Optimal Decremental Connectivity in Non-Sparse Graphs

Anders Aamand
MIT, Cambridge, MA, USA

Adam Karczmarz
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

Jakub Łącki
Google Research, New York, NY,USA

Nikos Parotsidis
Google Research, Zürich, Switzerland

Peter M. R. Rasmussen
BARC, University of Copenhagen, Denmark

Mikkel Thorup
BARC, University of Copenhagen, Denmark

Abstract

We present a dynamic algorithm for maintaining the connected and 2-edge-connected components in an undirected graph subject to edge deletions. The algorithm is Monte-Carlo randomized and processes any sequence of edge deletions in $O(m + n \operatorname{poly} \log n)$ total time. Interspersed with the deletions, it can answer queries whether any two given vertices currently belong to the same (2-edge-)connected component in constant time. Our result is based on a general Monte-Carlo randomized reduction from decremental c-edge-connectivity to a variant of fully-dynamic c-edge-connectivity on a sparse graph.

For non-sparse graphs with $\Omega(n \operatorname{poly} \log n)$ edges, our connectivity and 2-edge-connectivity algorithms handle all deletions in optimal linear total time, using existing algorithms for the respective fully-dynamic problems. This improves upon an $O(m \log(n^2/m) + n \operatorname{poly} \log n)$-time algorithm of Thorup [J.Alg. 1999], which runs in linear time only for graphs with $\Omega(n^2)$ edges.

Our constant amortized cost for edge deletions in decremental connectivity in non-sparse graphs should be contrasted with an $\Omega(\log n/ \log \log n)$ worst-case time lower bound in the decremental setting [Alstrup, Husfeldt, and Rauhe FOCS'98] as well as an $\Omega(\log n)$ amortized time lower-bound in the fully-dynamic setting [Patrascu and Demaine STOC'04].

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Mathematics of computing → Paths and connectivity problems; Mathematics of computing → Graph algorithms

Keywords and phrases decremental connectivity, dynamic connectivity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.6

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2111.09376`

Funding *Anders Aamand*: Supported by Thorup's VILLUM Investigator Grant 16582 and by a DFF-International Postdoc Grant 0164-00022B from the Independent Research Fund Denmark.
Adam Karczmarz: Partially supported by the ERC CoG grant TUgbOAT no 772346.
Peter M. R. Rasmussen: Supported by Thorup's VILLUM Investigator Grant 16582.
Mikkel Thorup: Supported by Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

1 Introduction

In this paper, we present Monte Carlo randomized decremental dynamic algorithms for maintaining the connected and 2-edge-connected components in an undirected graph subject to edge deletions. Starting from a graph with n vertices and m edges, the algorithm can process any sequence of edge deletions in $O(m + n \operatorname{polylog} n)$ total time while answering queries whether a pair of vertices is currently in the same (2-edge-)connected component. Each query is answered in constant time. The algorithm for decremental 2-edge-connectivity additionally reports all bridges as they appear.

Putting our results in perspective, we say a graph is *non-sparse* if it has $n \log^{\omega(1)} n$ edges. Large areas of algorithmic research are devoted to non-sparse graphs, e.g., the generic goal of sparsifying graphs to $O(n \operatorname{polylog} n)$ edges [6], or semi-streaming algorithms that aim to sketch graphs using $O(n \operatorname{polylog} n)$ space [11]. Our result states that for dynamic connectivity and 2-edge-connectivity, we can get down to amortized constant time per edge deletion if the initial input graph is non-sparse. Prior to this work, such a result was only known in the case where the initial input graph is very dense with $\Omega(n^2)$ edges, and in the case of some special classes of sparse graphs.

Achieving constant update and query time is generally the ideal target in data structures. What makes amortized constant time for decremental connectivity particularly interesting is that the most closely related problems have near-logarithmic cell-probe lower bounds. This concerns the problem of getting worst-case time bounds or getting a fully-dynamic algorithm (supporting both insertions and deletions of edges). The decremental setting and the fact that we allow for amortization is therefore just enough assumptions to barely push us into the world of constant update and query time (removing any of these assumptions, the polylogarithmic lower bounds would kick in) and as such, our result draws a fine line between the possible and the impossible. We shall discuss this further with precise references in Section 1.1. It is worth noting that for some dynamic graph problems related to maintaining (approximate) maximum matchings and colorings, constant amortized update bounds have been shown, see, e.g., [7, 20, 21, 37].

Our algorithms are Monte Carlo randomized and answer all queries correctly with high probability[1]. We note that since the correct answer to each query is *uniquely* determined from the input, the algorithms work against adaptive adversaries, that is, each deleted edge may depend on previous answers to queries and (in the case of decremental 2-edge-connectivity) on the alleged bridges reported by the algorithm[2]

Furthermore, our algorithms offer a *self-check* capability. At the end, after all updates and queries have been processed online, each algorithm can deterministically check if it might have made a mistake. If the self-check passes, it is *guaranteed* that no incorrect answer was given. Otherwise, the algorithm *may have* made a mistake. Given the self-check is deterministic, the probability that the self-check passes following the execution of the algorithm only

[1] We define high probability as probability $1 - O(n^{-\gamma})$ for any given γ.

[2] To be precise, with unique correct answers, for any adaptive adversary $\mathcal{A}_{\text{ad}}$, there is a non-adaptive adversary $\mathcal{A}_{\text{non-ad}}$ which provides the same sequence of edge deletions up to the first point in time where the algorithm potentially reports an incorrect answer. $\mathcal{A}_{\text{non-ad}}$ is simply defined to provide the same edge-deletions as $\mathcal{A}_{\text{ad}}$ would conditioned on it receiving the *unique* correct answers to every query. Intuitively, the adaptivity of the adversary only becomes relevant once the algorithm has already made a mistake. Illustrating the issue of non-uniqueness in the case of decremental connectivity, suppose we augmented our algorithm to report a (non-unique) *path* between queried pairs of vertices in the same component. The choice of path could reveal information about the random bits employed by our algorithm and this could be very problematic if $\mathcal{A}_{\text{ad}}$ decided to delete the reported path edges.

depends on the correctness of the algorithm execution. However, as we show in the following, the self-check passes with high probability. This feature implies that we can obtain Las Vegas algorithms for certain *non-dynamic* problems whose solutions employ decremental (2-edge-)connectivity algorithms as subroutines: we simply repeat trying to solve the static problem from scratch, each time with new random bits, until the final self-check is passed. With high probability, we are done already after the first round. A nice concrete example is the algorithm of Gabow, Kaplan, and Tarjan [15] for the static problem of deciding if a graph has a unique perfect matching. The algorithm uses a decremental 2-edge-connectivity algorithm as a subroutine. With our decremental 2-edge-connectivity algorithm, repeating until the self-check is passed, we obtain a Las Vegas algorithm for the unique perfect matching problem that is always correct, and which terminates in $O(m + n \operatorname{polylog} n)$ time with high probability.

