A Generalized Matching Reconfiguration Problem

Noam Solomon
Immunai, New York, NY, USA
noam@immunai.com

Shay Solomon
School of Electrical Engineering, Tel Aviv University, Israel
shayso@tauex.tau.ac.il

Abstract

The goal in *reconfiguration problems* is to compute a *gradual transformation* between two feasible solutions of a problem such that all intermediate solutions are also feasible. In the *Matching Reconfiguration Problem* (MRP), proposed in a pioneering work by Ito et al. from 2008, we are given a graph G and two matchings M and M', and we are asked whether there is a sequence of matchings in G starting with M and ending at M', each resulting from the previous one by either adding or deleting a single edge in G, without ever going through a matching of size $< \min\{|M|, |M'|\} - 1$. Ito et al. gave a polynomial time algorithm for the problem, which uses the Edmonds-Gallai decomposition.

In this paper we introduce a natural generalization of the MRP that depends on an integer parameter $\Delta \geq 1$: here we are allowed to make Δ changes to the current solution rather than 1 at each step of the transformation procedure. There is always a valid sequence of matchings transforming M to M' if Δ is sufficiently large, and naturally we would like to minimize Δ. We first devise an optimal transformation procedure for unweighted matching with $\Delta = 3$, and then extend it to weighted matchings to achieve asymptotically optimal guarantees. The running time of these procedures is linear.

We further demonstrate the applicability of this generalized problem to dynamic graph matchings. In this area, the number of changes to the maintained matching per update step (the *recourse bound*) is an important quality measure. Nevertheless, the *worst-case* recourse bounds of almost all known dynamic matching algorithms are prohibitively large, much larger than the corresponding update times. We fill in this gap via a surprisingly simple black-box reduction: Any dynamic algorithm for maintaining a β-approximate maximum cardinality matching with update time T, for any $\beta \geq 1$, T and $\varepsilon > 0$, can be *transformed* into an algorithm for maintaining a $(\beta(1+\varepsilon))$-approximate maximum cardinality matching with update time $T + O(1/\varepsilon)$ and worst-case recourse bound $O(1/\varepsilon)$. This result generalizes for approximate maximum weight matching, where the update time and worst-case recourse bound grow from $T + O(1/\varepsilon)$ and $O(1/\varepsilon)$ to $T + O(\psi/\varepsilon)$ and $O(\psi/\varepsilon)$, respectively; ψ is the graph *aspect-ratio*. We complement this positive result by showing that, for $\beta = 1 + \varepsilon$, the worst-case recourse bound of any algorithm produced by our reduction is optimal. As a corollary, several key dynamic approximate matching algorithms – with poor worst-case recourse bounds – are strengthened to achieve near-optimal worst-case recourse bounds with no loss in update time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Dynamic algorithms, graph matching, reconfiguration problem, recourse bound

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.57

Related Version A full version of the paper is available at `https://arxiv.org/pdf/1803.05825.pdf`.

Funding *Shay Solomon*: Partially supported by the Israel Science Foundation grant No.1991/19 and by Len Blavatnik and the Blavatnik Family foundation.

Acknowledgements The authors thank Thatchaphol Saranurak for fruitful discussions.

1 Introduction

The study of graph algorithms is mostly concerned with the measure of *(static) runtime*. Given a graph optimization problem, the standard objective is to design a fast (possibly approximation) algorithm, and ideally complement it with a matching lower bound on the runtime of any (approximation) algorithm for solving the problem. As an example, computing (from scratch) a 2-approximate minimum vertex cover (VC) can be done trivially in linear time, whereas a better-than-2 approximation for the minimum VC cannot be computed in polynomial time under the unique games conjecture [46].

The current paper is motivated by a natural need arising in networks that are prone to temporary or permanent changes. Such changes are sometimes part of the normal behavior of the network, as in *dynamic networks*, but changes could also be the result of unpredictable failures of nodes and edges, particularly in *faulty networks*. Consider a large-scale network $G = (V, E, w)$ for which we need to solve, perhaps approximately, some graph optimization problem, and the underlying solution (e.g., a maximum matching) is being used for some practical purpose (e.g., scheduling in packet switches) throughout a long time span. If the network changes over time, the quality of the used solution may degrade until it is too poor to be used in practice and it may even become infeasible.

Instead of the standard objectives of optimization, the questions that arise here concern *reoptimization*: Can we "efficiently" transform one given solution (the *source*) to another one (the *target*) under "real-life constraints"? The efficiency of the *transformation procedure* could be measured in terms of running time, but in some applications making even small changes to the currently used solution may incur huge costs, possibly much higher than the runtime cost of computing (from scratch) a better solution; we shall use "procedure" and "process" interchangeably. In particular, this is often the case whenever the edges of the currently used solution are "hard-wired" in some physical sense, as in road networks. Various real-life constraints or objectives may be studied; the one we focus on in this work is that at any step (or every few steps) throughout the transformation process the current solution should be both feasible and of quality no worse (by much) than that of either the source or target solutions. This constraint is natural as it might be prohibitively expensive or even impossible to carry out the transformation process *instantaneously*. Instead, the transformation can be broken into *phases* each performing $\leq \Delta$ changes to the transformed solution, where $\Delta \geq 1$ is some parameter, so that the solution obtained at the end of each phase – to be used instead of the source solution – is both feasible and of quality no (much) worse than either the source or target. The transformed solution is to eventually coincide with the target solution.

The arising *reoptimization* meta-problem generalizes the well-studied framework of *reconfiguration problems*, which we discuss in Section 1.1. It is interesting from both practical and theoretical perspectives, since even the most basic and well-understood optimization problems become open in this setting. E.g., for the VC problem, *given* a better-than-2 approximate target VC, can we transform to it from any source VC subject to the above constraints? This is an example for a problem that is computationally hard in the standard sense but might be easy from a reoptimization perspective. In contrast, perhaps computationally easy problems, such as approximate maximum matching, are hard from a reoptimization perspective?

This meta-problem captures tension between (1) the *global* objective of transforming one global solution to another, and (2) the *local* objective of transforming *gradually* while having a feasible and high quality solution throughout the process. A similar tension is captured by various models of computation that involve locality, including dynamic graph algorithms,

distributed computing, property testing and local computation algorithms (LCA). The study of the meta-problem presented here could borrow from these related research fields, but, more importantly, we anticipate that it will also contribute to them; indeed, we present here an application of this meta-problem to dynamic graph algorithms.

1.1 Graph Reconfiguration

The framework of *reconfiguration problems* has been subject to growing interest in recent years. The term *reconfiguration* was coined in the work of Ito et al. [41], which unified earlier related problems and terminology (see, e.g., [40, 31, 22]) into a single framework. The general goal is to compute a *transformation* between two feasible solutions of a problem such that all intermediate solutions are also feasible, where each pair of consecutive solutions need to be *adjacent* under a fixed polynomially testable symmetric adjacency relation on the set of feasible solutions. Such a transformation arises naturally in many contexts, such as solving puzzles, motion planning, questions of evolvability (can genotype evolve into another one via individual "adjacent" mutations?), and similarity of DNA sequences in computational genomics and particularly gene editing, which is among the hottest scientific topics these days; see the surveys of [58, 51] for further details. In most previous work, two solutions are called adjacent if their symmetric difference has size 1. The most well-studied problem under this framework is graph matching. For brevity, we shall only discuss here papers on graph matching; see the surveys [58, 51] for discussions on other problems.

In the *Matching Reconfiguration Problem* (MRP), proposed in [41], we are given a graph G and two matchings M and M', and we are asked whether there is a sequence of matchings in G starting with M and ending at M', each resulting from the previous one by either adding or deleting a *single edge* in G, without ever going through a matching of size $< \min\{|M|, |M'|\} - 1$. Ito et al. gave a polynomial time algorithm for the problem, which uses the Edmonds-Gallai decomposition. In particular, in some cases such a transformation does not exist, and much of the difficulty is in the decision problem (decide if exists or not). The problem of generalizing this algorithm for weighted matchings was proposed as an open problem in [41], and remained open to date, partially since the algorithm of [41] for unweighted matchings already relies on a rather intricate decomposition. The work of [41] triggered interesting followups on MRP [44, 42, 39, 43, 21, 27]. In all these followups, the symmetric difference between two adjacent matchings is rather strict: it is fixed by either 1 or 2 in [44, 42, 39, 27], whereas in the context of perfect matchings the symmmetric difference is an alternating cycle of length 4 [21, 43]. Perhaps since the symmetric difference in all the previous work is so strict, the goal was polynomial-time algorithms and hardness for the problem. The natural generalization of parameterizing the symmetric difference by an arbitrary $\Delta, \Delta \geq 1$ – as in our *reoptizmiation meta-problem*, was not studied in prior work.

1.2 Our contribution

We study two fundamental graph matching problems under the aforementioned meta-problem: (approximate) maximum cardinality matching (MCM) and maximum weight matching (MWM). Our meta-problem is, in fact, inherently different than the original MRP. We are not interested in the decision version of the problem – we take Δ to be large enough so that a transformation is *guaranteed* to exist. Thus we shift the focus from per-instance optimization to *existential optimization*, and our goal is to optimize Δ so that any source matching can be transformed to any target matching by performing at most Δ changes per step, while never reaching a *much worse* matching than either the source or the target along the way.

By "worse" we mean either in terms of size or weight, and we must indeed do a bit worse in some cases even for large Δ; the original MRP formulation for unweighted matchings allows to go down by 1 unit of size, and this slack is required also for large Δ. For weighted graphs, naturally, a bigger slack is required. For both unweighted and weighted matchings, we provide transformation procedures with near-optimal guarantees and linear running time. Our results are summarized next; the transformation for approximate MWM (Theorem 2) is the most technically challenging part of this work.

