Dynamic Graph Clustering Combining Modularity and Smoothness

ROBERT GÖRKE, Karlsruhe Institute of Technology, Germany
PASCAL MAILLARD, Université Pierre et Marie Curie, Paris
ANDREA SCHUMM, CHRISTIAN STAUDT, and DOROTHEA WAGNER, Karlsruhe Institute of Technology, Germany

Maximizing the quality index *modularity* has become one of the primary methods for identifying the clustering structure within a graph. Since many 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 sound recommendations for the choice of an algorithm.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; *network problems*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*; I.5.3 [**Computing Methodologies**]: Pattern Recognition—*Clustering*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Dynamic graph clustering, modularity, experimental evaluation, temporal smoothness

ACM Reference Format:
Görke, R., Maillard, P., Schumn, A., Staudt, C., and Wagner, D. 2013. Dynamic graph clustering combining modularity and smoothness. ACM J. Exp. Algor. 18, 1, Article 1.5 (April 2013), 29 pages.
DOI: http://dx.doi.org/10.1145/2444016.2444021

1. INTRODUCTION

Graph clustering is concerned with identifying and analyzing the group structure of networks[1]. 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 exists, good overviews are, for example, Brandes and Erlebach [2005] and Fortunato [2010]. In this work, we set our focus on the quality function *modularity*, coined by Newman and Girvan [2004], which has proven itself

[1]We use the terms *graph* and *network* interchangeably.

Author's address: A. Schumm, C. Staudt, R. Görke, and D. Wagner, Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Am Fasanengarten 5, Gebäude 50.34, 76131 Karlsruhe, Germany, {robert.goerke, andrea.schumm, christian.staudt, dorothea.wagner}@kit.edu
P. Maillard, Laboratoire de Probabilités et Modèles Aléatoires, Université Pierre et Marie Curie (Paris VI), Paris, France, pascal.maillard@upmc.fr

DOI: http://dx.doi.org/10.1145/2444016.2444021

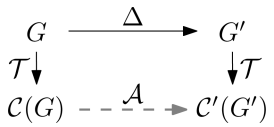


Fig. 1. Problem setting.

feasible and reliable in practice, especially as the target function for a maximization approach (see Brandes et al. [2008] for further references) that follows the paradigm of parameter-free community discovery [Keogh et al. 2004].

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 environments where computing power is limited [Schaeffer et al. 2006], even though very fast clustering methods have been proposed recently [Blondel et al. 2008; Delling et al. 2009]. Second, heuristics for the NP-hard [Brandes et al. 2008] 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 left-hand side of Figure 1 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 toward the same aim as a static technique $\mathcal{T}$ does.

1.1. Related Work

Dynamic graph clustering has so far been a rather untrodden field. Recent efforts [Görke et al. 2009] 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., Hopcroft et al. [2004] and Palla et al. [2007]), besides an array of case studies. Aggarwal and Yu [2005] proposed a parameter-based dynamic graph clustering method that allows user exploration. Parameters are avoided in Sun et al. [2007], where the minimum description length of a graph sequence is used to determine changes in clusterings and the number of clusters. Hübner [2008] proposed an explicitly bicriterial approach for low-difference updates and a partial ILP,[4] 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, the issue of clustering an evolving data set has been addressed in the field of data mining, for example, in Chakrabarti et al. [2006], where the authors share our goal of finding a smooth dynamic clustering. The literature on static *modularity*-maximization is quite broad. We omit a comprehensive review at this point and refer the reader to Brandes et al. [2008], Fortunato [2010], and Schaeffer [2007] for overviews, further references, and comparisons to other clustering techniques. The predominant spectral methods (e.g., White and Smyth [2005]) as well as the established techniques based on random walks [Pons and Latapy 2006; van Dongen 2000], do not lend themselves well to dynamization due to their noncontinuous nature. Variants of greedy agglomeration [Clauset et al. 2004; Blondel et al. 2008], however, are well suited, as we shall see.

For brevity, we mention but two fields where clustering is applied. In Schaeffer et al. [2006], communication overheads in changing sensor networks are reduced by a special clustering algorithm based on density, locality, and stability (see references therein for similar works). Scalability of peer-to-peer networks can be achieved by semantically

clustering the data (see, e.g., Vazirgiannis et al. [2006]); however, modularity has not been used yet.

This study is based on the preliminary paper [Görke et al. 2010a]. We broaden the range of evaluated algorithms and discuss our results in full detail; we divert several tedious details to our technical report [Görke et al. 2010b], but we announce this where we do so.

1.2. 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 [Clauset et al. 2004] and the fastest [Blondel et al. 2008] 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 integer linear program (ILP) algorithm [Brandes et al. 2008]. We compare the algorithms with their static counterparts and evaluate them experimentally on random preclustered dynamic graphs and on large real-world instances (up to 14K nodes and 200K changes). Then, shifting the focus toward large-scale changes per time step, we compare both the static algorithms and our dynamic versions with static variants incorporating an explicit trade-off between maximizing *modularity* and smoothness.

Our results reveal that the dynamic maintenance of a clustering yields higher quality than recomputation and guarantees much smoother clustering dynamics and much lower runtimes. Additionally, they yield strong evidence that small search spaces around the center of the graph change work best, and that actual local optimization (via an ILP) around this center is not the best choice. For large-scale changes, our static variant, which biases the process of identifying a clustering toward the previous results, excels.

1.3. Notation

Throughout this article, we will use the notation of Brandes and Erlebach [2005]. We assume that $G = (V, E, \omega)$ is an undirected, weighted, and simple graph[2] with the edge weight function $\omega\colon E \to \mathbb{R}_{\geq 0}$. The neighborhood of a node v is $N(v) := \{w \in V \mid \{v, w\} \in E\}$. 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 *clusters*. $\mathcal{C}(v)$ means $C \ni v$. A clustering is *trivial* if either $k = 1$ ($\mathcal{C}^1$) or all clusters contain only one element, that is, are *singletons* ($\mathcal{C}^V$). We identify a cluster C_i with its node-induced subgraph of G, which is $G(C_i, E(C_i))$. Then, $E(\mathcal{C}) := \bigcup_{i=1}^{k} E(C_i)$ are intracluster edges and $E \setminus E(\mathcal{C})$ intercluster edges, with cardinalities $m(\mathcal{C})$ and $\overline{m}(\mathcal{C})$, respectively. Further, we generalize degree $\deg(v)$ to clusters as $\deg(C) := \sum_{v \in C} \deg(v)$. While building up a clustering C, we will often deal with a so-called *preclustering* $\tilde{\mathcal{C}}$, which is a clustering on a subset $\tilde{V} \subseteq V$ but leaves $V \setminus \tilde{V}$ unclassified. When using edge weights, all the aforementioned 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 time steps comprises a batch sequence of b atomic events on G_t, which we detail later (see Section 2). In our setting, the sequence of changes arrives as a stream.

[2]A simple graph in this work is both loopless and has no parallel edges.

1.4. The Quality Index *Modularity*

In this work, we set our focus on *modularity* [Newman and Girvan 2004], a measure for the quality of a clustering. Just like any other quality index for clusterings (see, e.g., Brandes and Erlebach [2005] and Fortunato [2010]), *modularity* does have certain drawbacks such as nonlocality and scaling behavior [Brandes et al. 2008] or resolution limit [Fortunato and Barthélemy 2007]. However, being aware of these peculiarities, *modularity* can very well be considered a robust and 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

$$\mathrm{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\deg(v)\right)^2, \quad \text{or equivalently as} \tag{1}$$

$$=\sum_{\{u,v\}\in E}\left(\frac{1}{m}\delta_{uv}\right) - \sum_{(u,v)\in V\times V}\left(\frac{\deg(u)\cdot\deg(v)}{4m^2}\delta_{uv}\right), \quad \delta_{uv} = \begin{cases}1 & \text{if } \mathcal{C}(u)=\mathcal{C}(v)\\ 0 & \text{otherwise}\end{cases} \tag{2}$$

(weighted versions are analogous and merely require weighting edges and degrees).

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 Newman [2004] and Görke et al. [2010]. ModOpt, the problem of optimizing *modularity* is NP-hard [Brandes et al. 2008], but *modularity* can be computed in linear time and lends itself to a number of simple greedy maximization strategies. For the dynamic setting, the following simple corollary theoretically corroborates the use of heuristics, even if we do make the effort to compute an optimal initial clustering, as stated in the following text.

Problem 1 (*DynModOpt*). Given graph G, a *modularity*-optimal clustering $\mathcal{C}^{\mathrm{opt}}(G)$ and an atomic event Δ to G, yielding G'. Find a *modularity*-optimal clustering $\mathcal{C}^{\mathrm{opt}}(G')$.

Corollary 1.1. DynModOpt *is NP-hard*

Proof. We reduce an instance G of ModOpt to a linear number of instances of DynModOpt. Given graph G, there is a sequence $\mathcal{G}$ of graphs $(G_0, \ldots, G_\ell = G)$ of linear length such that (i) $\mathcal{G}$ starts with G_0 consisting of one edge e of G and its incident nodes u, v, (ii) $\mathcal{G}$ ends with G, and (iii) graph G_{i+1} results from G_i and an atomic event Δ_i. ModOpt can be solved in constant time for G_0 yielding $\mathcal{C}^{\mathrm{opt}}(G_0)$. Subsequently solving DynModOpt for instances $G_i, \mathcal{C}^{\mathrm{opt}}(G_i), \Delta_i$ yielding $\mathcal{C}^{\mathrm{opt}}(G_{i+1})$, we end with $\mathcal{C}^{\mathrm{opt}}(G_\ell) = \mathcal{C}^{\mathrm{opt}}(G)$, the solution to ModOpt. □

1.5. Measuring the Smoothness of a Dynamic Clustering

By comparing consecutive clusterings, we quantify how smooth an algorithm manages the transition from one output to the next, an aspect that is crucial to both readability and applicability. An array of measures exist that quantify the dissimilarity between two partitions of a set; for an overview and further references, see Delling et al. [2008]. Our results strongly suggest that most of these widely accepted measures are qualitatively equivalent in all our (nonpathological) instances (see Figure 20 for an example). We thus restrict our view to the graph-structural Rand index [Delling et al. 2008], being a well-known representative; it maps two clusterings into the interval [0, 1], that

is, from equality to maximum dissimilarity:

$$\mathcal{R}_g(\mathcal{C}, \mathcal{C}') := 1 - (|E_{11}| + |E_{00}|)/m, \tag{3}$$

$$\text{with } E_{11} = \{\{v, w\} \in E : \mathcal{C}(v) = \mathcal{C}(w) \wedge \mathcal{C}'(v) = \mathcal{C}'(w)\}, \tag{4}$$

and E_{00} the analog for inequality.

The weighted version is straightforward if we use $\omega(e)$ whenever we count edge e. Low distances correspond to smooth dynamics.

When we compare two clusterings $\mathcal{C}(G), \mathcal{C}'(G')$ of different graphs $G = (V, E) \neq G' = (V', E')$, the previously described measures are not well defined. A canonical solution is to use the *intersection* of the two graphs, that is, define $G'' = (V'', E'') = (V \cap V', E \cap E')$, and compare $\mathcal{C}_{|V''}(G'')$ and $\mathcal{C}'_{|V''}(G'')$. In fact, any other workaround seems unfair: Distance measures are based on either pair-counting, set overlaps, or entropy, but for none of them, the intuition conforms to classifying elements that are unknown in either G or G' in any particular way—be it smooth or distant. Simply ignoring new and discontinued elements avoids introducing a bias due to particular dynamics in a graph such as growth or sparsification.

