Modularity-Driven Clustering of Dynamic Graphs*

Robert Görke[1], Pascal Maillard[2], Christian Staudt[1], and Dorothea Wagner[1]

[1] Institute of Theoretical Informatics
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{rgoerke,christian.staudt,wagner}@ira.uka.de
[2] Laboratoire de Probabilités et Modèles Aléatoires
Université Pierre et Marie Curie (Paris VI), Paris, France
pascal.maillard@upmc.fr

Abstract. Maximizing the quality index *modularity* has become one of the primary methods for identifying the clustering structure within a graph. As contemporary networks are not static but evolve over time, traditional static approaches can be inappropriate for specific tasks. In this work we pioneer the NP-hard problem of online *dynamic modularity* maximization. We develop scalable dynamizations of the currently fastest and the most widespread static heuristics and engineer a heuristic dynamization of an optimal static algorithm. Our algorithms efficiently maintain a *modularity*-based clustering of a graph for which dynamic changes arrive as a stream. For our quickest heuristic we prove a tight bound on its number of operations. In an experimental evaluation on both a real-world dynamic network and on dynamic clustered random graphs, we show that the dynamic maintenance of a clustering of a changing graph yields higher *modularity* than recomputation, guarantees much smoother clustering dynamics and requires much lower runtimes. We conclude with giving recommendations for the choice of an algorithm.

1 Introduction

Graph clustering is concerned with identifying and analyzing the group structure of networks. Generally, a partition (i.e., a clustering) of the set of nodes is sought, and the size of the partition is a priori unknown. A plethora of formalizations for what a *good* clustering is exist, good overviews are, e.g., [21, 3]. In this work we set our focus on the quality function *modularity*, coined by Girvan and Newman [4], which has proven itself feasible and reliable in practice, especially as a target function for maximization (see [2] for further references), which follows the paradigm of parameter-free community discovery [5]. The foothold of this work is that most networks in practice are not static. Iteratively clustering snapshots of a dynamic graph from scratch with a static method has several disadvantages: First, runtime cannot be neglected for large instances or

*This work was partially supported by the DFG under grant WA 654/15-1. The full version of this extended abstract is available as a technical report [1].

environments where computing power is limited [6], even though very fast clustering methods have been proposed recently [7, 8]. Second, heuristics for the NP-hard [2] optimization of *modularity* suffer from local optima—this might be avoided by dynamically maintaining a good solution. Third, static heuristics are known not to react in a continuous way to small changes in a graph. The lefthand figure illustrates the general situation for updating clusterings. A graph G is updated by some change Δ, yielding G'. We investigate procedures $\mathcal{A}$ that update the clustering $\mathcal{C}(G)$ to $\mathcal{C}'(G')$ without reclustering from scratch, but work towards the same aim as a static technique $\mathcal{T}$ does.

$$\begin{tikzcd}
G \arrow[r, "\Delta"] \arrow[d, "\mathcal{T}"'] & G' \arrow[d, "\mathcal{T}"] \\
\mathcal{C}(G) \arrow[r, dashed, "\mathcal{A}"] & \mathcal{C}'(G')
\end{tikzcd}$$

Fig. 1: Problem setting

Related Work. Dynamic graph clustering has so far been a rather untrodden field. Recent efforts [9] yielded a method that can provably dynamically maintain a clustering that conforms to a specific bottleneck-quality requirement. Apart from that, there have been attempts to track communities over time and interpret their evolution, using static snapshots of the network, e.g. [10, 11], besides an array of case studies. In [12] a parameter-based dynamic graph clustering method is proposed which allows user exploration. Parameters are avoided in [13] where the minimum description length of a graph sequence is used to determine changes in clusterings and the number of clusters. In [14] an explicitly bicriterial approach for low-difference updates and a *partial* ILP are proposed, the latter of which we also discuss. To the best of our knowledge no fast procedures for updating *modularity*-based clustering in general dynamic graphs have been proposed yet. Beyond graph theory, in data mining the issue of clustering an evolving data set has been addressed in, e.g., [15], where the authors share our goal of finding a smooth dynamic clustering. The literature on static *modularity*-maximization is quite broad and we recommend [2, 3, 16] for further reading. Spectral methods, e.g., [17], and techniques based on random walks [18, 19], do not lend themselves well to dynamization due to their non-continuous nature. Variants of greedy agglomeration [20, 7], however, work well, as we shall see.

Our Contribution. In this work we present, analyze and evaluate a number of concepts for efficiently updating *modularity*-driven clusterings. We prove the NP-hardness of dynamic *modularity* optimization and develop heuristic dynamizations of the most widespread [20] and the fastest [7] static algorithms, alongside apt strategies to determine the search space. For our fastest procedure, we can prove a tight bound of $\Theta(\log n)$ on the expected number of operations required. We then evaluate these and a heuristic dynamization of an ILP. We compare the algorithms with their static counterparts and evaluate them experimentally on random preclustered dynamic graphs and on large real-world instances. We reveal that the dynamic maintenance of a clustering yields higher quality than recomputation, smoother clustering dynamics and lower runtimes.