▶ **Theorem 1** (MCM). *For any source and target matchings $\mathcal{M}$ and $\mathcal{M}'$, one can transform $\mathcal{M}$ into (a possibly superset of) $\mathcal{M}'$ via a sequence of phases consisting of ≤ 3 operations each (i.e., $\Delta = 3$), such that the matching at the end of each phase throughout this transformation is a valid matching for G of size $\geq \min\{|\mathcal{M}|, |\mathcal{M}'| - 1\}$. The runtime of this transformation procedure is $O(|\mathcal{M}| + |\mathcal{M}'|)$.*

▶ **Theorem 2** (MWM). *For any source and target matchings $\mathcal{M}$ and $\mathcal{M}'$ with $w(\mathcal{M}') > w(\mathcal{M})$, and any $\varepsilon > 0$, one can transform $\mathcal{M}$ into (a possibly superset of) $\mathcal{M}'$ via a sequence of phases consisting of $O(\frac{1}{\varepsilon})$ operations each (i.e., $\Delta = O(\frac{1}{\varepsilon})$), such that the matching obtained at the end of each phase throughout this transformation is a valid matching for G of weight $\geq \max\{w(\mathcal{M}) - W, (1-\varepsilon)w(\mathcal{M})\}$, where $W = \max_{e \in \mathcal{M}} w(e)$. The runtime of this transformation procedure is $O(|\mathcal{M}| + |\mathcal{M}'|)$.*

▶ Remark. Theorem 2 assumes that $w(\mathcal{M}') > w(\mathcal{M})$. This assumption is made without loss of generality, since, if $w(\mathcal{M}') \leq w(\mathcal{M})$, we can apply a reversed transformation, so that the matching will always be of weight $\geq \max\{w(\mathcal{M}') - W', (1-\varepsilon)w(\mathcal{M}')\}$, where $W' = \max_{e \in \mathcal{M}'} w(e)$.

In Section 5, we show that the guarantees provided by Theorems 1 and 2 are tight and asymptotically tight, respectively. Although our results may lead to the impression that there exists an efficient gradual transformation process to any graph optimization problem, we briefly discuss in Section 7 two trivial hardness results for the minimum VC and maximum independent set problems.

1.2.1 Application: A worst-case recourse bound for dynamic matching algorithms

In the standard *fully dynamic* setting we start from an empty graph G_0 on n fixed vertices, and at each time step i a single edge (u, v) is either inserted to the graph G_{i-1} or deleted from it, resulting in graph G_i. In the *vertex update* setting we have vertex updates instead of edge updates; this setting was mostly studied for bipartite graphs [24, 25, 14].

The problem of maintaining a large matching in fully dynamic graphs was subject to intensive interest recently [52, 10, 50, 38, 53, 56, 18, 14, 29, 3, 32, 13]. The basic goal is to devise an algorithm for maintaining a large matching while keeping a tab on the *update time*, i.e., the time required to update the matching at each step. One may try to optimize the *amortized* (average) update time of the algorithm or its *worst-case* (maximum) update time, but both measures are defined with respect to a *worst-case* sequence of graphs.

"Maintaining" a matching with update time u_T translates into maintaining a data structure with update time u_T, which answers queries regarding the matching with a low, ideally constant, *query time* q_T. For a queried vertex v the answer is the only matched edge incident on v, or NULL if v is free, while for a queried edge e the answer is whether edge e is matched or not. All queries made following the same update step i should be answered *consistently* with respect to the same matching, hereafter the *output matching (at step i)*,

but queries made in the next update step $i+1$ may be answered with respect to a completely different matching. Thus even if the worst-case update time is low, the output matching may change significantly from one update step to the next; some natural scenarios where the output matching changes significantly per update step are discussed in Section 2.

The number of changes (or replacements) to the output matching per update step is an important measure of quality, sometimes referred to as the *recourse bound*, and the problem of optimizing it has received growing attention recently [33, 30, 37, 25, 26, 14, 15, 2, 47]. In applications such as job scheduling, web hosting, streaming content delivery, data storage and hashing, a replacement of a matched edge by another one may be costly, possibly much more than the runtime of computing these replacements. Moreover, when the recourse bound is low, one can efficiently *output* all the changes to the matching following every update step, which could be important in practical scenarios. In particular, a low recourse bound is important when the matching algorithm is used as a black-box subroutine inside a larger data structure or algorithm [17, 1]; see Section 2.3 for more details. We remark that the recourse bound (generally defined as the number of changes to some underlying structure per update step) has been well studied in the areas of dynamic and online algorithms for a plethora of optimization problems besides graph matching, such as MIS, set cover, Steiner tree, flow and scheduling; see [34, 36, 37, 9, 48, 6, 28, 35, 54], and the references therein.

There is a strong separation between the state-of-the-art amortized versus worst-case bounds for dynamic matching algorithms, in terms of both the time and the recourse bounds. A similar separation exists for numerous other problems, such as dynamic minimum spanning forest. In various practical scenarios, particularly in systems designed to provide real-time responses, a strict tab on the *worst-case update time* or on the *worst-case recourse bound* is crucial, thus an algorithm with a low amortized guarantee but a high worst-case guarantee is useless.

Despite the importance of the recourse bound measure, all known algorithms but one in the area of dynamic matchings (described in detail in the full version [55]; see Appendix C therein) provide no nontrivial worst-case recourse bounds whatsoever! The sole exception is an algorithm for maintaining a maximal matching with a worst-case update time $O(\sqrt{m})$ and a constant recourse bound [50]. In this paper we fill in this gap via a surprisingly simple yet powerful black-box reduction (throughout *β-MCM* is a shortcut for β-approximate MCM):

▶ **Theorem 3.** *Any dynamic algorithm maintaining a β-MCM with update time T,*[1] *for any $\beta \geq 1$, T and $\varepsilon > 0$, can be* transformed *into an algorithm maintaining a $(\beta(1+\varepsilon))$-MCM with update time $T + O(1/\varepsilon)$ and worst-case recourse bound $O(1/\varepsilon)$. If the original time bound T is amortized/worst-case, so is the resulting time bound of $T + O(1/\varepsilon)$, while the recourse bound $O(1/\varepsilon)$ always holds in the worst-case. This applies to the fully dynamic setting under edge and/or vertex updates.*

The proof of Theorem 3 is carried out in two steps. First we prove Theorem 1 by showing a simple transformation process for any two matchings $\mathcal{M}$ and $\mathcal{M}'$ of the same *static* graph. The second step of the proof, which is the key insight behind it, is that the gradual transformation process can be used *essentially as is* in fully dynamic graphs, while incurring a negligible loss to the size and approximation guarantee of the transformed matching.

In Section 6 we complement the positive result provided by Theorem 3 by proving that the recourse bound $O(1/\varepsilon)$ is optimal (up to a constant factor) in the regime $\beta = 1 + \varepsilon$. In fact, the lower bound $\Omega(1/\varepsilon)$ on the recourse bound holds even in the amortized sense

[1] Besides answering queries, we naturally assume that at any update step the entire matching can be output within time (nearly) linear in its size. All known algorithms satisfy this assumption.

and even in the incremental (insertion only) and decremental (deletion only) settings. For larger values of β, taking ε to be a sufficiently small constant gives rise to an approximation guarantee arbitrarily close to β with a constant recourse bound.

A corollary of Theorem 3. As a corollary of Theorem 3, all previous algorithms [38, 16, 53, 29, 3, 13, 59] with low worst-case update time are strengthened to achieve a worst-case recourse bound of $O(1/\varepsilon)$ with only an additive overhead of $O(1/\varepsilon)$ to the update time. (Some of these results were already strengthened in this way by using a previous version of the current work, which was posted to arXiv in 2018.) Since the update time of all these algorithms is larger than $O(1/\varepsilon)$, we get a recourse bound of $O(1/\varepsilon)$ with no loss whatsoever in the update time! Moreover, all known algorithms with low amortized update time can be strengthened in the same way; e.g., in SODA'19 [32] (cf. [24]) it was shown that one can maintain a $(1+\varepsilon)$-MCM in the incremental edge update setting with a constant (depending exponentially on ε) amortized update time. While this algorithm yields a constant amortized recourse bound, no nontrivial (i.e., $o(n)$) worst-case recourse bound was known for this problem. Theorem 3 strengthens the result of [32] to maintain a $(1+\varepsilon)$-MCM with a constant amortized update time and the optimal worst-case recourse bound of $O(1/\varepsilon)$. Since the recourse bound is an important measure of quality, this provides a significant contribution to the area of dynamic matching algorithms.

Weighted matchings. The result of Theorem 3 can be generalized for approximate MWM in graphs with bounded aspect ratio ψ, by using the much more intricate transformation provided by Theorem 2 (compared to Theorem 1), as summarized in the next theorem. (The *aspect ratio* $\psi = \psi(G)$ of a weighted graph $G = (V, E, w)$ is defined as $\psi = \frac{\max_{e\in E} w(e)}{\min_{e\in E} w(e)}$.)