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 that 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 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 $\tilde{\mathcal{C}}(G_t)$. We call $\tilde{\mathcal{C}}$ the preclustering and nodes that are chosen for individual reassessment free, which 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, making up the sequence of changes between two graph states. Most commonly edge creations and removals take place, and they require the incident nodes to be present before and after the event. In case edges have weights, changing the latter obviously requires the presence of the corresponding edges. Node creations and removals, in turn, only handle isolated nodes, that is, 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.

2.1. Algorithms for Dynamic Updates of Clusterings

In the following, we describe the static algorithms we evaluate and our dynamic versions. Please note that we do not postprocess results with techniques like local optimization, as this blurs insights on the more fundamental algorithms. For an evaluation of the effect of postprocessing and its interaction with the main algorithm, see Noack and Rotta [2009].

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

```
1 while ∃C_i, C_j ∈ C : dQ_∪(C_i, C_j) ≥ 0 do
2 |   (C_1, C_2) ← arg max_{C_i,C_j∈C} dQ_∪(C_i, C_j)
3 |   merge(C_1, C_2)
```

2.1.1. The Global Greedy Algorithm. The most prominent algorithm for *modularity* maximization is a globally greedy algorithm [Clauset et al. 2004], which we call Global (Algorithm 1). Starting with a fine clustering (usually singletons), for each pair of clusters, it determines the increase in *modularity* $\mathrm{d}\mathbf{Q}_\cup$ that can be achieved by merging the pair. It chooses the merge most beneficial to *modularity* and performs it. This process is repeated until no more improvement is possible. Having this algorithm start with singletons (i.e., from scratch) at each time step actually makes it oblivious and thus static. For comparison, we use this algorithm as a pseudodynamic algorithm called sGlobal. In contrast, 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 on the basis of clusters and free nodes.

2.1.2. The Local Greedy Algorithm. In a recent work by Blondel et al. [2008], the simple mechanism of the aforementioned Global has been modified as to rely on local decisions (in terms of graph locality[3]), yielding an extremely fast and efficient maximization. Instead of looking globally for the best merge of two clusters, Local repeatedly lets each node consider moving to one of its neighbors' clusters if this improves *modularity*; the best move is then performed. Especially when starting with singletons, such moves potentially merge clusters, which is a special case of a move. We denote by $\mathrm{d}\mathbf{Q}_{\hookrightarrow}(v, C)$ the change in *modularity* incurred by moving node v into cluster C. From Equation (2), we can see that $\mathrm{d}\mathbf{Q}_{\hookrightarrow}(v, C)$ can quickly be computed by finding the neighbors of v in $\mathcal{C}(v)$ and in C, and by maintaining the degree sums of these clusters. Furthermore, it is easy to see that a move toward a nonneighbor is always less beneficial than isolating a node, thus checking only neighbors' clusters and isolation suffices. When no further node movements are performed (i.e., no movement can improve modularity), the current clustering is contracted: 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 such that we arrive at clusters of clusters. 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 that serves as the core for both a static and a dynamic variant of this approach, as shown in Algorithm 2. As the input, this algorithm takes a hierarchy of graphs and preclusterings 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., $h_{\max} = 0$), in which case line 11 creates all necessary higher levels. Roughly speaking, each iteration of the algorithm's outer loop takes the preclustering of the current level and turns it into a clustering.

Again posing as a pseudodynamic algorithm, the static variant (as in Blondel et al. [2008]), sLocal, passes only $(G_t, \tilde{\mathcal{C}}^V, P)$ to Local, where $\tilde{\mathcal{C}}^V$ means that it starts with singletons and all nodes freed, instead of a proper preclustering. The policy P is set to tell the algorithm to also start from scratch on all higher levels and to not work on

[3]We understand locality in a graph roughly as being only few hops apart. Blondel et al. [2008] only considers immediate neighborhoods in this context.

ALGORITHM 2: Local($G^{0\ldots h_{\max}}, \tilde{\mathcal{C}}^{0\ldots h_{\max}}, P$)

```
1  h ← 0
2  repeat
3  |  (G, C) ← (G^h, C^h)
4  |  repeat
5  |  |  forall the free v ∈ V do
6  |  |  |  C ← arg max_{C=C(u),u∈N(v)} dQ_↔(v, C)
7  |  |  |  if dQ_↔(v, C) > 0 then move(v, C)
8  |  |  |  else if dQ_↔(v, new Cluster) > 0 then move(v, new Cluster)
9  |  until no more changes
10 |  C^h ← C
11 |  (G^{h+1}, C̃^{h+1}) ← contract(G^h, C^h, P)
12 |  h ← h + 1
13 until no more real contractions
14 C(G^0) ← unfurl(C^{h−1})
```

previous results in line 11, that is, in $\tilde{\mathcal{C}}^{h+1}$ again all nodes in the *contracted* graph 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: $(G_t, G_{\text{old}}^{1,\ldots,h_{\max}}, \tilde{\mathcal{C}}, \mathcal{C}_{\text{old}}^{1,\ldots,h_{\max}}, P)$. 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 (or their neighbors as well, tunable by policy P) of the old next-level clustering $\mathcal{C}^{h+1}$ that have been affected by lower level changes just conducted, which yields $\tilde{\mathcal{C}}^{h+1}$ to be worked on next.

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. Thus, a run of Algorithm 2 avoids recomputing unrelated regions of the graph and resolving ambiguous or near-tie situations in a complementary fashion without necessity.

2.1.3. Time-Dependent Local Greedy. Suppose we face a problem instance where, despite a steady dynamicity in the graph, we are not often required to report a new clustering. Along the lines of the procedures described earlier, we could either use a static algorithm, arguing that after very large changes smoothness must be abandoned anyway, or we could employ a dynamic algorithm and either let it accumulate a huge search space (eventually becoming static), or have it eagerly maintain up-to-date clusterings. For suchlike instances, we thus briefly abandon our claim that a small search space suffices for implicitly bringing about smoothness and propose a third rather obvious alternative methodology: It has been observed by Good et al. [2010] that for *modularity*, the landscape of near-optimal clusterings is broad; if a change is large and many equally good solutions exist, why not try to mildly bias a static algorithm towards its previous result?

We follow Chakrabarti et al. [2006] and explicitly enforce smoothness by shaping an objective function TD_α by a convex combination of *modularity* and the Rand index:[4]

$$\text{TD}_\alpha := \alpha \cdot \Big(1 - \mathcal{R}_g\big(\mathcal{C}^{\text{old}}, \mathcal{C}^{\text{new}}\big)\Big) + (1-\alpha) \cdot \text{mod}\big(\mathcal{C}^{\text{new}}\big). \tag{5}$$

[4]Observe that since $\mathcal{R}_g$ is a distance measure, which we wish to minimize, we need to use $1-\mathcal{R}_g$ for combining it with *modularity*, which we wish to maximize.

Such an objective function embodies a scalable trade-off between quality and smoothness and could thus be used by a standard static algorithm. However, for many distance measures, a reformulation as an objective function for the case that one of the clusterings to be compared is not fixed (but is to be determined) is not immediately obvious. In the following, we show how to do this for the graph-structural Rand index. The traditional Rand index can similarly be incorporated. We restrict ourselves to building upon Local, as we shall later see that it is more reliable than Global. The algorithm we thus derive and explain in the is coined tdLocal@α, which stands for the algorithm Local operating on a graph encoding time-dependence, with parameter α scaling the trade-off between quality and smoothness.

Consider an old graph $G = (V, E)$ and a changed new graph $G' = (V', E')$ with clusterings $\mathcal{C}$ and $\mathcal{C}'$, respectively, and set $E'' = E \cap E'$. Using δ_{uv} and δ'_{uv} as in Equation (2), for $\mathcal{C}$ and $\mathcal{C}'$, respectively, we can rewrite $\mathcal{R}_g$ from Section 1.5 as

$$\mathcal{R}_g(\mathcal{C}, \mathcal{C}') = 1 - \frac{1}{|E''|} \sum_{\{uv\} \in E''} \left(\underbrace{\delta_{uv}\delta'_{uv}}_{=1 \text{ iff } \{uv\} \in E''_{11}} + \underbrace{(1-\delta_{uv})(1-\delta'_{uv})}_{=1 \text{ iff } \{uv\} \in E''_{00}} \right) \tag{6}$$

$$= 1 - \frac{1}{|E''|} \sum_{\{uv\} \in E''} (1 - \delta_{uv} + \delta'_{uv} \cdot (2\delta_{uv} - 1)) \ . \tag{7}$$

Being a contribution to the aforementioned term, let us define $\mathrm{dS}(u, v) := \frac{1}{|E''|}(2\delta_{uv} - 1)$. Then, we can see that $1 - \mathcal{R}_g(\mathcal{C}, \mathcal{C}')$ changes by

$$\frac{1}{|E''|} \left(\sum_{v \in C_b, \{u,v\} \in E''} \mathrm{dS}(u, v) - \sum_{w \in C_a, \{u,w\} \in E''} \mathrm{dS}(u, w) \right), \tag{8}$$

if we move u out of C_a and into C_b. We can thus annotate edges $\{u, v\} \in E''$ by dS, explicitly ignoring all discontinued $e \in E \setminus E'$ and all new $\{u, v\} \in E' \setminus E$. Algorithmically, this view is convenient, because for computing changes $\mathrm{dQ}_{\hookrightarrow}(u, v)$ in *modularity* in Algorithm 2 (lines 6, 8), we proceed similarly: As suggested by Equation (2), for each edge $e = \{u, v\}$, we record the change in *modularity* that putting v and u into the same cluster incurs. In order to arrive at TD_α, we can, therefore, simply replace the term $\frac{1}{m}$ in Equation (2) by $((1-\alpha) \cdot \frac{1}{m} + \alpha \cdot \frac{2\delta_{uv}-1}{|E''|})$ and use this term as edge value just as the purely *modularity*-based version of Algorithm 2 does (see, e.g., Blondel et al. [2008]).[5] By design, $\mathcal{R}_g$ does not require us to do any work for disconnected pairs.

One additional observation reveals that employing TD_α with any $\alpha \in [0, 1]$ allows us to use Algorithm 2 in the very way we stated it. In line 6 (8), we use the fact that v may only move to one of its neighbors' clusters (or start a new one) to strongly limit the options of node v. Pleasantly, this does not change when using TD_α: Suppose we move node v from cluster C_1 into a nonempty cluster C_2 with $C_2 \cap N(v) = \emptyset$. Since no edges lead from v into C_2, moving v into C_2 does not affect TD_α. By contrast, moving v out of C_1 might do so. However, putting v into a new cluster has exactly the same effect (but is always strictly favored by *modularity*). Thus, we need not check any additional options for v when using TD_α, and we avoid additional runtime.

2.1.4. ILP. While optimal *modularity* is out of reach, the problem can be cast as an ILP [Brandes et al. 2008]. Based on Equation (2), a (binary) distance relation $\mathcal{X}$

[5]Note that when contracting (line 11), we need to handle the contributions separately.

indicates whether elements are in the same cluster (set $\tilde{V} = V$ for now):

$$\mathcal{X}(\tilde{V}) := \left\{ X_{uv} : \{u, v\} \in \binom{\tilde{V}}{2} \right\} \quad \text{with} \quad X_{uv} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 1 & \text{otherwise} \end{cases}, \tag{9}$$

$$\text{constrained by } \forall \{u, v, w\} \in \binom{\tilde{V}}{3} : \begin{cases} X_{uv} + X_{vw} - X_{uw} \geq 0 \\ X_{uv} + X_{uw} - X_{vw} \geq 0 \\ X_{uw} + X_{vw} - X_{uv} \geq 0 \end{cases} \quad X \in \{0, 1\}, \tag{10}$$

$$\text{set as to minimize } \operatorname{mod}_{\text{ILP}}(G, \mathcal{C}_G) = \sum_{\{u,v\} \in \binom{\tilde{V}}{2}} \left(\omega(u, v) - \frac{\omega(u) \cdot \omega(v)}{2 \cdot \omega(E)} \right) X_{uv}. \tag{11}$$

To ensure the properties of an equivalence relation, Equation (10) represents transitivity. We can omit the other two: Reflexivity, $X_{uu} = 0$, is automatically ensured because a node is always in the same cluster as itself. Symmetry, $X_{uv} = X_{vu}$, is ensured because there is only one such variable. Note that the definition of X_{uv} renders this a minimization problem.

