# Fast Community Detection For Dynamic Complex Networks

Shweta Bansal, Sanjukta Bhowmick, Prashant Paymal

**Abstract** Dynamic complex networks are used to model the evolving relationships between entities in widely varying fields of research such as epidemiology, ecology, sociology, and economics. In the study of complex networks, a network is said to have community structure if it divides naturally into groups of vertices with dense connections within groups and sparser connections between groups. Detecting the evolution of communities within dynamically changing networks is crucial to understanding complex systems. In this paper, we develop a fast community detection algorithm for real-time dynamic network data. Our method takes advantage of community information from previous time steps and thereby improves efficiency while maintaining the quality of community detection. Our experiments on citation-based networks show that the execution time improves as much as 30% (average 13%) over static methods.

## 1 Introduction

Over the last decade, complex network models have advanced our understanding of systems at all scales, from protein interaction networks to global food webs and from physical transportation networks to online social networks [1, 2].

Researchers often build network models from empirical data and then seek to characterize and explain non-trivial structural properties such as heavy-tail degree distributions, clustering, short average path lengths, degree correlations and community structure [3, 4, 5, 6, 7]. These structural properties appear in diverse natural and man-made systems, and can fundamentally influence dynamical processes of and on these networks [7, 8].

Center for Infectious Disease Dynamics, Penn State University, University Park PA 16802, USA · e-mail: shweta@sbansal.com Department of Computer Science, University of Nebraska at Omaha, Omaha NB, USA e-mail: sbhowmick@mail.unomaha.edu,ppaymal@unomaha.edu

Community structure is a network characteristic describing the propensity of groups of vertices to form dense connections within the group than across. This characteristic is used in the analysis of networks for many applications including hierarchies of organizations [9], collaboration networks [10], protein interactions [1], and stability of electrical grids [11]. The problem of community detection involves finding such connected groups in a given network and has become a popular algorithmic problem in recent years. The quality of a particular division of a network into communities can be measured by modularity. One widely used method of community detection is modularity maximization which detects communities via approximate optimization methods such as simulated annealing, spectral clustering, or greedy algorithms, using modularity as the optimization function [12, 13].

Much of the current work in complex networks science is based on static networks, which are graphs where vertices and edges remain fixed permanently. However, the diverse interactions that make up empirical complex networks are often quite fluid: new connections form, while others dissolve, providing opportunities for topological changes that can have a major impact on dynamics over the network. Examples include evolving social contact networks over which infectious diseases can disperse and dynamic computer networks over which users can communicate and share resources. A dynamic representation of complex networks, in which vertices and edges shift according to changes in the system, more reflects this reality.

Community detection on dynamic networks has not received much attention until recently. Though consequent temporal configurations of a dynamic network vary by only a small amount (i.e. one node or edge added or deleted), most community detection methods treat each configuration as a separate network. The information regarding communities from the previous configuration is not used and the communities have to be recomputed as a whole, requiring redundant computations. The efficiency of these algorithms can be greatly improved if the re-computation is limited only to the portions of the network that are affected by the modifications.

We propose a fast community detection algorithm for real-time dynamic networks that takes advantage of community information computed in previous time steps and thereby increases the efficiency of the detected community structure. In Section 2, we review the static community detection algorithm on which we base our dynamic algorithm, as well as discuss other approaches to community detection in dynamic graphs. In the next two sections, we discuss our contribution to the dynamic community detection problem, and demonstrate results of our analysis algorithm on a citation-based empirical network. We finish with our conclusions, presenting the benefits of our algorithm and future directions.

## 2 Background and Previous Work

The idea of communities (or metapopulations in ecology, modules in physics, and cohesive subgroups in sociology) is one that is appealing to many disciplines, and is closely-related to the problem of graph partitioning in graph theory, graph clustering

in computer science and block modeling in sociology. In this section, we focus on work in the network science literature based on the hierarchical clustering approach for community detection. For a complete review of other popular methods, see [13].