▶ **Theorem 4.** *Any dynamic algorithm for maintaining a β-approximate MWM (shortly, β-*MWM*) with update time T in a dynamic graph with aspect ratio always bounded by ψ, for any $\beta \geq 1$, $T, \varepsilon > 0$ and ψ, can be* transformed *into an algorithm for maintaining a $(\beta(1+\varepsilon))$-MWM with update time $T + O(\psi/\varepsilon)$ and worst-case recourse bound $O(\psi/\varepsilon)$. If the original time bound T is amortized/worst-case, so is the resulting time bound of $T + O(\psi/\varepsilon)$, while the recourse bound $O(\psi/\varepsilon)$ always holds in the worst-case. This applies to the fully dynamic setting under edge and/or vertex updates.*

Scenarios with high recourse bounds. There are various scenarios where high recourse bounds may naturally arise. In such scenarios our reductions (Theorems 3 and 4) can come into play to achieve low worst-case recourse bounds. Furthermore, although a direct application of our reductions may only hurt the update time, we demonstrate the usefulness of these reductions in achieving low update time bounds in some natural settings (where we might not care at all about recourse bounds); this, we believe, provides another strong motivation for our reductions. The details are provided in Section 2.

1.3 Related work

We discussed in Section 1.1 prior work on graph reconfiguration problems. Other than this line of work, there are also inherently different lines of work on "reoptimiziation", which indeed can be interpreted broadly – there is an extensive and diverse body of research devoted to various notions of reoptimization; see [57, 8, 23, 20, 7, 11, 12, 54, 19], and the references therein. The common goal in all previous work on reoptimization (besides the one discussed in Section 1.1 on reconfiguration) is to (efficiently) compute an exact or approximate solution

to a new problem instance by using the solution for the old instance, where typically the solution for the new instance should be close to the original one under certain distance measure. Our work is inherently different than all such previous work, since our starting point is that *some solution to the new problem instance is given*, and the goal is to compute a *gradual transformation process* (subject to some constraints) between the two given solutions. Also, our work is inherently different than previous work on reconfiguration, as explained in Section 1.2.

1.4 Organization

We start (Section 2) with discussing some scenarios where high recourse bounds may naturally arise. We continue (Section 3) by describing a basic scheme for dynamic approximate matchings that was introduced in [38]. In Section 4.1 we present a simple transformation process for MCM in static graphs, thus proving Theorem 1. This result is generalized for MWM via a more intricate transformation process that proves Theorem 2, which is deferred to the full version [55] (see Appendix D therein) due to space constraints. These transformations, which apply to static graphs, are adapted to the fully dynamic setting in Sections 4.2 and 4.3, thus proving Theorems 3 and 4, respectively. The optimality of these transformations is discussed in Section 5. Our lower bound of $\Omega(1/\varepsilon)$ on the recourse bound of $(1+\varepsilon)$-MCMs is provided in Section 6. We conclude with a discussion in Section 7.

2 Scenarios with high recourse bounds

In this section we discuss some scenarios where high recourse bounds may naturally arise. In all such scenarios our reductions (Theorems 3 and 4) can come into play to achieve low worst-case recourse bounds; for clarity we focus in this discussion, sometimes implicitly, on large (unweighted) matching, but the entire discussion carries over with very minor changes to the generalized setting of weighted matchings.

Section 2.3 demonstrates that, although we *may not care at all about recourse bounds*, maintaining a large (weight) matching with a low update time requires in some cases the use of a dynamic matching algorithm with a low recourse bound; this is another situation where our reductions can come into play, but more than that, we believe that it provides an additional strong motivation for our reductions.

2.1 Randomized algorithms

Multiple matchings. Given a randomized algorithm for maintaining a large matching in a dynamic graph, it may be advantageous to run multiple instances of the algorithm (say $\mathsf{polylog}(n)$), since this may increase the chances that at least one of those instances provides a large matching with high probability (w.h.p.) at any point in time. Notice, however, that it is not the same matching that is guaranteed to be large throughout the entire update sequence, hence the ultimate algorithm (or data structure), which outputs the largest among the $\mathsf{polylog}(n)$ matchings, may need to switch between a pool of possibly very different matchings when going from one update step to the next. Thus even if the recourse bound of the given randomized algorithm is low, and so each of the maintained matchings changes gradually over time, we do not get any nontrivial recourse bound for the ultimate algorithm.

Large matchings. Sometimes the approximation guarantee of the given randomized algorithm holds w.h.p. only when the matching is sufficiently large. This is the case with the algorithm of [29] that achieves $\mathsf{polylog}(n)$ worst-case update time, where the approximation

guarantee of $2+\varepsilon$ holds w.h.p. only when the size of the matching is $\Omega(\log^5 n/\varepsilon^4)$. To perform efficiently, [29] also maintains a matching that is guaranteed to be maximal (and thus provide a 2-MCM) when the maximum matching size is smaller than $\delta = O(\log^5 n/\varepsilon^4)$, via a deterministic procedure with a worst-case update time of $O(\delta)$. The ultimate algorithm of [29] switches between the matching given by the randomized algorithm and that by the deterministic procedure, taking the larger of the two. Thus even if the recourse bounds of both the randomized algorithm and the deterministic procedure are low, the worst-case recourse bound of the ultimate algorithm, which might be of the order of the "large matching" threshold, could be very high. (The large matching threshold is the threshold on the matching size above which a high probability bound on the approximation guarantee holds.) In [29] the large matching threshold is $\delta = O(\log^5 n/\varepsilon^4)$, so the recourse bound is reasonably low. (This is not the bottleneck for the recourse bound of [29], as discussed next.) In general, however, the large matching threshold may be significantly higher than $\mathsf{polylog}(n)$.

Long update sequences. For the probabilistic guarantees of a randomized dynamic algorithm to hold w.h.p., the update sequence must be of bounded length. In particular, polylogarithmic guarantees on the update time usually require that the length of the update sequence will be polynomially bounded. This is the case with numerous dynamic graph algorithms also outside the scope of graph matchings (cf. [45, 1]), and the basic idea is to partition the update sequence into sub-sequences of polynomial length each and to run a fresh instance of the dynamic algorithm in each sub-sequence. In the context of matchings, the algorithm of [29] uses this approach. Notice, however, that an arbitrary sub-sequence (other than the first) does not start from an empty graph. Hence, for the ultimate algorithm of [29] to provide a low worst-case update time, it has to gradually construct the graph at the beginning of each sub-sequence from scratch and maintain for it a new gradually growing matching, while re-using the "old" matching used for the previous sub-sequence throughout this gradual process. Once the gradually constructed graph coincides with the true graph, the ultimate algorithm switches from the old matching to the new one. (See [29] for further details.) While this approach guarantees that the worst-case update time of the algorithm is in check, it does not provide any nontrivial worst-case recourse bound.

2.2 From amortized to worst-case

There are techniques for transforming algorithms with low amortized bounds into algorithms with similar worst-case bounds. For approximate matchings, such a technique was first presented in [38]. Alas, the transformed algorithms do not achieve any nontrivial worst-case recourse bound; see Section 3 for details.

2.3 When low update time requires low recourse bound

When a dynamic matching algorithm is used as a black-box subroutine inside a larger data structure or algorithm, a low recourse bound of the algorithm used as a subroutine is needed for achieving a low update time for the larger algorithm. We next consider a natural question motivating this need; one may refer to [17, 1] for additional motivation.

▶ **Question 1.** *Given k dynamic matchings of a dynamic graph G, whose union is guaranteed to contain a large matching for G at any time, for an arbitrary parameter k, can we combine those k matchings into a large dynamic matching for G* efficiently*?*

This question may arise when there are physical limitations, such as memory constraints, e.g., as captured by MapReduce-style computation, where the edges of the graph are partitioned into k parties. More specifically, consider a fully dynamic graph G of huge scale, for which we want to maintain a large matching with low update time. The edges of the graph are dynamically partitioned into k parties due to memory constraints, each capable of maintaining a large matching for the graph induced by its own edges with low update time, and the only guarantee on those k dynamically changing matchings is the following global one: The *union* of the k matchings at any point in time *contains* a large matching for the entire dynamic graph G. (E.g., if we maintain at each update step the invariant that the edges of G are partitioned across the k parties uniformly at random, such a global guarantee can be provided via the framework of *composable randomized coresets* [49, 5, 4].)

This question may also arise when the input data set is noisy. Coping with noisy input usually requires *randomization*, which may lead to high recourse bounds as discussed in Section 2.1. Let us revisit the scenario where we run multiple instances of a randomized dynamic algorithm with low update time; denote the number of such instances by k. If the input is noisy, we may not be able to guarantee that at least one of the k maintained matchings is large w.h.p. at any point in time, as suggested in Section 2.1. A weaker, more reasonable assumption is that the union of those k matchings contains a large matching.

The key observation is that it is insufficient to maintain each of the k matchings with low update time, even in the worst-case, as each such matching may change significantly following a single update step, thereby changing significantly the union of those matchings. "Feeding" this union to *any* dynamic matching algorithm would result with poor update time bounds, even in the amortized sense. Consequently, to resolve Question 1, each of the k maintained matchings must change *gradually* over time, or in other words, the underlying algorithm(s) needed for maintaining those matchings should guarantee a low recourse bound. A low amortized/worst-case recourse bound of the underlying algorithm(s) translates into a low amortized/worst-case update time of the ultimate algorithm, provided of course that the underlying algorithm(s) for maintaining those k matchings, as well as the dynamic matching algorithm to which their union is fed, all achieve a low amortized/worst-case update time.

3 The scheme of [38]

This section provides a short overview of a basic scheme for dynamic approximate matchings from [38]. Although such an overview is not required for proving Theorems 3 and 4, it is instructive to provide it, as it shows that the scheme of [38] is insufficient for providing any nontrivial worst-case recourse bound. Also, the scheme of [38] exploits a basic *stability* property of matchings, which we use for proving Theorems 3 and 4, thus an overview of this scheme may facilitate the understanding of our proof.