Since runtimes for the full ILP reach days for more than 200 nodes, we neither tackle a pure version nor one based on TD_α, tempting though that might be. A promising idea pioneered by Hübner [2008] is to solve a partial ILP (pILP), using $\tilde{V} \subsetneq V$. Such a program takes a preclustering—of much smaller complexity—as the input and solves this instance, that is, finishes the clustering, optimally via an ILP; a singleton preclustering yields a full ILP ($\tilde{V} = V$). We introduce two variants: (i) The argument noMerge prohibits merging preclusters 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 Equations (9) and (10) and change Equation (11).

For (i), variables $\mathcal{Y}$ indicating whether a node $u \in \tilde{V}$ is contained in a cluster $C \in \tilde{\mathcal{C}}$ are introduced (again as a binary "distance", i.e., 0 iff $u \in C$) and triplets of constraints similar to Equation (10) ensure their transitive consistency with $\mathcal{X}$. Moreover, a node must only join one cluster, and the target function must evaluate such joins. Formally, this translates to using Equations (9) and (10) plus the following:

$$\mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}}) := \left\{ Y_{uC} : \{u, C\} \in \tilde{V} \times \tilde{\mathcal{C}} \right\} \quad \text{with} \quad Y_{uC} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = C \\ 1 & \text{otherwise} \end{cases}, \tag{12}$$

$$\text{constr. by } \forall \{u, v, C\} \in \binom{\tilde{V}}{2} \times \tilde{\mathcal{C}} : \begin{cases} X_{uv} + Y_{uC} - Y_{vC} \geq 0 \\ X_{uv} + Y_{vC} - Y_{uC} \geq 0 \\ Y_{uC} + Y_{vC} - X_{uv} \geq 0 \end{cases}, Y_{uC} \in \{0, 1\}, \tag{13}$$

$$\text{and also by } \forall u \in \tilde{V} : \sum_{C \in \tilde{\mathcal{C}}} Y_{uC} \geq k - 1 \quad \text{(a node's cluster must be unique)}, \tag{14}$$

$$\begin{aligned} \text{as to minimize } \operatorname{mod}_{\substack{\text{pILP} \\ \text{no merge}}}(G, \mathcal{C}) = & \sum_{X_{uv} \in \mathcal{X}(\tilde{V})} \left(\omega(u, v) - \frac{\omega(u) \cdot \omega(v)}{2\omega(E)} \right) X_{uv} \\ & + \sum_{Y_{uC} \in \mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}})} \left(\sum_{w \in C} \left(\omega(u, w) - \frac{\omega(u) \cdot \omega(w)}{2\omega(E)} \right) \right) Y_{uC}. \end{aligned} \tag{15}$$

Table I. 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)	$\{u\}, \mathcal{C}(u) \setminus u$

Table II. Reactions of the Algorithms to Graph Events
Each isolated node forms a singleton when inserted and simply deleted when removed. With "$\to S$" we indicate that a prep strategy prepares a preclustering. Algorithm tdLocal@α is covered by sLocal

	algorithms' reactions					
event	sGlobal	dGlobal	sLocal	dLocal	dILP	dEOO
	static, globally greedy merging (from $\mathcal{C}^V$)	dynamic, globally greedy merging (from $\tilde{\mathcal{C}}$)	static, locally greedy moving (from $\mathcal{C}^V$)	dynamic, locally greedy moving (from $\tilde{\mathcal{C}}$)	dynamic ILP (from $\tilde{\mathcal{C}}$)	dynamic elemental operations (from $\tilde{\mathcal{C}}$)
Δ	$-$	$\to S$	$-$	$\to S$	$\to S$, pILP(args)	$-$
$t+1$	Global ($G_t, \mathcal{C}^V$)	Global ($G_t, \tilde{\mathcal{C}}$)	Local(G_t $\mathcal{C}^V$, all)	Local($G_{t-1}^{1\ldots h_{\max}}$, $\tilde{\mathcal{C}}, \mathcal{C}_{t-1}^{1\ldots h_{\max}}, P$)	-	EOO(G_{t+1}, $\mathcal{C}_{t+1}$, args)

In the following, we only roughly sketch out case (ii). If clusters are allowed to merge, we additionally need variables $Z_{CC'}$ for the distance between clusters, constrained just as in Equation (10). Details on this formulation can be found in Görke et al. [2010b]. Note that if in addition to the merging of clusters we also allow splitting, we actually arrive at the full ILP again.

The dynamic clustering algorithms that first solicit a preclustering and then call pILP are called dILP. Note that they react on any edge event, accumulating events until a time step occurs can result in prohibitive runtimes.

2.1.5. Elemental Operations Optimizer. Method EOO performs a bounded number of operations, trying to increase the quality. Specifically, we allow moving (to a neighbor's cluster) or splitting off (isolating) a node, and merging its cluster with one of the others, as listed in Table I. In a random order, for each node, the most beneficial of these operations is executed, terminating if either no node admits any improvement or if some maximum number $\mathrm{op}_{\max}$ of operations have been executed. Our dynamic algorithm dEOO (or dEOO@$\mathrm{op}_{\max}$) simply calls EOO at each time step, doing nothing in between. Thus, no search space is accumulated for EOO to focus on; instead, the algorithm simply tries to improve all parts of its previous clustering as to better fit the changed graph, ignorant of the actual changes that have taken place. We can view EOO as a control, as it has all the options dLocal and dGlobal together have but lacks both a pool of free nodes and higher-level capabilities. EOO or very similar tools for local optimization can also be used as postprocessing tools.

2.2. Strategies for Building the Preclustering

We now describe prep strategies, which generate a preclustering $\tilde{\mathcal{C}}$, that is, define the search space. We distinguish the *backtrack strategy*, which refines a clustering and is only applicable to Global, 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

Table III. Overview of How Strategies Handle Graph Events
Changes to edges' weights are analog to creations/removals. Isolated nodes are universally made singletons when inserted and ignored when removed.

cause	reaction				
event	BT		BU, $\tilde{V} =$	N, $\tilde{V} =$	BN, $\tilde{V} =$
$E + \{u, v\}$	sep(u, v)	$\mathcal{C}(u) = \mathcal{C}(v)$	$\mathcal{C}(u) \cup \mathcal{C}(v)$	$N_d(u) \cup N_d(v)$	$\mathrm{BFS}\{u, v\}_{\mid s}$
	iso(u), iso(v)	$\mathcal{C}(u) \neq \mathcal{C}(v)$			
$E - \{u, v\}$	iso(u), iso(v)	$\mathcal{C}(u) = \mathcal{C}(v)$	$\mathcal{C}(u) \cup \mathcal{C}(v)$	$N_d(u) \cup N_d(v)$	$\mathrm{BFS}\{u, v\}_{\mid s}$
	–	$\mathcal{C}(u) \neq \mathcal{C}(v)$			

is necessarily local. Both output a half-finished preclustering. We detail the two approaches in the following two subsections.

2.3. Subset Strategies

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 that 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.

2.4. Backtrack Strategy

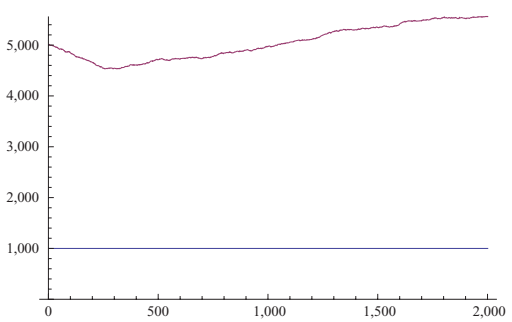
The backtrack strategy (BT) records the merge operations of Global and backtracks them if a graph modification suggests their reconsideration. In Görke et al. [2010b], we rigorously detail what we mean by "suggests," but for brevity we just state that the actions listed for BT provably require very little 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, that is, we separate them. For intercluster additions, we track back u and v individually, until we isolate them as singletons, such that Global can reclassify and potentially merge them. Intercluster deletions are not reacted on. On intracluster deletions, we again isolate both u and v such that Global has the possibility to find separate clusters for them. For more details on these operations, see Görke et al. [2010b]. 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. We leave its rather technical proof to Görke et al. [2010b].

THEOREM 2.1. *Assume that a backtrack step divides a cluster randomly. Then, for the number I of steps* isolate*(v) requires, it holds:* $\mathbb{E}\{I\} \in \Theta(\ln n)$.

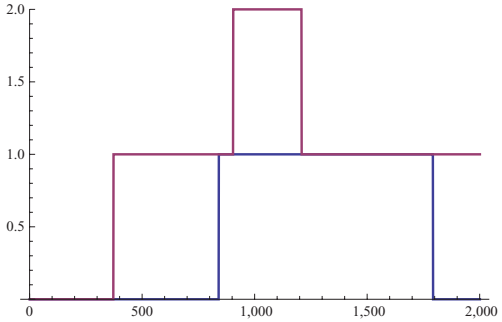
3. EXPERIMENTAL EVALUATION OF DYNAMIC ALGORITHMS

3.1. Instances

We use both generated graphs and real-world instances. We briefly describe them here, but for more details, please see Görke and Staudt [2009] and Hübner [2008].



(a) Numbers of **nodes** end **edges**. This instance does not allow node events it only allows edge events for well-controlled experiments. $\mathcal{G}_r^1$ uses $b = 10$ atomic events per time step.



(b) Illustration of how the ground-truth clustering gradually changes; numbers and types (**split**, **merge**) of ongoing changes over time. Jumps indicate initiated splits/merges, and drops indicate completed ones. $\mathcal{G}_r^1$ has one merge and two splits, one of them incomplete.

Fig. 2. Graph $\mathcal{G}_r^1$, nondefault parameters of its generation: $t_{\max} = 2k$, $n_0 = 1k$, $|\mathcal{C}| = 20$, $b = 10$, $p_\omega = 10^{-3}$ ($= P$[cluster-event/time step]), $p_{\text{in}} = 0.1$, $p_{\text{out}} = 0.005$.

3.1.1. Random Graphs $\mathcal{G}_r$. Our Erdős-Rényi-type generator builds upon Brandes et al. [2003] and adds to this community structure and dynamicity in all graph elements and in the clustering, that is, nodes and edges are inserted and removed and ground-truth clusters merged and split, always complying with sound probabilities. Our generator and a ready-to-use software package are thoroughly described in Görke and Staudt [2009]; we here briefly summarize its behavior. The changing (see Figure 2(b)) ground-truth clustering of the generator defines edge probabilities (p_{in} inside clusters and p_{out} in between) and thereby steers what clustering can and should be detected and how the graph evolves. Roughly speaking, within ground-truth clusters edges accumulate and in between they become sparse. The generator additionally maintains a reference clustering, which follows the ground-truth clustering, as soon as changes in the latter actually manifest in the edge structure; we use this reference clustering to compare our algorithms. We conducted experiments for a large number of settings, varying size, density, node/edge-volatility, stability of clusters, and so on, and in the following only give representative plots, and point out specific peculiarities. These instances range between 1K and 10K nodes and between 1K and 100K changes. A number of plots use a graph coined $\mathcal{G}_r^1$, one of our simpler test instances. It is is used in many examples, as behavior on it is largely archetypal; Figure 2 depicts its rough statistics.