## *2.1 Evaluating Community Structure*

To develop a method for community identification, one needs an evaluation criteria to judge the quality of the detected community structure. One such measure was proposed by Newman and Girvan in [14] and is based on the intuitive idea that random networks do not exhibit (strong) community structure. Given an arbitrary partition of a network into $N_c$ communities, it is possible to define a matrix $C$ (of size $N_c^2$) where the elements $C_{ij}$ represent the fraction of total links starting at a node in group $i$ and ending at a node in group $j$. Then, the sum of any row of $C$, $a_i = \sum_j C_{ij}$ corresponds to the fraction of links connected to subgroup $i$. If the network does not exhibit community structure, or if the partitions are allocated without any regard to the underlying structure, the expected value of the fraction of links within groups can be estimated. It is simply the product of the probability that a link begins at a node in $i$, $a_i$, and the probability of links that end at a node in $i$, $a_i$. Thus, the expected number of within-community links is $a_i^2$. The actual fraction of links within each group, however, is $C_{ii}$. So, a comparison of the actual and expected values, summed over all groups of the partition gives us the deviation of the partition from randomness: $Q(C) = \sum_i (C_{ii} - a_i^2)$. $Q$ is known as modularity, and has become a widely used optimization criteria for community identification algorithms.

The search for the optimal (i.e. largest) modularity value is a NP-hard problem, however, because the space of possible partitions grows faster than any power of the network size. Thus, heuristic search strategies must be used to solve this optimization problem. In recent years, it has been pointed out that the modularity measure has a resolution limit in that it may fail to identify modules smaller than a particular scale (depending on the network size and the degree of module interconnectedness) [15, 16]. This phenomenon stems from the fact that modularity is a sum of terms, and thus modularity maximization amounts to searching for the optimal tradeoff between the number of terms (i.e. the number of modules) and the value of each term. Although other quality measures for community structure do exist [17, 18], many have the same weakness as the modularity measure because they are calculated as sums over all modules and thus have the same tradeoff as described above.

## *2.2 Community Detection*

**Static Community Detection:** The hierarchical clustering approach can be dichotomized into divisive and agglomerative strategies, and both require a similarity measure to be defined between vertices or groups of vertices. Agglomerative al-

gorithms [14], begin by initially considering every node in the network to belong to an individual community, and then proceed to combine vertices that are closely related (based on the similarity measure) to form larger communities. Divisive algorithms [12], on the other hand, begin by initially considering all network vertices to belong to a single community and then proceed to remove edges between pairs of vertices that are the least similar. This subdivides the graph into smaller but tighter communities. Hierarchical clustering is a popular community detection method because in addition to providing the identification of communities in the network, it also provides a hierarchical structure in the communities. In, [14], Newman and Girvan, propose a greedy agglomerative approach based on maximization of modularity for hierarchical community identification. In [19], Clauset, Newman and Moore, propose an algorithm (to be referred to as the CNM algorithm from here on forth) which operates on the same principle but is more efficient due to their use of data structures. The algorithm starts from every node in its own community (like all agglomerative approaches), followed by a computation for each pair of communities of the expected increase in modularity if the pair of communities was to be merged. For efficiency, this computation is only made for pairs of communities that are connected, since joining two communities with no connections between them can never result in an increase in $Q$. The algorithm has running time $O(mdlogn)$ for a network with $n$ vertices, $m$ edges, and a depth, $d$ of the hierarchical community structure, and is thus known to perform efficiently on vertices up to 500,000 vertices [20].

**Dynamic Community Detection:** The problem of community identification for dynamic network data has received less attention in the field. During the course of the past few years, there have been a few proposed methods, which we review here. These methods fall within two classes: one designed for data which is evolving in real time known as incremental or online community detection; and the other for data where all the changes of the network evolution are known a priori, known as offline community detection.