3.1 The amortization scheme of [38]

The *stability* property of matchings used in [38] is that the maximum matching size changes by at most 1 following each update step. Thus if we have a β-MCM, for any $\beta \geq 1$, the approximation guarantee of the matching will remain close to β throughout a long update sequence. Formally, the following lemma is a simple adaptation of Lemma 3.1 from [38]; its proof is given in Appendix E of the full version [55]. (Lemma 3.1 of [38] is stated for approximation guarantee $1 + \varepsilon$ and for edge updates, whereas Lemma 5 here holds for any approximation guarantee and also for vertex updates.)

▶ **Lemma 5.** *Let $\varepsilon' \leq 1/2$. Suppose $\mathcal{M}_t$ is a β-MCM for G_t, for any $\beta \geq 1$. For $i = t, t+1, \ldots, t + \lfloor \varepsilon' \cdot |\mathcal{M}_t| \rfloor$, let $\mathcal{M}_t^{(i)}$ denote the matching $\mathcal{M}_t$ after removing from it all edges that got deleted during updates $t+1, \ldots, i$. Then $\mathcal{M}_t^{(i)}$ is a $(\beta(1+2\varepsilon'))$-MCM for G_i.*

For concreteness, we shall focus on the regime of approximation guarantee $1+\varepsilon$, and sketch the argument of [38] for maintaining a $(1+\varepsilon)$-MCM in fully dynamic graphs. (As Lemma 5 applies to any approximation guarantee $\beta \geq 1+\varepsilon$, it is readily verified that the same argument carries over to any approximation guarantee.)

One can compute a $(1+\varepsilon/4)$-MCM $\mathcal{M}_t$ at a certain update step t, and then re-use the same matching $\mathcal{M}_t^{(i)}$ throughout all update steps $i = t, t+1, \ldots, t' = t + \lfloor \varepsilon/4 \cdot |\mathcal{M}_t| \rfloor$ (after removing from it all edges that got deleted from the graph between steps t and i). By Lemma 5, assuming $\varepsilon \leq 1/2$, $\mathcal{M}_t^{(i)}$ provides a $(1+\varepsilon)$-MCM for all graphs G_i. Next compute a fresh $(1+\varepsilon/4)$-MCM $\mathcal{M}_{t'}$ following update step t' and re-use it throughout all update steps $t', t'+1, \ldots, t' + \lfloor \varepsilon/4 \cdot |\mathcal{M}_{t'}| \rfloor$, and repeat. In this way the static time complexity of computing a $(1+\varepsilon)$-MCM $\mathcal{M}$ is *amortized* over $1 + \lfloor \varepsilon/4 \cdot |\mathcal{M}| \rfloor = \Omega(\varepsilon \cdot |\mathcal{M}|)$ update steps. Note that the static computation time of an approximate matching is $O(|\mathcal{M}| \cdot \alpha/\varepsilon^2)$, where α is the arboricity bound; refer to Appendix F in the full version [55]. (This bound on the static computation time was established in [53]; it reduces to $O(|\mathcal{M}| \cdot \sqrt{m}/\varepsilon^2)$ and $O(|\mathcal{M}| \cdot \Delta/\varepsilon^2)$ for general graphs and graphs of degree bounded by Δ, respectively, which are the bounds provided by [38].)

3.2 A Worst-Case Update time

In the amortization scheme of [38] described above, a $(1+\varepsilon/4)$-MCM $\mathcal{M}$ is computed *from scratch*, and then being re-used throughout $\lfloor \varepsilon/4 \cdot |\mathcal{M}| \rfloor$ additional update steps. The worst-case update time is thus the static computation time of an approximate matching, namely, $O(|\mathcal{M}| \cdot \alpha/\varepsilon^2)$. To improve the worst-case guarantee, the tweak used in [38] is to simulate the static approximate matching computation within a "time window" of $1 + \lfloor \varepsilon/4 \cdot |\mathcal{M}| \rfloor$ consecutive update steps, so that following each update step the algorithm simulates only $O(|\mathcal{M}| \cdot \alpha/\varepsilon^2)/(1 + \lfloor \varepsilon/4 \cdot |\mathcal{M}| \rfloor = O(\alpha \cdot \varepsilon^{-3})$ steps of the static computation. During this time window the gradually-computed matching, denoted by $\mathcal{M}'$, is useless, so the previously-computed matching $\mathcal{M}$ is re-used as the output matching. This means that each matching is re-used throughout a time window of twice as many update steps, hence the approximation guarantee increases from $1+\varepsilon$ to $1+2\varepsilon$, but we can reduce it back to $1+\varepsilon$ by a straightforward scaling argument. Note that the gradually-computed matching does not include edges that got deleted from the graph during the time window.

3.3 Recourse bounds

Consider an arbitrary time window used in the amortization scheme of [38], and note that the same matching is being re-used throughout the entire window. Hence there are no changes to the matching in the "interior" of the window except for those triggered by adversarial deletions, which may trigger at most one change to the matching per update step. On the other hand, at the start of any time window (except for the first), the output matching is switched from the old matching $\mathcal{M}$ to the new one $\mathcal{M}'$, which may require $|\mathcal{M}| + |\mathcal{M}'|$ replacements to the output matching at that time. Note that the amortized number of replacements per update step is quite low, being upper bounded by $(|\mathcal{M}| + |\mathcal{M}'|)/(1 + \lfloor \varepsilon/4 \cdot |\mathcal{M}| \rfloor)$. In the regime of approximation guarantee $\beta = O(1)$, we have $|\mathcal{M}| = O(|\mathcal{M}'|)$, hence the amortized recourse bound is bounded by $O(1/\varepsilon)$. For a general approximation guarantee β, the naive amortized recourse bound is $O(\beta/\varepsilon)$.

On the negative side, the worst-case recourse bound may still be as high as $|\mathcal{M}| + |\mathcal{M}'|$, even after performing the above tweak. Indeed, that tweak only causes the time windows to be twice longer, and it does not change the fact that once the computation of $\mathcal{M}'$ finishes, the output matching is switched from the old matching $\mathcal{M}$ to the new one $\mathcal{M}'$ *instantaneously*, which may require $|\mathcal{M}| + |\mathcal{M}'|$ replacements to the output matching at that time.

4 Proofs of Theorems 3 and 4

This section is mostly devoted (see Sections 4.1 and 4.2) to the proof of Theorem 3. At the end of this section (Section 4.3) we sketch the adjustments needed for deriving the result of Theorem 4, whose proof follows along similar lines to those of Theorem 3.

4.1 A simple transformation in static graphs

This section is devoted to the proof of Theorem 1, which provides the first step in the proof of Theorem 3. We remark that this proof can be viewed as a "warm up" to that of Theorem 2 for MWM, which is deferred to the full version [55] (see Appendix D therein), and is considerably more technically involved.

Let $\mathcal{M}$ and $\mathcal{M}'$ be two matchings for the same graph G. Our goal is to gradually transform $\mathcal{M}$ into (a possibly superset of) $\mathcal{M}'$ via a sequence of constant-time operations to be described next, each making at most 3 changes to the matching, such that the matching obtained at any point throughout this transformation process is a valid matching for G of size at least $\min\{|\mathcal{M}|, |\mathcal{M}'| - 1\}$. It is technically convenient to denote by $\mathcal{M}^*$ the *transformed* matching, which is initialized as $\mathcal{M}$ at the outset, and being gradually transformed into $\mathcal{M}'$; we refer to $\mathcal{M}$ and $\mathcal{M}'$ as the *source* and *target* matchings, respectively. Each operation starts by adding a single edge of $\mathcal{M}' \setminus \mathcal{M}^*$ to $\mathcal{M}^*$ and then removing from $\mathcal{M}^*$ the at most two edges incident on the newly added edge; thus at most 3 changes to the matching are made per operation. It is instructive to assume that $|\mathcal{M}'| > |\mathcal{M}|$, as the motivation for applying this transformation, which will become clear in Section 4.2, is to increase the matching size; in this case the size $|\mathcal{M}^*|$ of the transformed matching $\mathcal{M}^*$ never goes below the size $|\mathcal{M}|$ of the source matching $\mathcal{M}$.

We say that an edge of $\mathcal{M}' \setminus \mathcal{M}^*$ that is incident on at most one edge of $\mathcal{M}^*$ is *good*, otherwise it is *bad*, being incident on two edges of $\mathcal{M}^*$. Since $\mathcal{M}^*$ has to be a valid matching throughout the transformation process, adding a bad edge to $\mathcal{M}^*$ must trigger the removal of two edges from $\mathcal{M}^*$. Thus if we keep adding bad edges to $\mathcal{M}^*$, the size of $\mathcal{M}^*$ may halve throughout the transformation process. The following lemma shows that if all edges of $\mathcal{M}' \setminus \mathcal{M}^*$ are bad, the transformed matching $\mathcal{M}^*$ is at least as large as the target matching $\mathcal{M}'$.