The tradition of looking for linear time algorithms for non-sparse graphs goes back at least to Fibonacci heaps, which can be used for solving single source shortest paths in $O(m + n \log n)$ time [14]. Our results show that another fundamental graph problem can be solved in linear time in the non-sparse case.

The previous best time bounds for the decremental connectivity and 2-edge-connectivity problems were provided by Thorup [39]. His algorithms run in $O(m \log(n^2/m) + n \operatorname{polylog} n)$ total time. This is amortized constant time per edge deletion only for very dense graphs starting with $\Omega(n^2)$ edges. For graphs with $O(n^{1.99})$ edges, this is $O(\log n)$ amortized time per edge deletion.

Both our algorithm and the previous one by Thorup are based on a general reduction from decremental c-edge-connectivity to fully-dynamic c-edge-connectivity on a sparse c-certificate graph with $\tilde{O}(cn)$ updates.

The contribution of this paper is a new type of sparse c-certificate that is much more efficient to maintain during edge deletions, reducing amortized time per deletion from $O(\log(n^2/m))$ to the optimal $O(1)$. We hope that this new sparse c-certificate will inspire other applications. We shall discuss it further in Section 2.

It should be noted that [39] used Las Vegas randomization, that is, correctness was guaranteed, but the running time bound only held with high probability. Our algorithms are Monte Carlo randomized, but offer the final self-check. Another difference is that our new algorithms need only a polylogarithmic number of random bits, whereas the ones from [39] used $\Theta(m)$ random bits.

We will now give a more detailed discussion of our results in the context of related work.

1.1 Connectivity

Dynamic connectivity is the most fundamental dynamic graph problem. The fully dynamic version has been extensively studied [8, 9, 12, 22, 23, 26, 28, 31, 34, 35, 36, 40, 42, 43] from both the lower and upper bound perspective, even though close to optimal amortized update bounds have been known since the 90s [22, 23, 40]. Currently, the best known amortized update time bounds are $O(\log^2 n / \log\log n)$ deterministic [43] and $O(\log n \cdot (\log\log n)^2)$ expected time [26].

Note that Thorup's $O(\log(n^2/m))$ bound for decremental connectivity is essentially only a $(\log\log n)^2$ factor better than the latter of these bounds for fully-dynamic connectivity, while our new bound brings the decremental cost down to a constant (for non-sparse graphs). Getting down to a constant is particularly interesting when we compare with related lower bounds as discussed below.

Connectivity Lower Bounds. Our result implies that decremental connectivity is provably easier than fully-dynamic connectivity for a wide range of graph densities. Specifically, let t_u be the update time of a fully dynamic connectivity algorithm and let t_q be its query time. Pătraşcu and Demaine [35] showed a lower bound of $\Omega(\log n)$ on $\max(t_u, t_q)$ in the cell-probe model. Pătraşcu and Thorup [36] also showed that $t_u = o(\log n)$ implies $t_q = \Omega(n^{1-o(1)})$. These lower bounds hold for all graph densities and allow for both amortization and randomization. As a result, no fully-dynamic connectivity algorithm can answer connectivity queries in constant time and have an amortized update time of $o(\log n)$.

In sharp contrast, assuming that $m = \Omega(n \operatorname{poly} \log n)$ edges are deleted, our algorithm shows that one can solve decremental connectivity handling both queries and updates in constant amortized time.

We note that such a result is possible only because we allow for amortization, as any decremental connectivity algorithm with *worst-case* update time $O(\operatorname{polylog} n)$ must have worst-case query time $\Omega\left(\frac{\log n}{\log\log n}\right)$ [3]. This lower bound holds even for trees supporting restricted connectivity queries of the form "are u and v connected?" for a fixed "root" u. This lower bound also holds for dense graphs, as we can always add a large static clique to the problem.

An optimal incremental connectivity algorithm has been known for over 40 years. Namely, to handle $m \geq n$ edge insertion and q connectivity queries, one can use the union-find data structure [38] with $n-1$ unions and $2(m+q)$ finds. The total running time is $\Theta((m+q)\alpha((m+q), n))$, which is linear for all but very sparse graphs (since $\alpha(\Omega(n \log n), n) = O(1)$). It was later shown that this running time is optimal for incremental connectivity [13]. Interestingly, incremental connectivity can be solved in optimal linear time in the case of forests provided that the final shape of the forest is known in advance [16].

Similarly to the decremental case, one cannot hope to obtain an analogous result with a worst-case update time in the incremental setting: Pătraşcu and Thorup [36] showed that any incremental connectivity data structure with $o\left(\frac{\log n}{\log\log n}\right)$ worst-case update time must have worst-case $\Omega(n^{1-o(1)})$ query time in the cell-probe model.

Other cases of optimal decremental connectivity. There is much previous work on cases where decremental connectivity can be supported in $O(m)$ total time. Alstrup, Secher, and Spork [5] showed that decremental connectivity can be solved in optimal $O(m)$ total time on forests, answering queries in $O(1)$ time.[3] This was later extended to other classes of sparse graphs: planar graphs [32], and minor-free graphs [24]. All these special graph classes are sparse with $m = O(n)$ edges.

For general graphs, we only have the previously mentioned work by Thorup [39], yielding a total running time of $O(m)$ for very dense graphs with $m = \Omega(n^2)$ edges. We now obtain the same linear time bound for all non-sparse graphs with $m = \Omega(n \operatorname{poly} \log n)$ edges.

1.2 General reduction for c-edge-connectivity

Our algorithm for decremental connectivity is based on a general randomized reduction from decremental c-edge-connectivity (assuming all m edges are deleted) to fully-dynamic c-edge-connectivity on a sparse graph with $\tilde{O}(cn)$ updates. The reduction has a polylogarithmic cost per vertex as well as a constant cost per edge. The previous decremental connectivity

[3] The general *word encoding* trick behind [5, 16] that brings the update time to amortized constant has been even shown to have practical relevance [4].

algorithm of Thorup [39] was also based on such a general reduction, but the cost per edge was $O(\log(n^2/m))$ which is $O(1)$ only for very dense graphs with $m = \Omega(n^2)$. Below we will describe the format of the reductions in more detail.