Tantipathananandh et al [21] propose an offline clustering framework based on (the NP-hard problem of) finding optimal graph colorings. They present heuristic algorithms which find near optimal solutions and are demonstrated on small networks with little evolution. However, these algorithms are likely not scalable in their current form. Ning et al [22] propose an incremental (online) algorithm which is initialized by a standard spectral clustering algorithm, followed by updates of the spectra as the dataset evolves. Compared with re-computation by standard spectral clustering for web blog data, their algorithm achieves similar accuracy but smaller computational costs. More recently, Leung et al [23] discuss the potential of the label propagation algorithm (originally proposed in [24]) for dynamic network data (without any experiments). The label propagation algorithm initializes each node with a unique label, and progresses by allowing each node to adopt the label most popular among its neighbors. This iterative process produces densely connected groups of vertices from the current consensus. In this iterative process densely connected groups of vertices form a consensus on a unique label. The static version of the label propagation algorithm is efficient (near-linear time [24]) and the method could

likely be applied to dynamic data. Lastly, earlier in 2010, Mucha et al [25] generalized the Laplacian dynamics approach to obtain a version of the modularity measure for multi-slice (i.e. dynamic) networks. This new measure can then be coupled with existing heuristic methods for dynamic community detection.

## 3 Our Contribution: Dynamic Community Detection

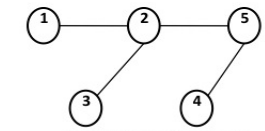
We introduce a dynamic community detection algorithm for real-time online changes, which involve the addition or deletion of edges in the network. Our algorithm, based on the greedy agglomerative technique of the CNM algorithm, follows a hierarchical clustering approach, where two communities are merged at each step to optimize the increase in modularity of the network. As mentioned in Section 2.1, modularity maximization does have limitations; however, we choose to base our method on the CNM algorithm and the modularity measure because they are well-studied in the field, making comparisons possible.

We observe that if the total number of edges is sufficiently large, then a small change in the number of edges would not significantly affect the fraction of edges in the graph, i.e. the values of $C_{ij}$. Therefore, until a vertex associated with the modified edge is merged, all earlier merging steps should proceed exactly as in the previous time step. Based on these observations, our dynamic community detection algorithm is designed as follows:
*Given a modified edge* **(a,b)***, replicate the combination steps of the previous time steps until vertex* **a** *or vertex* **b** *is encountered. Then switch back to the original agglomerative algorithm and continue as in the static case.*

The primary advantage of our dynamic algorithm is that we reduce redundant computations for identifying communities that are to be merged (and for the merging operation itself) by replicating the merge operations that are common between two consecutive time steps. Depending on the position of the vertices **a** or **b** in the dendrogram representing the hierarchical clustering, our algorithm can reduce as much as 30% of algorithm time compared to a repetition of the static method for each time step. In order to replicate the merges, we store the dendrogram from the previous time step, and this requires $O(n)$ additional memory space,where $n$ is the number of vertices in the network.

We note that our dynamic community detection method is not limited to modularity for the quality function or the CNM technique for hierarchical clustering. We have designed the method so it is conducive to efficient implementation and can be easily added as module to existing agglomerative community detection methods, with any quality functions (local or global).

**Pseudocode for Dynamic Community Detection Algorithm**
**Input:** Network $G_0$ and list of modified edges over time steps where $t = 1, \ldots, T$.
**Output:** Community structure at time steps $t = 1, \ldots, T$.



