua Yang, Dong Wen, Lu Qin, Ying Zhang, Xubo Wang, and Xuemin Lin. 2019. Fully Dynamic Depth-First Search in Directed Graphs. *Proc. VLDB Endow.* 13, 2 (2019), 142–154. https://doi.org/10.14778/3364324.3364329

[258] Chia-Chen Yen, Mi-Yen Yeh, and Ming-Syan Chen. 2013. An Efficient Approach to Updating Closeness Centrality and Average Path Length in Dynamic Networks. In *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu (Eds.). IEEE Computer Society, 867–876. https://doi.org/10.1109/ICDM.2013.135

[259] Anita Zakrzewska and David A. Bader. 2016. Fast Incremental Community Detection on Dynamic Graphs. In *Parallel Processing and Applied Mathematics*, Roman Wyrzykowski, Ewa Deelman, Jack Dongarra, Konrad Karczewski, Jacek Kitowski, and Kazimierz Wiatr (Eds.). Springer International Publishing, Cham, 207–217. https://doi.org/10.1007/978-3-319-32149-3_20

[260] Christos D Zaroliagis. 2002. Implementations and experimental studies of dynamic graph algorithms. In *Experimental algorithmics*. Springer, 229–278. https://doi.org/10.1007/3-540-36383-1_11

[261] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A Fast Order-Based Approach for Core Maintenance. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 337–348. https://doi.org/10.1109/ICDE.2017.93

[262] Weiguo Zheng, Chengzhi Piao, Hong Cheng, and Jeffrey Xu Yu. 2019. Computing a Near-Maximum Independent Set in Dynamic Graphs. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 76–87. https://doi.org/10.1109/ICDE.2019.00016

[263] Weiguo Zheng, Qichen Wang, Jeffrey Xu Yu, Hong Cheng, and Lei Zou. 2018. Efficient Computation of a Near-Maximum Independent Set over Evolving Graphs. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 869–880. https://doi.org/10.1109/ICDE.2018.00083

[264] Lei Zhu, Shaoning Pang, Abdolhossein Sarrafzadeh, Tao Ban, and Daisuke Inoue. 2016. Incremental and Decremental Max-Flow for Online Semi-Supervised Learning. *IEEE Trans. Knowl. Data Eng.* 28, 8 (2016), 2115–2127. https://doi.org/10.1109/TKDE.2016.2550042

[265] Di Zhuang, Morris J Chang, and Mingchen Li. 2019. DynaMo: Dynamic community detection by incrementally maximizing modularity. *IEEE Transactions on Knowledge and Data Engineering* (2019). https://doi.org/10.1109/TKDE.2019.2951419

[266] David Zuckerman (Ed.). 2019. *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. IEEE Computer Society. https://ieeexplore.ieee.org/xpl/conhome/8936052/proceeding