Because there are different notions of c-edge-connectivity, we first need to clarify our definitions. We say that two vertices u, v are c-edge-connected iff there exist c edge-disjoint paths between u and v in G. It is known that c-edge-connectivity is an equivalence relation; we call its classes the *c-edge-connected classes*. However, for $c \geq 3$, a c-edge-connected class may induce a subgraph of G which is not connected, so it also makes sense to consider *c-edge-connected components*, i.e., the maximal c-edge-connected induced subgraphs of G.[4] It is important to note that the c-edge-connected components and the c-edge-connected classes are *uniquely defined* and both induce a natural partition of the vertices of the underlying graph. Moreover, each c-edge-connected component of G is a subset of some c-edge-connected class of G. For $c = 1, 2$, the c-edge-connected classes are c-edge-connected, so the two notions coincide. To illustrate the difference, let us fix $c \geq 3$ and consider a graph with $c + 2$ vertices $v_s, v_t, v_1, \ldots, v_c$ and edges $\{v_s, v_t\} \times \{v_1, \ldots, v_c\}$; while all c-edge-connected components in this graph are singletons, there is one c-edge-connected class, which is not a singleton, namely $\{v_s, v_t\}$.

We define a *c-certificate* of G to be a subgraph H of G that contains all edges not in c-edge-connected components, and also contains a c-edge-connected subgraph of each c-edge-connected component. Both Thorup's and our reduction maintains a c-certificate H of G. Then, for any $c' \leq c$, we have that the c'-edge-connected equivalence classes and the c'-edge-connected components are the same in G and H. As the edges from G are deleted, we maintain a c-certificate with $\widetilde{O}(cn)$ edges undergoing only $\widetilde{O}(cn)$ edge insertions and deletions in total.

The (uniquely defined) c-edge-connected components of a graph can be found using the following algorithm: while the graph contains a cut of size at most $c - 1$, remove all edges of this cut. For the reductions, we need algorithms that can help us in this process. We therefore define the *fully dynamic c-edge-cut problem* as follows. Suppose a graph G is subject to edge insertions and/or deletions. Then, a fully dynamic c-edge-cut data structure should report, after each update, some edge e that belongs to some cut of size less than c. A typical application of such a data structure is to repeatedly remove such edges e belonging to cuts of size less than c, which splits G into its c-edge-connected components. For each $c \geq 1$, denote by $T_c(n)$ the amortized time needed by the data structure to find an edge belonging to a cut of size less than c. For example, for $c = 1$ we have $T_1(n) = O(1)$ since we do not have to maintain anything. For $c = 2$, the data structure is required to maintain some *bridge* of G and it is known that $T_2(n) = O((\log n \cdot \log\log n)^2)$ [25]. For $c \geq 3$, in turn, we have $T_c(n) = O(n^{1/2} \operatorname{poly}(c))$ [41].

Given a fully dynamic c-edge-cut data structure, whose update time for a graph on n vertices is $T_c(n)$, Thorup's [39] reduction maintains, in $O(m \log(n^2/m)) + \widetilde{O}(c \cdot n \cdot T_c(n))$ total time, a c-certificate H of the decremental graph G starting with n vertices and m edges. The certificate undergoes only $\widetilde{O}(cn)$ edge insertions and deletions throughout any sequence of deletions issued to G. We reduce here the total time to $O(m) + \widetilde{O}(c \cdot n \cdot T_c(n))$.

Combining our reduction with the polylogarithmic fully-dynamic connectivity and 2-edge-connectivity algorithm of Holm, de Lichtenberg, and Thorup [23], we can now solve decremental connectivity and 2-edge-connectivity in $O(m) + \widetilde{O}(n)$ time.

[4] There is no consensus in the literature on the terminology relating to c-edge-connected components and classes. Some authors (e.g., [17, 18]) reserve the term c-edge-connected components for what we in this paper call c-edge-connected classes.

We can also apply the fully dynamic min-cut algorithm of Thorup [41] which identifies cuts of size $n^{o(1)}$ in $n^{1/2+o(1)}$ worst-case time. For $c = n^{o(1)}$, we then maintain a c-certificate H in $O(m + n^{3/2+o(1)})$ total time. This includes telling which vertices are in the same c-edge-connected component. If we further want to answer queries about c-edge-connectivity between pairs of vertices, we can apply the fully-dynamic data structure of Jin and Sun [27] to the c-certificate H. By definition, the answers to these queries are the same in H and G, and the algorithm takes $n^{o(1)}$ time per update or query. Hence the total time for the updates remains $O(m + n^{3/2+o(1)})$, and we can tell if two vertices are c-edge-connected in $n^{o(1)}$ time.

1.3 Results

We will now give a more precise description of our reduction, including the log-factors hidden in the $\widetilde{O}(cn)$ bound. Let the *decremental c-certificate* problem be that of maintaining a c-certificate of G when G is subject to edge deletions. Recall that $T_c(n)$ denotes the amortized update time of a fully-dynamic c-edge-cut data structure. Thorup [39] showed the following.

▶ **Theorem 1** (Thorup [39]). *There exists a Las Vegas randomized algorithm for the decremental c-certificate problem with expected total update time $O(m \log(n^2/m) + n(c + \log n) \cdot T_c(n) \log^2 n)$. The maintained certificate undergoes $O(n \cdot (c + \log n))$ expected edge insertions and deletions throughout, assuming $\Theta(m)$ random bits are provided. These bounds similarly hold with high probability.*

In particular the total update time is $O(m)$ for very dense graphs with $\Omega(n^2)$ edges. Our main result, which we state below, shows that amortized constant update time can be obtained as long as the initial graph has $\Omega(n \operatorname{polylog}(n))$ edges.

▶ **Theorem 2.** *There exists a Monte Carlo randomized algorithm for the decremental c-certificate problem with total update time $O(m + n(c + \log n) \cdot T_c(n) \log^3 n + nc \log^7 n)$. The maintained certificate undergoes $O(nc \log^4 n)$ edge insertions and deletions throughout. The algorithm is correct with high probability. Within this time bound, the algorithm offers a final self-check after processing all updates.*

In fact, our algorithm is itself a reduction to $O(\log n)$ instances of the decremental c-certificate problem on a subgraph of G with $m' = O(m/\log^2 n)$ edges. To handle each of these instances, we use the state-of-the-art data structure (Theorem 1) which costs only $O(m' \log m') = O(m/\log n)$ (for non-sparse graphs), yielding a combined cost of $O(m)$. As a result, our improved reduction (Theorem 2) requires $\Theta(m/\log n)$ random bits to hold.