3.1.2. E-Mail Graph $\mathcal{G}_e$. The network of e-mail 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 e-mails during the past 7 days; thus, edges can completely time out. Isolated nodes are removed from the network. This network, $\mathcal{G}_e$, has between 100 and 1,500 nodes, depending on the time of year, and about 700K events spanning about 2.5 years. It features a strong power-law degree distribution. Figure 3 shows the temporal development of this graph in terms of n (lower) and m (upper) per 100 events. The first peak stems from a spam attack in late 2006, the two large drops are from Christmas breaks, and the smaller drops from spring and autumn breaks.[6]

[6]For further details on $\mathcal{G}_e$ and for downloading the whole dataset, please visit http://i11www.iti.uni-karlsruhe.de/en/projects/spp1307/emaildata.

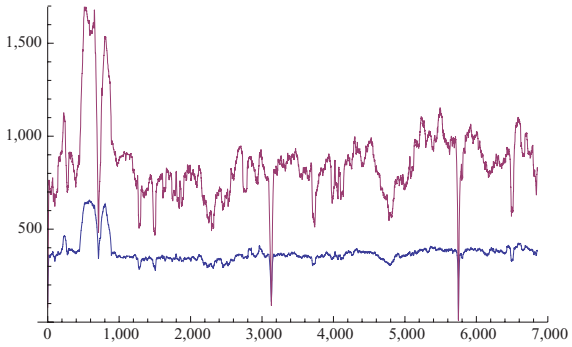


Fig. 3. Nodes and edges of $\mathcal{G}_e$.

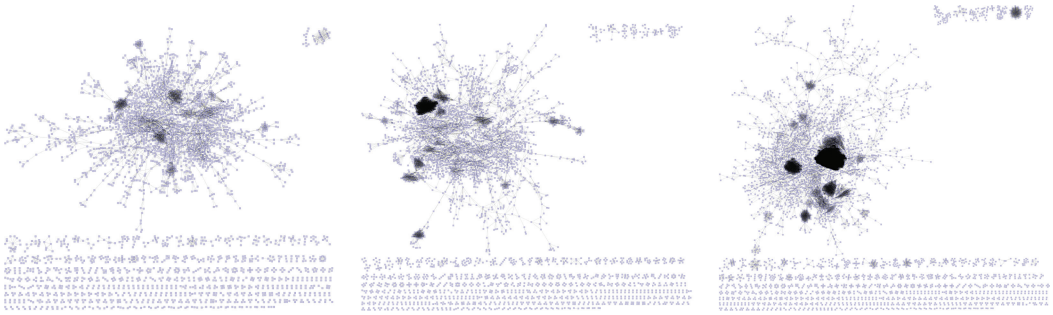


Fig. 4. *ArXiv* category *Nuclear Theory* ($\mathcal{G}_x^1$), cont. 33K e-prints, after 2001, 2005, and 2009.

3.1.3. arXiv Graphs $\mathcal{G}_x$. Since 1992, the arXiv.org e-print archive[7] is a popular repository for scientific e-prints, stored in several categories alongside timestamped metadata. We extracted networks of collaboration between scientists based on coauthorship. For each e-print, we add equally weighted clique-edges among the contributors such that each author gains a total edge weight of 1.0 per e-print contributed to (see Figure 4 for three examples). We let e-prints time out after 2 years and remove disconnected authors. As these networks are ill-natured for local updates, we shall use them as tough trials. We show results for two categories (14K/25K authors, 33K/14K e-prints) that feature a large connected component.

3.2. Fundamental Results

For the sake of readability, we use a moving average in plots for distance and quality to smoothen the raw data, separately looking at variances. We consider the following performance measures: the quality (*modularity*) of the clustering, the smoothness ($\mathcal{R}_g$) of the changes performed on it, and the runtime (ms) of the algorithm; we additionally keep track of $|\mathcal{C}|$. Generally speaking, the x-axis always indicates the current time step, and the y-axis gives the measurement, as described in the corresponding legend.

Behavior of dEOO. In a first feasibility test, dEOO generally falls behind the other algorithms in terms of quality; the more severely (up to 10%), the more volatile and clear the observable clustering is. On some graphs, dEOO manages to keep up for quite a while, before eventually falling behind. We generally observed that very few

[7]Website of e-print repository: arxiv.org; for our tools for collecting and processing the data, see i11www.iti.uni-karlsruhe.de/projects/spp1307/dyneval.

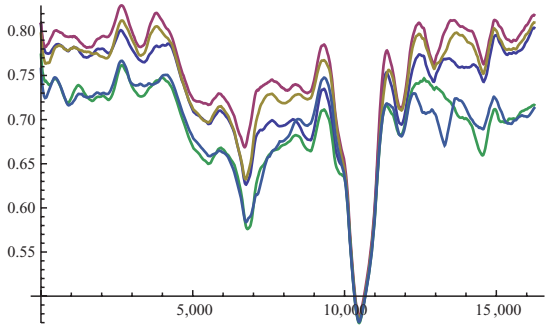


Fig. 5. *Modularity* on the first quarter of $\mathcal{G}_e$, $b = 10$: (bottom to top) both EOO@10 and EOO@100 fail to follow dLocal@BN_8, dGlobal@BN_8, and dGlobal@BT. Larger batches to do not help (see Figure 21).

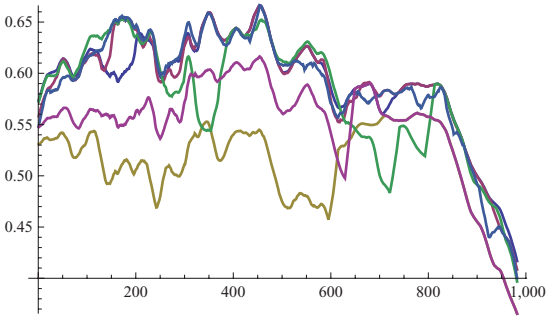


Fig. 6. *Modularity* on $\mathcal{G}_r^2$ ($\mathcal{G}_r^1$ with increased clustering volatility), $b = 1$: (bottom to top) EOO@10 and EOO@100 are worse than the reference, dLocal@BN_8, dGlobal@BN_8, and dGlobal@BT.

operations were actually conducted by dEOO; on most graphs (e.g., $\mathcal{G}_e$), not even $\text{op}_{\max} = 10$ was a limiting value. This indicates that dEOO lacks the ability to sufficiently renovate the clustering, since it is unable to reconsider a larger number of nodes at once. Exemplary plots for quality on $\mathcal{G}_e$ and a random graph are shown in Figures 5 and 6. As a consequence, smoothness is often slightly better than for other dynamic algorithms, runtimes are comparable (independent of $\text{op}_{\max}$), and the number of identified clusters is comparably low, on average (as large-scale splits of clusters cannot be performed). For some random graphs with no significant reference clustering, dEOO managed to compete well in terms of quality. Generally speaking, dEOO used as the sole technique to update a clustering is ill-suited, as it lacks the abilities to operate more generously on a freed search space of nodes and to perform compound operations (this does not devalue its potential as a postprocessing tool).

Parameters of Local. It has been stated by Blondel et al. [2008] 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 slightly less smooth; for dLocal, the latter observation does not hold. Since a random order is likely to be more robust, we universally use it for Local; we divert plots on discrepancies between orders to Görke et al. [2010b] for brevity. We found that considering only affected nodes or also their neighbors in higher levels, does not affect any criterion on average (we omit plots on this). Thus, we prefer the affected policy, since it is the simpler variant.

pILP Variants. Allowing the ILP to merge existing clusters takes longer and produces a coarser clustering—which is quite intuitive—but also yields a slightly worse *modularity*. We conjecture that the reason for this is that merging invites hazardous local

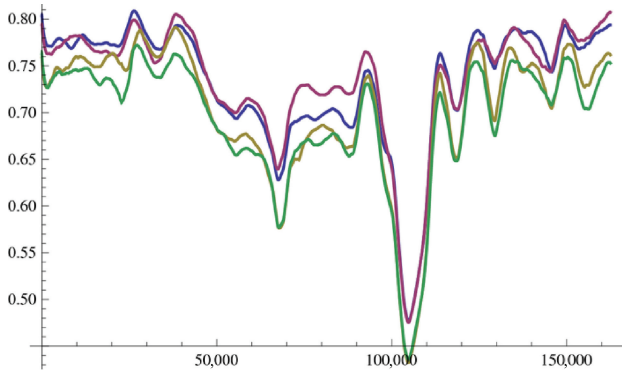


Fig. 7. *Modularity* for dLocal, dGlobal, dILP (noMerge), and dILP (merge) on the first quarter of $\mathcal{G}_e$, Batch size 1, strategy BN_8.

optima. We made this observation on almost all tested instances, and we, therefore, reject merge for dILP. In terms of the number of clusters, merge and noMerge roughly bound both dLocal and dGlobal from below and above, respectively (see Figure 19 for an example).

Heuristics vs. dILP. A striking observation about dILP is the fact that it yields worse quality than dLocal and sLocal with identical prep strategies, as shown in Figure 7. Intuitively speaking, dILP solves similar problems (using potentially different preclusterings but the same set of free nodes) in each time step as the other real heuristics do, but dILP solves them optimally. Thus, we clearly expect dILP to yield better quality, but this does not happen. Being locally optimal seems to overfit and get stuck in inferior local optima, 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. This observation has been made for other (huge) instances by Blondel et al. [2008], and we can confirm it on all our generated instances; these results are paralleled by the dynamic counterparts. An exception is instance $\mathcal{G}_e$, as discussed later. In terms of runtime, however, sGlobal is hardly slower than 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.

3.3. 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, empirically, runtime grows sublinearly, which suggests $d = 1$. Similar facts hold for other instances and batch sizes. Note that large batch sizes b let a prep strategy accumulate many free nodes yielding a larger search space; however, we observed that a small b does not benefit from larger search spaces. 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 essentially equivalent to increasing d, only on a finer scale. Consequently, we can

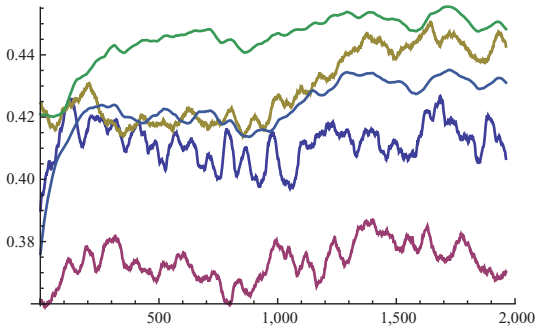


Fig. 8. *Modularity* on $\mathcal{G}_r^1$: dGlobal@BT (lower blue) and dGlobal@BN_{16} (upper blue) beat sGlobal; dLocal@BN_4 beats sLocal.

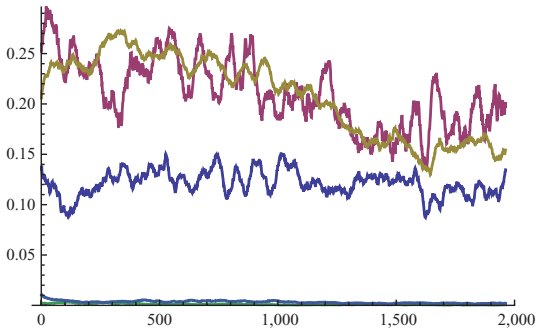


Fig. 9. $\mathcal{R}_g$ on $\mathcal{G}_r^1$: sGlobal and sLocal are less smooth (factor 100) than dLocal@BN_4, dGlobal@BN_{16} (bottom); dGlobal@BT competes well.

report similar observations as for N_d. For dLocal, BN_4 proved slightly superior. dGlobal's quality benefits from increasing s, but again at the cost of runtime and smoothness, so BN_{16} is a reasonable choice.