▶ **Lemma 6.** *If all edges of $\mathcal{M}' \setminus \mathcal{M}^*$ are bad, then $|\mathcal{M}^*| \geq |\mathcal{M}'|$.*

Proof. Consider a bipartite graph $L \cup R$, where each vertex in L corresponds to an edge of $\mathcal{M}' \setminus \mathcal{M}^*$ and each vertex in R corresponds to an edge of $\mathcal{M}^* \setminus \mathcal{M}'$, and there is an edge between a vertex in L and a vertex in R iff the corresponding matched edges share a common vertex in the original graph. If all edges of $\mathcal{M}' \setminus \mathcal{M}^*$ are bad, then any edge of $\mathcal{M}' \setminus \mathcal{M}^*$ is incident on two edges of $\mathcal{M}^*$, and since $\mathcal{M}'$ is a valid matching, those two edges cannot be in $\mathcal{M}'$. In other words, the degree of each vertex in L is exactly 2. Also, the degree of each vertex in R is at most 2, as $\mathcal{M}'$ is a valid matching. It follows that $|R| \geq |L|$, or in other words $|\mathcal{M}^* \setminus \mathcal{M}'| \geq |\mathcal{M}' \setminus \mathcal{M}^*|$, yielding $|\mathcal{M}^*| \geq |\mathcal{M}'|$. ◀

The transformation process is carried out as follows. At the outset we initialize $\mathcal{M}^* = \mathcal{M}$ and compute the sets $\mathcal{G}$ and $\mathcal{B}$ of good and bad edges in $\mathcal{M}' \setminus \mathcal{M}^* = \mathcal{M}' \setminus \mathcal{M}$ within time $O(|\mathcal{M}| + |\mathcal{M}'|)$ in the obvious way, and store them in doubly-linked lists. We keep mutual pointers between each edge of $\mathcal{M}^*$ and its at most two incident edges in the corresponding linked lists $\mathcal{G}$ and $\mathcal{B}$. Then we perform a sequence of operations, where each operation starts by adding an edge of $\mathcal{M}' \setminus \mathcal{M}^*$ to $\mathcal{M}^*$, giving precedence to good edges (i.e., adding a bad edge to $\mathcal{M}^*$ only when there are no good edges to add), and then removing from $\mathcal{M}^*$ the at most two edges incident on the newly added edge. Following each such operation, we update the lists $\mathcal{G}$ and $\mathcal{B}$ of good and bad edges in $\mathcal{M}' \setminus \mathcal{M}^*$ within constant time in the obvious way. This process is repeated until $\mathcal{M}' \setminus \mathcal{M}^* = \emptyset$, at which stage we have $\mathcal{M}^* \supseteq \mathcal{M}'$. Note that the number of operations performed before emptying $\mathcal{M}' \setminus \mathcal{M}^*$ is bounded by $|\mathcal{M}'|$, since each operation removes at least one edge from $\mathcal{M}' \setminus \mathcal{M}^*$. It follows that the total runtime of the transformation process is bounded by $O(|\mathcal{M}| + |\mathcal{M}'|)$.

It is immediate that $\mathcal{M}^*$ remains a valid matching throughout the transformation process, as we pro-actively remove from it edges that share a common vertex with new edges added to it. To complete the proof of Theorem 1 it remains to prove the following lemma.

▶ **Lemma 7.** *At any moment in time we have* $|\mathcal{M}^*| \geq \min\{|\mathcal{M}|, |\mathcal{M}'| - 1\}$.

Proof. Suppose for contradiction that the lemma does not hold, and consider the first time step t^* throughout the transformation process in which $|\mathcal{M}^*| < \min\{|\mathcal{M}|, |\mathcal{M}'| - 1\}$. Since initially $|\mathcal{M}^*| = |\mathcal{M}|$ and as every addition of a good edge to $\mathcal{M}^*$ triggers at most one edge removal from it, time step t^* must occur after an addition of a bad edge. Recall that a bad edge is added to $\mathcal{M}^*$ only when there are no good edges to add. Just before this addition we have $|\mathcal{M}^*| \geq |\mathcal{M}'|$ by Lemma 6, thus we have $|\mathcal{M}^*| \geq |\mathcal{M}'| - 1$ after adding that edge to $\mathcal{M}^*$ and removing the two edges incident on it from there, yielding a contradiction. ◀

▶ Remark 8. When $|\mathcal{M}| < |\mathcal{M}'|$, it is possible to gradually transform $\mathcal{M}$ to $\mathcal{M}'$ without ever being in deficit compared to the initial value of $\mathcal{M}$, i.e., $|\mathcal{M}^*| \geq |\mathcal{M}|$ throughout the transformation process. However, if $|\mathcal{M}'| \leq |\mathcal{M}|$, this no longer holds true; refer to Section 5.1 for more details.

4.2 The Fully Dynamic Setting

In this section we provide the second step in the proof of Theorem 3, showing that the simple transformation process described in Section 4.1 for static graphs can be generalized for the fully dynamic setting, thus completing the proof of Theorem 3.

Consider an arbitrary dynamic algorithm, Algorithm $\mathcal{A}$, for maintaining a β-MCM with an update time of T, for any $\beta \geq 1$ and T. The matching maintained by Algorithm $\mathcal{A}$, denoted by $\mathcal{M}_i^A$, for $i = 1, 2, \ldots$, may change significantly following a single update step. All that is guaranteed by Algorithm $\mathcal{A}$ is that it can update the matching following every update step within a time bound of T, either in the worst-case sense or in the amortized sense, following which queries regarding the matching can be answered in (nearly) constant time. Recall also that we assume that, for any update step i, the matching $\mathcal{M}_i^A$ provided by Algorithm $\mathcal{A}$ at step i can be output within time (nearly) linear in the matching size.

Our goal is to output a matching $\tilde{\mathcal{M}} = \tilde{\mathcal{M}}_i$, for $i = 1, 2, \ldots$, possibly very different from $\mathcal{M}^A = \mathcal{M}_i^A$, which changes very slightly from one update step to the next. To this end, the basic idea is to use the matching $\mathcal{M}^A$ provided by Algorithm $\mathcal{A}$ at a certain update step, and then re-use it (gradually removing from it edges that get deleted from the graph) throughout a sufficiently long window of $\Theta(\varepsilon \cdot |\mathcal{M}^A|)$ consecutive update steps, while gradually transforming it into a larger matching, provided again by Algorithm $\mathcal{A}$ at some later step.

The *gradual transformation process* is obtained by adapting the process described in Section 4.1 for static graphs to the fully dynamic setting. Next, we describe this adaptation. We assume that $\beta = O(1)$; the case of a general β is addressed in Section 4.2.1.

Consider the beginning of a new time window, at some update step t. Denote the matching provided by Algorithm $\mathcal{A}$ at that stage by $\mathcal{M}' = \mathcal{M}_t^A$ and the matching output by our algorithm by $\mathcal{M} = \tilde{\mathcal{M}}_t$. Recall that the entire matching $\mathcal{M}' = \mathcal{M}_t^A$ can be output in time (nearly) linear in its size, and we henceforth assume that $\mathcal{M}'$ is given as a list of edges. (For concreteness, we assume that the time needed for storing the edges of $\mathcal{M}'$ in an appropriate list is $O(|\mathcal{M}'|.)$ While $\mathcal{M}'$ is guaranteed to provide a β-MCM at any update step, including t, the approximation guarantee of $\mathcal{M}$ may be worse. Nevertheless, we will show (Lemma 9) that $\mathcal{M}$ provides a $(\beta(1+2\varepsilon'))$-MCM for G_t. Under the assumption that $\beta = O(1)$, we thus have $|\mathcal{M}| = O(|\mathcal{M}'|)$. The length of the time window is $W = \Theta(\varepsilon \cdot |\mathcal{M}|)$, i.e., it starts at update step t and ends at update step $t' = t + W - 1$. During this time window, we gradually transform $\mathcal{M}$ into (a possibly superset of) $\mathcal{M}'$, using the transformation described in Section 4.1 for static graphs; recall that the matching output throughout this transformation process is denoted by $\mathcal{M}^*$. We may assume that $|\mathcal{M}|, |\mathcal{M}'| = \Omega(1/\varepsilon)$, where the constant hiding in the Ω-notation is sufficiently large; indeed, otherwise $|\mathcal{M}| + |\mathcal{M}'| = O(1/\varepsilon)$ and there is no need to apply the transformation process, as the trivial worst-case recourse bound is $O(1/\varepsilon)$.

We will show (Lemma 9) that the output matching $\tilde{\mathcal{M}}_i$ provides a $(\beta(1+O(\varepsilon))$-MCM at any update step i. Two simple adjustments are needed for adapting the transformed matching $\mathcal{M}^*$ of the static setting to the fully dynamic setting:

- To achieve a low worst-case recourse bound and guarantee that the overhead in the update time (with respect to the original update time) is low in the worst-case, we cannot carry out the entire computation at once (i.e. following a single update step), but should rather *simulate it gradually* over the entire time window of the transformation process. Specifically, recall that the transformation process for static graphs consists of two phases, a preprocessing phase in which the matching $\mathcal{M}' = \mathcal{M}_t^A$ and the sets $\mathcal{G}$ and $\mathcal{B}$ of good and bad edges in $\mathcal{M}' \setminus \mathcal{M}$ are computed, and the actual transformation phase that transforms $\mathcal{M}^*$, which is initialized as $\mathcal{M}$, into (a possibly superset of) $\mathcal{M}'$. Each of these phases requires time $O(|\mathcal{M}| + |\mathcal{M}'|) = O(|\mathcal{M}|)$. The first phase does not make any replacements to $\mathcal{M}^*$, whereas the second phase consists of a sequence of at most $|\mathcal{M}'|$ constant-time operations, each of which may trigger a constant number of replacements to $\mathcal{M}^*$. The computation of the first phase is simulated in the first $W/2$ update steps of the window, performing $O(|\mathcal{M}| + |\mathcal{M}'|)/(W/2) = O(1/\varepsilon)$ computation steps and zero replacements to $\mathcal{M}^*$ following every update step. The computation of the second phase is simulated in the second $W/2$ update steps of the window, performing $O(|\mathcal{M}| + |\mathcal{M}'|)/(W/2) = O(1/\varepsilon)$ computation steps and replacements to $\mathcal{M}^*$ following every update step.
- Denote by $\mathcal{M}_i^*$ the matching output at the ith update step by the resulting gradual transformation process, which simulates $O(1/\varepsilon)$ computation steps and replacements to the output matching following every update step. While $\mathcal{M}_i^*$ is a valid matching for the (static) graph G_t at the beginning of the time window, some of its edges may get deleted from the graph in subsequent update steps $i = t+1, t+2, \ldots, t'$. Consequently, the matching that we shall output for graph G_i, denoted by $\tilde{\mathcal{M}}_i$, is the one obtained from $\mathcal{M}_i^*$ by removing from it all edges that got deleted from the graph between steps t and i.