We can reduce the need for random bits dramatically paying a little extra cost per vertex. Our new randomized c-certificate that is the key to obtaining the new reduction requires only pairwise independent sampling to work. This is in sharp contrast with the certificate of Karger [30], used in the construction of Thorup's data structure (Theorem 1), which requires full independence, i.e., $\Theta(m)$ random bits. We show that we may instead plug our new certificate into Thorup's data structure at the cost of a single additional logarithmic factor in the running time. Since Karger's certificate constitutes the only use of randomness in Thorup's data structure, and full independence in our construction is required only for invoking Theorem 1, we obtain the below low-randomness version of our main result.

▶ **Theorem 3.** *There exists a Monte Carlo randomized algorithm for the decremental c-certificate problem with total update time $O(m + nc \cdot T_c(n) \log^4 n + nc \log^7 n)$. The maintained certificate undergoes $O(nc \log^4 n)$ edge insertions and deletions throughout. The algorithm is correct with high probability if $O(\operatorname{polylog} n)$ random bits are provided. Within this time bound, the algorithm offers the final self-check after processing all updates.*

By using Theorem 3 with best known fully dynamic algorithms for different values of c [23, 27, 41], we obtain:

▶ **Theorem 4.** *There exists Monte Carlo randomized decremental connectivity and decremental 2-edge-connectivity algorithms with $O(m + n\log^7 n)$ total update time and $O(1)$ query time.*

▶ **Theorem 5.** *Let $c = (\log n)^{o(1)}$. There exists a Monte Carlo randomized decremental c-edge-connectivity data structure which can answer queries to whether two vertices are in the same c-edge connected class in $O(n^{o(1)})$ time, and which has $O(m) + \tilde{O}(n^{3/2})$ total update time.*

▶ **Theorem 6.** *Let $c = O(n^{o(1)})$. There exists a Monte Carlo randomized decremental c-edge-connected components data structure with $O(m + n^{3/2+o(1)})$ total update time and $O(1)$ query time.*

While Theorems 5 and 6 are only optimal for graphs with $m = \Omega(n^{3/2+o(1)})$ edges, we do note that the improvement in runtime from $O(mT_c(n))$ to $O(m + nT_c(n) \operatorname{polylog} n)$ is in general more impressive when $T_c(n)$ is large. E.g., if $T_c(n) = \sqrt{n}$, for dense graphs with $\Omega(n^2)$ edges, the former bound is $O(m^{5/4})$ while the later is $O(m)$ which is a polynomial improvement.

All the above applications of our main result work using only $O(\operatorname{polylog} n)$ random bits. They moreover each have the self-check property as well. As discussed before, our new 2-edge-connectivity data structure implies an optimal $O(m)$-time unique perfect matching algorithm for $m = \Omega(n \operatorname{polylog} n)$.

1.3.1 Adaptive updates and unique perfect matching

All our time bounds are amortized. Amortized time bounds are particularly relevant for dynamic data structures used inside algorithms solving problems for static graphs. In such contexts, future updates often depend on answers to previous queries, and therefore we need algorithms that work with adaptive updates.

Our reduction works against adaptive updates as long as all the information it provides is uniquely defined from the input graph and the update sequence, hence not revealing any information about the random choices in our c-certificate H. We assume some linear orderings of the vertices and the edges, and define the representative (or ID) of a c-edge connected component to be the smallest vertex in it. The reduction will safely maintain the following public information about the c-edge-connected components of G: between deletions, each vertex stores a pointer to the representative of its c-edge connected component, so two vertices are in the same c-edge-connected component if and only if they have they point to the same representative. With the representative, we store the size of the c-edge connected component, and list its vertices in sorted order. Finally, we have a sorted list of all edges that go between c-edge-connected components. After each update, we can also reveal the representatives of the new c-edge-connected components, and the edges between these components. For the case of 2-edge-connectivity, the above means that we can maintain the bridges of a decremental graph and we can also maintain the connected components and their sizes without revealing what the current randomized certificate looks like. All this is needed for the unique perfect matching algorithm of Gabow, Kaplan, and Tarjan [15]. The algorithm is an extremely simple recursion based on the fact that a graph with a unique

Algorithm 1 Algorithm computing Thorup's certificate in the static setting.

```
Input    : A graph $G = (V, E)$, where $n = |V|$, sampling probability $P$, parameter $c$
Returns : A set of $\tilde{O}(c \cdot n/P)$ edges giving a $c$-certificate of $G$

1 Function ThorupCertificate($V, E, P, c$):
2     if $|E| \leq c \cdot n$ then
3         return $E$
4     $S \leftarrow$ subset of $E$, in which each edge is included independently with prob. $P$;
5     $D \leftarrow$ edges of $E$ connecting distinct $c$-edge-connected components of $(V, S)$;
6     return $D \cup$ ThorupCertificate($V, S, P, c$)
```

perfect matching has a bridge and all components have even sizes. The algorithm first asks for a bridge (u, v) of some component. If there is none, there is no unique matching. Otherwise we remove (u, v) and check the sizes of the components of u and v. If they are odd, (u, v) is in the unique matching, and we remove all other incident edges. Otherwise (u, v) is not in the unique matching. The important thing here is that the bridges do not tell us anything about our random 2-certificate of the 2-edge-connected components.

Thus we solve the static problem of deciding if a graph has a unique perfect matching in $O(m) + \widetilde{O}(n)$ time. If the self-verification reports a possible mistake, we simply rerun. Consequently we get a Las Vegas algorithm that terminates in $O(m) + \widetilde{O}(n)$ time with high probability.

Outline. Due to space constraints, in the remaining part of this extended abstract we give a rather extensive technical overview of our data structure. All the details and proofs can be found in the full version of this paper.

2 Technical overview

Our main technical contribution is a new construction of a sparse randomized c-certificate that witnesses the c-edge-connected components of G and can be maintained in constant time per edge deletion in G (assuming that the initial graph is not too sparse). In the static case, deterministic certificates of this kind have been known for decades [33]. However, they are not very robust in the decremental setting, where an adversary can constantly remove its edges forcing it to update frequently. Consequently, Thorup [39] used a randomized sample-based certificate to obtain his reduction. The general idea behind this approach is to ensure that the certificate is sparse and undergoes few updates. Ideally, the sparse certificate will only have to be updated whenever an edge from the certificate is deleted. Using a fully dynamic data structure on the certificate, we may obtain efficient algorithms provided that we don't spend too much time on maintaining the certificate. Thorup's reduction had an additive overhead $O(m \log(n^2/m))$ for maintaining the certificate, which we will reduce to the optimal $O(m)$. We shall, in fact, use Thorup's reduction as a subroutine, called on $O(\log n)$ decremental subproblems each starting with $O(m/\log^2 n)$ edges.