Notation. Throughout this paper, we will use the notation of [21]. We assume that $G = (V, E, \omega)$ is an undirected, weighted, and simple graph with the edge weight function $\omega\colon E \to \mathbb{R}_{\geq 0}$. We set $|V| =: n, |E| =: m$ and $\mathcal{C} = \{C_1, \ldots, C_k\}$ to be a partition of V. We call $\mathcal{C}$ a *clustering* of G and sets C_i *clus-*

ters. $\mathcal{C}(v)$ is $C \ni v$. A clustering is *trivial* if either $k = 1$ ($\mathcal{C}^{\mathbf{1}}$), or all clusters contain only one element, i.e., are *singletons* ($\mathcal{C}^V$). We identify a cluster C_i with its node-induced subgraph of G. Then $E(\mathcal{C}) := \bigcup_{i=1}^{k} E(C_i)$ are *intra-cluster* edges and $E \setminus E(\mathcal{C})$ *inter-cluster* edges, with cardinalities $m(\mathcal{C})$ and $\overline{m}(\mathcal{C})$, respectively. Further, we generalize degree $\deg(v)$ to clusters as $\deg(C) := \sum_{v \in C} \deg(v)$. When using edge weights, all the above definitions generalize naturally by using $\omega(e)$ instead of 1 when counting edge e. Weighted node degrees are called $\omega(v)$. A *dynamic graph* $\mathcal{G} = (G_0, \ldots, G_{t_{\max}})$ is a sequence of graphs, with $G_t = (V_t, E_t, \omega_t)$ being the state of the dynamic graph at time step t. The *change* $\Delta(G_t, G_{t+1})$ between timesteps comprises a sequence of b *atomic* events on G_t, which we detail later. We have the sequence of changes arrive as a stream.

The Quality Index *Modularity*. In this work we set our focus on *modularity* [4], a measure for the goodness of a clustering. Just like any other quality index for clusterings (see, e.g., [21, 3]), *modularity* does have certain drawbacks such as *non-locality* and *scaling behavior* [2] or *resolution limit* [22]. However, being aware of these peculiarities, *modularity* can very well be considered a useful measure that closely agrees with intuition on a wide range of real-world graphs, as observed by myriad studies. *Modularity* can be formulated as

$$\operatorname{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2 \quad \text{(weighted analogous)} \; . \qquad (1)$$

Roughly speaking, *modularity* measures the fraction of edges which are covered by a clustering and compares this value to its expected value, given a random rewiring of the edges which, on average, respects node degrees. This definition generalizes in a natural way as to take edge weights $\omega(e)$ into account, for a discussion thereof see [23] and [24]. ModOpt, the problem of optimizing *modularity* is NP-hard [2], but *modularity* can be computed in linear time and lends itself to a number of greedy maximization strategies. For the dynamic setting, the following corollary corroborates the use of heuristics (see [1] for a proof).

Corollary 1 (DynModOpt is NP-hard). *Given graph G, a modularity-optimal clustering $\mathcal{C}^{\text{opt}}(G)$ and an atomic event Δ to G, yielding G'. It is NP-hard to find a modularity-optimal clustering $\mathcal{C}^{\text{opt}}(G')$.*

Measuring the Smoothness of a Dynamic Clustering. By comparing consecutive clusterings, we quantify how smooth an algorithm manages the transition between two steps, an aspect which is crucial to both readability and applicability. An array of measures exist that quantify the (dis)similarity between two partitions of a set; for an overview and further references, see [25]. Our results strongly suggest that most of these widely accepted measures are qualitatively equivalent in all our (non-pathological) instances (see full version [1]). We thus restrict our view to the *(graph-structural) Rand* index [25], being a well known representative; it maps two clusterings into the interval $[0, 1]$, i.e., from equality to maximum dissimilarity: $\mathcal{R}_g(\mathcal{C}, \mathcal{C}') := 1 - (|E_{11}| + |E_{00}|)/m$, with $E_{11} = \{\{v, w\} \in E : \mathcal{C}(v) = \mathcal{C}(w) \wedge \mathcal{C}'(v) = \mathcal{C}'(w)\}$, and E_{00} the analog for inequality. We use the intersection of two graphs when comparing their clusterings. *Low* distances correspond to *smooth* dynamics.