The strategy that simply breaks up all clusters affected by changes, BU, clearly falls behind in terms of all criteria compared to the other strategies and often mimics the static algorithms. As expected, we can discard this strategy and rather consider it as a "control." Note that this is a very basic confirmation of the assumption that local updates are a good idea.

dGlobal using BT is by far the fastest algorithm, confirming our theoretical predictions from Section 2.2, but still produces competitive quality. However, it often yields a smoothness almost in the range of sGlobal.

In summary, our best dynamic candidates are dGlobal@BT, dGlobal@BN_{16} (they achieve a speed-up over sGlobal of up to 1K and of 20 at 1K nodes, respectively), and algorithm dLocal@BN_4 (with a speed-up of 5 over sLocal). Figure 22 exemplarily shows a comparison of prep strategies for dGlobal in terms of *modularity* on $\mathcal{G}_r^1$.

3.4. Comparison of the Best and Their Static Counterparts

We now take a focused look at those dynamic clustering algorithms and prep strategies, which we observed to be the most promising and compare them with their static counterpart. As a general observation, as depicted in Figure 8, each dynamic candidate beats its static counterpart in terms of *modularity*. On the generated graphs, dLocal is faster and superior to dGlobal; this is not the case for the e-mail network—here both Global algorithms beat each Local algorithm. In terms of smoothness (Figure 9),

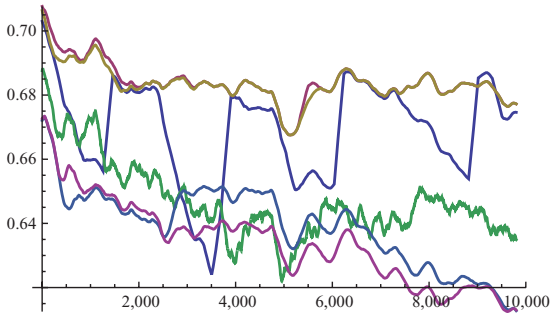


Fig. 10. *Modularity* on a longer (10K×10 events) random graph $\mathcal{G}_r^4$. Reference exhibits jumps where the clustering changes, before which it clearly decreases. In contrast, the dynamics adapt quickly to changes. Top to bottom: dLocal@N_1, dLocal@BN_4, reference, dGlobal@BT, dGlobal@BN_8 dGlobal@BN_{16} which does not benefit from its larger search space.

dynamics (except for dGlobal@BT) are superior to statics by a factor of approximately 100, but even dGlobal@BT beats them. For large batch sizes (e.g., 100 and beyond), dynamics are still clearly smoother, but eventually fall behind the statics in terms of quality by a few percent. We discuss runtimes in detail later in Section 3.7.

3.5. Dynamic Algorithms React Quickly to Changing Clusterings

Throughout our experiments, we observed that the dynamic algorithms exhibit the ability to react quickly and aptly to changes in the ground-truth clustering. Figure 10 shows an example where our best dynamic algorithms quickly cope with rapid changes to the clustering—in contrast to the reference clustering, with its rather clumsy, stepwise adaption: The very simple method of the reference clustering to follow the ground-truth relies solely on how clearly the split of a ground-truth cluster, or the merge of two ground-truth clusters has manifested in terms of the affected edge mass. The changes in the ground-truth clustering are visible in the form of the drops in the reference quality. At each such change, after brief depressions, the *modularity* values of all algorithms rise to their old levels. Only dGlobal@BN_{16} seems to need some more time to adapt to the last clustering event. This instance is a growing network with 10K changes of batch size 10. Its few changes in the clustering are rapidly realized by a decent frequency of node insertions in ways consistent with the coming clustering. It is thus a more "difficult" instance for an algorithm to prove its reactivity; on other instances, we observed even better results. The quick reactivity of a dynamic algorithm is of particular importance, as, static counterpart algorithms clearly are not subject to such issues, since they "forget" their previous work.

3.6. Time-Dependent Static Algorithms for Large Batches

The effect of scaling parameter α on the quality of tdLocal@α is showcased in Figure 11. As expected, it is immediate that quality monotonously decreases with increasing α, significantly so beyond approximately 0.4, since with a maximum *modularity* of about 0.8 on $\mathcal{G}_e$, smoothness then gets the upper hand (see formulas in Section 2.1.3). Compared to sLocal, we observed a doubling in smoothness for tdLocal@0.1 already, with a slower but monotonous improvement for larger α. Runtimes did not consistently differ. Observing the average modularity, which impacts the effect of α, and the comparison in Figure 12, we focus on tdLocal@0.2, which is a reasonable compromise. We let tdLocal@0.2 compete against sLocal and dLocal@BN_4 on a very long random graph $\mathcal{G}_r^3$ with 10K time steps with $1,000$ events each, and we can confirm that the time-dependent strategy holds the best of both worlds for large batches. Figure 13 plainly exemplifies how

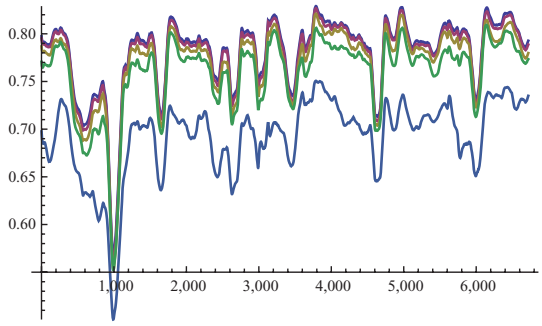


Fig. 11. *Modularity* on $\mathcal{G}_e$: Batch size 100 e-mails: top to bottom: tdLocal@0.1, tdLocal@0.2, tdLocal@0.3, tdLocal@0.4, and tdLocal@0.6.

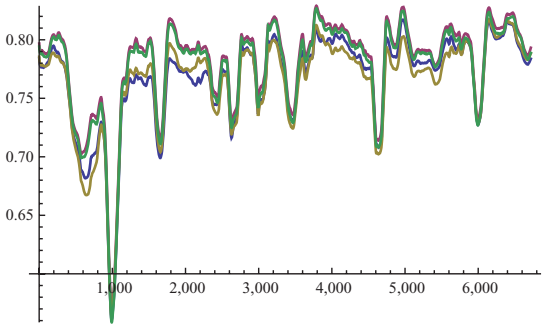
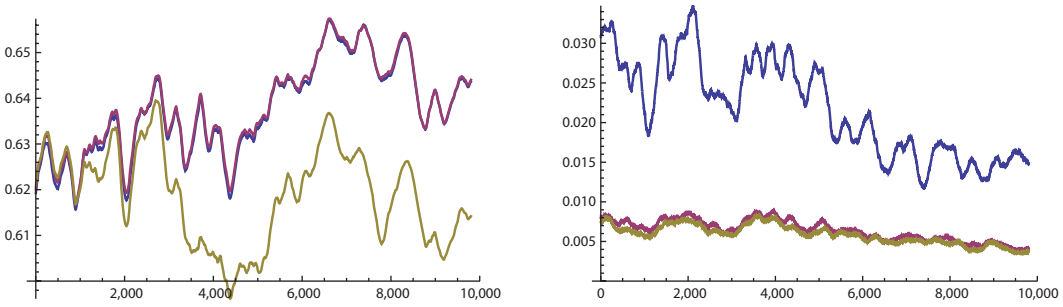


Fig. 12. *Modularity* on $\mathcal{G}_e$: Batch size 100 e-mails: top to bottom: sLocal, tdLocal@0.2, dGlobal@BN_{128}, and dGlobal@N_1.

tdLocal@0.2 equals the conventional static algorithm in quality and how it matches the dynamic algorithm in terms of smoothness. In fact, its average quality is even slightly higher than that of sLocal, mildly suggesting that stability can even help quality; we observed this phenomenon on several instances. Runtimes are still clearly lower for BN_4, however, larger search spaces let dLocal eventually approach sLocal. We observed that high values of α let tdLocal very mildly favor finer clusterings; by Equation (8), it is not hard to see that this is due to the tendency of $\mathcal{R}_g$ to first put a set of nodes into a separate cluster if *modularity* intends to migrate them into a different cluster.

3.7. Discussion of Absolute Runtimes

In order to improve comparability, we designed our implementation such that all algorithms follow a uniform style and share subroutines where possible. That said, our implementation is prototypical with no focus on low runtimes. In absolute terms, average runtimes per time step on smaller instances like $\mathcal{G}_e$ are given in Table IV. Generally speaking, runtimes compare as expected, but still a few insights are worth pointing out. Enlarging the search space of dynamics indefinitely still does not eventually cause runtimes as for the statics, this is due to the fact that datastructures do not need to be rebuild from scratch. Using prep strategy N_1 or N_2, runtimes are very similar to those of BN; however, an unreasonable search depth of three or more lets runtimes rise. As a general observation, dLocal is faster than dGlobal by a constant factor, except for the backtrack strategy dGlobal@BT, which universally is the fastest of our variants. dGlobal@BT is always faster than other dynamics by a factor of 10, and the



(a) *Modularity*, $\mathcal{G}_r^3$, $b = 1{,}000$ (b) $\mathcal{R}_g$, $\mathcal{G}_r^3$, $b = 1{,}000$

Fig. 13. Results on random graph $\mathcal{G}_r^3$ with about 1,000 nodes, $10K \times 1K$ events, and expected intra- and intercluster degrees of 6 and 4, respectively; 12 cluster events. For quality (top to bottom), tdLocal@0.2 even bests sLocal with dLocal@BN_4 having trouble to catch up after a series of cluster events (completed ones, though). In terms of distance, tdLocal@0.2 equals dLocal@BN_4, with sLocal struggling at 3 times the former's average distance.

Table IV. Average Runtimes per Time Step of Various Algorithms $\mathcal{G}_e$, $b = 100$

algorithm	ms
dLocal@BN_2	4.6
dLocal@BN_4	6
dLocal@BN_8	7.2
dGlobal@BT	0.4
dGlobal@BN_4	13.6
dGlobal@BN_8	20.6
sGlobal	49.2
sLocal	86.1
tdLocal@0.1	89.4
tdLocal@0.3	89.2

speed-up over statics easily reaches 1,000 for large graphs. On $\mathcal{G}_e$, sGlobal is faster than sLocal, thanks to the large overhead of the latter algorithm's contraction mechanism; on larger graphs, the converse holds. It is interesting to see that adding time-dependence to sLocal does not change runtimes at all. On a graph with 10K nodes, the previously mentioned observations uniformly scale such that sLocal requires around 5s per time step and the dLocal@BN_4 requires 78ms and dGlobal@BT around 4ms.

Figure 14 illustrates how a selection of algorithms react to changes in the size of the instance. As pointed out in Section 3.1, $\mathcal{G}_e$ shrinks during Christmastime and quickly grows again afterward. This process is followed by all runtimes and becomes visible by dents in the runtime plots. Static algorithms are far more sensitive to size than dynamics (note the logarithmic y-axis). Outliers are due to the activity of Java's garbage collection.

