Karlsruhe Reports in Informatics 2011,11
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Dynamic Graph Clustering Combining Modularity and Smoothness

Robert Görke, Pascal Maillard, Andrea Schumm, Christian Staudt, and Dorothea Wagner

2011

Dynamic Graph Clustering Combining Modularity and Smoothness*

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

[1] Institute of Theoretical Informatics
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{rgoerke,schumm,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 sound recommendations for the choice of an algorithm.

1 Introduction

Graph clustering is concerned with identifying and analyzing the group structure of networks[3]. 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., (Brandes and Erlebach 2005; Fortunato 2009). In this work we set our focus on the quality function *modularity*, coined by Newman and Girvan (2004), which has proven itself feasible and reliable in practice, especially as the target function for a maximization approach (see (Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 2008) for further references) that follows the paradigm of parameter-free community discovery (Keogh, Lonardi, and Ratanamahatana 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, Marinoni, Särelä, and Nikander 2006), even though very fast clustering methods have been proposed recently (Blondel, Guillaume, Lambiotte, and Lefebvre 2008; Delling, Görke, Schulz, and Wagner 2009). Second, heuristics for the NP-hard (Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 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 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 re-clustering 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.

*This work was partially supported by the DFG under grant WA 654/15-2/3.

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

1.1 Related Work

Dynamic graph clustering has so far been a rather untrodden field. Recent efforts (Görke, Hartmann, and Wagner 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, Khan, Kulis, and Selman 2004; Palla, Barabási, and Vicsek 2007), besides an array of case studies. Aggarwal and Yu (2005) proposed a parameter-based dynamic graph clustering method which allows user exploration. Parameters are avoided in (Sun, Yu, Papadimitriou, and Faloutsos 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, e.g., in (Chakrabarti, Kumar, and Tomkins 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, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 2008; Fortunato 2009; Schaeffer 2007) for overviews, further references and comparisons to other clustering techniques. Spectral methods, e.g., (White and Smyth 2005), and techniques based on random walks (Pons and Latapy 2006; van Dongen 2000), do not lend themselves well to dynamization due to their non-continuous nature. Variants of greedy agglomeration (Clauset, Newman, and Moore 2004; Blondel, Guillaume, Lambiotte, and Lefebvre 2008), however, are well suited, as we shall see.

This study is based on the preliminary paper (Görke, Maillard, Staudt, and Wagner 2010a). We broaden the range of evaluated algorithms and discuss our results in full detail; we divert several tedious details to our previous technical report (Görke, Maillard, Staudt, and Wagner 2010b), but 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, Newman, and Moore 2004) and the fastest (Blondel, Guillaume, Lambiotte, and Lefebvre 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 ILP[4]-algorithm (Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 2008). We compare the algorithms with their static counterparts and evaluate them experimentally on random preclustered dynamic graphs and on large real-world instances. Then, shifting the focus towards 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 towards the previous results excels.

1.3 Notation

Throughout this paper, we will use the notation of Brandes and Erlebach (2005). We assume that $G = (V, E, \omega)$ is an undirected, weighted, and simple graph[5] with the edge weight function $\omega\colon E \to \mathbb{R}_{\geq 0}$. The

[4]ILP stands for Integer Linear Program.

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

neighborhood of a node v is $N(v) := \{w \in V \mid \{v,w\} \in E\}$. We set $|V| =: n, |E| =: m$ and $\mathcal{C} = \{C_1, \dots, C_k\}$ 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 $k = 1$ ($\mathcal{C}^{\mathbf{1}}$), 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, which is $G(C_i, E(C_i))$. 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 $\deg(C) := \sum_{v \in C} \deg(v)$. 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 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* $\mathcal{G} = (G_0, \dots, G_{t_{\max}})$ 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.

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; Fortunato 2009)), *modularity* does have certain drawbacks such as *non-locality* and *scaling behavior* (Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 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

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

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

(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, Gaertler, Hübner, and Wagner 2010). ModOpt, the problem of optimizing *modularity* is NP-hard (Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 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.

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 $\mathcal{C}^{\text{opt}}(G')$.*

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 $(G_0, \dots, G_\ell = G)$ 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, (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 $\mathcal{C}^{\text{opt}}(G_0)$. Subsequently solving DynModOpt for instances $G_i, \mathcal{C}^{\text{opt}}(G_i), \Delta_i$ yielding $\mathcal{C}^{\text{opt}}(G_{i+1})$, we end with $\mathcal{C}^{\text{opt}}(G_\ell) = \mathcal{C}^{\text{opt}}(G)$, 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 which 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, Gaertler, Görke, and Wagner 2008). Our results strongly suggest that most of these widely accepted measures are qualitatively equivalent in all our (non-pathological) instances (see Figure 19 for an example). We thus restrict our view to the *graph-structural Rand* index (Delling, Gaertler, Görke, and Wagner 2008), 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 (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 clustering $\mathcal{C}(G), \mathcal{C}'(G')$ of different graphs $G = (V,E) \neq G' = (V',E')$, the above measures are not well-defined. A canonical solution is to use the *intersection* of the two graphs, i.e., 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 on 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 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 $\tilde{\mathcal{C}}(G_t)$. 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. 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.

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).

The Global Greedy Algorithm The most prominent algorithm for *modularity* maximization is a globally greedy algorithm (Clauset, Newman, and Moore 2004), which we call Global (Algorithm 1). Starting with singletons, for each pair of clusters, it determines the increase in *modularity* $\mathrm{dQ}_{\cup}$ that

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

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

can be achieved by merging the pair and performs the most beneficial merge. This is repeated until no more improvement is possible. As the pseudo-dynamic algorithm sGlobal, we let this algorithm cluster from scratch at each time step 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 and free nodes.

The Local Greedy Algorithm In a recent work (Blondel, Guillaume, Lambiotte, and Lefebvre 2008) 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 repeatedly lets each node consider moving to one of its neighbors' clusters, if this improves *modularity*. Especially when starting with singletons, moves potentially merge clusters. We denote by $\mathrm{dQ}_{\hookrightarrow}(v, C)$ the change in *modularity* incurred by moving node v into cluster C. From Equation 2 we can see that $\mathrm{dQ}_{\hookrightarrow}(v, C)$ can be computed by finding v's neighbors 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 towards a non-neighbor is always less beneficial than isolating a node, thus checking neighbors' clusters and isolation suffices. 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.