Once the current time window terminates, a new time window starts, and the same transformation process is repeated, with $\tilde{\mathcal{M}}_{t'}$ serving as $\mathcal{M}$ and $\mathcal{M}_{t'}^A$ serving as $\mathcal{M}'$. Since all time windows are handled in the same way, it suffices to analyze the output matching of the current time window, and this analysis would carry over to the entire update sequence.

It is immediate that the output matching $\tilde{\mathcal{M}}_i$ is a valid matching for any $i = t, t+1, \ldots, t'$. Moreover, since we make sure to simulate $O(1/\varepsilon)$ computation steps and replacements following every update step, the worst-case recourse bound of the resulting algorithm is bounded by $O(1/\varepsilon)$ and the update time is bounded by $T + O(1/\varepsilon)$, where this time bound is worst-case/amortized if the time bound T of Algorithm $\mathcal{A}$ is worst-case/amortized.

It is left to bound the approximation guarantee of the output matching $\tilde{\mathcal{M}}_i$. Recall that $W = \Theta(\varepsilon \cdot |\mathcal{M}|)$, and write $W = \varepsilon' \cdot |\mathcal{M}|$, with $\varepsilon' = \Theta(\varepsilon)$. (We assume that ε is sufficiently small so that $\varepsilon' \leq 1/2$. We need this restriction on ε' to apply Lemma 5.)

▶ **Lemma 9.** *$\tilde{\mathcal{M}}_t$ and $\tilde{\mathcal{M}}_{t'}$ provide a $(\beta(1+2\varepsilon'))$-MCM for G_t and $G_{t'}$, respectively. Moreover, $\tilde{\mathcal{M}}_i$ provides a $(\beta((1+2\varepsilon')^2))$-MCM for G_i, for any $i = t, t+1, \ldots, t'$.*

Proof. First, we bound the approximation guarantee of the matching $\tilde{\mathcal{M}}_{t'}$, which is obtained from $\mathcal{M}^*_{t'}$ by removing from it all edges that got deleted from the graph throughout the time window. By the description of the transformation process, $\mathcal{M}^*_{t'}$ is a superset of $\mathcal{M}'$, hence $\tilde{\mathcal{M}}_{t'}$ is a superset of the matching obtained from $\mathcal{M}'$ by removing from it all edges that got deleted throughout the time window. Since $\mathcal{M}'$ is a β-MCM for G_t, Lemma 5 implies that $\tilde{\mathcal{M}}_{t'}$ is a $(\beta(1+2\varepsilon'))$-MCM for $G_{t'}$. More generally, this argument shows that the matching obtained at the end of any time window is a $(\beta(1+2\varepsilon'))$-MCM for the graph at that step.

Next, we argue that the matching obtained at the start of any time window (as described above) is a $(\beta(1+2\varepsilon'))$-MCM for the graph at that step. This assertion is trivially true for the first time window, where both the matching and the graph are empty. For any subsequent time window, this assertion follows from the fact that the matching at the start of a new time window is the one obtained at the end of the old time window, for which we have already shown that the required approximation guarantee holds. It follows that $\tilde{\mathcal{M}}_t = \mathcal{M}$ is a $(\beta(1+2\varepsilon'))$-MCM for G_t.

Finally, we bound the approximation guarantee of the output matching $\tilde{\mathcal{M}}_i$ in the entire time window. (It suffices to consider the interior of the window.) Lemma 7 implies that $|\mathcal{M}^*_i| \geq \min\{|\mathcal{M}|, |\mathcal{M}'| - 1\}$, for any $i = t, t+1, \ldots, t'$. We argue that $\mathcal{M}^*_i$ is a $(\beta(1+2\varepsilon'))$-MCM for G_t. If $|\mathcal{M}^*_i| \geq |\mathcal{M}|$, then this assertion follows from the fact that $\mathcal{M}$ provides such an approximation guarantee. We henceforth assume that $|\mathcal{M}^*_i| \geq |\mathcal{M}'| - 1$. Recall that $|\mathcal{M}'| = \Omega(1/\varepsilon) = \Omega(1/\varepsilon')$, where the constants hiding in the Ω-notation are sufficiently large, hence removing a single edge from $\mathcal{M}'$ cannot hurt the approximation guarantee by more than an additive factor of, say ε', i.e., less than $\beta(2\varepsilon')$. Since $\mathcal{M}'$ provides a β-MCM for G_t, it follows that $\mathcal{M}^*_i$ is indeed a $(\beta(1+2\varepsilon'))$-MCM for G_t, which completes the proof of the above assertion. Consequently, Lemma 5 implies that $\tilde{\mathcal{M}}_i$, which is obtained from $\mathcal{M}^*_i$ by removing from it all edges that got deleted from the graph between steps t and i, is a $(\beta((1+2\varepsilon')^2))$-MCM for G_i. ◀

4.2.1 A general approximation guarantee

In this section we consider the case of a general approximation parameter $\beta \geq 1$. The bound on the approximation guarantee of the output matching provided by Lemma 9, namely $(\beta((1+2\varepsilon')^2))$, remains unchanged. Recalling that $\varepsilon' \leq 1/2$, it follows that the size of $\mathcal{M}'$ cannot be larger than that of $\mathcal{M}$ by more than a factor of $(\beta((1+2\varepsilon')^2)) \leq 2\beta$. Consequently, the number of computation steps and replacements performed per update step, namely, $O(|\mathcal{M}| + |\mathcal{M}'|)/(W/2)$, is no longer bounded by $O(1/\varepsilon)$, but rather by $O(\beta/\varepsilon)$. To achieve a bound of $O(1/\varepsilon)$ for a general β, we shall use a matching $\mathcal{M}''$ different from $\mathcal{M}'$, which includes a possibly small fraction of the edges of $\mathcal{M}'$. Recall that we can output ℓ arbitrary edges of the matching $\mathcal{M}' = \mathcal{M}^A_t$ in time (nearly) linear in ℓ, for any integer

$\ell = 1, 2, \ldots, |\mathcal{M}'|$. Let $\mathcal{M}''$ be a matching that consists of (up to) $2|\mathcal{M}|$ arbitrary edges of $\mathcal{M}'$; that is, if $|\mathcal{M}'| > 2|\mathcal{M}|$, $\mathcal{M}''$ consists of $2|\mathcal{M}|$ arbitrary edges of $\mathcal{M}'$, otherwise $\mathcal{M}'' = \mathcal{M}'$. We argue that $\mathcal{M}''$ is a β-MCM for G_t. Indeed, if $|\mathcal{M}'| > 2|\mathcal{M}|$ the approximation guarantee follows from the approximation guarantee of $\mathcal{M}$ and the fact that $\mathcal{M}''$ is twice larger than $\mathcal{M}$, whereas in the complementary case the approximation guarantee follows from that of $\mathcal{M}'$. In any case it is immediate that $|\mathcal{M}''| = O(|\mathcal{M}|)$. (For concreteness, we assume that the time needed for storing the edges of $\mathcal{M}''$ in an appropriate list is $O(|\mathcal{M}''|) = O(|\mathcal{M}|)$.) We may henceforth carry out the entire transformation process with $\mathcal{M}''$ taking the role of $\mathcal{M}'$, and in this way guarantee that the number of computation steps and replacements to the output matching performed per update step is reduced from $O(\beta/\varepsilon)$ to $O(1/\varepsilon)$.

4.3 Proof of Theorem 4

The proof of Theorem 4 is very similar to the one of Theorem 3. Specifically, we derive Theorem 4 by making a couple of simple adjustments to the proof of Theorem 3 given above, which we sketch next. First, instead of using the transformation of Theorem 1, we use the one of Theorem 2, whose proof appears in Appendix D of the full version [55]. Second, the *stability* property of unweighted matchings used in the proof of Theorem 3 is that the maximum matching size changes by at most 1 following each update step. This stability property enables us in the proof of Theorem 3 to consider a time window of $W = \Theta(\varepsilon \cdot |\mathcal{M}|)$ update steps, so that any β-MCM computed at the beginning of the window will provide (after removing from it all the edges that get deleted from the graph) a $(\beta(1+\varepsilon))$-MCM throughout the entire window, for any $\beta \geq 1$. It is easy to see that this stability property generalizes for weighted matchings, where the maximum matching weight may change by an additive factor of at most ψ. (Recall that the aspect ratio of the dynamic graph is always bounded by ψ; also, we may assume by scaling that the minimum edge weight is 1.) In order to obtain a $(\beta(1+\varepsilon))$-MWM throughout the entire time window, it suffices to consider a time window of $W' = W'_\psi = W/\psi = \Theta(\varepsilon \cdot |\mathcal{M}|/\psi)$, i.e., a time window shorter than that used for unweighted matchings by a factor of ψ, and as a result the update time of the resulting algorithm will grow from $T + O(1/\varepsilon)$ to $T + O(\psi/\varepsilon)$ and the worst-case recourse bound will grow from $O(1/\varepsilon)$ to $O(\psi/\varepsilon)$.