3.8. Trials on arXiv Data

As an independent data set, we use our arXiv graphs for testing our results from $\mathcal{G}_e$ and the generated instances. As explained in Section 3.1, a single e-print constitutes a weighted clique, with the contributing authors as its nodes. As e-prints arrive as a stream, these graphs consist solely of glued cliques of authors, established within

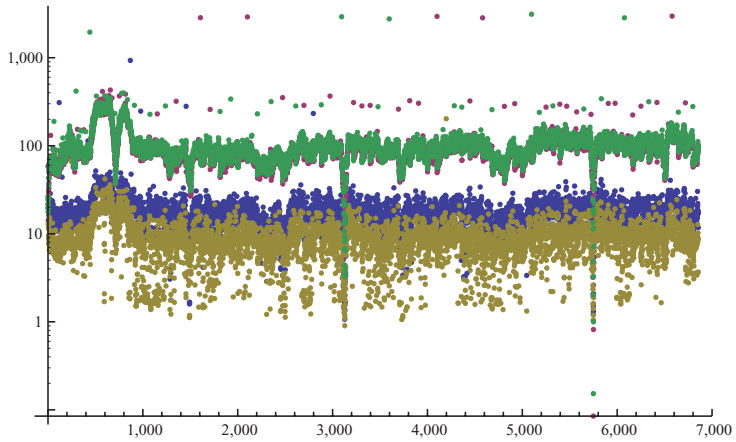


Fig. 14. $\mathcal{G}_e$, $b = 100$, runtimes. Bottom to top: dLocal@N_1, dLocal@BN_{128}, sLocal, and tdLocal@0.2.

single time steps each where potentially many new nodes and edges are introduced. Together with modularity's resolution limit [Fortunato and Barthélemy 2007] and its fondness of balanced clusters and a nonarbitrary number thereof in large graphs [Good et al. 2010], 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 (Figure 4), while dynamics using N or BN will sometimes have no choice but to further enlarge some growing cluster, as they may only change the clustering in the vicinity of the event. Locally this is a good choice, but globally some far-away cut might qualify as an improvement over pure componentwise growth because modularity does favor rather balanced cluster sizes.

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 recluster; however, still they are only worse by about 1%. Figures 15 and 16 show *modularity* for coarse and fine batches, respectively, using the arXiv category *Nuclear Theory* (1992–2010, 33K e-prints, 200K elementary events, 14K authors, yielding $\mathcal{G}_x^1$). As before, dynamics are faster and smoother. For the coarse batches, speed-ups of 10 to 2K (BT) are attained; for fine batches, these are 100 to 2K. In line with the previous observations, their clusterings are slightly coarser (except for dGlobal@BT). See Appendixes B and C for insightful further results that exhibit the appropriateness of dGlobal@BT.

Figure 17 exemplifies for $b = 200$ e-prints (on *Nuclear Theory*) how tdLocal performs, compared to its pure static and the dynamic counterpart. The results clearly confirm that mildly pushing a clustering toward its previous structure does not have to be payed for in terms of *modularity*, but significantly improves smoothness. In this setting, except for runtime, dLocal falls behind the time-dependent approach. Cross-comparing with results for dGlobal@BT (as illustrated in Figure 18), we can state that the global backtrack strategy competes very well with the time-dependent strategy, with the slight advantage of the latter in terms of both quality and smoothness getting larger with larger batches. If for such cases runtime remains a central issue, dGlobal@BT is still by far the best choice for graphs like $\mathcal{G}_x^1$ (see Figure 23 for analogous observations on $\mathcal{G}_e$).

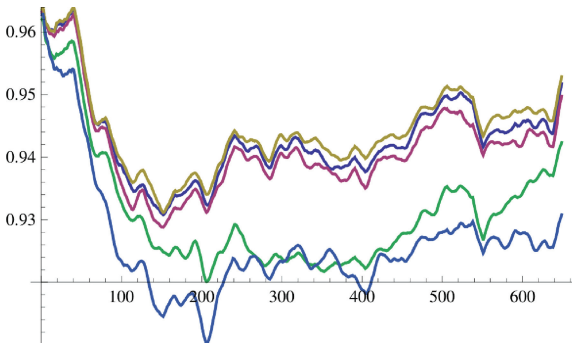


Fig. 15. *Modularity*, $\mathcal{G}_x^1$, batch size 50 e-prints: Backtracking (dGlobal@BT) easily follows the static algorithms (sLocal and sGlobal); even dLocal@BN_4 and dGlobal@BN_{16} are worse by only about 1%.

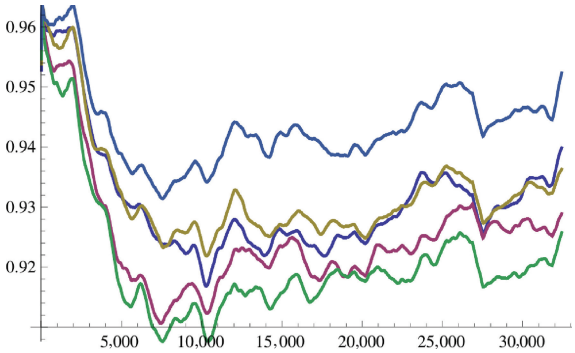
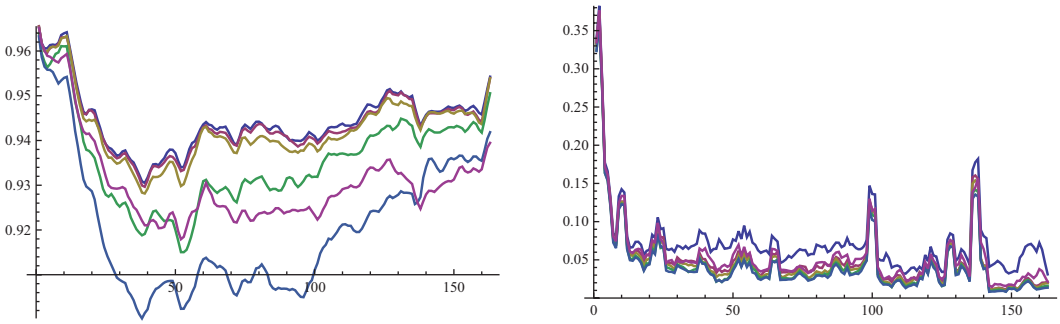
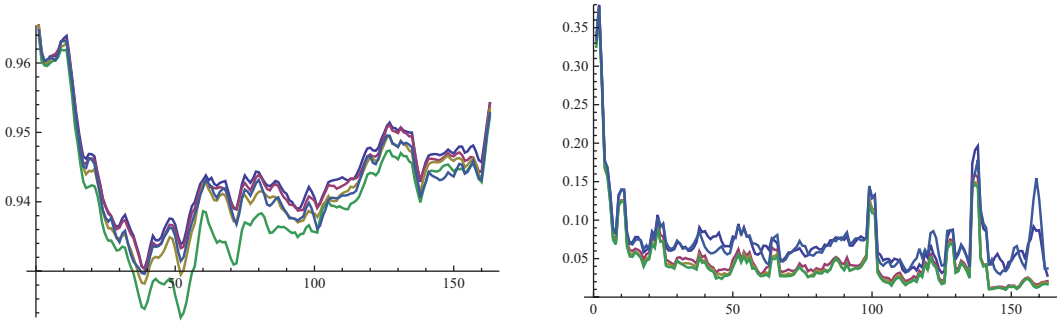


Fig. 16. *Modularity*, $\mathcal{G}_x^1$, batch size 1 e-print, dynamics only: dGlobal@BT excels, followed by dLocal@N_1 and dLocal@BN_4, and then dGlobal@BN_{16} and dGlobal@N_1, which do not benefit from finer batches.



(a) *Modularity*, $\mathcal{G}_x^1$, $b = 200$ e-prints. (b) $\mathcal{R}_g$, $\mathcal{G}_x^1$, $b = 200$ e-prints.

Fig. 17. Nicely stacked as expected, we find (top to bottom for *modularity*) sLocal, tdLocal@0.1, tdLocal@0.2, tdLocal@0.4, and tdLocal@0.6, with sLocal@BN_4 ranging somewhere between 0.4 and 0.6. This order is reversed for distance $\mathcal{R}_g$, with only tdLocal@0.4 being an exception and ranging around tdLocal@0.1. The dynamic algorithm is still an order of magnitude faster. Other large batch sizes yield similar results.



(a) *Modularity*, $\mathcal{G}_x^1$, $b = 200$ e-prints. (b) $\mathcal{R}_g$, $\mathcal{G}_x^1$, $b = 200$ e-prints.

Fig. 18. Almost equivalent in terms of quality are (top to bottom for *modularity*) sLocal, tdLocal@0.1, tdLocal@0.2, and dGlobal@BT, with even tdLocal@0.3 ranging less than 1% behind. In terms of smoothness, the parameter of tdLocal hardly matters: All lie at about 60% the distance of dGlobal@BT and sLocal.

3.9. Summary of Insights

Since we deal with a confusing array of degrees of freedom in the discussion of results, we summarize our findings in the following. The outcomes of our evaluation are very favorable for the dynamic approach in terms of all three criteria. They are quicker, smoother, and yield higher-quality clusterings. In addition, they are by no means sluggish, but they adapt to ground-truth changes quickly without major dents in quality. 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 runtime, 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 (while being rather slow) for *modularity* maximization [Guimerà and Amaral 2005]. In fact, our findings on the quality that dILP yields—an algorithm that largely impedes the escape from a local maximum—corroborate this: The combination of a prep strategy and a maximization heuristic surpassed dILP. In some instances, we even observed a behavior that resembles an asymptotic convergence toward a "consolidated" result.

Although the majority of our findings can be claimed to be very general with respect to the different instances we tested, our data indicate that the best choice for an algorithm in terms of quality may still depend on the nature of the target graph. In particular, we point out that while dLocal surpasses dGlobal on almost all generated graphs, dGlobal is superior on our real-world instance $\mathcal{G}_e$—independent of the batch size. 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. Note that this behavior has not been observed for the static counterparts [Blondel et al. 2008]. In turn, our arXiv trial graphs, which grow and shrink in a volatile but local manner, allow for a small margin of quality improvement if the clustering is regularly adapted globally (rebalanced and coarsened/refined). Only the statics and dGlobal@BT are capable of such an adaption, however, at the cost of smoothness. Universally, the latter algorithm is the fastest.

Time-dependent static clustering appears superior for large batches, as allowing dynamics to accumulate large search spaces draws near a purely static algorithm.

By contrast, an eager dynamic maintenance of a good clustering in between time steps avoids this. However, the aforementioned effect that repeatedly stirring up a clustering allows a dynamic algorithm to escape local optima certainly weakens for long, large, and rough dynamics, which are difficult to react to with a limited search space—even if many trials are allowed.

In conclusion, 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. Large-scale dynamics are best tackled with a mildly time-dependent static algorithm if runtime is not the prime concern.

4. IMPLEMENTATION NOTES

We conducted our experiments on up to eight cores, but using only one core per experiment, of a dual Intel Xeon 5430 running SUSE Linux 11.1. The machine is clocked at 2.6GHz and has 32GB of RAM and 2×1MB of L2 cache. Our algorithms and measures are implemented in Java 1.6.0 13, partially using the yFiles graph library,[8] and run on a 64-bit server VM. The evaluations, plots, and set-ups of experiments were conducted via a frontend programmed in Mathematica (version 7.0.1.0). As priority queue, we use a `java.util.PriorityQueue`. As a data structure that supports backtrack, instead of using a rather involved fully dynamic union-find structure, we maintain a similar structure, a binary forest with actual nodes as leaves and the merge operations as internal tree-nodes.

4.1. A Brief Discussion of Memory Consumption

Compared to runtimes being the limiting factor in our experiments, the memory consumption of our algorithms is low; thus, we do not set our focus on it. In our experiments, the memory required for logging all intermediate results for the evaluation dominated the requirements of the algorithms by far. Algorithms building upon Local require linear space and practically benefit from a compact representation of the graph and of intermediate cluster assignments. Algorithms building upon Global also require linear space; in addition, they require a compact priority queue for their globally greedy decisions and, when using the backtrack prep strategy, a binary forest representing the merge operations. In absolute terms, depending on the implementation, the memory consumption of both approaches is, roughly speaking, about two times the space required for storing the graph itself. The only extra memory required by time-dependent algorithms is that needed for keeping the previous time step in memory.