Algorithm 2: Local($G^{0\dots h_{\max}}$, $\tilde{\mathcal{C}}^{0\dots h_{\max}}$, 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              $C \leftarrow \arg\max_{C=\mathcal{C}(u), u \in N(v)} \mathrm{dQ}_{\hookrightarrow}(v, C)$
7              if $\mathrm{dQ}_{\hookrightarrow}(v, C) > 0$ then move$(v, C)$
8              if $\mathrm{dQ}_{\hookrightarrow}(v, \text{new Cluster}) > 0$ then move$(v, \text{new Cluster})$
9      until no more changes
10     $\mathcal{C}^h \leftarrow \mathcal{C}$
11     $(G^{h+1}, \tilde{\mathcal{C}}^{h+1}) \leftarrow$ contract$(G^h, \mathcal{C}^h, P)$
12     $h \leftarrow h + 1$
13 until no more real contractions
14 $\mathcal{C}(G^0) \leftarrow$ unfurl$(\mathcal{C}^{h-1})$
```

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 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., $h_{\max} = 0$, then 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 pseudo-dynamic algorithm, the static variant (as in (Blondel, Guillaume, Lambiotte, and Lefebvre 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 previous results in line 11, i.e., 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: $(G_t,\, G_{\text{old}}^{1,\dots,h_{\max}},\, \tilde{\mathcal{C}},\, \mathcal{C}_{\text{old}}^{1,\dots,h_{\max}}, P)$. 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 (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.

Time-Dependent Local Greedy Suppose we face a problem instance where, despite a steady dynamicity in the graph, we are rather infrequently required to report a new clustering. Along the lines of the procedures described above, 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, de Montjoye, and Clauset (2009) that for *modularity*, the landscape of near-optimal clusterings is broad; if now 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, Kumar, and Tomkins (2006) and *explicitly* enforce smoothness by shaping into an objective function TD_α a convex combination of *modularity* and the *Rand* index: $\mathrm{TD}_\alpha := \alpha \cdot (1 - \mathcal{R}_g(\mathcal{C}^{\text{old}}, \mathcal{C}^{\text{new}})) + (1-\alpha)\cdot \mathrm{mod}(\mathcal{C}^{\text{new}})$.[6] 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 re-formulation 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 is coined tdLocal@α.

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

$$\mathcal{R}_g(\mathcal{C},\mathcal{C}') = 1 - \frac{1}{|E''|}\sum_{\{uv\}\in E''}\Big(\underbrace{\delta_{uv}\delta'_{uv}}_{=1 \text{ if } \{uv\}\in E''_{11}} + \underbrace{(1-\delta_{uv})(1-\delta'_{uv})}_{=1 \text{ if } \{uv\}\in E''_{00}}\Big) \tag{3}$$

$$= 1 - \frac{1}{|E''|}\sum_{\{uv\}\in E''}\left(1 - \delta_{uv} + \delta'_{uv}\cdot(2\delta_{uv} - 1)\right) \tag{4}$$

Being a contribution to the above 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

$$\sum_{v\in C_b,\{u,v\}\in E''} \mathrm{dS}(u,v) - \sum_{w\in C_a,\{u,w\}\in E''} \mathrm{dS}(u,w)\ , \tag{5}$$

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 $\{u,v\} \in E' \setminus E$. This view is convenient, algorithmically, since for computing changes $\mathrm{dQ}_{\hookrightarrow}(u,v)$ in *modularity* in Algorithm 2 (lines 6, 8), we proceed similarly: As suggested by Equation 2, we record for each edge $e = \{u,v\}$, the change in *modularity* that putting v and u into

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

the same cluster incurs. In order to actually use TD_α, we can therefore simply replace the term $\mathrm{dQ}_{\hookrightarrow}$ in Algorithm 2 by $((1-\alpha)\cdot \mathrm{dQ}_{\hookrightarrow} + \alpha \cdot \frac{2\delta_{uv}-1}{|E''|})$ and use this term as edge value just as the purely *modularity*-based version of the algorithm does (see, e.g., (Blondel, Guillaume, Lambiotte, and Lefebvre 2008)).[7] 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 v's options. 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 avoid additional asymptotic runtime.

ILP While optimal *modularity* is out of reach, the problem *can* be cast as an ILP (Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner 2008). Based on Equation 2, a (binary) distance relation $\mathcal{X}$ indicates whether elements are in the same cluster (set $\tilde{V} = V$ for now):

$$\mathcal{X}(\tilde{V}) := \{X_{uv} : \{u,v\} \in \binom{\tilde{V}}{2}\} \quad \text{with} \quad X_{uv} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = \mathcal{C}(v) \\ 1 & \text{otherwise} \end{cases} , \tag{6}$$

$$\text{constrained by } \forall \{u,v,w\} \in \binom{\tilde{V}}{3} : \begin{cases} X_{uv} + X_{vw} - X_{uw} \geq 0 \\ X_{uv} + X_{uw} - X_{vw} \geq 0 \\ X_{uw} + X_{vw} - X_{uv} \geq 0 \end{cases} , X \in \{0,1\} , \tag{7}$$

$$\text{set as to minimize } \mathrm{mod}_{\mathrm{ILP}}(G, \mathcal{C}_G) = \sum_{\{u,v\} \in \binom{\tilde{V}}{2}} \left(\omega(u,v) - \frac{\omega(u)\cdot\omega(v)}{2\cdot\omega(E)} \right) X_{uv} . \tag{8}$$