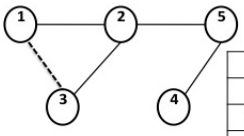

a. Network of 5 vertices

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | W1 | - | - | - |
| 2 | W2 | - | W3 | - | W4 |
| 3 | - | W5 | - | - | - |
| 4 | - | - | - | - | W6 |
| 5 | - | W7 | - | W8 | - |

b. Adjacency matrix for the network

| C_ID | 1 | 2 | 3 | 4 | 5 | | | |
|---|---|---|---|---|---|---|---|---|
| Index | 1 | 2 | 5 | 6 | 7 | 9 | | |
| Columns | 2 | 1 | 3 | 5 | 2 | 5 | 2 | 4 |
| Values | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 |

c. CSR format for the original network




d. Network of 5 vertices
With edge (1,3) added

| C_ID | 1 | 2 | 3 | 4 | 5 | **1** | **3** | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Index | 1 | 2 | 5 | 6 | 7 | 9 | **10** | **11** | | |
| Columns | 2 | 1 | 3 | 5 | 2 | 5 | 2 | 4 | **3** | **1** |
| Values | W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | W9 | W10 |

e. CSR format for the modified network

**Fig. 1** CSR representation of a dynamic network. Figure a: The original network. Figure b. The sparse adjacency matrix corresponding to the network. The values represent the increase in modularity if the row and column are to be merged. Figure c. The CSR format for the sparse matrix. Figure d. The original network with a new edge (1,3) added, Figure e. Modification to the original sparse matrix to add entries for edge (1,3) and (3,1)

1. The community structure of the input network $G_0$ is initialized using the original greedy agglomerative algorithm.
2. Each combination step is stored as a triplet $< i, j, dQ >, t = 0$, where $i$ and $j$ are communities that have merged and $dQ$ is the increase in modularity due to the merge.
3. For iterations over timesteps $t = 1, \ldots, T$
   a. Obtain change in edges. Let $a$ and $b$ be the vertices involved in the edge change.
   b. Update network $G_{t-1}$ to $G_t$ to include the change
   c. Replicate combination steps of $G_{t-1}$ until vertex $a$ or $b$ is encountered
   d. Revert to original agglomerative algorithm.
   e. Continue until increase of modularity, $dQ$ is negative
   f. Delete combination steps for $G_{t-1}$
   g. Store all the combination steps $G_t$
4. End

**Dynamic Updates to Networks:** Updating the network structure for each modification is a computationally intensive operation, and whose efficiency depends on the underlying data structure. Data structures for dynamic networks include adjacency lists, such as those used in [26], which are easy to modify through addition

and deletion of elements to the list. However, adjacency lists can potentially occupy non-contiguous addresses, thereby not rendering efficient memory utilization.

In our implementation, we have used the compressed row storage method [27], a popular format for representing sparse matrices, which stores values associated with the links in the network in an array, i.e. within a contiguous memory location. When an edge is deleted, the corresponding value is set to zero; when an edge is created, a new entry and value is added to the existing array.

CSR is based on expressing the network as sparse matrix, as shown in Figure 1. The array *Index* points to the first non-zero element in each row. The *Columns* array represents the corresponding columns of the matrix with non-zero values and the *Values* array represents the increase in modularity in the communities if the corresponding row and column are joined. To identify the community structure, we add an extra row *C_ID* to the traditional CSR format which corresponds to the community of each row (vertex). As the agglomeration algorithm progresses the entries in *C_ID* and *Values* change to reflect the evolving community structure.

The CSR data structure ensures high cache utilization and is easy to implement. However, due to the addition of edges and no deletion (the deleted edges are represented by zeros), the network tends to become larger as the number of modifications increase. In future implementations, we plan to design representations where changes can be consolidated after a certain number of modifications.

## 4 Empirical Results

In this section, we describe the results of our online community detection method on a publicly available empirical network dataset. Although our algorithm is designed for use on real-time data, there are few publicly available datasets of this kind. Thus, to test our algorithm, we use dynamic network data where all temporal snapshots are available a priori, but are processed one step at a time.

The real world network dataset is based on the DBLP database [1] which presents information on computer science publications listed in the DBLP Computer Science Bibliography [28]. The available database provides a snapshot of the bibliography as of April 12, 2006 with article titles, authors, editors, publication dates, venues (journal or conference name) and citation information. From this data, we derive a dynamic co-authorship network spanning the year 2000 to 2001.