5. 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. While this article is far from being a comprehensive experimental evaluation of dynamic graph clustering algorithms, it is a first step toward dynamism in a field that has received much interest in the recent past. We propose a feasible methodology for evaluating dynamic graph clustering algorithms, and we hope that since we provde the tools, instances, and random graph generators we used, future work on the topic can be conducted in a comparable and well-founded way. The techniques for augmenting static algorithms to dynamic algorithms might also serve as a blueprint for other families of algorithms; however, this is subject to a lot more future work.

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

[8]Licensed from yWorks; for more information, see www.yworks.com.

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 not only saves 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, just as a history-oblivious elemental optimizer alone cannot compete with the proposed algorithms. 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. For large-scale changes, where a big search space can accumulate, the dynamic algorithms draw near to their static counterparts. For such settings, we proposed and evaluated a time-dependent approach that clusters from scratch but maximizes an explicit combination of *modularity* and smoothness; this technique proved to be superior to the other strategies for large-scale changes.

APPENDIX

A. FURTHER EXPERIMENTS AND VARIOUS OBSERVATIONS

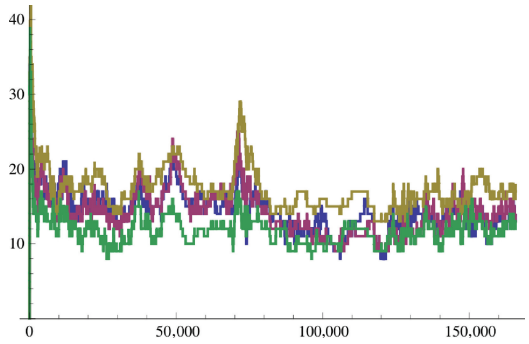


Fig. 19. Cluster count (on an excerpt of $\mathcal{G}_e$): dILP(merge), and dILP (noMerge), roughly bound dLocal and dGlobal from below and above. For merge, (noMerge) merges are hard to revert (emulate).

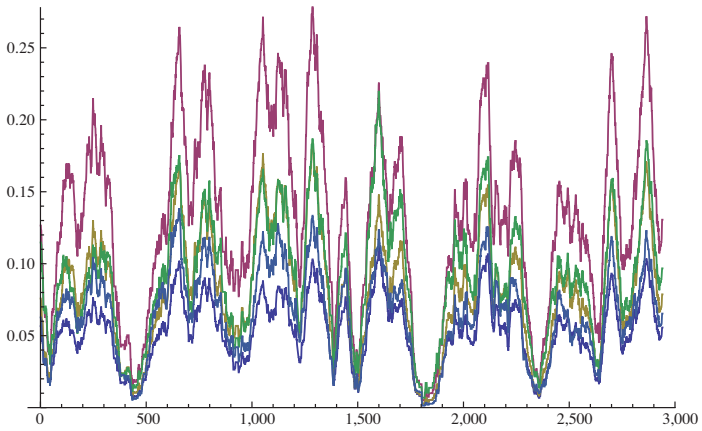


Fig. 20. Different distance measures (for sGlobal on an excerpt of $\mathcal{G}_e$) strongly agreed, qualitatively, in our experiments: Jaccard, Fred & Jain, Fowlkes-Mallows, van Dongen, and Rand.

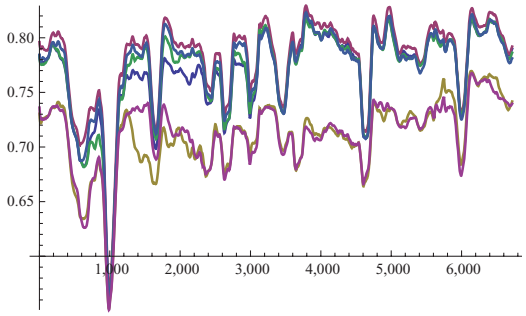


Fig. 21. *Modularity* on the first quarter of $\mathcal{G}_e$, $b = 100$: (bottom to top) as in Figure 5 for $b = 10$, both EOO@10 and EOO@100 fail to follow dLocal@BN_8, dGlobal@N_1, dGlobal@BN_8, and dGlobal@BT.

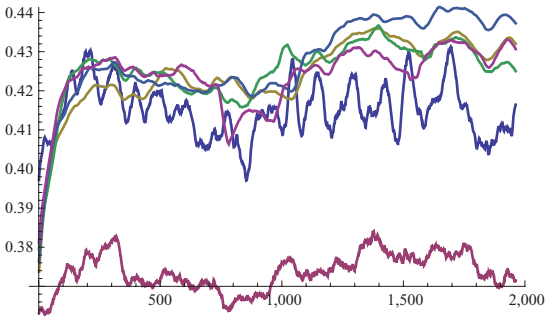
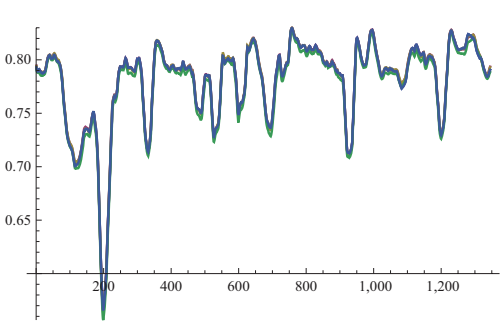
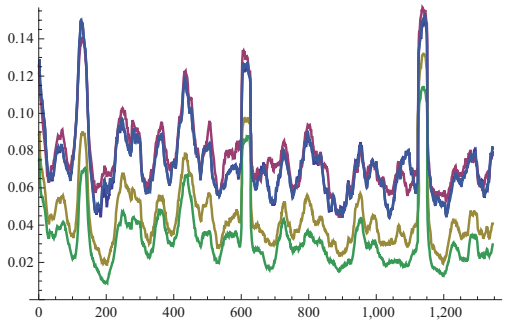


Fig. 22. *Modularity* on $\mathcal{G}_r^1$ (bottom to top): dGlobal@BU, dGlobal@BT, dGlobal@BN_{64}, dGlobal@N_2, dGlobal@N_1, and dGlobal@BN_{16}. Distance plots similarly (but inverted), and runtimes range between the extremes dGlobal@BT (fastest) and dGlobal@BU.



(a) *Modularity* on $\mathcal{G}_e$, $b = 500$ e-mails.



(b) $\mathcal{R}_g$ on $\mathcal{G}_e$, $b = 500$ e-mails.

Fig. 23. Practically equivalent in terms of quality are (bottom to top for $\mathcal{R}_g$) tdLocal@0.2, tdLocal@0.1, sGlobal, dGlobal@BT, and sLocal. In terms of distance, for this large batch size, the time-dependent approach is better by a factor of about 2.

B. TRIALS WITH ARXIV DATA: CATEGORY *NUCLEAR THEORY*

Dynamics and statics, *batch size* 50:

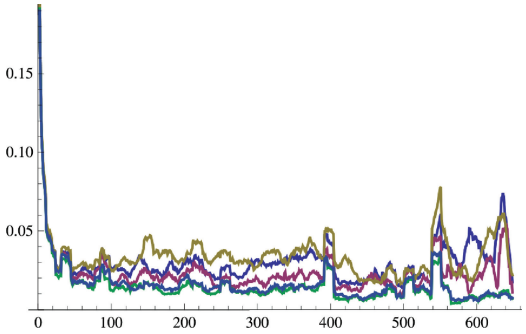


Fig. 24. $\mathcal{R}_g$, $\mathcal{G}_x^1$: On a very low level, the statics (sLocal and sGlobal) and dGlobal@BT are slightly less smooth than dLocal@BN_4 and dGlobal@BN_{16}. The overall level indicates the graph's stability.

Various dynamics, *batch size* 1:

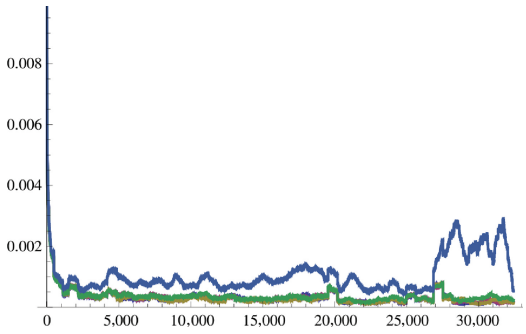


Fig. 25. $\mathcal{R}_g$, $\mathcal{G}_x^1$: On an extremely low level, only dGlobal@BT reacts to the harsh graph changes (see Figure 28); dLocal@N_1, dLocal@BN_4, dGlobal@BN_{16}, and dGlobal@N_1 hardly spot out.

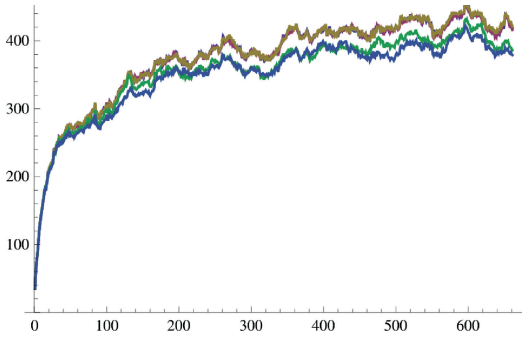


Fig. 26. $|\mathcal{C}|$, $\mathcal{G}_x^1$: Clearly, dGlobal@BT and the statics (sLocal and sGlobal) rebalance and reorganize the clustering, while dLocal@BN_4 and dGlobal@BN_{16} more often enlarge existing clusters.

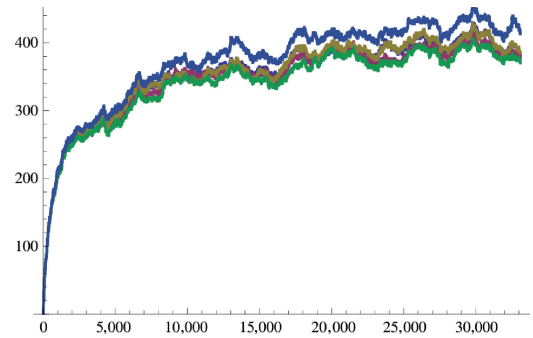


Fig. 27. $|\mathcal{C}|$, $\mathcal{G}_x^1$: As in Figure 26, dGlobal@BT rebalances for higher quality; the locals (dLocal@N_1, dLocal@BN_4) do slightly better than the globals (dGlobal@BN_{16}, dGlobal@N_1).

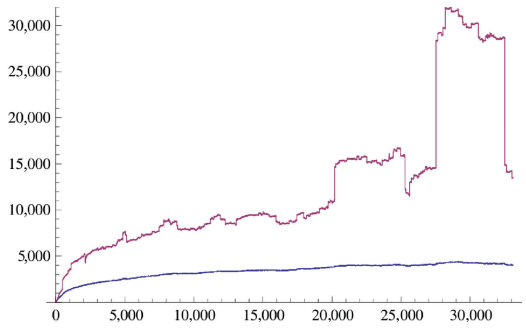


Fig. 28. $\mathcal{G}_x^1$: The number of nodes (lower blue plot) and especially the number of edges (upper red plot) for *Nuclear Theory* are rather volatile. Clique-wise growth proves local-unfriendly.