2.1 Thorup's construction [39]

Let us first briefly describe how Thorup's algorithm operates on certificates and highlight difficulties in improving his reduction to linear time. First of all, the c-certificate is constructed as follows (see Algorithm 1 for pseudocode). Initially, sample edges of G uniformly with probability $P \leq 1/2$, thus obtaining a subgraph S. Then, compute the c-edge-connected components of S and form a certificate H out of two parts: (1) A *recursive* certificate of S, and (2) the subgraph D consisting of edges of G connecting distinct c-edge-connected components of S.

As proved by Karger [30], D has size $\widetilde{O}(cn/P)$ with high probability. Thorup [39] generalizes this by proving that D undergoes only $\widetilde{O}(cn/P)$ insertions throughout any sequence of edge deletions to S. Since D depends only on the c-edge-connected components of S, it is enough to have a c-certificate of S in order to define D. Hence, a c-certificate of S (which is a graph a size $O(mP)$, i.e., a constant factor smaller) is maintained under edge deletions recursively. The recursion stops when the size of the input graph is $O(cn)$. To maintain D at each recursive level, we first need to maintain the c-edge-connected components of the (recursive) certificate of S under edge deletions. The certificate of S can be (inductively) seen to have $\widetilde{O}(cn/P)$ edges and undergo $\widetilde{O}(cn/P)$ updates. As a result, for $P = 1/2$ maintaining its c-edge-connected components costs $\widetilde{O}(cn \cdot T_c(n))$ total time using the fully-dynamic c-edge-cut data structure. Since at each recursion level the certificate size decreases geometrically, the expected cost of all the dynamic c-edge-cut data structures is $\widetilde{O}(cn \cdot T_c(n))$. For $c = 1, 2$, $\widetilde{O}(cn \cdot T_c(n)) = \widetilde{O}(n)$.

The bottle-neck in Thorup's reduction. For non-sparse graphs, the bottleneck in Thorup's reduction is the additional cost of $O(m \log(n^2/m))$ which comes from the fact that, at each level of the recursion, when a c-edge-connected component in S splits into two components as a result of an edge deletion, we need to find edges of G between these two components in order to update D. This takes $O(m \log(n^2/m))$ total time throughout using a standard technique of iterating through the edges incident to the vertices in the smaller component every time a split happens [10]. The $O(\log(n^2/m))$ (instead of $O(\log(n))$) cost comes by noticing that a vertex can at most have q neighbors in a component of order q, and that after we go through the edges of a vertex i times it is in a component of order $\leq n/2^i$; hence it is only the first $O(\log(n/\deg(v)))$ times that all neighbors of v have to be considered, so, by applying Jensen's inequality, the total time spent on this becomes $O\left(\sum_{v \in V} \deg(v) \log(n/\deg(v))\right) = O(m \log(n^2/m))$.

It turns out very challenging to get rid of the $O(m \log(n^2/m))$ term associated with finding cuts when components split in Thorup's reduction. If we knew that all of these cuts were *small*, say of size at most δ, then we could apply a whole bag of tricks for efficiently finding them in a total time of $\tilde{O}(n\delta)$, e.g., using invertible Bloom lookup tables [19], or the XOR-trick [1, 2, 29]. Unfortunately, the bound of $\tilde{O}(cn/P)$ only gives an average bound on the number of edges between pairs of components, and in fact there can be pairs of components having as many as $\Omega(n^{1/3})$ edges between them, as we will later show. In order to resolve this, we have to introduce a new type of sample based c-edge certificate obtained by only removing cuts of size at most $\delta = O(c \operatorname{polylog} n)$ from G. In the following three subsections, we describe the ideas behind this new certificate, the technical challenges encountered in efficiently maintaining it, and why such a certificate is relevant for decremental connectivity algorithms.

2.2 Our c-certificate based on small cut samples

In this section we describe the construction of our c-certificate. For simplicity, we assume $c = 1$ for now.

The (simplified) algorithm for computing the certificate in the static setting is given as Algorithm 2. In order to obtain a conceptually simpler picture of the certificate, Algorithm 2 is described recursively where each recursive call takes as input a *minor* G' of G, namely

Algorithm 2 Algorithm computing our new certificate in the static setting.

```
Input   : A graph $G = (V, E)$ where $n = |V|$ and $m = |E|$, sampling probability $P$,
          parameter $\delta$
Returns : A set of $O(mP \log n + n\delta \log n)$ edges giving a 1-certificate of $G$

Function NewCertificate($V, E, P, \delta$):
    if $E = \emptyset$ then
        return $\emptyset$
    $D = \emptyset$;
    while $G$ has a non-isolated vertex $v$ of degree $\leq \delta$ do
        Remove from $E$ all edges incident to $v$ and add them to $D$
    $S \leftarrow$ subset of $E$, in which each edge is included independently with prob. $P$;
    $H \leftarrow (V, S)$;
    $G' \leftarrow$ graph obtained from $G$ by contracting connected components of $H$;
    return $S \cup D \cup$ NewCertificate($V(G'), E(G'), P, \delta$)
```

the graph obtained by contracting the connected components of $H = (V, S)$, where S is a subset of edges of G (after pruning G of small cuts in lines 5-6) sampled with probability P. Adding an edge e of G' to the certificate, simply means that we add the corresponding edge of G. While Algorithm 2 gives a precise description of the static certificate at any given point, maintaining these minors in the dynamic setting is too costly. Because of that, in the dynamic algorithm, instead of using minors we work with a sequence of subgraphs of the initial graph that are easier to maintain dynamically.

Denote by ℓ the depth of the recursion in Algorithm 2. For $i = 1, \ldots, \ell$, let S_i be the union of samples S on the recursive levels $1, \ldots, i$ of Algorithm 2, so that S_ℓ contains all the edges sampled in the process. When an edge is deleted from G, it is removed from all the sampled subsets S in the recursion, and thus also from all the relevant subsets S_i. This way, after any sequence of deletions the certificate that we maintain only depends on the initial samples $S_1, S_2 \setminus S_1, \ldots, S_\ell \setminus S_{\ell-1}$ and the current graph G, not on the sequence of edge updates made to G so far. We may therefore describe the certificate statically.

The critical idea behind our certificate is to introduce a small-cut-parameter δ. Our certificate is obtained by iteratively removing certain cuts from G where each cut is allowed to be of size at most δ. We denote by $D \subset G$ the graph whose edge set consists of the edges removed in this process. The overall goal is to define this cut removal process in a way so that (1) each connected component of $G \setminus D$ is connected in S_l, and (2) it is easy to detect new small cuts under edge deletions issued to G. We then use $S_\ell \cup D$ as our connectivity certificate of G. Importantly, we want δ to be *as small as possible*, ideally $\delta = O(\text{polylog}(n))$. This is because $\tilde{O}(\delta n)$ will show up as an additive cost in our algorithm for maintaining the certificate.