5 Optimality of our Transformations

5.1 Unweighted matchings

In the unweighted case, when $|\mathcal{M}| < |\mathcal{M}'|$, Theorem 1 states that $\mathcal{M}$ can gradually transform into $\mathcal{M}'$ without ever being in deficit compared to the initial value of $\mathcal{M}$, i.e., $|\mathcal{M}^*| \geq |\mathcal{M}|$ throughout the entire transformation process. If $|\mathcal{M}'| \leq |\mathcal{M}|$, however, this no longer holds; in this case the theorem states that we'll reach a deficit of at most 1 unit. To see that this bound is tight, consider the case when $|\mathcal{M}| = |\mathcal{M}'|$ and $H = \mathcal{M} \oplus \mathcal{M}'$ is a simple alternating cycle that consists of all edges in $\mathcal{M}$ and $\mathcal{M}'$, and thus of length $2|\mathcal{M}|$. Throughout any transformation process and until handling the last edge of the cycle, it must be that $|\mathcal{M}^*| < |\mathcal{M}|$ if $\Delta < 2|\mathcal{M}|$.

▶ Remark. In fact, the same situation will occur if $\Delta = 2$. In the particular case of $\Delta = 1$, we'll be in deficit of up to 2 throughout the process – adding the first edge of $\mathcal{M}'$ requires us to delete its two incident edges in $\mathcal{M}$, which already leads to a deficit of 2 units.

5.2 Weighted matchings

In the weighted case, quantifying this deficit throughout the process is more subtle, but the worst-case scenario remains essentially the same: $|\mathcal{M}| = |\mathcal{M}'|$, all edges have weight W and $H = \mathcal{M} \oplus \mathcal{M}'$ is a simple alternating cycle that consists of all edges in $\mathcal{M}$ and $\mathcal{M}'$. Throughout any transformation process and until handling the last edge of the cycle, it must be that $w(\mathcal{M}^*) \leq w(\mathcal{M}) - W$ if $\Delta < 2|\mathcal{M}|$. In general, the deficit to the weight of the matching is inverse linear in Δ, hence taking Δ to be $\Theta(1/\varepsilon)$ ensures that the weight of the matching throughout the process never goes below $(1-\varepsilon)w(\mathcal{M})$. Interestingly, here a similar situation occurs also when $w(\mathcal{M}') > w(\mathcal{M})$. Specifically, consider the same example as above but add η to each edge weight of $\mathcal{M}'$, i.e., assume that the edge weights of $\mathcal{M}$ and $\mathcal{M}'$ are now W and $W' = W + \eta$, respectively. Then the deficit is no longer W as before (for $1 < \Delta < 2|\mathcal{M}|$), but rather $W - \lfloor \Delta/2 \rfloor \cdot \eta$ (for $1 < \Delta < 2|\mathcal{M}|$). Indeed, adding the first edge of $\mathcal{M}'$ requires the deletion of its two incident edges in $\mathcal{M}$, at which stage the deficit is $W - \eta$; from that moment onwards, a single edge of $\mathcal{M}$ is deleted so that another edge of $\mathcal{M}'$ can be added, which reduces η from the deficit each time. Therefore, if $\Delta \cdot \eta = W$, the deficit is always at least $W/2$, while $w(\mathcal{M}') > w(\mathcal{M}) + W/2$. This scenario shows that the bound of Theorem 2 is asymptotically tight.

▶ Remark. If $\Delta = 1$, we'll be in deficit of $2W$ (rather than W) throughout the process, similarly to the unweighted case. In the degenerate case that $\mathcal{M}$ and $\mathcal{M}'$ consist of a single edge each and $\Delta = 1$, the weight after the first edge deletion reduces to 0.

6 Optimality of the Recourse Bound

In this section we show that an approximation guarantee of $(1+\varepsilon)$ requires a recourse bound of $\Omega(1/\varepsilon)$, even in the amortized sense and even in the incremental (insertion only) and decremental (deletion only) settings. We only consider edge updates, but the argument extends seamlessly to vertex updates. This lower bound of $\Omega(1/\varepsilon)$ on the recourse bound does not depend on the update time of the algorithm in any way. Let us fix ε to be any parameter satisfying $\varepsilon = \Omega(1/n), \varepsilon \ll 1$, where n is the (fixed) number of vertices.

Consider a simple path $P_\ell = (v_1, v_2, \ldots, v_{2\ell})$ of length $2\ell - 1$, for an integer $\ell = c(1/\varepsilon)$ such that $\ell \geq 1$ and c is a sufficiently small constant. (Thus P_ℓ spans at least two but no more than n vertices.) There is a single maximum matching $\mathcal{M}_\ell^{OPT}$ for P_ℓ, of size ℓ, which is also the only $(1+\varepsilon)$-MCM for P_ℓ. After adding the two edges (v_0, v_1) and $(v_{2\ell}, v_{2\ell+1})$ to P_ℓ, the maximum matching $\mathcal{M}_\ell^{OPT}$ for the old path P_ℓ does not provide a $(1+\varepsilon)$-MCM for the new path, $(v_0, v_1, \ldots, v_{2\ell+1})$, which we may rewrite as $P_{\ell+1} = (v_1, v_2, \ldots, v_{2(\ell+1)})$. The only way to restore a $(1+\varepsilon)$-approximation guarantee is by removing all ℓ edges of $\mathcal{M}_\ell^{OPT}$ and adding the remaining $\ell + 1$ edges instead, which yields $\mathcal{M}_{\ell+1}^{OPT}$. One may carry out this argument repeatedly until the length of the path reaches, say, $4\ell - 1$. The amortized number of replacements to the matching per update step throughout this process is $\Omega(1/\varepsilon)$. Moreover, the same amortized bound, up to a small constant factor, holds if we start from an empty path instead of a path of length $2\ell - 1$. We then delete all $4\ell - 1$ edges of the final path and start again from scratch, which may reduce the amortized bound by another small constant. In this way we get an amortized recourse bound of $\Omega(1/\varepsilon)$ for the fully dynamic setting.

To adapt this lower bound to the incremental setting, we construct $n' = \Theta(\varepsilon \cdot n)$ vertex-disjoint copies $P^1, P^2, \ldots, P^{n'}$ of the aforementioned incremental path, one after another, in the following way. Consider the ith copy P^i, from the moment its length becomes $2\ell - 1$ and until it reaches $4\ell - 1$. If at any moment during this gradual construction of P^i, the matching

restricted to P^i is not the (only) maximum matching for P^i, we *halt* the construction of P^i and move on to constructing the $(i+1)$th copy P^{i+1}, and then subsequent copies, in the same way. A copy whose construction started but was halted is called *incomplete*; otherwise it is *complete*. (There are also *empty* copies, whose construction has not started yet.) For any incomplete copy P^j, the matching restricted to it is not the maximum matching for P^j, hence its approximation guarantee is worse than $1+\varepsilon$; more precisely, the approximation guarantee provided by any matching other than the maximum matching for P^j is at least $1+c'\cdot\varepsilon$, for a constant c' that can be made as large as we want by decreasing the aforementioned constant c, or equivalently, ℓ. (Recall that $\ell = c(1/\varepsilon)$.) If the matching restricted to P^j is changed to the maximum matching for P^j at some later moment in time, we return to that incomplete copy and resume its construction from where we left off, thereby *temporarily suspending* the construction of some other copy $P_{j'}$. The construction of P^j may get halted again, in which case we return to handling the temporarily suspended copy $P_{j'}$, otherwise we return to handling $P_{j'}$ only after the construction of P^j is complete, and so forth. In this way we maintain the invariant that the approximation guarantee of the matching restricted to any incomplete copy (whose construction is not temporarily suspended) is at least $1+c'\cdot\varepsilon$, for a sufficiently large constant c'. While incomplete copies may get completed later on, a complete copy remains complete throughout the entire update sequence. At the end of the update sequence no copy is empty or temporarily suspended, i.e., any copy at the end of the update sequence is either incomplete or complete. The above argument implies that any complete copy has an amortized recourse bound of $\Omega(1/\varepsilon)$, over the update steps restricted to that copy. Observe also that at least a constant fraction of the n' copies must be complete at the end of the update sequence, otherwise the entire matching cannot provide a $(1+\varepsilon)$-MCM for the entire graph, i.e., the graph obtained from the union of these n' copies. It follows that the amortized recourse bound over the entire update sequence is $\Omega(1/\varepsilon)$.

The lower bound for the incremental setting can be extended to the decremental setting using a symmetric argument to the one given above.

7 Discussion

This paper introduces a natural generalization of the MRP, and provides near-optimal transformations for the problems of MCM and MWM.

One application of this meta-problem is to dynamic graph algorithms. In particular, by building on our transformation for maximum cardinality matching we have shown that any algorithm for maintaining a β-MCM can be transformed into an algorithm for maintaining a $\beta(1+\varepsilon)$-MCM with essentially the same update time as that of the original algorithm and with a worst-case recourse bound of $O(1/\varepsilon)$, for any $\beta \geq 1$ and $\varepsilon > 0$. This recourse bound is optimal for the regime $\beta = 1+\varepsilon$. We also extended this result for weighted matchings, but there is a linear dependence on the aspect-ratio of the graph in the update time and recourse bounds. It would be interesting to improve this dependency to be polylogaritmic in the aspect-ratio.

A natural direction for future work is to study additional basic graph problems under this generalized framework. Although our positive results may lead to the impression that there exists an efficient gradual transformation process to any optimization graph problem, we conclude with a sketch of two trivial hardness results.