The networks have 3252 vertices and from 10997 (for year 2000) to 11159 (for year 2001) edges. Each temporal snapshot network represents authors by vertices and co-authorship by edges. There are 2169 separate changes in the edges ( 1124 additions and 1044 deletions) from 2000 to 2001. The input to our dynamic algorithm consists of network snapshots given one at a time (to mimic the behavior of a real-time data stream). We compare the time the performance of our dynamic algorithm to that of the static algorithm for the same network at every time step.

[1] Data acquired from http://kdl.cs.umass.edu/data/dblp/dblp-info.html and http://www.public.asu.edu/ ltang9/heterogeneous_network.html

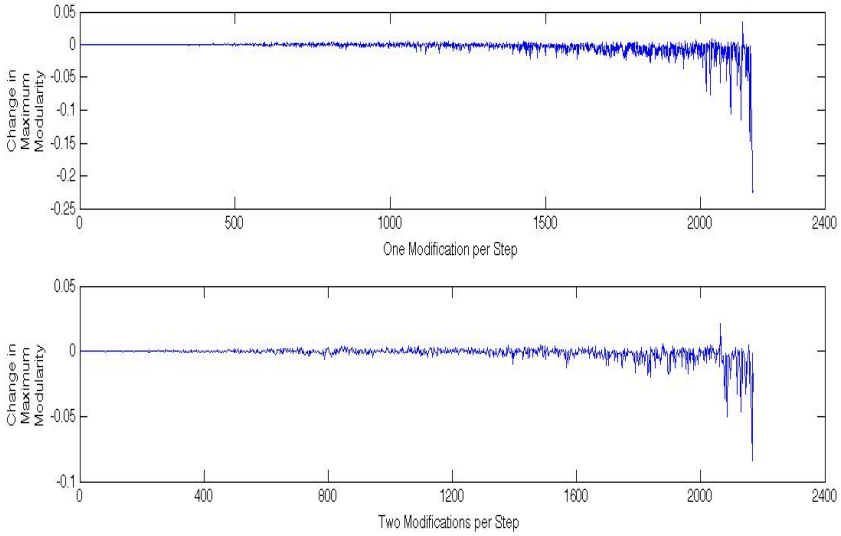



**Fig. 2** Difference in maximum modularity of the static and dynamic method over each network snapshot. The X-axis plots the number of modifications and the Y-axis plots the difference in the modularity. Top: One change per time step. Bottom: Two changes per time step.

We experiment with multiple step sizes on 1 change per time step to 2,4,8 and 10 changes per time step. The results and observations of our experiments are described below.

Our dynamic algorithm assumes that small changes in the network will ensure that the total number of edges, and therefore the values of $C_{ij}$, will remain nearly constant. To fulfill this condition we alternate the online modifications between addition and deletion of edges. However, the total modifications from year 2000 to 2001 in the DBLP dataset, are comprised of 80 more additions than deletions, resulting in a 2% change in the number of edges in the network. As we will discuss below, this change does mildly affect results during the final modifications.

**Quality of Solution:** To validate the correctness of our algorithm, we compare the solutions of our dynamic algorithm with the analagous static method repeated for every time step. As seen in Figures 2 and 3, the maximum modularity obtained by the two methods remains nearly the same until the final modification steps, where they diverge. The variance in modularity is generally within 5%, except in the case of 1 change per step, where the variance increases to almost 25% during the final modifications. In general, the more changes per step size, the more closely the maximum modularity value from the dynamic method adheres to the static case.

