An Incremental Batch Technique for Community Detection

Wen Haw Chong, Loo Nin Teow
DSO National Laboratories
20 Science Park Drive Singapore 118230
{cwenhaw, tloonin}@dso.org.sg

Abstract—In the analysis of real world networks, it is often of interest to partition nodes into groups referred to as communities, whereby each community is densely connected and different communities are sparsely connected to one another. While community detection on static networks has been extensively researched on, updating community structures efficiently and accurately on evolving networks is a relatively new research area. In this paper, we discuss the inadequacies of previous techniques as well as justify the need for a new class of techniques that can handle complex batch changes in networks. We then propose one such incremental technique. Compared to earlier work, the proposed technique is much more efficient in scenarios where a network evolves significantly while maintaining a high level of accuracy. Experiments on both artificial and real world networks validate the utility of the proposed technique.

Keywords—Community Detection, Incremental Update, Modularity, Complex Networks

I. INTRODUCTION

In graph mining and social network analysis applications, community detection forms an important research area. The problem is also known as clustering in graphs whereby one aims to derive groups of densely connected nodes. Typically communities can be described with the following: within each community, the connection density is high between nodes whereas between communities, connections are sparse. Communities have practical value in real world networks since they often correspond to functional groups. For example, they can consist of cliques of friends or members of an organization in social networks, web pages of similar topics in the world wide web, researchers working on similar fields in co-authorship networks, computing nodes in the same sub-net etc. Hierarchical community structures may also exist in the sense that larger communities encompass smaller sub-communities and so on. This is manifested in real world examples such as departments or sub-units within larger organizations.

Given the practical importance of detecting communities, numerous criterions to quantify community structure, along with optimization techniques have been proposed. Earlier criterions such as ratio cut [1], ratio association [2] are well known. More recent criterions include modularity [3], [4] modularity density [5] and surprise [6]. The modularity criterion was proposed by Newman and has emerged in recent years as one of the most powerful and widely used optimization criterion in the literature, although research is continuously ongoing. The criterion has demonstrated excellent results and can be optimized via a variety of techniques, ranging from local greedy optimization [9] to spectral bisection or clustering [3], [5] etc. Due to its widespread prevalence and impact, this paper shall focus on modularity optimization.

Traditionally, community detection consists of obtaining a static network snapshot and applying a one-time computation of the chosen technique. If the network undergoes any update to its structure such as edge/node deletions or additions, one then simply re-applies the technique. Given that many real-world networks are massive, easily running up to millions of nodes and time-evolving in nature, this is fast becoming an unsatisfactory approach in terms of computation time. While a one-time expensive computation may be tolerable in retrospective studies, it is impractical to incur the same cost at frequent periodic intervals, especially when timely information about communities is required. It is hence essential to develop incremental community detection techniques that can update detected communities in a dynamic manner as the network evolves. Such techniques have the potential to reduce result generation time by avoiding wasteful re-computation. The savings are especially significant if the general community structures evolve in a measured pace over time, such that one can use previously computed results as a bootstrap or prior information to help compute the current community assignments. It is also important that in the process, accuracy is not compromised or any loss is minimized.

In the next section, we first review a popular static community detection technique, followed by existing incremental techniques for detecting communities across small network changes, i.e. fine grained techniques. This is followed by the discussion of various use cases/factors that justify the need for a different class of techniques which can update communities across large network changes. We then show that for the existing incremental fine-grained techniques, there are significant inadequacies that result in suboptimal modularity values as the network evolves. In the process, we bring forth several evolution properties that can be tied to the formation and disappearance of sub-communities. Based on the discussed use cases and newly derived properties, we then propose a new incremental technique that is more readily applicable if a network undergoes large changes or when very fine-grained community snapshots over time are not required. Section V then presents experimental results on both artificial and real world networks. We conclude in section VI.
II. BACKGROUND

A. Modularity

Given a network with \( n \) nodes and \( m \) edges (or links), let \( A \) be the corresponding adjacency matrix, whose element \( A_{ij} \) is the edge weight between nodes \( i \) and \( j \). This is either 1 or 0 for unweighted networks and may be of any other values for weighted networks. Let \( k_i \) be the degree of node \( i \). We further denote the community containing node \( i \) as its parent community \( C(i) \). The expected number of edges between nodes \( i \) and \( j \) is \( k_i k_j / 2m \) if edges are placed at random. Modularity, denoted as \( Q \), is then quantified by the difference between the actual and expected number of edges:

\[
Q = \frac{1}{2m} \sum_{ij} [A_{ij} - k_i k_j / 2m] \delta(C(i), C(j))
\]