2 The Clustering Algorithms

Formally, a dynamic clustering algorithm is a procedure which, given the previous state of a dynamic graph G_{t-1}, a sequence of graph events $\Delta(G_{t-1}, G_t)$ and a clustering $\mathcal{C}(G_{t-1})$ of the previous state, returns a clustering $\mathcal{C}'(G_t)$ of the current state. While the algorithm may discard $\mathcal{C}(G_{t-1})$ and simply start from scratch, a good dynamic algorithm will harness the results of its previous work. A natural approach to dynamizing an agglomerative clustering algorithm is to break up those local parts of its previous clustering, which are most likely to require a reassessment after some changes to the graph. The half finished instance is then given to the agglomerative algorithm for completion. A crucial ingredient thus is a *prep strategy* S which decides on the search space which is to be reassessed. We will discuss such strategies later, until then we simply assume that S breaks up a reasonable part of $\mathcal{C}(G_{t-1})$, yielding $\tilde{\mathcal{C}}(G_{t-1})$ (or $\tilde{\mathcal{C}}(G_t)$ if including the changes in the graph itself). We call $\tilde{\mathcal{C}}$ the *preclustering* and nodes that are chosen for individual reassessment *free* (can be viewed as singletons).

Formalization of Graph Events. We describe our test instances in more detail later, but for a proper description of our algorithms, we now briefly formalize the graph events we distinguish. Most commonly *edge creations* and *removals* take place, and they require the incident nodes to be present before and after the event. Given edge *weights*, changes require an edge's presence. *Node creations* and *removals* in turn only handle degree zero nodes, i.e., for an intuitive node deletion we first have to remove all incident edges. To summarize such compound events we use *time step* events, which indicate to an algorithm that an updated clustering must now be supplied. Between time steps it is up to the algorithm how it maintains its intermediate clustering. Additionally, *batch* updates allow for only running an algorithm after a scalable number of b timesteps.

2.1 Algorithms for Dynamic Updates of Clusterings

The Global Greedy Algorithm. The most prominent algorithm for *modularity* maximization is a global greedy algorithm [20], which we call Global (Alg. 1). Starting with singletons, for each pair of clusters, it determines the increase in *modularity* dQ that can be achieved by merging the pair and performs the most beneficial merge. This is repeated until no more improvement is possible. As the **static** (pseudo-dynamic) algorithm sGlobal[3], we let this algorithm cluster from scratch at each timestep for comparison. By passing a *preclustering* $\tilde{\mathcal{C}}(G_t)$ to Global we can define the properly **dynamic** algorithm dGlobal. Starting from $\tilde{\mathcal{C}}(G_t)$ this algorithm lets Global perform greedy agglomerations of clusters.

Alg. 1: Global$(G, \mathcal{C})$

```
1 while $\exists C_i, C_j \in \mathcal{C} : \mathrm{dQ}(C_i, C_j) \geq 0$ do
2     $(C_1, C_2) \leftarrow \arg\max_{C_i, C_j \in \mathcal{C}} \mathrm{dQ}(C_i, C_j)$
3     merge$(C_1, C_2)$
```

[3]For historical reasons, sGlobal appears in plots as StaticNewman, dGlobal as Newman, sLocal as StaticBlondel and dLocal as Blondel, based on the algorithms' authors.

The Local Greedy Algorithm. In a recent work [7] the simple mechanism of the aforementioned Global has been modified as to rely on local decisions (in terms of graph locality), yielding an extremely fast and efficient maximization. Instead of looking globally for the best merge of two clusters, Local, as sketched out in Alg. 2, repeatedly lets each node consider moving to one of its neighbors' clusters, if this improves *modularity*; this potentially merges clusters, especially when starting with singletons. As soon as no more nodes move, the current clustering is *contracted*, i.e., each cluster is contracted to a single node, and adjacencies and edge weights between them are summarized. Then, the process is repeated on the resulting graph which constitutes a higher level of abstraction; in the end, the highest level clustering is decisive about the returned clustering: The operation unfurl assigns each elementary node to a cluster represented by the highest level cluster it is contained in. We again sketch out an algorithm which serves as the core for both a static and a dynamic variant of this approach, as shown in Alg. 2. As the input, this algorithm takes a hierarchy of graphs and clusterings and a search space policy P. Policy P affects the graph *contractions*, in that P decides which nodes of the next level graph should be free to move. Note that the input hierarchy can also be flat, i.e., $h_{\max} = 0$, then line 11 creates all necessary higher levels. Again posing as a pseudo-dynamic algorithm, the **static variant** (as in [7]), sLocal, passes only $(G_t,\tilde{\mathcal{C}}^V)$ to Local, such that it starts with singletons and all nodes freed, instead of a proper *preclustering*. Policy P is set to tell the algorithm to start from scratch on all higher levels and to not work on previous results in line 11, i.e., in $\tilde{\mathcal{C}}^{h+1}$ again all nodes in the contraction are free singletons. The **dynamic variant** dLocal remembers its old results. It passes the changed graph, a current *preclustering* of it and all higher-level contracted structures from its previous run to Local: $(G_t, G_{\text{old}}^{1,\dots,h_{\max}}, \tilde{\mathcal{C}}, \mathcal{C}_{\text{old}}^{1,\dots,h_{\max}}, P)$. In level 0, the preclustering $\tilde{\mathcal{C}}$ defines the set of free nodes. In levels beyond 0, policy P is set to have the contract-procedure free only those nodes of the next level, that have been affected by lower level changes (or their neighbors as well, tunable by policy P). Roughly speaking, dLocal starts by letting all free (elementary) nodes reconsider their cluster. Then it lets all those (super-)nodes on higher levels reconsider their cluster, whose content has changed due to lower level revisions.