The reason for the later discrepancies in the modularity are twofold. First, as discussed earlier, our algorithm is based on the assumption that the total number of edges in the network remain nearly constant. As the number of edges change,

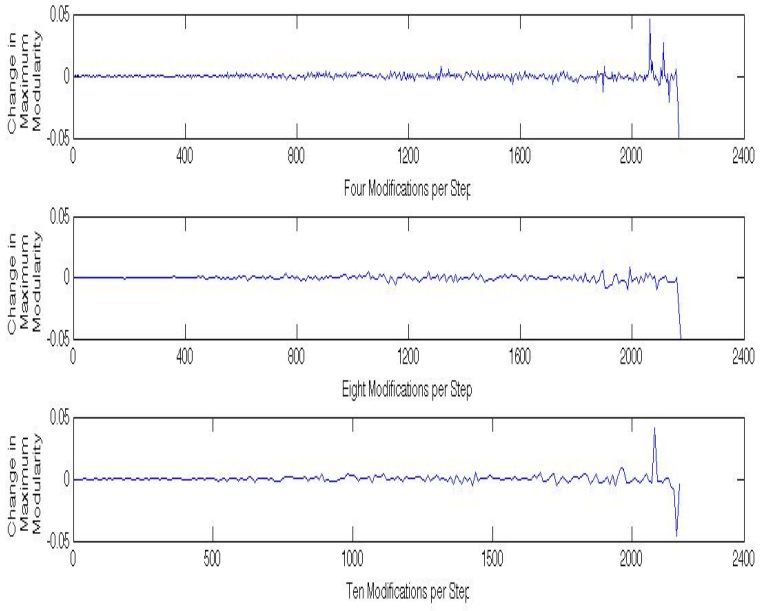



**Fig. 3** Difference in maximum modularity of the static and dynamic method over each network snapshot. The X-axis plots the number of modifications and the Y-axis plots the difference in the modularity. Top: Four changes per time step. Middle: Eight changes per time step. Bottom: Ten changes per time step.

the answers tend to diverge. The second reason for the variation in modularity is as follows. The initial merging operations are based on those of the previous time step. However, as we continue adding changes to the network, the early merging steps are determined by the *dynamic* method from the previous steps. The discrepancies from the static algorithm, if any, tend to accumulate to lead to a wider divergence as the number of changes grow. This effect can be ameliorated by reverting to the complete static algorithm after a certain number of network modifications.

**Performance Results:** Comparing the execution time of community detection for the DBLP dynamic network for the year 2000 to 2001, we find that our dynamic algorithm is either faster or of the same speed as the static case, with efficiency increasing with the number of modifications, which in turn increases the CSR array size. We measure results only for time steps 1 to 2100, since the solutions of the static and dynamic method are equivalent in this range. The percentage of improvement is obtained by computing $\frac{(StaticTime - DynamicTime)}{StaticTime} * 100\%$. As seen from Figures 4 and 5, the speedup can be as much as 30% with an average of 13%. The curves with more changes per time step (Figure 5) are smoother due to fewer calls to the dynamic method, otherwise the progression of the speedup curve is nearly the same over all experiments.

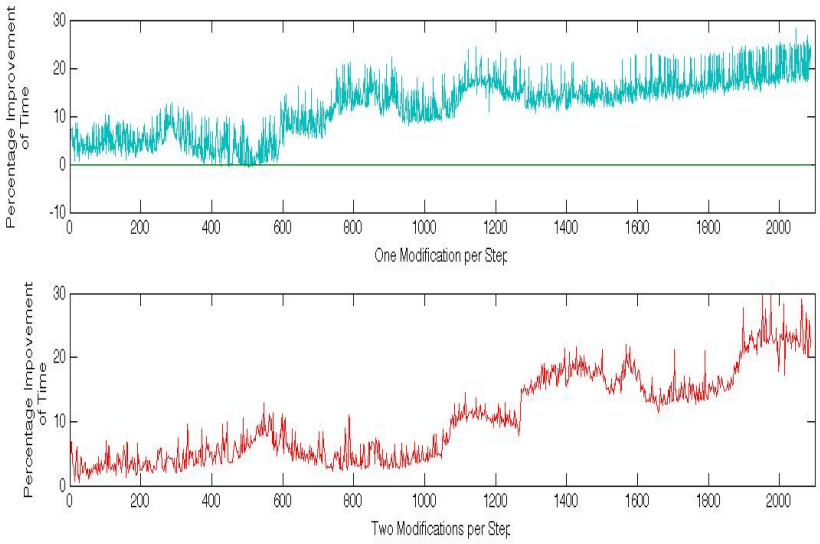



**Fig. 4** Percentage speedup of the dynamic method over the static method at each network snapshot. The X-axis plots the number of modifications and the Y-axis plots the speedup. Top: One change per time step. Bottom: Two changes per time step.