To ensure the properties of an equivalence relation, Equation 7 represents transitivity, we can omit the other two: Reflexivity, $X_{uu} = 0$, is automatically ensured since a node is always in the same cluster as itself. Symmetry, $X_{uv} = X_{vu}$, is ensured since 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, i.e., 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 *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 Equations 6 and 7 and change Equation 8.

For (i), variables $\mathcal{Y}$ indicating the distance of node $u \in \tilde{V}$ to cluster $C \in \tilde{\mathcal{C}}$ are introduced (again binary, i.e., 0 iff $u \in C$) and triplets of constraints similar to Equation 7 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 6 and 7 plus the following:

$$\mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}}) := \{Y_{uC} : \{u, C\} \in \tilde{V} \times \tilde{\mathcal{C}}\} \quad \text{with} \quad Y_{uC} = \begin{cases} 0 & \text{if } \mathcal{C}(u) = C \\ 1 & \text{otherwise} \end{cases} , \tag{9}$$

$$\text{constrained by } \forall \{u,v,C\} \in \binom{\tilde{V}}{2} \times \tilde{\mathcal{C}} : \begin{cases} X_{uv} + Y_{uC} - Y_{vC} \geq 0 \\ X_{uv} + Y_{vC} - Y_{uC} \geq 0 \\ Y_{uC} + Y_{vC} - X_{uv} \geq 0 \end{cases} , Y_{uC} \in \{0,1\} \tag{10}$$

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

Table 2: Reactions of the algorithms to graph events. Isolated nodes are made singletons when inserted and simply deleted when removed. With "$\rightarrow S$" we indicate that a *prep strategy* prepares a *preclustering*. Algorithm tdLocal@α is covered by sLocal.

cause	algorithms' reactions					
event	sGlobal	dGlobal	sLocal	dLocal	dILP	dEOO
Δ	–	$\rightarrow S$	–	$\rightarrow S$	$\rightarrow S$, 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)

$$\text{and also by } \forall u \in \tilde{V} : \sum_{C \in \tilde{\mathcal{C}}} Y_{uC} \geq k-1 \quad \text{(a node's cluster must be unique)}, \tag{11}$$

$$\text{as to minimize } \operatorname{mod}_{\substack{\mathsf{pILP} \\ \text{no merge}}}(G, \mathcal{C}) = \sum_{X_{uv} \in \mathcal{X}(\tilde{V})} \left(\omega(u,v) - \frac{\omega(u) \cdot \omega(v)}{2\omega(E)} \right) X_{uv} \tag{12}$$
$$+ \sum_{Y_{uC} \in \mathcal{Y}(\tilde{V}, \tilde{\mathcal{C}})} \left(\sum_{w \in C} \left(\omega(u,w) - \frac{\omega(u) \cdot \omega(w)}{2\omega(E)} \right) \right) Y_{uC}$$

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 7. Details on this formulation can be found in (Görke, Maillard, Staudt, and Wagner 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 which 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.

Elemental Operations Optimizer Method EOO performs a bounded number of operations, trying to increase the quality. Specifically, we allow moving (to a neighbor) or splitting off (isolating) a node and merging its cluster with one of the others, as listed in Table 1. 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 $\text{op}_{\max}$ of operations have been executed. Our dynamic algorithm dEOO (or dEOO@$\text{op}_{\max}$) 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 post-processing tools.

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

Table 3: Overview of how strategies handle graph events. Changes to edges' weights are analog to creations/removals. Degree-0 nodes are universally made singletons when inserted and ignored when removed.

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

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 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.

2.4 Backtrack Strategy

The *backtrack strategy* (BT) records the merge operations of Global and backtracks them if a graph modification suggests their reconsideration. We rigorously detail in (Görke, Maillard, Staudt, and Wagner 2010b) 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 re-classify 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. For more details on these operations see (Görke, Maillard, Staudt, and Wagner 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, Maillard, Staudt, and Wagner 2010b).

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

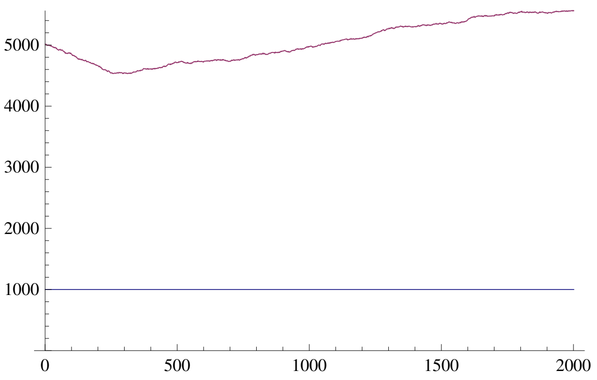
3 Experimental Evaluation of Dynamic Algorithms[8]

3.1 Instances

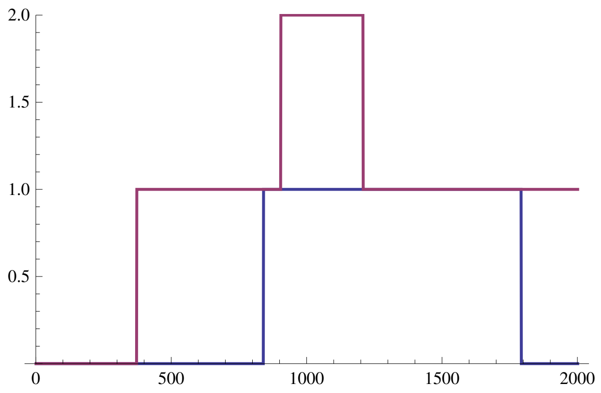
We use both generated graphs and real-world instances. We briefly describe them here, but for more details please see (Görke and Staudt 2009) and (Hübner 2008).

[8]For implementation notes see App. 4, supplementary information is stored at i11www.iti.uni-karlsruhe.de/projects/spp1307/dyneval