Alg. 2: Local$(G^{0\dots h_{\max}}, \mathcal{C}^{0\dots h_{\max}}, P)$

```
1  $h \leftarrow 0$
2  repeat
3      $(G,\mathcal{C}) \leftarrow (G^h,\mathcal{C}^h)$
4      repeat
5          forall free $v \in V$ do
6              if $\max_{v \in N(u)} \mathrm{dQ}_{uv} \geq 0$ then
7                  $w \leftarrow \arg\max_{v \in N(u)} \mathrm{dQ}_{uv}$
8                  move$(u,\mathcal{C}(w))$
9      until no more changes
10     $\mathcal{C}^h \leftarrow \mathcal{C}$
11     $(G^{h+1},\tilde{\mathcal{C}}^{h+1}) \leftarrow \mathrm{contract}_P(G^h,\mathcal{C}^h)$
12     $h \leftarrow h+1$
13 until no more real contractions
14 $\mathcal{C}(G^0) \leftarrow \mathrm{unfurl}(\mathcal{C}^{h-1})$
```

ILP. While optimality is out of reach, the problem *can* be cast as an ILP [2]. A distance relation X_{uv} indicates whether elements u and v are in the same cluster, and simple constraints keep these X-variables consistent. Since runtimes for the full ILP reach days for more than 200 nodes, a promising idea pioneered in [14] is to solve a *partial ILP* (pILP). Such a program takes a *preclustering*—of much smaller complexity—as the input, and solves this instance, i.e., finishes the clustering, optimally via an ILP; a singleton *preclustering* yields a full ILP. We introduce two variants, (i) the argument noMerge prohibits merging *pre-clusters*, and only allows free nodes to join clusters or form new ones, and (ii) merge allows existing clusters to merge. For both variants we need to add constraints and terms to the standard formulation using solely variables X_{uv}. Roughly speaking, for (i), variables Y_{uC} indicating the distance of node u to cluster C are introduced constraints ensure their consistency with the X-variables; for (ii), we additionally need variables $Z_{CC'}$ for the distance between clusters, constrained just as X_{uv}. See the full paper [1] for details on all these ILP formulations. The dynamic clustering algorithms which first solicit a *preclustering* and then call pILP are called dILP. Note that they react on any edge event; accumulating events until a timestep occurs can result in prohibitive runtimes.

Elemental Optimizer The *elemental operations optimizer*, EOO, performs a limited number of operations, trying to increase the quality. Specifically, we allow moving or splitting off nodes and merging clusters, as listed in Table 1. Although rather limited in its options, EOO or very similar tools for local optimization are often used as post-processing tools (see [26] for a discussion). Our algorithm dEOO simply calls EOO at each time step.

Table 1: EOO operations, allowed/disallowed via parameters

Operation	Effect
merge(u,v)	$\mathcal{C}(u) \cup \mathcal{C}(v)$
shift(u,v)	$\mathcal{C}(u) - u, \mathcal{C}(v) + u$
split(u)	$(\{u\}, \mathcal{C}(u) \setminus u) \leftarrow \mathcal{C}(u)$

2.2 Strategies for Building the Preclustering

We now describe *prep strategies* which generate a *preclustering* $\tilde{\mathcal{C}}$, i.e., define the search space. We distinguish the *backtrack strategy*, which refines a clustering, and *subset strategies*, which free nodes. The rationale behind the *backtrack strategy* is that selectively backtracking the clustering produced by Global enables it to respect changes to the graph. On the other hand, *subset strategies* are based on the assumption that the effect of a change on the clustering structure is necessarily local. Both output a half-finished *preclustering*.

The *backtrack strategy* (BT) records the merge operations of Global and backtracks them if a graph modification suggests their reconsideration. We detail in the full paper [1] what we mean by "suggests", but for brevity we just state that the actions listed for BT provably require very little asymptotic effort and offer Global a good chance to find an improvement. Speaking intuitively, the reactions to a change in (non-)edge $\{u, v\}$ are as follows (weight changes are analogous): For intra-cluster additions we backtrack those merge operations that led to u and v being in the same cluster and allow Global to find a tighter cluster for them, i.e., we separate them. For inter-cluster additions we track back u

and v individually, until we isolate them as singletons, such that Global can reclassify and potentially merge them. Inter-cluster deletions are not reacted on. On intra-cluster deletions we again isolate both u and v such that Global may have them find separate clusters. Note that this strategy is only applicable to Global; conferring it to Local is neither straightforward nor promising as Local is based on node *migrations* in addition to *agglomerations*. Anticipating this strategy's low runtime, we can give a bound on the expected number of backtrack steps for a single call of the crucial operation isolate (proven in the full paper [1]).