We will describe shortly how this type of certificate can be used in the design of efficient decremental connectivity algorithms, but let us first demonstrate that the existence of such a cut removal process (satisfying both (1) and (2)) for a small δ is non-trivial.

First of all, we could simply remove *all* cuts from G of size at most δ leaving us with the $(\delta + 1)$-connected components. Karger's result [30] implies that with $\delta = O((c + \log n)/P)$ sufficiently large, these components will remain c-edge connected in S. However, in order to maintain the small cuts, we would need a decremental δ-edge connectivity algorithm. As $\delta > c$, this approach simply reduces our problem to a much harder one.

Suppose on the other hand that we attempted to use Thorup's sampling certificate [39] described above. To simplify the exposition, let's assume that $P = 1/2$. If D is the set of edges between connected components of S, $D \cup S$ is a certificate. Thorup's algorithm recurses on S to find a final certificate of G. At first sight it may seem like D can be constructed by iteratively removing cuts of size at most $\delta = O(\log n)$ between the connected components of S. After all, isn't it unlikely that a connected component of S has more than, say, $100 \log n$ unsampled outgoing edges when the sampling probability is $P = 1/2$? As alluring as this logic may be, it is flawed. Indeed, there exist graphs of maximum degree $O(\log n)$ such that after sampling with $P = 1/2$, some two connected components of the sampled subgraph, C_1 and C_2, will have $\Omega(n^{1/3})$ unsampled edges between them. At some point in the iterative process, we are thus forced to remove a cut of size $\Omega(n^{1/3})$ splitting C_1 and C_2, and we would have to choose δ of at least this size (but it is possible that other examples could show that δ would have to be even larger). Our algorithms spend total time $\tilde{O}(n\delta)$ on finding these cuts, and if $\delta = \Omega(n^{1/3})$, this is not good enough for a linear time algorithm for non-sparse graphs.

We remark that in this example, each vertex of G has degree $O(\log n)$ with high probability. Therefore, an alternative approach yielding cuts of size $O(\log n)$ would be to cut out one vertex at a time moving all incident edges to D. In particular this would cut the large sampled components C_1 and C_2 into singletons, one vertex at a time. We cannot proceed like this for general graphs which may have many vertices of large degree. Nevertheless, this simple idea will be critically used in our construction which we will now discuss.

Our actual certificate uses $\delta = \Theta(\frac{\log n}{P})$ and $P = 1/\operatorname{polylog} n$. To construct our certificate, we start by iteratively pruning G of the edges incident to vertices of degree less than δ, moving these edges to D. The graph left after the pruning $G_1 = G \setminus D$ satisfies that each vertex of positive degree has degree at least δ. Next, S_1 defines a sample of G_1, $H_1 = S_1 \cap G_1$. The expected degree of each vertex in H_1 which is not isolated in G_1 is at least $\delta \cdot P = \Theta(\log n)$, and thus we get that with constant probability a fraction of $3/4$ of the vertices with positive degree in the sampled subgraph H_1 have degree ≥ 4.

Using this property we show that H_1 can have at most $5n/6$ connected components. As a result, if we contract the connected components of H_1 in the pruned graph G_1, the resulting graph G'_1 has at most $5n/6$ vertices. Finally, we construct a certificate for G'_1 recursively using the samples $S_2 \setminus S_1, S_3 \setminus S_2, \ldots$, stopping when the contracted graph has no edges between the contracted vertices (here G played the role of G'_0). The constant factor decay in the number of components ensures that we are done after $\ell = O(\log n)$ steps with high probability. All edges of D are obtained as the removed edges of cuts of G of size less than δ, so D will have size $O(n\delta)$. Our certificate will simply be $S_\ell \cup D$ which we prove is in fact a certificate.

With this, we have thus completed the goal of obtaining a small cut sample certificate with δ as small as $O(\frac{\log n}{P})$. Abstractly, our certificate has a quite simple description: we alternate between sampling, removing small cuts around connected components in the sample, and finally contracting these components. However, in our implementation, we cannot afford to perform the contractions as described above explicitly, as updating them dynamically would be costly. As a result we end up solving a more challenging problem in the dynamic setting. Given a graph G and its subgraph H undergoing edge deletions, determine if any connected components of H is incident to at most δ edges of $G \setminus D$, i.e., induces a cut of at most δ edges. It turns out that since we are only concerned with cuts of size at most δ, we can in fact identify these cuts in total time $O(m) + \tilde{O}(\delta n)$. We will describe this in the following section.

A final property of our new randomized decremental certificate algorithm is that it requires only $O(\log^2 n)$ random bits to yield high-probability correctness bounds. This is in sharp contrast with Thorup's algorithm [39] which requires $\Omega(m)$ random bits. On a high level, the reason we can do with few random bits is that in each step of the construction of our certificate, we only need the bounds on the number of contracted components to hold with constant probability. Indeed, we will still only have $O(\log n)$ recursive levels with high probability. This means that for the probability bounds within a single recursive level, it suffices to apply Chebyshev's inequality. While the reduction of the number of required random bits is nice, the main point, however, is that with our new certificate we can get down to constant amortized update time per edge-deletion for decremental (2-edge)-connectivity for all but the sparsest graphs.

2.3 Maintaining our certificate

As edges are deleted from G, the recursive structure of the c-certificate H changes. Indeed, a deletion of an edge may cause the following changes in one of the recursive layers of H: (1) introduce a cut of size less than δ surrounding a c-edge-connected component or (2) break a c-edge-connected component in two. In the first case, the edges of the cut have to be moved to D, and deleted from the current and later layers of H, causing further cascading. When a c-edge-connected component (in a recursive layer) of H breaks in two, we need to determine whether either of the new components has less than δ outgoing edges in $G \setminus D$. If we use the standard technique of iterating over all the edges incident to vertices of the smaller component, this again incurs an $O(\log(n^2/m))$ cost per edge which is insufficient. However, as we only care about components with at most δ outgoing edges, it turns out that we can do better. We define the *boundary* $\partial_G(C)$ of a component C of some graph $H \subset G$ to be the set of edges of G with one endpoint in C, and another in $V \setminus C$. To overcome the $O(\log(n^2/m))$ cost per edge, we prove that we can maintain boundaries of size at most δ under splits using a Monte Carlo randomized algorithm in $O(m + n\delta \operatorname{polylog} n)$ total time. We achieve this by developing a fully dynamic data structure summarized as follows, that we believe may be of independent interest.