Random Graphs {ran} Our Erdős-Rényi-type generator builds upon (Brandes, Gaertler, and Wagner 2003) 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. Our generator and a ready-to-use software package are thoroughly described in (Görke and Staudt 2009); we here briefly summarize its behavior. The generator's ground-truth clustering defines edge probabilities (p_{in} inside clusters and p_{out} in between) and thereby steers 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 to. We conducted experiments for a large number of settings, varying size, density, node/edge-volatility, stability of clusters, etc., and in the following only give representative plots, and point out specific peculiarities. A number of plots uses a graph coined $\mathcal{G}_1$, 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.



(a) Numbers of **nodes** end **edges**, this instance does not allow node events but only edge events for well controlled experiments. $\mathcal{G}_1$ uses $b = 10$ atomic events per time step.



(b) Numbers and types (**split**, **merge**) of ongoing changes in the clustering, plateaus indicate that a ground-truth change has not yet been reacted to by the reference, i.e., arguably is not yet well visible in the graph.

Fig. 2: Graph $\mathcal{G}_1$, non-default parameters of its generation: $t_{\max} = 2k$, $n_0 = 1k$, $|\mathcal{C}| = 20$, $b = 10$, $p_\omega = 10^{-3}$ ($=P$[cluster-event/time step]), $p_{\text{in}} = 0.1$, $p_{\text{out}} = 0.005$.

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. This 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. Fig. 3 shows the temporal development of this graph in terms of n (**lower**) and m (**upper**) per 100 events. The first peak stems

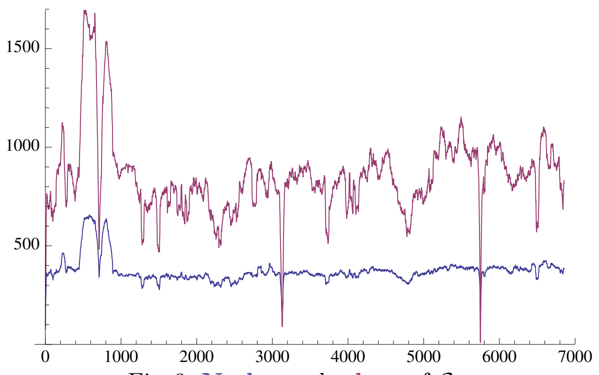


Fig. 3: **Nodes** and **edges** of $\mathcal{G}_e$.

from a spam attack in late '06, the two large drops from Christmas breaks and the smaller drops from spring and autumn breaks.

arXiv Graphs {arx} Since 1992 the *arXiv.org e-Print archive*[9] 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 Fig. 4 for three examples. We let e-prints time out after two 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 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 criteria quality (*modularity*), *smoothness* ($\mathcal{R}_g$) and runtime (ms), and additionally $|\mathcal{C}|$. 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 quite a while, before eventually falling behind. We generally observed that very few operations were actually conducted by dEOO; on most graphs (e.g., $\mathcal{G}_e$) not even $\mathrm{op}_{\max} = 10$ was a limiting value. This indicates that dEOO lacks the ability to sufficiently renovate the clustering—owing to it being 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 $\mathrm{op}_{\max}$), and the number of identified clusters is comparably low on the 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, Guillaume, Lambiotte, and Lefebvre (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

[9]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 .

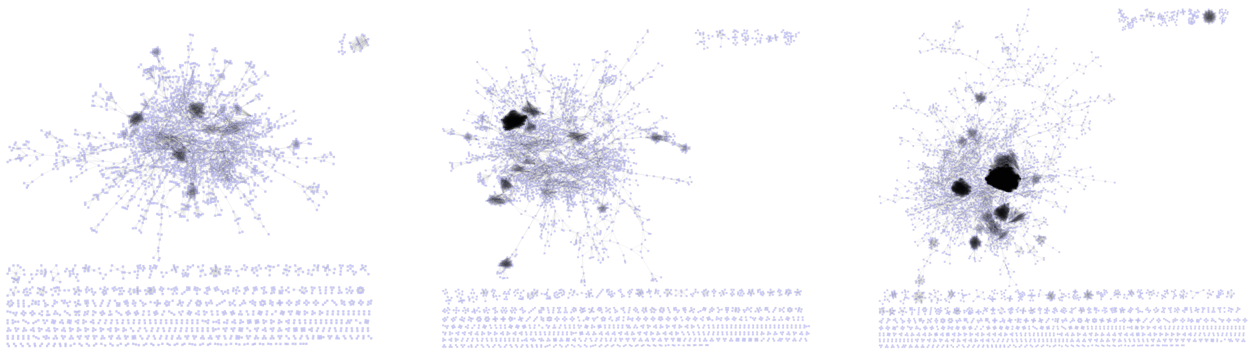


Fig. 4: *ArXiv* category *Nuclear Theory*, containing 33k e-prints, after '01, '05 and '09.

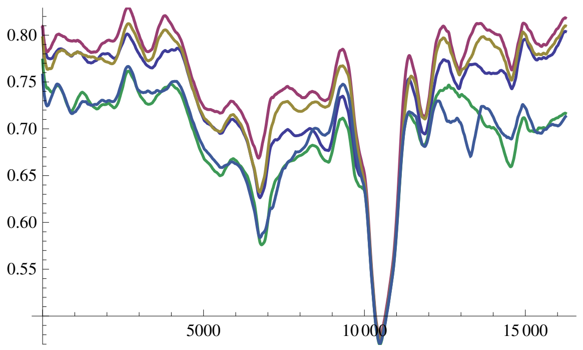


Fig. 5: *Modularity* on the first quarter of $\mathcal{G}_e$, $b = 10$: (bottom to top) both **EOO@10** and **EOO@100** fail to follow **dLocal@BN**$_8$, **dGlobal@BN**$_8$ and **dGlobal@BN**$_BT$. Larger batches to do not help, see Figure 20.

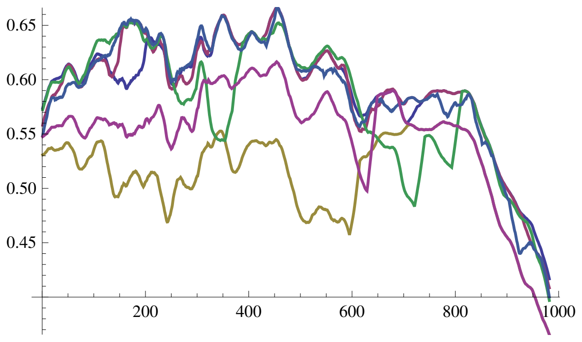


Fig. 6: *Modularity* on G'_1 (G_1 with increased clustering volatility), $b = 1$: (bottom to top) **EOO@10** and **EOO@100** lag behind the **reference** and **dLocal@BN**$_8$, **dGlobal@BN**$_8$ and **dGlobal@BN**$_BT$.

plots on discrepancies between orders to (Görke, Maillard, Staudt, and Wagner 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, being the simpler variant.

pILP Variants. Allowing the ILP to merge existing clusters takes longer, and clusters coarser—which is quite intuitive—but also yields a slightly worse *modularity*. We conjecture that the reason for this is that merging invites hazardous local 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 18 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 Fig. 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. We thus 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.

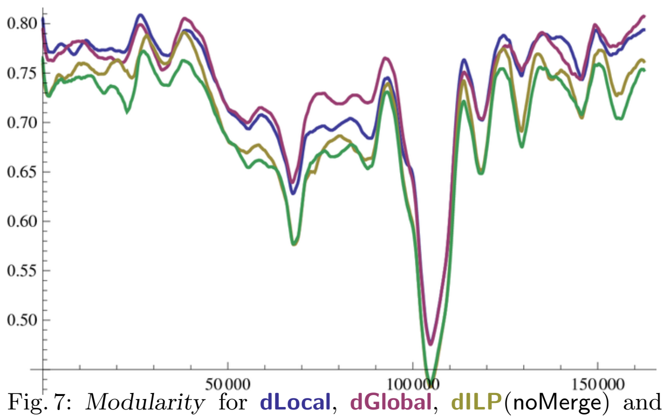


Fig. 7: *Modularity* for **dLocal**, **dGlobal**, **dILP**(noMerge) and **dILP**(merge) on the first quarter of $\mathcal{G}_e$, batch size 1, strategy BN$_8$.

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, Guillaume, Lambiotte, and Lefebvre (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 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.

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