Theorem 1. *Assume that a backtrack step divides a cluster randomly. Then, for the number I of steps isolate(v) requires, it holds: $E\{I\} \in \Theta(\ln n)$.*

A *subset strategy* is applicable to all dynamic algorithms. It frees a subset $\tilde{V}$ of individual nodes that need reassessment and extracts them from their clusters. We distinguish three variants which are all based on the hypothesis that local reactions to graph changes are appropriate. Consider an edge event involving $\{u, v\}$. The *breakup strategy* (BU) marks the affected clusters $\tilde{V} = \mathcal{C}(u) \cup \mathcal{C}(v)$; the *neighborhood strategy* (N_d) with parameter d marks $\tilde{V} = N_d(u) \cup N_d(v)$, where $N_d(w)$ is the d-hop neighborhood of w; the *bounded neighborhood strategy* (BN_s) with parameter s marks the first s nodes found by a breadth-first search simultaneously starting from u and v.

3 Experimental Evaluation of Dynamic Algorithms[4]

Instances. We use both generated graphs and real-world instances. We briefly describe them here, but for more details please see [27] and [14].

Random Graphs {*ran*}. Our Erdős-Rényi-type generator builds upon [28] and adds to this dynamicity in all graph elements and in the clustering, i.e., nodes and edges are inserted and removed and ground-truth clusters merged and split, always complying with sound probabilities. The generator's own clustering serves as a reference to compare our algorithms to, see [27] for details. In later plots we use selected random instances, however, descriptions apply to all such graphs.[4]

EMail Graph $\mathcal{G}_e$. The network of email contacts at the department of computer science at KIT is an ever-changing graph with an inherent clustering: Workgroups and projects cause increased communication. We weigh edges by the number of exchanged emails during the past seven days, thus edges can completely time out; degree-0 nodes are removed from the network. $\mathcal{G}_e$ has between 100 and 1500 nodes depending on the time of year, and about 700K events spanning about 2.5 years. It features a strong power-law degree distribution.

arXiv Graphs {*arx*}. Since 1992 the *arXiv.org e-Print archive*[5] is a popular repository for scientific e-prints, stored in several categories alongside timestamped metadata. We extracted networks of collaboration between scientists

[4]For many more experimental results and plots as well as for implementation notes see the full paper [1], supplementary information is stored at i11www.iti.uni-karlsruhe.de/projects/spp1307/dyneval

[5]Website of e-print repository: arxiv.org

based on coauthorship. E-prints induce equally weighted clique-edges among the contributors such that each author gains a total edge weight of 1.0 per e-print contributed to. E-prints time out after two years and disconnected authors are removed.[5] As these networks are ill-natured for local updates, we use them as tough trials. We show results on two categories with large connected components.

Fundamental Results. For the sake of readability, we use a moving average in plots for distance and quality in order to smoothen the raw data. We consider the criteria quality (*modularity*), smoothness ($\mathcal{R}_g$) and runtime (ms), and additionally $|\mathcal{C}|$ as a structural indicator.

Discarding dEOO. In a first feasibility test, dEOO immediately falls behind all other algorithms in terms of quality (see full paper [1]), an observation substantiated by the fact that dEOO works better if related to some base algorithm [26]. Moreover, runtimes for dEOO as the sole technique are infeasible for large graphs.

Local Parameters. It has been stated in [7] that the order in which Local considers nodes is irrelevant. In terms of average runtime and quality we can confirm this for sLocal, though a random order tends to be less smooth; for dLocal the same observation holds (see full version [1]). However, since node order *does* influence specific values, a random order can compensate the effects this might have in pathological cases. Considering only affected nodes or also their neighbors in higher levels, does not affect any criterion on average.

pILP Variants. Allowing the ILP to merge existing clusters takes longer, and clusters coarser and with a slightly worse *modularity*; we therefore reject it.

Heuristics vs. dILP. A striking observation about dILP is the fact that it yields worse quality than dLocal and sLocal with identical *prep strategies*. Being locally optimal seems to overfit, a phenomenon that does not weaken over time and persists throughout most instances. Together with its high runtime and only small advantages in smoothness, dILP is ill-suited for updates on large graphs.

Static Algorithms. Briefly comparing sGlobal and sLocal we can state that sLocal consistently yields better quality and a finer yet less smooth clustering (see full version [1]). This generally applies to the corresponding dynamic algorithms as well. In terms of speed, however, sGlobal hardly lags behind sLocal, especially for small graphs with many connected components, where sLocal cannot capitalize on its strength of quickly reducing the size of a large instance. For such instances, separately maintaining and handling connected components could thus reasonably speed up sLocal, but would also do so for sGlobal.