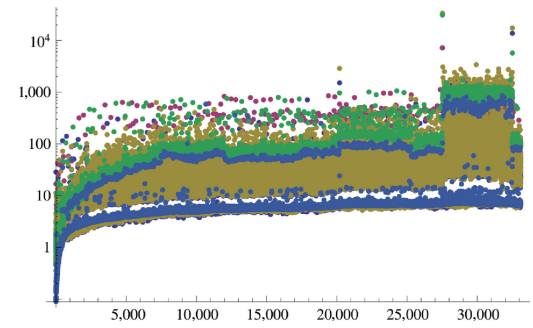


Fig. 29. Runtimes, $\mathcal{G}_x^1$: Only dGlobal@BT is largely unaffected by graph growth. Prep strategy N (dLocal@N_1, dGlobal@N_1, is slower than BN (dLocal@BN_4, dGlobal@BN_{16}).

C. TRIALS WITH ARXIV DATA: CATEGORY *COMPUTER SCIENCE*

Category Computer Science (yielding graph $\mathcal{G}_x^2$) with all subcategories consists of 14K e-prints and 25K authors. We use *batch size* 10 for dynamics, compared to statics using *batch size* 100 in Figure 35. dGlobal@BT excels.

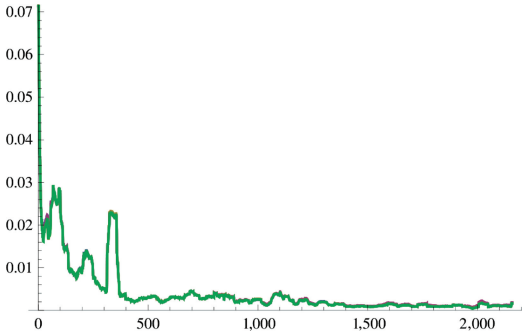


Fig. 30. $\mathcal{R}_g$, $\mathcal{G}_x^2$: Concerning smoothness, all tested dynamic algorithms (dGlobal@BT, dLocal@BN$_4$, dLocal@N$_1$, dGlobal@BN$_{16}$) behave identically.

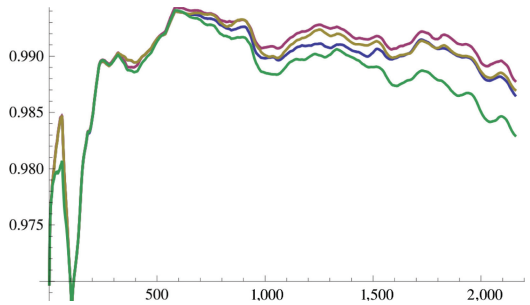


Fig. 31. *Modularity*, $\mathcal{G}_x^2$: On a very high level, only dGlobal@BT starts to slightly stand out after a while, and dGlobal@BN$_{16}$ falls behind; dLocal@BN$_4$ and dLocal@N$_1$ hardly differ.

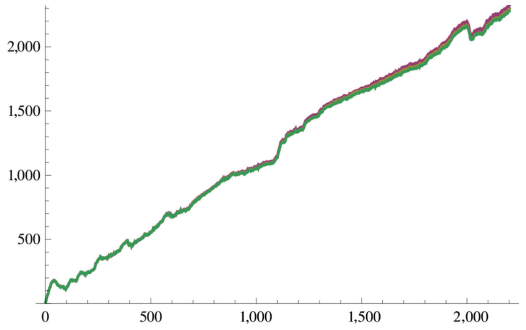


Fig. 32. $|\mathcal{C}|$, $\mathcal{G}_x^2$: All algorithms identify a linear growth in $|\mathcal{C}|$ over time; with dGlobal@BT slightly beyond the others (dLocal@BN$_4$, dLocal@N$_1$, dGlobal@BN$_{16}$).

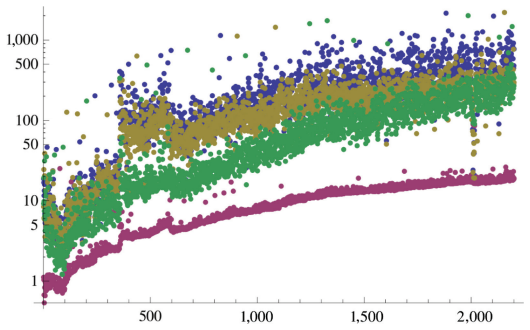


Fig. 33. Runtimes, $\mathcal{G}_x^2$: Even clearer than for *Nuclear Theory*, dGlobal@BT scales well, while the other algorithms slow down more strongly (dLocal@BN$_4$, dLocal@N$_1$, dGlobal@BN$_{16}$).

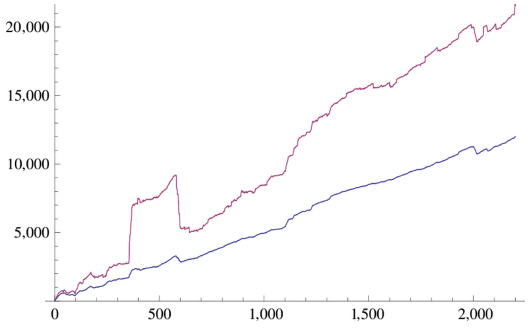


Fig. 34. $\mathcal{G}_x^2$: There seems to be an unflinching growth in popularity of arXiv's categories of computer science. Both the number of nodes (lower blue plot) and the number of edges (upper red plot) grow sharply and steadily.

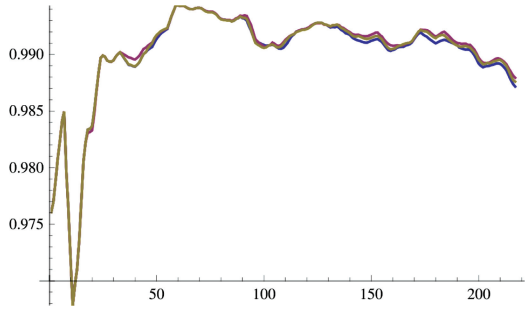


Fig. 35. *Modularity*, $\mathcal{G}_x^2$: Batch size 100 for statics vs. dGlobal@BT. By margins of 0.01%, dGlobal@BT lies between sLocal and sGlobal. dGlobal@BT has virtually the same $\mathcal{R}_g$ and $|\mathcal{C}|$ as the statics but is faster by factors beyond 10^4.

REFERENCES

AGGARWAL, C. C. AND YU, P. S. 2005. Online analysis of community evolution in data streams. In *Proceedings of the 5th SIAM International Conference on Data Mining (SDM '05)*. SIAM.

BLONDEL, V., GUILLAUME, J.-L., LAMBIOTTE, R., AND LEFEBVRE, E. 2008. Fast unfolding of communities in large networks. *J. Stat. Mech. Theory Exp. 2008,* 10.

BRANDES, U., DELLING, D., GAERTLER, M., GÖRKE, R., HÖFER, M., NIKOLOSKI, Z., AND WAGNER, D. 2008. On modularity clustering. *IEEE Trans. Knowl. Data Eng. 20,* 2, 172–188.

BRANDES, U. AND ERLEBACH, T., Eds. 2005. *Network Analysis: Methodological Foundations*. Springer.

BRANDES, U., GAERTLER, M., AND WAGNER, D. 2003. Experiments on graph clustering algorithms. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA '03)*. Springer, 568–579.

CHAKRABARTI, D., KUMAR, R., AND TOMKINS, A. S. 2006. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 554–560.

CLAUSET, A., NEWMAN, M. E. J., AND MOORE, C. 2004. Finding community structure in very large networks. *Phys. Rev. E 70,* 066111.

DELLING, D., GAERTLER, M., GÖRKE, R., AND WAGNER, D. 2008. Engineering comparators for graph clusterings. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM '08)*. Springer, 131–142.

DELLING, D., GÖRKE, R., SCHULZ, C., AND WAGNER, D. 2009. ORCA reduction and contraction graph clustering. In *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM '09)*. Springer, 152–165.

FORTUNATO, S. 2010. Community detection in graphs. *Phys. Rep. 486,* 3–5, 75–174.

FORTUNATO, S. AND BARTHÉLEMY, M. 2007. Resolution limit in community detection. *Proc. Natl. Acad. Sci. 104,* 1, 36–41.

GOOD, B. H., DE MONTJOYE, Y.-A., AND CLAUSET, A. 2010. Performance of modularity maximization in practical contexts. *Phys. Rev. E 81,* 046106.

GÖRKE, R., GAERTLER, M., HÜBNER, F., AND WAGNER, D. 2010. Computational aspects of lucidity-driven graph clustering. *J. Graph Algor. Appl. 14,* 2, 165–197.

GÖRKE, R., HARTMANN, T., AND WAGNER, D. 2009. Dynamic graph clustering using minimum-cut trees. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures (WADS '09)*. Springer, 339–350.

GÖRKE, R., MAILLARD, P., STAUDT, C., AND WAGNER, D. 2010a. Modularity-driven clustering of dynamic graphs. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA '10)*. Springer, 436–448.

GÖRKE, R., MAILLARD, P., STAUDT, C., AND WAGNER, D. 2010b. Modularity-driven clustering of dynamic graphs. Tech. rep. TR 2010-5, ITI Wagner, Department of Informatics, Karlsruhe Institute of Technology, Informatik, Uni Karlsruhe, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000016558.

GÖRKE, R. AND STAUDT, C. 2009. A generator of dynamic clustered random graphs. Tech. rep. TR 2009-7, ITI Wagner, Faculty of Informatics, Universität Karlsruhe, Informatik, Uni Karlsruhe, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000012167.

GUIMERÀ, R. AND AMARAL, L. A. N. 2005. Functional cartography of complex metabolic networks. *Nature 433,* 895–900.

HOPCROFT, J. E., KHAN, O., KULIS, B., AND SELMAN, B. 2004. Tracking evolving communities in large linked networks. *Proc. Natl. Acad. Sci. 101,* 5249–5253.

HÜBNER, F. 2008. The dynamic graph clustering Problem—ILP-based approaches balancing optimality and the mental map. M.S. thesis, Department of Informatics, Diplomarbeit.

KEOGH, E., LONARDI, S., AND RATANAMAHATANA, C. A. 2004. Towards parameter-free data mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 206–215.

NEWMAN, M. E. J. 2004. Analysis of weighted networks. *Phys. Rev. E 70,* 056131, 1–9.

NEWMAN, M. E. J. AND GIRVAN, M. 2004. Finding and evaluating community structure in networks. *Phys. Rev. E 69,* 026113, 1–16.

NOACK, A. AND ROTTA, R. 2009. Multi-level algorithms for modularity clustering. In *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA '09)*. Springer, 257–268.

PALLA, G., BARABÁSI, A.-L., AND VICSEK, T. 2007. Quantifying social group evolution. *Nature 446*, 664–667.

PONS, P. AND LATAPY, M. 2006. Computing communities in large networks using random walks. *J. Graph Algor. Appl. 10,* 2, 191–218.

SCHAEFFER, S. E. 2007. Graph clustering. *Comput. Sci. Rev. 1,* 1, 27–64.

SCHAEFFER, S. E., MARINONI, S., SÄRELÄ, M., AND NIKANDER, P. 2006. Dynamic local clustering for hierarchical ad hoc networks. In *Proceedings of the IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*. IEEE, 667–672.

SUN, J., YU, P. S., PAPADIMITRIOU, S., AND FALOUTSOS, C. 2007. GraphScope: Parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 687–696.

VAN DONGEN, S. M. 2000. Graph clustering by flow simulation. Ph.D. thesis, University of Utrecht.

VAZIRGIANNIS, M., NØRVÅG, K., AND DOULKERIDIS, C. 2006. Peer-to-peer clustering for semantic overlay network generation. In *Proceedings of the 6th International Workshop on Pattern Recognition in Information Systems (PRIS '06)*.

WHITE, S. AND SMYTH, P. 2005. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 5th SIAM International Conference on Data Mining (SDM '05)*. SIAM, 274–285.

Received November 2010; revised September 2012; accepted September 2012