## 5 Discussion and Future Work

We see from the results, that our real-time dynamic algorithm can improve the execution time of a static agglomerative method, while maintaining the quality of solution as measured by the maximum modularity of the network. However, repeated applications of the dynamic method or too many changes of the same type can hamper the quality of results.

Our goal has primarily been to design an efficient algorithm for dynamic community detection by extending a static agglomerative technique and benchmarking our results with the static algorithm results. We do not claim to improve upon the static algorithm results, nor deal with the issues of the underlying algorithm (as discussed in [15, 16]). Finding the ”correct” community structure is also an open problem. However, due to the modularity of our algorithm it is easy to add this dynamic component to any current or future agglomerative method that uses hierarchical clustering.

Our future research plans include designing a dynamic component for divisive community detection methods. We also plan to improve the efficiency of the algorithm by a more selective search of the dendrogram and develop an offline-version of our algorithm which we anticipate would perform at least as well as the online version.

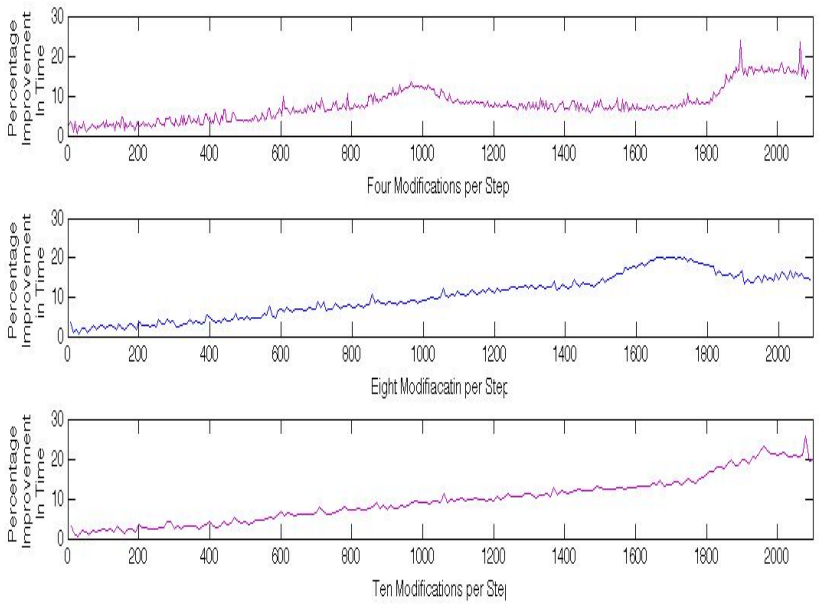



**Fig. 5** Percentage speedup of the dynamic method over the static method at each network snapshot. The X-axis plots the number of modifications and the Y-axis plots the speedup. Top: Four changes per time step. Middle: Eight changes per time step. Bottom: Ten changes per time step.

**Acknowledgements** The authors thank two anonymous reviewers for their insightful feedback. S. Bansal acknowledges support from the RAPIDD program of the Science & Technology Directorate, Department of Homeland Security, and the Fogarty International Center, National Institutes of Health. S. Bhowmick acknowledges the support from the Nebraska EPSCoR First Award and the College of IS&T, University of Nebraska at Omaha.