The strategy which 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 Sec. 2.2, but still produces competitive quality. However, it often yields a smoothness almost 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 of 20 at 1k nodes, respectively) and algorithm dLocal@BN_4 (with a speedup of 5 over sLocal). Figure 21 exemplarily illustrates a comparison of prep strategies for dGlobal in terms of *modularity* on $\mathcal{G}_1$.

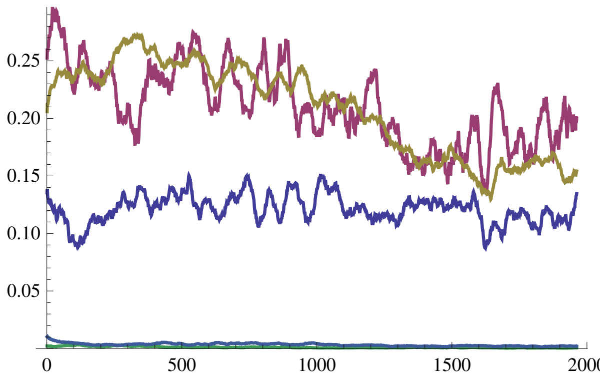


Fig. 8: $\mathcal{R}_g$ on $\mathcal{G}_1$: **sGlobal** and **sLocal** are less smooth (factor 100) than **dLocal@BN$_4$**, **dGlobal@BN$_{16}$** (bottom); **dGlobal@BT** competes well.

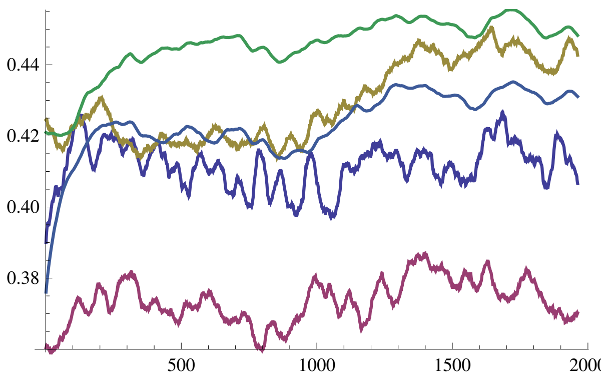


Fig. 9: *Modularity* on $\mathcal{G}_1$: **dGlobal@BT** (**lower blue**) and **dGlobal@BN$_{16}$** (**upper blue**) beat **sGlobal**; **dLocal@BN$_4$** beats **sLocal**.

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 Fig. 9, each dynamic candidate beats its static counterpart in terms of *modularity*. On the generated graphs, dLocal is superior to dGlobal, and faster, this is not the case for the email network—here both Global algorithms beat each Local algorithm. In terms of smoothness (Fig. 8), dynamics (except for dGlobal@BT) are superior to statics by a factor of ca. 100, but even dGlobal@BT beats them. For large batch sizes (e.g., 100 and beyond), dynamics are still clearly smoother, but eventually lag behind the statics in terms of quality by a few percent.

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 by 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, clearly, static counterpart algorithms are not subject to such issues, since they "forget " their previous work.

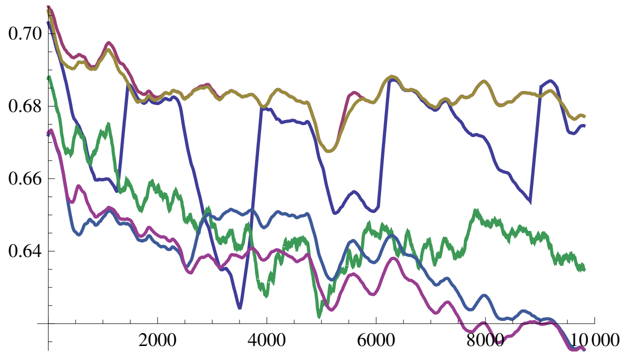


Fig. 10: *Modularity* on a long (10k×10 events) random graph. **Reference** exhibits jumps where the clustering changes, before which it clearly decreases. In contrast, the dynamics adapt quickly to changes. Top to bottom: **dLocal@N$_1$**, **dLocal@BN$_4$**, **reference**, **dGlobal@BT**, **dGlobal@BN$_8$** **dGlobal@BN$_{16}$** which does not benefit from its larger search space.

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 ~ 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). 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. Keeping an eye on the average *modularity*, which impacts α's effect, we focus on tdLocal@0.2, which is a reasonable compromise. Having compete tdLocal@0.2 against sLocal and dLocal@BN$_4$ on a long random graph with 1000 time steps

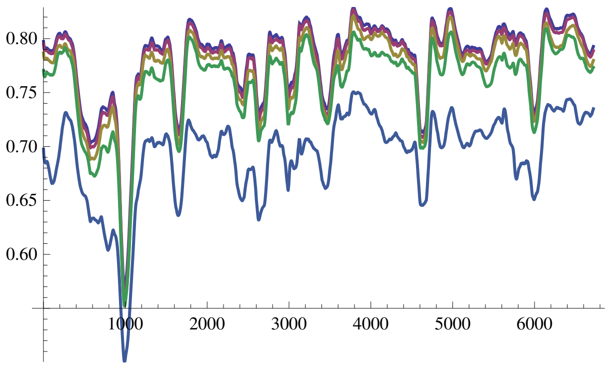


Fig. 11: *Modularity*, on $\mathcal{G}_e$: batch size 100 e-mails: top to bottom: **tdLocal@0.1**, **tdLocal@0.2**, **tdLocal@0.3** **tdLocal@0.4** and **tdLocal@0.6**.

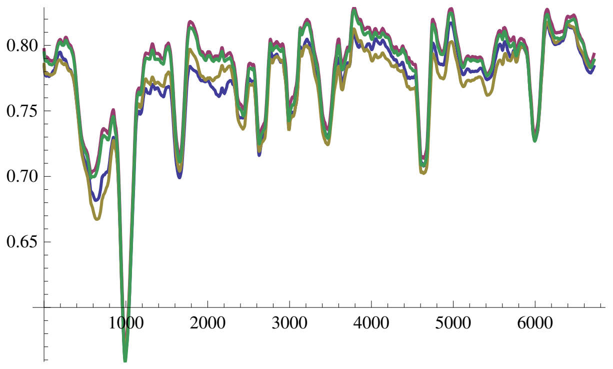


Fig. 12: *Modularity*, on $\mathcal{G}_e$: batch size 100 e-mails: top to bottom: **sLocal**, **tdLocal@0.2**, **dGlobal@BN$_{128}$**, **dGlobal@N$_1$**,

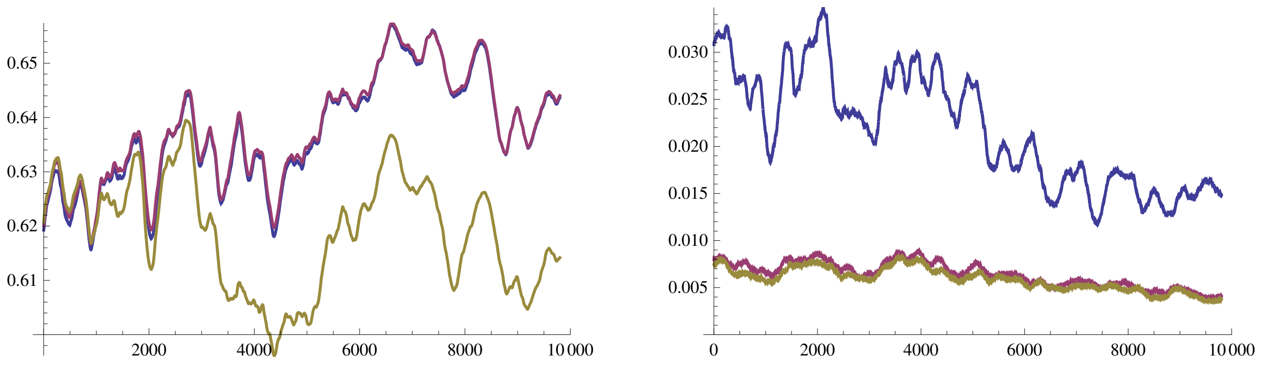
with 1000 events each, we can confirm that the time-dependent strategy holds the best of both worlds for large batches: Figure 13 plainly exemplifies how tdLocal@0.2 equals the conventional static algorithm in quality, and matches the dynamic algorithm in terms of smoothness. In fact, its average quality 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 5 it is not hard to see that this is due to $\mathcal{R}_g$'s tendency to first put a set of nodes into a separate cluster, if *modularity* intends to migrate them into a different cluster.

3.7 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. These graphs consist solely of glued cliques of authors (papers), established within single time steps 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 non-arbitrary number thereof in large graphs (Good, de Montjoye, and Clauset 2009), 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. 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 14 and 15 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 Apps. B and C for insightful further results that exhibit dGlobal@BT's appropriateness.

Figure 16 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 towards 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 lags behind the time-dependent approach. Cross-comparing with results for dGlobal@BT (as illustrated in Figure 17), we can state that the global backtrack



(a) *Modularity*, {ran}, $b = 1000$. (b) $\mathcal{R}_g$, {ran}, $b = 1000$.

Fig. 13: Results on a random graph with around 1000 nodes, $10k \times 1k$ events, and expected intra- and inter-cluster 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 formers average distance.

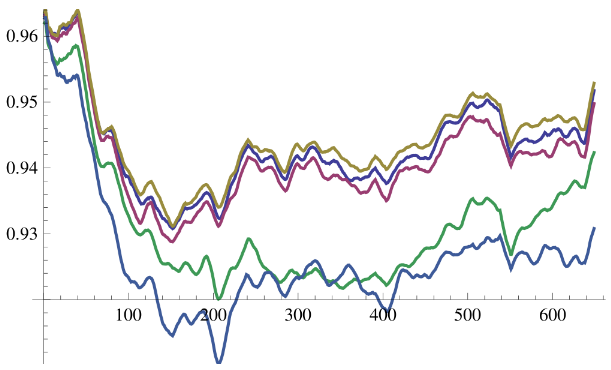