Prep Strategies. We now determine the best choice of *prep strategies* and their parameters for dGlobal and dLocal. In particular, we evaluate N_d for $d \in \{0, 1, 2, 3\}$ and BN_s for $s \in \{2, 4, 8, 16, 32\}$, alongside BU and BT. Throughout our experiments $d = 0$ (or $s = 2$) proved insufficient, and is therefore ignored in the following. For dLocal, increasing d has only a marginal effect on quality and smoothness, while runtime grows sublinearly, which suggests $d = 1$. For dGlobal, N_d risks high runtimes for depths $d > 1$, especially for dense graphs. In terms of quality N_1 is the best choice, higher depths seem to deteriorate quality—a strong indication that large search spaces contain local optima. Smoothness approaches the bad values of sGlobal for $d > 2$. For BN, increasing s is essen-

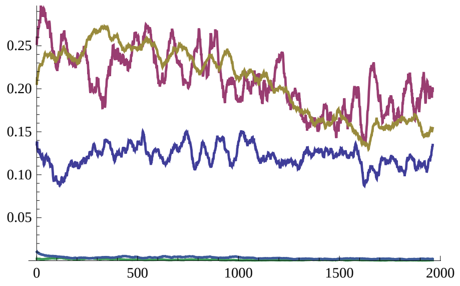

Fig. 2: $\mathcal{R}_g$, {ran} (top to bottom at right end): **sGlobal** (1st) and **sLocal** (2nd) are less smooth (factor 100) than **dLocal@BN**$_4$, **dGlobal@BN**$_{16}$ (bottom); **dGlobal@BT** (3rd) competes well.

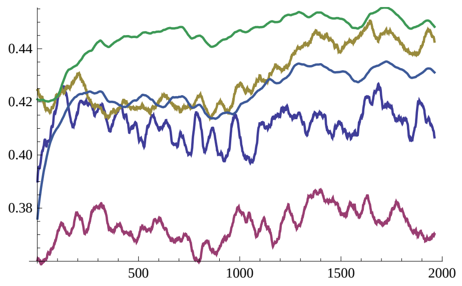

Fig. 3: *Modularity*, {ran} (top to bottom at right end): **dGlobal@BT** (4th) and **dGlobal@BN**$_{16}$ (3rd) beat **sGlobal** (5th); **dLocal@BN**$_4$ (1st) beats **sLocal** (2nd).

tially equivalent to increasing d, only on a finer scale. Consequently, we can report similar observations. For dLocal, BN$_4$ proved slightly superior. dGlobal's quality benefits from increasing s in this range, but again at the cost of speed and smoothness, so that BN$_{16}$ is a reasonable choice. BU clearly falls behind in terms of all criteria compared to the other strategies, and often mimics the static algorithms. dGlobal using BT is by far the fastest algorithm, confirming our theoretical predictions from Sec. 2.2, but still produces competitive quality. However, it often yields a smoothness in the range of sGlobal. Summarizing, our best dynamic candidates are the algorithms dGlobal@BT and dGlobal@BN$_{16}$ (achieving a speedup over sGlobal of up to 1k and 20 at 1k nodes, respectively) and algorithm dLocal@BN$_4$(speedup of 5 over sLocal).

Comparison of the Best. As a general observation, as depicted in Fig. 3, each dynamic candidate beats its static counterpart in terms of *modularity*. On the generated graphs, dLocal is superior to dGlobal, and faster. In terms of smoothness (Fig. 2), dynamics (except for dGlobal@BT) are superior to statics by a factor of ca. 100, but even dGlobal@BT beats them.

Trials on *arXiv* Data. As an independent data set, we use our *arXiv* grahps for testing our results from $\mathcal{G}_e$ and the random instances. These graphs consist solely of glued cliques of authors (papers), established within single timesteps where potentially many new nodes and edges are introduced. Together with modularity's *resolution limit* [22] and its fondness of balanced clusters and a non-arbitrary number thereof in large graphs [30], these degenerate dynamics are adequate for fooling local algorithms that cannot regroup cliques all over as to modularity's liking: Static algorithms constantly reassess a growing component, while dynamics using N or BN will sometimes have no choice but to further enlarge some growing cluster. Locally this is a good choice, but globally some far-away cut might qualify as an improvement over pure componentwise growth.

However, we measured that dGlobal@BT easily keeps up with the static algorithms' *modularity*, being able to adapt its number of clusters appropriately. The dynamic algorithms using other *prep strategies* do struggle to make up for their inability to re-cluster; however, they still only lag behind by about 1%. Figures 4 and 5 show *modularity* for coarse and fine batches, respectively, using the *arXiv* category *Nuclear Theory* (1992-2010, 33K e-prints, 200K elementary events, 14K authors). As before, dynamics are faster and smoother. For the coarse batches, speedups of 10 to 2K (BT) are attained; for fine batches, these are 100 to 2K. In line with the above observations, their clusterings are slightly coarser (except for dGlobal@BT) (see full paper [1] for further insights).