## References

1. K. Voevodski, S. H. Teng, Y. Xia, Finding local communities in protein networks, BMC Bioinformatics 10 (10) (2009) 297.
2. A. Vazquez, R. Dobrin, D. Sergi, J. P. Eckmann, Z. N. Oltvai, A. L. Barabási, The topological relationship between the large-scale attributes and local interaction patterns of complex networks., PNAS 101 (2004) 17940–17945.
3. D. Watts, S. Strogatz, Collective dynamics of small world networks, Nature 393 (6684) (441) (1998) 440–42.
4. R. Albert, H. Jeong, A. Barabasi, Diameter of the world-wide web, Nature 401 (1999) 130–131.

5. M. Newman, J. Park, Why social networks are different from other types of networks, Phys. Rev. E 68 (036122) (2003) 036122.
6. M. Newman, Assortative mixing in networks, Phys. Rev. Lett 89 (2002) 208701.
7. M. Boguna, R. Pastor-Satorras, Vespignani, Statistical Mechanics of Complex Networks, Vol. 625 of Lecture Notes in Physics, 2003, Ch. Epidemic spreading in complex networks with degree correlations, pp. 127–147.
8. R. Albert, A. L. Barabasi, Statistical mechanics of complex networks, Reviews of Modern Physics 74 (2002) 47–97.
9. M. Porter, N. Mucha, P. J., M. E. J., A. J. Friend, Community structure in the united states house of representatives, Physica A. 386 (2007) 414–438.
10. A. L. Barabasi, H. Jeong, E. Ravasz, Z. Neda, A. Schuberts, T. Vicsek, Evolution of the social network of scientific collaborations, Physica A. 311 (2002) 590–614.
11. K. Atkins, J. Chen, V. S. Anil Kumar, A. Marathe, Structure of electrical networks: A graph theory based analysis., International Journal of Critical Infrastructures 5 (2009) 265–284.
12. M. Girvan, M. Newman, Community structure in social and biological networks., PNAS 99 (2002) 7821–7826.
13. M. Newman, Detecting community structure in networks, Eur. Phys. J. B 38 (2004) 321–330.
14. M. E. J. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
15. S. Fortunato, M. Barthlemy, Resolution limit in community detection, PNAS 104 (1) (2007) 36–41.
16. B. H. Good, Y. de Montjoye, A. Clauset, The performance of modularity maximization in practical contexts, Phys 82 (2010) 046106.
17. K. Steinhaeuser, N. V. Chawla, Identifying and evaluating community structure in complex networks, Pattern Recognition Letters 31(5) (2010) 413–21.
18. M. Gaertler, Network Anal, Springer-Verlag, 2005, Ch. Clustering, pp. 178–215.
19. A. Clauset, M. E. J. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004) 066111.
20. K. Wakita, T. Tsurumi, Finding community structure in mega-scale social networks, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 1275–76.
21. C. Tantipathananandh, T. Berger-Wolf, D. Kempe, A framework for community identification in dynamic social networks, in: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining, 2007, pp. 717–726.
22. H. Ning, W. Xu, Y. Chi, Y. Gong, T. Huang, Incremental spectral clustering with application to monitoring of evolving blog communities, in: SIAM Int. Conf. on Data Mining, 2007, pp. 261–72.
23. I. X. Y. Leung, P. Hui, P. Liò, J. Crowcroft, Towards real-time community detection in large networks, Phys. Rev. E 79 (2009) 066107.
24. U. N. Raghavan, R. Albert, S. Kumara, Near linear time algorithm to detect community structures in large-scale networks, Phys. Rev. E 76 (2007) 036106.
25. P. J. Mucha, T. Richardson, K. Macon, M. A. Porter, J.-P. Onnela, Community structure in time-dependent, multiscale, and multiplex networks, Science 328 (2010) 876–878.
26. D. A. Bader, A. Amos-Binks, C. Chavarrsa-Miranda, D. andHastings, K. Madduri, P. S. C., STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation, Tech. rep., Georgia Institute of Technology (2009).
27. Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing Company,, 1995.
28. The DBLP Computer Science Bibliography, http://dblpVis.uni-trier.de.