Fig. 14: *Modularity*, {arx}, batch size 50 e-prints: Backtracking (**dGlobal@BT**) easily follows the static algorithms (**sLocal** and **sGlobal**); even **dLocal@BN**$_4$ and **dGlobal@BN**$_{16}$ lag behind by only $\sim 1\%$.

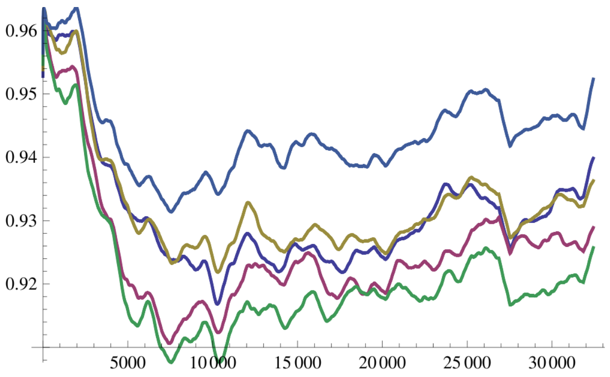
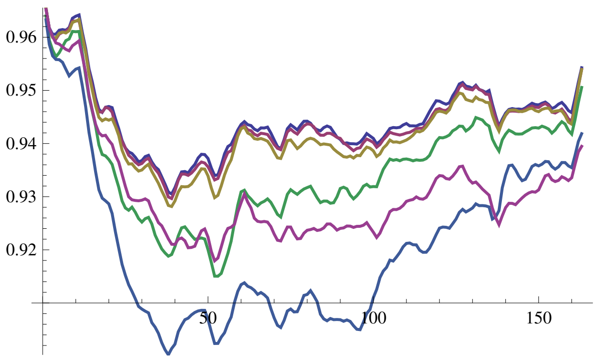


Fig. 15: *Modularity*, {arx}, batch size 1 e-print, dynamics only: **dGlobal@BT** excels, followed by **dLocal@N**$_1$ and **dLocal@BN**$_4$ and then **dGlobal@BN**$_{16}$ and **dGlobal@N**$_1$ which don't benefit from finer batches.

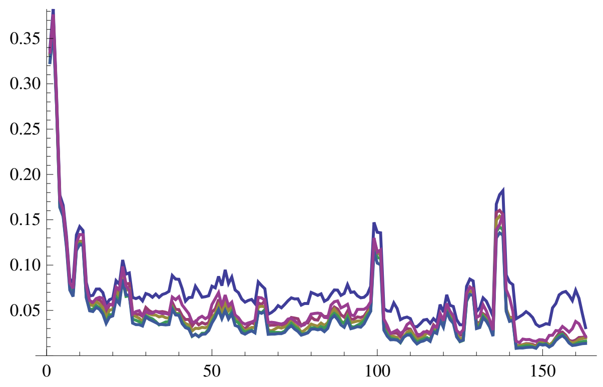
strategy competes very well with the time-dependent strategy, with the slight advantage of the latter in terms of both quality and smoothness getting larger with larger batches. If for such cases runtime remains a central issue, dGlobal@BT is still by far the best choice for graphs like {arx} (see Figure 22 for analogous observations on $\mathcal{G}_e$).

3.8 Summary of Insights

Since we deal with a confusing array of degrees of freedom in the discussion of results above, 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, and in addition, they are by no means sluggish, but 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

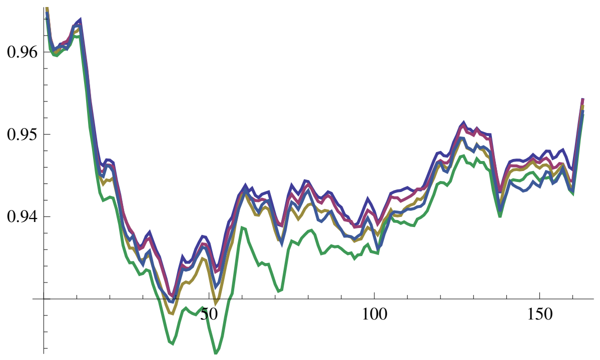


(a) *Modularity*, {arx}, $b = 200$ e-prints.

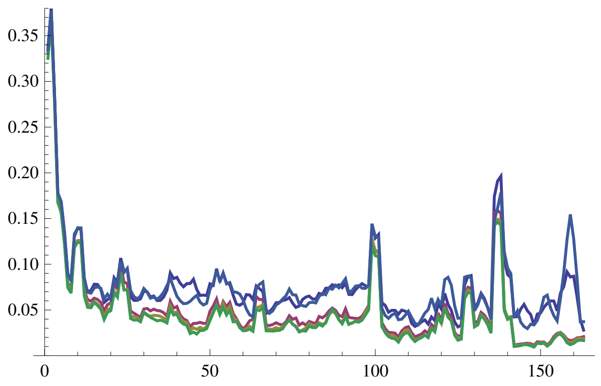


(b) $\mathcal{R}_g$, {arx}, $b = 200$ e-prints.

Fig. 16: 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*, {arx}, $b = 200$ e-prints.



(b) $\mathcal{R}_g$, {arx}, $b = 200$ e-prints.

Fig. 17: 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**.

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 (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 towards 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 indicates 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, Guillaume, Lambiotte, and Lefebvre 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 (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.

Time-dependent static clustering appears superior for large batches, as having dynamics 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 abovementioned 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.

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. 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, 1 per experiment, of a dual *Intel Xeon* 5430 running *SUSE Linux* 11.1. The machine is clocked at 2.6GHz, 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[10], and run on a 64-Bit Server VM. Evaluations, plots and the setups 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 which 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.

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. 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, 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 which 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.

[10]Licensed from *yWorks* for more information, see www.yworks.com .

References

[Aggarwal and Yu 2005] Aggarwal, C. C. and P. S. Yu (2005). Online Analysis of Community Evolution in Data Streams. See ?? (2005).

[Blondel, Guillaume, Lambiotte, and Lefebvre (2008] Blondel, V., J.-L. Guillaume, R. Lambiotte, and E. Lefebvre (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment 2008*(10).

[Brandes, Delling, Gaertler, Görke, Höfer, Nikoloski, and Wagner (2008] Brandes, U., D. Delling, M. Gaertler, R. Görke, M. Höfer, Z. Nikoloski, and D. Wagner (2008, February). On Modularity Clustering. *IEEE Transactions on Knowledge and Data Engineering 20*(2), 172–188.

[Brandes and Erlebach (2005] Brandes, U. and T. Erlebach (Eds.) (2005, February). *Network Analysis: Methodological Foundations*, Volume 3418 of *Lecture Notes in Computer Science*. Springer.

[Brandes, Gaertler, and Wagner (2003] Brandes, U., M. Gaertler, and D. Wagner (2003). Experiments on Graph Clustering Algorithms. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, Volume 2832 of *Lecture Notes in Computer Science*, pp. 568–579. Springer.

[Chakrabarti, Kumar, and Tomkins (2006] Chakrabarti, D., R. Kumar, and A. S. Tomkins (2006). Evolutionary Clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 554–560. ACM Press.

[Clauset, Newman, and Moore (2004] Clauset, A., M. E. J. Newman, and C. Moore (2004). Finding community structure in very large networks. *Physical Review E 70*(066111).

[Delling, Gaertler, Görke, and Wagner (2008] Delling, D., M. Gaertler, R. Görke, and D. Wagner (2008, June). Engineering Comparators for Graph Clusterings. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management (AAIM'08)*, Volume 5034 of *Lecture Notes in Computer Science*, pp. 131–142. Springer.

[Delling, Görke, Schulz, and Wagner (2009] Delling, D., R. Görke, C. Schulz, and D. Wagner (2009, June). ORCA Reduction and ContrAction Graph Clustering. In A. V. Goldberg and Y. Zhou (Eds.), *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM'09)*, Volume 5564 of *Lecture Notes in Computer Science*, pp. 152–165. Springer.