For the *maximum independent set problem* any gradual transformation process cannot provide any nontrivial approximation guarantee, regardless of the approximation guarantees of the source and target independent sets. To see this, denote the source approximate

maximum independent set (the one we start from) by $\mathcal{S}$ and the target approximate maximum independent set (the one we gradually transform into) by $\mathcal{S}'$, and suppose there is a complete bipartite graph between $\mathcal{S}$ and $\mathcal{S}'$. Since we cannot add even a single vertex of $\mathcal{S}'$ to the output independent set $\mathcal{S}^*$ (which is initialized as $\mathcal{S}$) before removing from it all vertices of $\mathcal{S}$ and assuming each step of the transformation process makes only Δ changes to $\mathcal{S}^*$, the approximation guarantee of the output independent set must reach $\Omega(|\mathcal{S}'|/\Delta)$ at some moment throughout the transformation process. In other words, the approximation guarantee may be arbitrarily large.

As another example, an analogous argument shows that for the *minimum vertex cover problem*, any gradual transformation process cannot provide an approximation guarantee better than $\frac{|\mathcal{C}|+|\mathcal{C}'|}{|\mathcal{C}'|} > 2$, where $\mathcal{C}$ and $\mathcal{C}'$ are the source and target vertex covers, respectively. On the other hand, one can easily see that the approximation guarantee throughout the entire transformation process does not exceed $\frac{|\mathcal{C}|+|\mathcal{C}'|}{|\mathcal{C}^{OPT}|}$, where $\mathcal{C}^{OPT}$ is a minimum vertex cover for the graph, by gradually adding all vertices of the target vertex cover $\mathcal{C}'$ to the output vertex cover $\mathcal{C}^*$ (which is initialized as $\mathcal{C}$), and later gradually removing the vertices of $\mathcal{C}$ from the output vertex cover $\mathcal{C}^*$.

These examples demonstrate a basic limitation of our generalized framework, and suggest that further research of this framework is required. One interesting direction for further research is studying the maximum independent set and minimum vertex cover problems for bounded degree graphs; note that the trivial hardness results mentioned above do not apply directly to bounded degree graphs. More generally, studying additional combinatorial optimization problems under this framework may contribute to a deeper understanding of its inherent limitations and strengths, and in particular, to finding additional applications of this framework, possibly outside the area of dynamic matching algorithms.

References

1. Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Proc. of 57th FOCS*, pages 335–344, 2016.
2. Spyros Angelopoulos, Christoph Dürr, and Shendan Jin. Online maximum matching with recourse. In *Proc. of 43rd MFCS*, pages 8:1–8:15, 2018.
3. Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *Proc. 45th ICALP*, pages 7:1–7:16, 2018.
4. Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. *CoRR*, abs/1711.03076, 2017. `arXiv:1711.03076`.
5. Sepehr Assadi and Sanjeev Khanna. Randomized composable coresets for matching and vertex cover. In *Proc. of 29th SPAA*, pages 3–12, 2017.
6. Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *Proc. 50th STOC*, 2018.
7. Giorgio Ausiello, Vincenzo Bonifaci, and Bruno Escoffier. Complexity and approximation in reoptimization. In *Computability in Context: Computation and Logic in the Real World*, pages 101–129. World Scientific, 2011.
8. Giorgio Ausiello, Bruno Escoffier, Jérôme Monnot, and Vangelis Th. Paschos. Reoptimization of minimum and maximum traveling salesman's tours. *J. Discrete Algorithms*, 7(4):453–463, 2009.
9. Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Kirk Pruhs, Kevin Schewior, and Clifford Stein. A 2-competitive algorithm for online convex optimization with switching costs. In *Proc. of APPROX-RANDOM*, pages 96–109, 2015.
10. S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Proc. of 52nd FOCS*, pages 383–392, 2011.

11 Michael A Bender, Martin Farach-Colton, Sándor P Fekete, Jeremy T Fineman, and Seth Gilbert. Reallocation problems in scheduling. *Algorithmica*, 73(2):389–409, 2015.

12 Michael A Bender, Martin Farach-Colton, Sándor P Fekete, Jeremy T Fineman, and Seth Gilbert. Cost-oblivious storage reallocation. *TALG*, 13(3):38, 2017.

13 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In *Proc. SODA*, pages 1899–1918, 2019.

14 Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized $o(\log^2 n)$ replacements. In *Proc. of 28th SODA*, pages 692–711, 2018.

15 Aaron Bernstein, Tsvi Kopelowitz, Seth Pettie, Ely Porat, and Clifford Stein. Simultaneously load balancing for every p-norm, with reassignments. In *Proc. ITCS*, pages 51:1–51:14, 2017.

16 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Proc. 42nd ICALP*, pages 167–179, 2015.

17 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proc. of 26th SODA*, pages 692–711, 2016.

18 S. Bhattacharya, M. Henzinger, and D. Nanongkai. Fully dynamic maximum matching and vertex cover in $o(\log^3 n)$ worst case update time. In *Proc. of 28th SODA*, pages 470–489, 2017.

19 Davide Bilò. New algorithms for steiner tree reoptimization. In *Proc. of 45th ICALP*, pages 19:1–19:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.19.

20 Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2):227–251, 2011.

21 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In *Proc. of 44th MFCS*, volume 138 of *LIPIcs*, pages 80:1–80:14, 2019.

22 Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: Pspace-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009.

23 Nicolas Boria and Vangelis Th. Paschos. Fast reoptimization for the minimum spanning tree problem. *J. Discrete Algorithms*, 8(3):296–310, 2010.

24 Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *Proc. 55th FOCS*, pages 384–393, 2014.

25 Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Shortest augmenting paths for online matchings on trees. In *Proc. of 13th WAOA*, pages 59–71, 2015.

26 Bartłomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych-Pawlewicz. A tight bound for shortest augmenting paths on trees. In *Proc. 13th LATIN*, pages 201–216, 2018.

27 Nicolas Bousquet, Tatsuhiko Hatanaka, Takehiro Ito, and Moritz Mühlenthaler. Shortest reconfiguration of matchings. In *Proc. of 45th WG*, pages 162–174. Springer, 2019.

28 Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In *Proc. of PODC*, pages 217–226, 2016.

29 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *Proc. 45th ICALP*, pages 33:1–33:14, 2018.

30 Kamalika Chaudhuri, Constantinos Daskalakis, Robert D. Kleinberg, and Henry Lin. Online bipartite perfect matching with augmentations. In *Proc. INFOCOM*, pages 1044–1052, 2009.

31 P. Gopalan, P. G Kolaitis, E. Maneva, and C. H Papadimitriou. The connectivity of boolean satisfiability: computational and structural dichotomies. *SICOMP*, 38(6):2330–2355, 2009.

32 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proc. of 30th SODA*, pages 1886–1898, 2019.

33 Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Online perfect matching and mobile computing. In *Proc. of 45th Wads*, pages 194–205, 1995.

34 Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. In *Proc. of STOC*, pages 525–534, 2013.

35 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proc. 49th STOC*, pages 537–550, 2017.

36 Anupam Gupta and Amit Kumar. Online steiner tree with deletions. In *Proc. of 25th SODA*, pages 455–467, 2014.

37 Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proc. 25th SODA*, pages 468–479, 2014.

38 M. Gupta and R. Peng. Fully dynamic $(1 + \epsilon)$-approximate matchings. In *54th FOCS*, pages 548–557, 2013.

39 Manoj Gupta, Hitesh Kumar, and Neeldhara Misra. On the complexity of optimal matching reconfiguration. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 221–233. Springer, 2019.

40 Robert A. Hearn and Erik D. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In *Proc. ICALP*, pages 401–413. Springer, 2002.

41 T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *TCS*, 412(12-14):1054–1065, 2011.

42 T. Ito, N. Kakimura, N. Kamiyama, Y. Kobayashi, and Y. Okamoto. Reconfiguration of maximum-weight b-matchings in a graph. *J. Comb. Optim.*, 37(2):454–464, 2019.

43 Takehiro Ito, Naonori Kakimura, Naoyuki Kamiyama, Yusuke Kobayashi, and Yoshio Okamoto. Shortest reconfiguration of perfect matchings via alternating cycles. In *Proc. of 27th ESA*, volume 144 of *LIPIcs*, pages 61:1–61:15, 2019.

44 Marcin Kaminski, Paul Medvedev, and Martin Milanic. Complexity of independent set reconfigurability problems. *Theor. Comput. Sci.*, 439:9–15, 2012.

45 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proc. of 24th SODA*, pages 1131–1142, 2013.

46 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

47 Jannik Matuschke, Ulrike Schmidt-Kraepelin, and José Verschae. Maintaining perfect matchings at low cost. *CoRR*, abs/1811.10580, 2018. `arXiv:1811.10580`.

48 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016.

49 Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *Proc. 47th STOC*, pages 153–162, 2015.

50 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proc. 45th STOC*, pages 745–754, 2013.

51 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):52, 2018.

52 K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proc. of 42nd STOC*, pages 457–464, 2010.

53 D. Peleg and S. Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: A density-sensitive approach. In *Proc. of 26th SODA*, 2016.

54 Baruch Schieber, Hadas Shachnai, Gal Tamir, and Tami Tamir. A theory and algorithms for combinatorial reoptimization. *Algorithmica*, 80(2):576–607, 2018.

55 Noam Solomon and Shay Solomon. A generalized matching reconfiguration problem. *CoRR*, abs/1803.05825, 2020. `arXiv:1803.05825`.

56 S. Solomon. Fully dynamic maximal matching in constant update time. In *Proc. 57th FOCS*, pages 325–334, 2016.

57 Babacar Thiongane, Anass Nagih, and Gérard Plateau. Lagrangean heuristics combined with reoptimization for the 0–1 bidimensional knapsack problem. *Discrete applied mathematics*, 154(15):2200–2211, 2006.

58 Jan van den Heuvel. The complexity of change. In Simon R. Blackburn, Stefanie Gerke, and Mark Wildon, editors, *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013.

59 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *Proc. of 52nd STOC*, pages 194–207. ACM, 2020.