Summary of Insights. The outcomes of our evaluation are very favorable for the dynamic approach in terms of all three criteria. Furthermore, the dynamics exhibit the ability to react quickly and adequately to changes in the random generator's ground-truth clustering (see full paper [1]).

We observed that dLocal is less susceptible to an increase of the search space than dGlobal. However, our results argue strongly for the locality assumption in both cases—an increase in the search space beyond a very limited range is not justified when trading off runtime against quality. On the contrary, quality and smoothness may even suffer for dLocal. Consequently, N and BN strategies with a limited range are capable of producing high-quality clusterings while excelling at smoothness. The BT strategy for dGlobal yields competitive quality at unrivaled speed, but at the expense of smoothness. For dLocal a gradual improvement of quality and smoothness over time is observable, which can be interpreted as an effect reminiscent of *simulated annealing*, a technique that has been shown to work well for *modularity* maximization [29]. Our data indicates that the best choice for an algorithm in terms of quality may also depend on the nature of the target graph. While dLocal surpasses dGlobal on almost all generated graphs, dGlobal is superior on our real-world instance $\mathcal{G}_e$. We speculate that this is due to $\mathcal{G}_e$ featuring a power law degree distribution in contrast to the Erdős-Rényi-type generated instances. In turn, our *arXiv* trial graphs, which grow and shrink in a

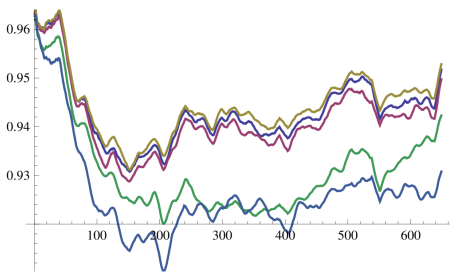

Fig. 4: *Modularity*, {arx}, batch size 50 e-prints (top to bottom at right end): Backtracking (**dGlobal@BT**) (2nd) easily follows the static algorithms (**sLocal** (1st) and **sGlobal** (3rd)); even **dLocal@BN**$_4$ (4th) and **dGlobal@BN**$_{16}$ (5th) lag behind by only $\sim 1\%$.

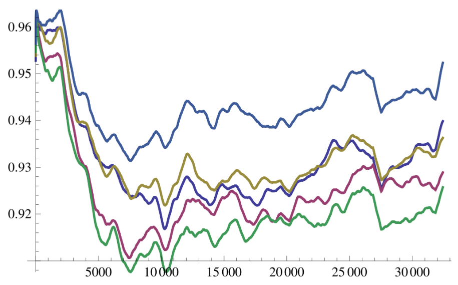

Fig. 5: *Modularity*, {arx}, batch size 1 e-print, dynamics only (top to bottom at right end): **dGlobal@BT** (1st) excels, followed by **dLocal@N**$_1$ (3rd) and **dLocal@BN**$_4$ (2nd) and then **dGlobal@BN**$_{16}$ (4th) and **dGlobal@N**$_1$ (5th) whom finer batches don't help.

volatile but local manner, allow a for a small margin of quality improvement, if the clustering is regularly adapted globally (re-balanced and coarsened/refined). Only the statics and dGlobal@BT are able to do this, however, at the cost of smoothness. Universally, the latter algorithm is the fastest. Concluding, some dynamic algorithm always beats the static algorithms; backtracking is preferable for locally concentrated or monotonic graph dynamics and a small search space is to be used for randomly distributed changes in a graph.

4 Conclusion

As the first work on *modularity*-driven clustering of dynamic graphs, we deal with the NP-hard problem of updating a *modularity*-optimal clustering after a change in the graph. We developed dynamizations of the currently fastest and the most widespread heuristics for *modularity*-maximization and evaluated them and a dynamic partial ILP for local optimality. For our fastest update strategy, we can prove a tight bound of $\Theta(\log n)$ on the expected number of backtrack steps required. Our experimental evaluation on real-world dynamic networks and on dynamic clustered random graphs revealed that dynamically maintaining a clustering of a changing graph does not only save time, but also yields higher *modularity* than recomputation—except for degenerate graph dynamics—and guarantees much smoother clustering dynamics. Moreover, heuristics are better than being locally optimal at this task. Surprisingly small search spaces work best, avoid trapping local optima well and adapt quickly and aptly to changes in the ground-truth clustering, which strongly argues for the assumption that changes in the graph ask for local updates on the clustering.