[Fortunato (2009] Fortunato, S. (2009). Community detection in graphs. *Physics Reports 486*(3–5), 75–174.

[Fortunato and Barthélemy (2007] Fortunato, S. and M. Barthélemy (2007). Resolution limit in community detection. *Proceedings of the National Academy of Science of the United States of America 104*(1), 36–41.

[Good, de Montjoye, and Clauset (2009] Good, B. H., Y.-A. de Montjoye, and A. Clauset (2009, October). The performance of modularity maximization in practical contexts.

[Görke, Gaertler, Hübner, and Wagner (2010] Görke, R., M. Gaertler, F. Hübner, and D. Wagner (2010). Computational Aspects of Lucidity-Driven Graph Clustering. *Journal of Graph Algorithms and Applications 14*(2), 165–197.

[Görke, Hartmann, and Wagner (2009] Görke, R., T. Hartmann, and D. Wagner (2009, August). Dynamic Graph Clustering Using Minimum-Cut Trees. In F. Dehne, J.-R. Sack, and R. Tamassia (Eds.), *Algorithms and Data Structures, 11th International Workshop (WADS'09)*, Volume 5664 of *Lecture Notes in Computer Science*, pp. 339–350. Springer.

[Görke, Maillard, Staudt, and Wagner (2010a] Görke, R., P. Maillard, C. Staudt, and D. Wagner (2010a, May). Modularity-Driven Clustering of Dynamic Graphs. In P. Festa (Ed.), *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, Volume 6049 of *Lecture Notes in Computer Science*, pp. 436–448. Springer.

[Görke, Maillard, Staudt, and Wagner (2010b] Görke, R., P. Maillard, C. Staudt, and D. Wagner (2010b). Modularity-Driven Clustering of Dynamic Graphs. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH). Informatik, Uni Karlsruhe, TR 2010-5.

[Görke and Staudt (2009] Görke, R. and C. Staudt (2009). A Generator for Dynamic Clustered Random Graphs. Technical report, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH). Informatik, Uni Karlsruhe, TR 2009-7.

[Guimerà and Amaral (2005] Guimerà, R. and L. A. N. Amaral (2005, February). Functional Cartography of Complex Metabolic Networks. *Nature 433*, 895–900.

[Hopcroft, Khan, Kulis, and Selman (2004] Hopcroft, J. E., O. Khan, B. Kulis, and B. Selman (2004, April). Tracking Evolving Communities in Large Linked Networks. *Proceedings of the National Academy of Science of the United States of America 101.*

[Hübner (2008] Hübner, F. (2008, May). The Dynamic Graph Clustering Problem - ILP-Based Approaches Balancing Optimality and the Mental Map. Master's thesis, Department of Informatics. Diplomarbeit.

[Keogh, Lonardi, and Ratanamahatana (2004] Keogh, E., S. Lonardi, and C. A. Ratanamahatana (2004). Towards Parameter-Free Data Mining. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 206–215. ACM Press.

[Newman (2004] Newman, M. E. J. (2004). Analysis of Weighted Networks. *Physical Review E 70*(056131), 1–9.

[Newman and Girvan (2004] Newman, M. E. J. and M. Girvan (2004). Finding and evaluating community structure in networks. *Physical Review E 69*(026113).

[Noack and Rotta (2009] Noack, A. and R. Rotta (2009, June). Multi-level Algorithms for Modularity Clustering. In J. Vahrenhold (Ed.), *Proceedings of the 8th International Symposium on Experimental Algorithms (SEA'09)*, Volume 5526 of *Lecture Notes in Computer Science*, pp. 257–268. Springer.

[Palla, Barabási, and Vicsek (2007] Palla, G., A.-L. Barabási, and T. Vicsek (2007, April). Quantifying social group evolution. *Nature 446*, 664–667.

[Pons and Latapy (2006] Pons, P. and M. Latapy (2006). Computing Communities in Large Networks Using Random Walks. *Journal of Graph Algorithms and Applications 10*(2), 191–218.

[Schaeffer (2007] Schaeffer, S. E. (2007, August). Graph Clustering. *Computer Science Review 1*(1), 27–64.

[Schaeffer, Marinoni, Särelä, and Nikander (2006] Schaeffer, S. E., S. Marinoni, M. Särelä, and P. Nikander (2006, September). Dynamic Local Clustering for Hierarchical Ad Hoc Networks. In *Proceedings of Sensor and Ad Hoc Communications and Networks, 2006.*, Volume 2, pp. 667–672. IEEE Computer Society.

[?? (2005] SDM'05 (2005). *Proceedings of the fifth SIAM International Conference on Data Mining.* SIAM.

[Sun, Yu, Papadimitriou, and Faloutsos (2007] Sun, J., P. S. Yu, S. Papadimitriou, and C. Faloutsos (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*, pp. 687–696. ACM Press.

[van Dongen (2000] van Dongen, S. M. (2000). *Graph Clustering by Flow Simulation.* Ph. D. thesis, University of Utrecht.

[White and Smyth (2005] White, S. and P. Smyth (2005). A Spectral Clustering Approach to Finding Communities in Graphs. See ?? (2005), pp. 274–285.

A Further Experiments and Various Observations

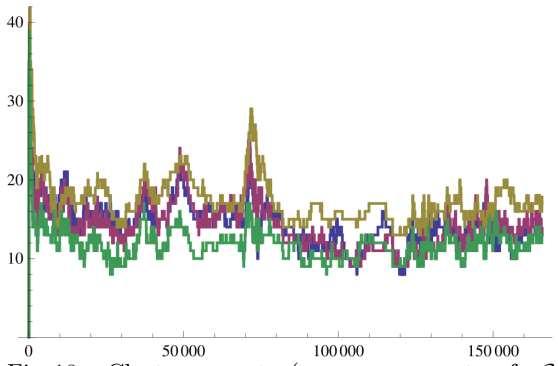


Fig. 18: Cluster count (on an excerpt of $\mathcal{G}_e$): **dILP(merge)** and **dILP(noMerge)** roughly bound **dLocal** and **dGlobal** from below and above. For **merge** (**noMerge**) merges are hard to revert (emulate).