► **Theorem 7.** *Let $G = (V, E)$ be an initially empty graph subject to edge insertions and deletions and let s, $1 \le s \le n$, be an integral parameter. There exists a data structure that can process up to $O(\operatorname{poly} n)$ queries of the form "given some $S \subseteq V$, compute $\partial_G(S)$", where $\partial_G(S) = E(S, V \setminus S)$, so that with high probability each query is answered correctly in $O\left(|S|s + |E(S,V)| \cdot \frac{|\partial_G(S)|}{s} + \log n\right)$ time. The data structure is initialized in $O(ns)$ time and can be updated in constant time.*

We realize this result by deploying the so-called XOR-trick [29][5] for deciding if a boundary of some subset of vertices is non-empty, albeit in a somewhat unusual manner. We now briefly describe the method. Suppose each $e \in E$ is assigned a random bit-string x_e of length $\Theta(\log n)$, which fits in $O(1)$ machine words. Let $x_v = \bigoplus_{vw=e\in E} x_e$ denote the XOR of the respective bit-strings of edges incident to v. Then, one can prove that, given $S \subseteq V$, with high probability the XOR $\bigoplus_{u\in S} x_u$ is non-zero if and only if $\partial_G(S) \neq \emptyset$. The underlying idea is that if an edge e incident to $v \in S$ has its other endpoint also contained in S, its corresponding bit string x_e appears exactly twice in $\bigoplus_{u\in S} x_u$, and thus cancels out. So, emptiness of $\partial_G(S)$ can be tested in $O(|S|)$ time.

[5] See also [1, 2] for uses of the same idea in other contexts.

The XOR trick can also be used with no change to retrieve a non-empty boundary $\partial_G(S)$, but only when that boundary has precisely one element. In order to retrieve *some* element of $\partial_G(S)$, existing applications of the XOR-trick consider a polylogarithmic number of independent edge set samples, chosen such that one of the samples intersects $\partial_G(S)$ precisely on one edge (with high probability). This unavoidably introduces a polylogarithmic dependence in the cost per edge of the graph, which is prohibitive in our scenario.

The main idea behind Theorem 7 which allows us to deal with this problem is as follows. We partition the edge set E into $E_1, \dots, E_s$. Each $e \in E$ is assigned to one of these sets uniformly at random. We apply the XOR-trick for each of the edge-disjoint subgraphs (V, E_i) separately. This takes $O(s|S|)$ time and computes a set I of all i such that $\partial_G(S) \cap E_i \neq \emptyset$ (with high probability). Clearly, in order to find $\partial_G(S)$, we only need to look for this boundary's elements in $\left(\bigcup_{i \in I} E_i\right) \cap E_G(S, V)$. Note that the expected size of this set is $(|I|/s) \cdot |E_G(S, V)| \leq (|\partial_G(S)|/s) \cdot |E_G(S, V)|$. If we set s to be larger than the maximum size of a boundary that we would like to retrieve (in the algorithm we ensure that the ratio is polylogarithmic), we significantly reduce the set of candidate edges to consider and can search through them exhaustively. In total, as we show, only $O(|\partial_G(S)| + |E_G(S, V)| \cdot |\partial_G(S)|/s + \log n)$ edges are explored with high probability.

In our application, we end up using the data structure of Theorem 7 storing the (dynamic) graph $G \setminus D$, and handling small boundary (of size no more than $\delta = \operatorname{polylog} n$) queries for smaller sides $C \subseteq V$ of decomposing components of $\ell = O(\log n)$ dynamic subgraphs of $G \setminus D$. Throughout, the total size of the queried subsets C is $O(n \log^2 n)$. Consequently, the sum of $|E(C, V)|$ over these sets is $O(m \log^2 n)$. By setting $s = \delta \log^2 n$ in Theorem 7, we obtain that the required queries for δ-bounded boundaries $\partial_{G \setminus D}(C)$ can be processed in $O(n \operatorname{polylog} n + m)$ total time.

2.4 Combining our certificate with Thorup's algorithm

With the certificate as above, the overall idea for a decremental connectivity algorithm is to maintain a c-certificate of (each recursive layer of) the *decremental* graph $H = S \setminus D$ using the algorithm by Thorup [39]. By choosing $P = 1/\log^2 n$, S has $m' = O(m/\log^2 n)$ edges with high probability, so employing the algorithm of Theorem 1 on each recursive layer takes total time $O(m' \log^2 n + ncT_c(n) \operatorname{polylog} n) = O(m + ncT_c(n) \operatorname{polylog} n)$ with high probability. Let H^* be the c-certificate thus obtained for H. Using a fully dynamic c-edge-connectivity algorithm on $H^* \cup D$ (which undergoes $O(cn \operatorname{polylog} n)$ updates), we maintain a c-edge certificate of G. As $H^* \cup D$ undergoes $O(cn \operatorname{polylog} n)$ updates, running the fully dynamic algorithm takes total time $O(cnT_c(n) \operatorname{polylog} n)$.

We remark that for $c = 1, 2$ we could instead use a fully dynamic c-edge connectivity algorithm on H with polylogaritmic update and query time at the price of a smaller P (which would incur more log-factors in our final time bound). For $c > 2$, however, we only know that $T_c(n) = O(n^{1/2} \operatorname{poly}(c))$. Since running a fully dynamic algorithm on H takes total time $\Omega(mT_c(n)/\operatorname{polylog} n)$, this is insufficient to obtain linear time algorithms for dense graphs.

2.5 Final self-check

Let us finally describe the ideas behind the final self-checks claimed in Theorem 2 and 3 in a more general context. In particular, we show that if a randomized Monte Carlo dynamic algorithm satisfies some generic conditions then it can be augmented to detect, at the end of its execution, whether there is any chance that it answered any query incorrectly. That is, if the self-check passes then it is guaranteed that all queries were answered correctly throughout the execution of the algorithm. Otherwise, it indicates that some queries might have been answered incorrectly. The self-check property is particularly useful in applications of dynamic algorithms as subroutines in algorithms solving static problems, that is, it enables static algorithms to exhibit Las Vegas guarantees instead of the Monte Carlo guarantees provided by the dynamic algorithm, as they can simply re-run the static algorithm with fresh randomness until the self-check passes.

The properties of a dynamic algorithm amenable to a self-check behavior are as follows:

- Once an error is made by the dynamic algorithm it should be detectable and any subsequent updates of the algorithm should not correct the error before it is detected.
- If the dynamic algorithm is stopped at any point in time, it should be able to still perform the self-check within the guaranteed running time of the algorithm.