References

1. Görke, R., Maillard, P., Staudt, C., Wagner, D.: Modularity-Driven Clustering of Dynamic Graphs. Technical report, Universität Karlsruhe (TH) (2010), Informatik, TR 2010-5.
2. Brandes, U., Delling, D., Gaertler, M., Görke, R., Höfer, M., Nikoloski, Z., Wagner, D.: On Modularity Clustering. IEEE TKDE **20**(2) (2008) 172–188
3. Fortunato, S.: Community detection in graphs. Elsevier Phys. R. **486**(3–5) (2009)
4. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69**(026113) (2004)
5. Keogh, E., Lonardi, S., Ratanamahatana, C.A.: Towards Parameter-Free Data Mining. In: Proc. of the 10th ACM SIGKDD Int. Conf., ACM (2004) 206–215
6. Schaeffer, S.E., Marinoni, S., Särelä, M., Nikander, P.: Dynamic Local Clustering for Hierarchical Ad Hoc Networks. In: Proc. of Sensor and Ad Hoc Communications and Networks, 2006. Volume 2., IEEE 667–672
7. Blondel, V., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: The. and Exp. **2008**(10)
8. Delling, D., Görke, R., Schulz, C., Wagner, D.: ORCA Reduction and ContrAction Graph Clustering. In Goldberg, A.V., Zhou, Y., eds.: Proc. of the 5th Int. Conf. on Alg. Asp. in Information and Management. LNCS 5564, Springer (2009) 152–165
9. Görke, R., Hartmann, T., Wagner, D.: Dynamic Graph Clustering Using Minimum-Cut Trees. In: Alg. and Data Structures, 11th Int. WS. LNCS 5664, Springer (2009)

10. Hopcroft, J.E., Khan, O., Kulis, B., Selman, B.: Tracking Evolving Communities in Large Linked Networks. Proceedings of the National Academy of Science of the United States of America **101** (April 2004)
11. Palla, G., Barabási, A.L., Vicsek, T.: Quantifying social group evolution. Nature **446** (April 2007) 664–667
12. Aggarwal, C.C., Yu, P.S.: Online Analysis of Community Evolution in Data Streams. [31]
13. Sun, J., Yu, P.S., Papadimitriou, S., Faloutsos, C.: GraphScope: Parameter-Free Mining of Large Time-Evolving Graphs. In: Proc. of the 13th ACM SIGKDD Int. Conference, ACM Press (2007) 687–696
14. Hübner, F.: The Dynamic Graph Clustering Problem - ILP-Based Approaches Balancing Optimality and the Mental Map. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik (May 2008)
15. Chakrabarti, D., Kumar, R., Tomkins, A.S.: Evolutionary Clustering. In: Proc. of the 12th ACM SIGKDD Int. Conference, ACM Press (2006) 554–560
16. Schaeffer, S.E.: Graph Clustering. Computer Science Review **1**(1) (2007) 27–64
17. White, S., Smyth, P.: A Spectral Clustering Approach to Finding Communities in Graphs. [31] 274–285
18. Pons, P., Latapy, M.: Computing Communities in Large Networks Using Random Walks. Journal of Graph Algorithms and Applications **10**(2) (2006) 191–218
19. van Dongen, S.M.: Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht (2000)
20. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E **70**(066111) (2004)
21. Brandes, U., Erlebach, T., eds.: Network Analysis: Methodological Foundations. Volume 3418 of Lecture Notes in Computer Science. Springer (February 2005)
22. Fortunato, S., Barthélemy, M.: Resolution limit in community detection. PNAS **104**(1) (2007) 36–41
23. Newman, M.E.J.: Analysis of Weighted Networks. P. R. E**70**(056131) (2004) 1–9
24. Görke, R., Gaertler, M., Hübner, F., and Wagner, D.: Computational Aspects of Lucidity-Driven Graph Clustering. *JGAA*, **14**(2) (2010)
25. Delling, D., Gaertler, M., Görke, R., Wagner, D.: Engineering Comparators for Graph Clusterings. In: Proc. of the 4th Int. Conf. on Alg. Asp. in Information and Management (AAIM'08). LNCS 5034., Springer (June 2008) 131–142
26. Noack, A., Rotta, R.: Multi-level Algorithms for Modularity Clustering. In: Proc. of the 8th Int. Symp. on Exp. Algorithms. LNCS 5526, Springer (2009) 257–268
27. Görke, R., Staudt, C.: A Generator for Dynamic Clustered Random Graphs. Technical report, Universität Karlsruhe (TH) (2009), Informatik, TR 2009-7.
28. Brandes, U., Gaertler, M., Wagner, D.: Experiments on Graph Clustering Algorithms. In: Proc. of the 11th Annual European Symposium on Algorithms (ESA'03). LNCS 2832 , Springer (2003) 568–579
29. Guimerà, R., Amaral, L.A.N.: Functional Cartography of Complex Metabolic Networks. Nature **433** (February 2005) 895–900
30. Good, B. H., de Montjoye, Y. and Clauset, A.: The performance of modularity maximization in practical contexts. arxiv.org/abs/0910.0165 (2009)
31. Proceedings of the fifth SIAM International Conference on Data Mining. In: Proceedings of the fifth SIAM International Conference on Data Mining, SIAM (2005)