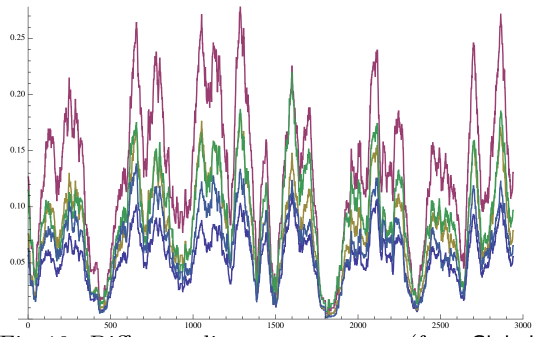


Fig. 19: Different distance measures (for sGlobal on an excerpt of $\mathcal{G}_e$) strongly agreed, qualitatively, in our experiments: **Jaccard**, **Fred & Jain**, **Fowlkes-Mallows**, **van Dongen** and **Rand**.

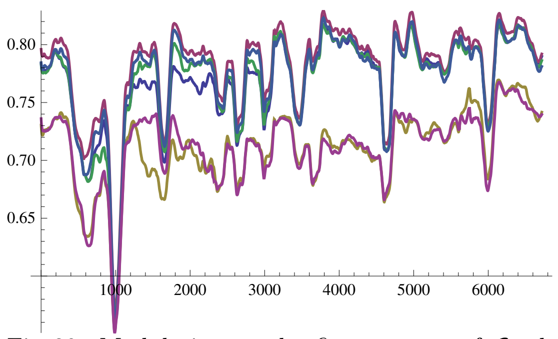


Fig. 20: *Modularity* on the first quarter of $\mathcal{G}_e$, $b = 100$: (bottom to top) as in Figure 5 for $b = 10$, both **EOO@10** and **EOO@100** fail to follow **dLocal@BN**$_8$, **dGlobal@N**$_1$, **dGlobal@BN**$_8$, **dGlobal@BN**$_{BT}$.

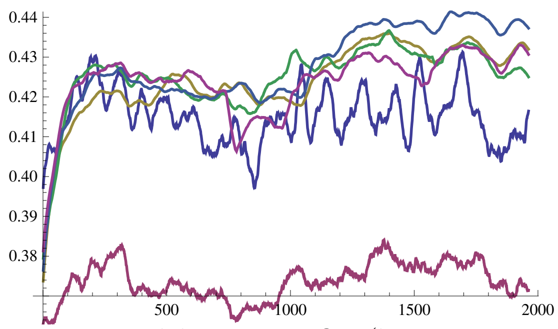
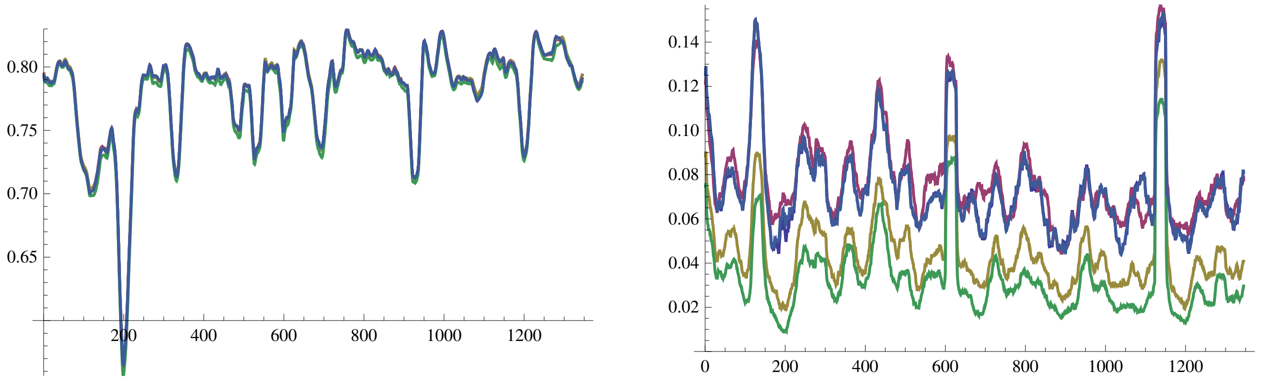


Fig. 21: *Modularity* on $\mathcal{G}_1$ (bottom to top): **dGlobal@BU**, **dGlobal@BT**, **dGlobal@BN**$_{64}$, **dGlobal@N**$_2$, **dGlobal@N**$_1$ and **dGlobal@BN**$_{16}$. Distance plots similarly (but inverted), and runtimes range between the extremes **dGlobal@BT** (fastest) and **dGlobal@BU**.



(a) *Modularity* on $\mathcal{G}_e$, $b = 500$ emails. (b) $\mathcal{R}_g$ on $\mathcal{G}_e$, $b = 500$ emails.

Fig. 22: Practically equivalent in terms of quality are (bottom to top for $\mathcal{R}_g$) **tdLocal@**0.2, **tdLocal@**0.1, **sGlobal**, and **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:

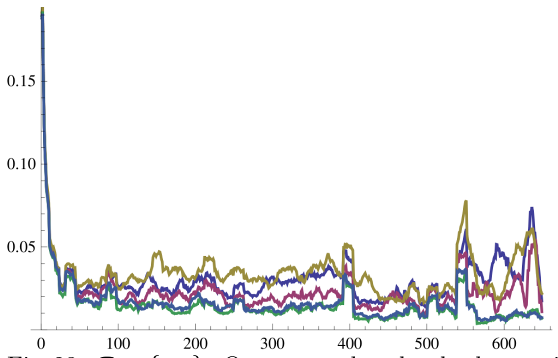


Fig. 23: $\mathcal{R}_g$, {arx}: On a very low level, the statics (**sLocal** and **sGlobal**) and **dGlobal@BT** are slightly less smooth than **dLocal@BN**$_4$ and **dGlobal@BN**$_{16}$. The overall level indicates the graph's stability.

Various dynamics, *batch size* 1:

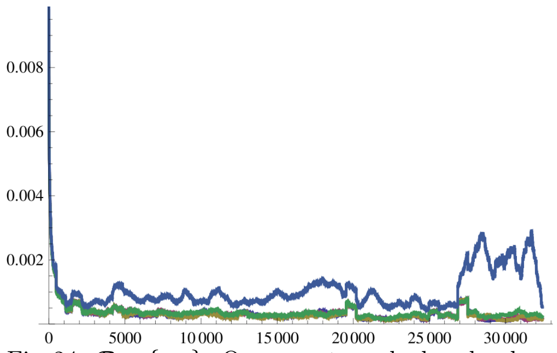


Fig. 24: $\mathcal{R}_g$, {arx}: On an extremely low level, only **dGlobal@BT** reacts to the harsh graph changes (see Fig. 27); **dLocal@N**$_1$, **dLocal@BN**$_4$, **dGlobal@BN**$_{16}$ and **dGlobal@N**$_1$ hardly spot out.

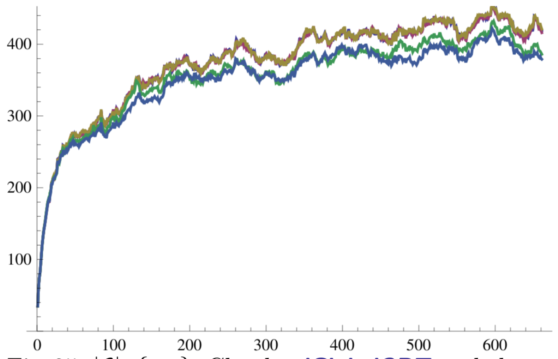


Fig. 25: $|\mathcal{C}|$, {arx}: Clearly, **dGlobal@BT** and the statics (**sLocal** and **sGlobal**) rebalance and reorganize the clustering, while **dLocal@BN**$_4$ and **dGlobal@BN**$_{16}$ more often enlarge existing clusters.

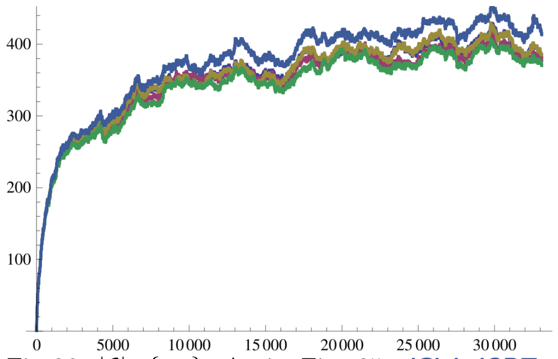


Fig. 26: $|\mathcal{C}|$, {arx}: As in Fig. 25, **dGlobal@BT** rebalances for higher quality; the locals (**dLocal@N**$_1$, **dLocal@BN**$_4$) do slightly better than the globals (**dGlobal@BN**$_{16}$, **dGlobal@N**$_1$).

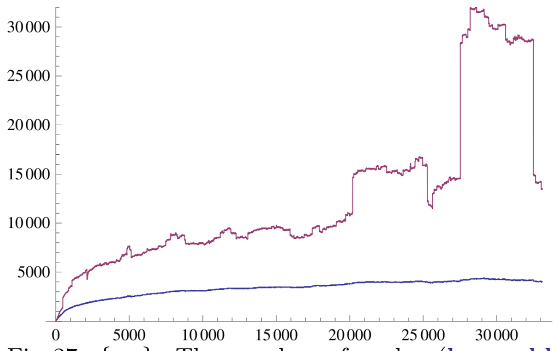


Fig. 27: {arx}: The number of nodes (**lower blue plot**) and especially the number of edges (**upper red plot**) for *Nuclear Theory* are rather volatile. Clique-wise growth proves local-unfriendly.

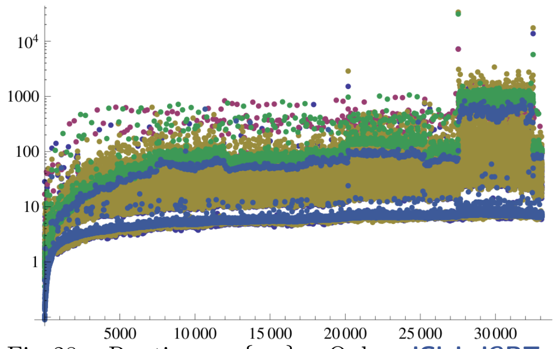


Fig. 28: Runtimes, {arx}: Only **dGlobal@BT** is largely unaffected by graph growth. Prep strategy N (**dLocal@N**$_1$, **dGlobal@N**$_1$, is slower than BN (**dLocal@BN**$_4$, **dGlobal@BN**$_{16}$).

C Trials with arXiv Data: Category *Computer Science*

Category CS with all subcategories consists of 14K e-prints, 25K authors. We use *batch size* 10 for dynamics, compared to statics using *batch size* 100 in Fig. 34. dGlobal@BT excels.

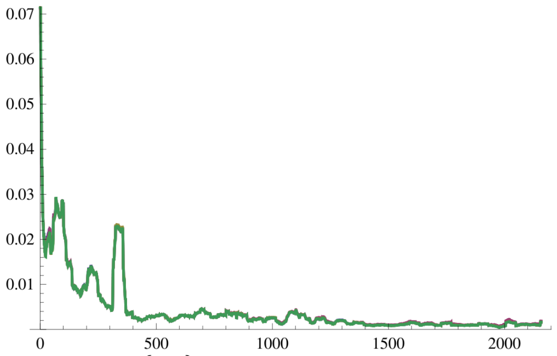


Fig. 29: $\mathcal{R}_g$, {arx}: Concerning smoothness, all tested dynamic algorithms (**dGlobal@BT**, **dLocal@BN**$_4$, **dLocal@N**$_1$, **dGlobal@BN**$_{16}$) behave identically.

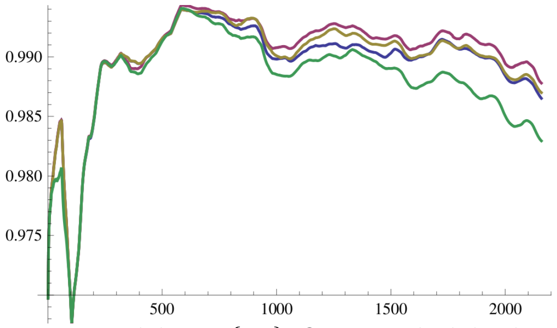


Fig. 30: *Modularity*, {arx}: On a very high level, only **dGlobal@BT** starts to slightly stand out after a while, and **dGlobal@BN**$_{16}$ falls behind; **dLocal@BN**$_4$ and **dLocal@N**$_1$ hardly differ.

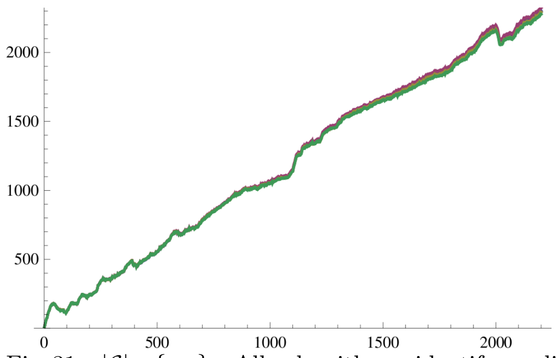


Fig. 31: $|\mathcal{C}|$, {arx}: All algorithms identify a linear growth in $|\mathcal{C}|$ over time; with **dGlobal@BT** slightly beyond the others (**dLocal@BN**$_4$, **dLocal@N**$_1$, **dGlobal@BN**$_{16}$).

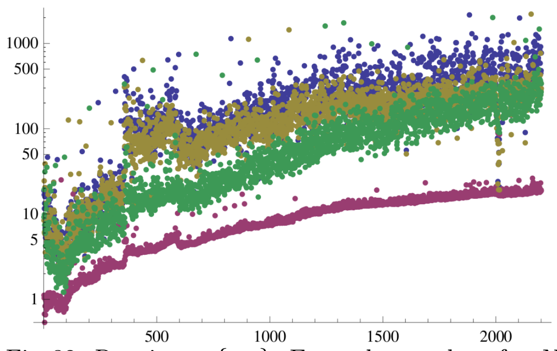


Fig. 32: Runtimes, {arx}: Even clearer than for *Nuclear Theory*, **dGlobal@BT** scales well, while the other algorithms slow down more strongly (**dLocal@BN**$_4$, **dLocal@N**$_1$, **dGlobal@BN**$_{16}$).

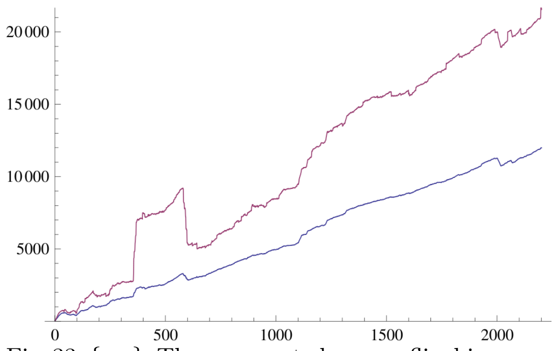


Fig. 33: {arx}: There seems to be an unflinching growth in popularity of *arXiv's* categories of computer science. Both the number of nodes (**lower blue plot**) and the number of edges (**upper red plot**) grow sharply and steadily.

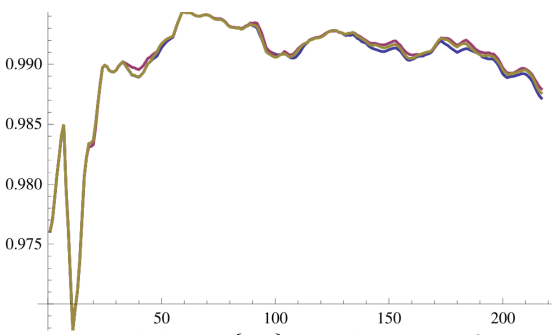


Fig. 34: *Modularity*, {arx}: Batch size 100 for statics vs. **dGlobal@BT**. By margins of 0.01% **dGlobal@BT** lies between **sLocal** and **sGlobal**. **dGlobal@BT** has virtually the same $\mathcal{R}_g$ and $|\mathcal{C}|$ as the statics but is faster by factors beyond 10^4.