In our algorithm, as long as the c-certificate maintained by our algorithm is correct, the c-edge-connectivity queries answered by our algorithm exhibit the same guarantees as the fully dynamic c-edge-connectivity algorithm running on the c-certificate H. Hence, we only need to detect potential errors in the process of maintaining the c-certificate H. Such errors only happen with probability $n^{-\Omega(1)}$.

By definition, a c-certificate $H \subseteq G$ of G is correct if for every "non-witness" edge (u,v) from $G \setminus H$, we have that u and v are c-edge-connected in H. We use $H = S_\ell \cup D$ where S_ℓ is decremental, and we impose the stronger requirement that if $(u,v) \in G \setminus H$, then u and v are c-edge-connected in S_ℓ. If this is not the case, we consider it an error.

Suppose we have an error with (u,v). Since S_ℓ is decremental, u and v cannot later become c-edge connected in S_ℓ. Thus, the error can only disappear if (u,v) is deleted from G or (u,v) is added to H. Therefore, all our self-checker needs to do is this: Whenever an edge from $G \setminus H$ is about to be deleted from G or about to be added to H, we first check that u and v are c-edge-connected in S_ℓ; otherwise we found an error. Also, if the algorithm is terminated before all edges are deleted, we perform that above check on all remaining edges. If any check finds an error, we flag the execution as invalid.

If an execution of our algorithm has not been flagged, we know that all queries have been answered correctly. Moreover, the execution is only flagged with probability $n^{-\Omega(1)}$.

As a final note, every vertex will maintain an ID of its c-edge-connected component in S_ℓ. Then u and v are the c-edge-connected in S_ℓ if and only if they have the same ID. This is checked in constant time, so these extra checks do not affect our overall asymptotic time bounds.

References

1 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012. doi:10.1137/1.9781611973099.40.

2 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012. doi:10.1145/2213556.2213560.

3 S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 1998. doi:10.1109/SFCS.1998.743504.

4 Stephen Alstrup, Jens P. Secher, and Mikkel Thorup. Word encoding tree connectivity works. In David B. Shmoys, editor, *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*, pages 498–499. ACM/SIAM, 2000. URL: `http://dl.acm.org/citation.cfm?id=338219.338598`.

5 Stephen Alstrup, Jens Peter Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *Information Processing Letters*, 64(4):161–164, 1997. `doi:10.1016/S0020-0190(97)00170-1`.

6 Joshua D. Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *Commun. ACM*, 56(8):87–94, 2013. `doi:10.1145/2492007.2492029`.

7 Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, and Shay Solomon. Fully dynamic $(\Delta+1)$-coloring in $O(1)$ update time. *ACM Trans. Algorithms*, 18(2):10:1–10:25, 2022. `doi:10.1145/3494539`.

8 Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019. `arXiv:1910.08025`.

9 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification – A technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. `doi:10.1145/265910.265914`.

10 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981. `doi:10.1145/322234.322235`.

11 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. `doi:10.1016/j.tcs.2005.09.013`.

12 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. `doi:10.1137/0214055`.

13 M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing (STOC)*, STOC '89, pages 345–354, New York, NY, USA, 1989. Association for Computing Machinery. `doi:10.1145/73007.73040`.

14 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987. `doi:10.1145/28869.28874`.

15 Harold N. Gabow, Haim Kaplan, and Robert Endre Tarjan. Unique maximum matching algorithms. *J. Algorithms*, 40(2):159–183, 2001. `doi:10.1006/jagm.2001.1167`.

16 Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985. `doi:10.1016/0022-0000(85)90014-5`.

17 Zvi Galil and Giuseppe F. Italiano. Maintaining the 3-edge-connected components of a graph on-line. *SIAM J. Comput.*, 22(1):11–28, 1993. `doi:10.1137/0222002`.

18 Dora Giammarresi and Giuseppe F. Italiano. Decremental 2- and 3-connectivity on planar graphs. *Algorithmica*, 16(3):263–287, 1996. `doi:10.1007/BF01955676`.

19 Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011.

20 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1886–1898. SIAM, 2019. `doi:10.1137/1.9781611975482.114`.

21 Monika Henzinger and Pan Peng. Constant-time dynamic $(\Delta+1)$-coloring. *ACM Trans. Algorithms*, 18(2):16:1–16:21, 2022. `doi:10.1145/3501403`.

22 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999. doi:10.1145/320211.320215.

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

24 Jacob Holm and Eva Rotenberg. Good r-divisions imply optimal amortised decremental biconnectivity. *CoRR*, abs/1808.02568, 2018. arXiv:1808.02568.

25 Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\tilde{O}(\log^2 n)$ amortized time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 35–52, 2018. doi:10.1137/1.9781611975031.3.

26 Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log n(\log\log n)^2)$ amortized expected time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 510–520, 2017. doi:10.1137/1.9781611974782.32.

27 Wenyu Jin and Xiaorui Sun. Fully dynamic s-t edge connectivity in subpolynomial time (extended abstract). In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 861–872. IEEE, 2021. doi:10.1109/FOCS52979.2021.00088.

28 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

29 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1131–1142, USA, 2013. Society for Industrial and Applied Mathematics.

30 David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999. doi:10.1287/moor.24.2.383.

31 Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster worst case deterministic dynamic connectivity. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 53:1–53:15, 2016. doi:10.4230/LIPIcs.ESA.2016.53.

32 Jakub Lacki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. *Theory Comput. Syst.*, 61(4):1037–1053, 2017. doi:10.1007/s00224-016-9709-x.

33 Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992. doi:10.1007/BF01758778.

34 Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961, 2017. doi:10.1109/FOCS.2017.92.

35 Mihai Pǎtraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 546–553, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007352.1007435.

36 Mihai Pǎtraşcu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 559–568, 2011. doi:10.1145/1993636.1993711.

37 Shay Solomon. Fully dynamic maximal matching in constant update time. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.43.

38 Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975. `doi:10.1145/321879.321884`.

39 Mikkel Thorup. Decremental dynamic connectivity. *Journal of Algorithms*, 33(2):229–243, 1999. `doi:10.1006/jagm.1999.1033`.

40 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350, 2000. `doi:10.1145/335305.335345`.

41 Mikkel Thorup. Fully-dynamic min-cut. *Comb.*, 27(1):91–127, 2007. `doi:10.1007/s00493-007-0045-2`.

42 Zhengyu Wang. An improved randomized data structure for dynamic graph connectivity. *CoRR*, abs/1510.04590, 2015. `arXiv:1510.04590`.

43 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1757–1769. SIAM, 2013. `doi:10.1137/1.9781611973105.126`.