(1)

where \( \delta(C(i), C(j)) \) is 1 if \( C(i) \) equals \( C(j) \) and is 0 otherwise. Alternatively, it can be written as

\[
Q = \sum_{i} \left[ \frac{l_i}{m} - \left( \frac{d_i}{2m} \right)^2 \right]
\]

(2)

where \( Y \) is the number of communities, \( l_i \) is the intra edge count and \( d_i \) sums the total degrees of all nodes in community \( i \). We shall be using (2) in subsequent proofs. The criterion assumes a null model where the expected number of edges between node pairs is proportional to the product of their degrees, and correspondingly there is an expected (by chance) number of intra-community edges for each community groupings. Given a specific set of groupings, if the intra-community edges are significantly higher than the expected number, then the groupings are “surprising” with respect to the null model and there is strong evidence of community structures. The modularity optimization problem is hence one of finding node groupings so as to maximize statistical surprise represented by the modularity value. High modularity values indicate stronger community structures.

B. Techniques

While a number of techniques had been proposed to optimize modularity [3], [7], [8], they have largely been superseded by the Louvain technique [9], which had emerged as one of the fastest and most accurate technique. Due to its computation efficiency, it is widely used and easily applicable on huge networks of millions of nodes. The technique is based on greedy local optimization of modularity and comprises 2 phases: node movement phase and network construction phase. The algorithm starts with each node defined as a community. The steps are as follows:

1. **Node movement**: For each node \( i \), move it out of its community to its neighbor’s community where modularity gain is positive and maximum. If no positive gain is possible, node \( i \) remains in its community.

2. Repeat step 1 over all nodes multiple times until modularity has converged to its maximum value.

3. **Network construction**: Construct a new network whose nodes now represent the communities. The edges between the new nodes are calculated as the sum of the edge weights between nodes in the corresponding communities.

Repeat the whole process until there are no possible node movements (in step 1) which will increase modularity. At this stage, the algorithm terminates.

Note that in this algorithm, there is no merging of communities, only inter-community movements of nodes. Communities which end up with no nodes are then dropped from subsequent processing. On running the algorithm to completion, the end result is a hierarchy of communities. Since many real world communities are hierarchical in structure, the results are more informative than returning a flat level of communities.

Due to the local, micro-nature of its optimization scheme, the Louvain technique is highly amenable to handle incremental changes in networks. Essentially, given a new update to the network, e.g. node/edge addition, node movements can be localized and community memberships of adjacent/nearby nodes can be reassigned by using information about the nature of the network changes. These has been exploited by [10, 11] in their adaptation of the Louvain technique. In addition, we highlight that the technique itself can be easily initialized with different node groupings. In the typical application of community detection on a static network, the algorithm is initialized with singleton communities, i.e. each node as its own community. While this provides a great degree of freedom during optimization, singleton initialization is by no means optimal in terms of convergence speed and achieving high modularity values. In particular, if prior information such as previous community assignments is available, this can be used to bootstrap the initial assignment. If there had been limited network changes such that the general community structures remain highly similar, the technique can converge much faster when compared to singleton initialization. The ease of initialization with prior information further supports the adaption of the Louvain technique for incremental community detection.

C. Assignment-based Incremental Techniques

Prior to describing the incremental techniques, we formalize the changes that a network can undergo. These are

- **Edge addition**: Addition of an edge between node pairs where one or both of the connected nodes may be new or existing nodes
- **Node addition**: Addition of a node to the network. The new node may connect to multiple nodes, which may be a mixture of existing or newly added nodes. The new node can also be an isolated node with no connections.
- **Edge deletion**: Deletion of an edge between two existing nodes
• Node deletion: Deletion of an existing node and all of its edges to neighboring nodes.

As is evident, node and edge additions are intimately related since they often occur jointly. The same applies for node and edge deletions. New edges can also be characterized based on whether they connect node pairs in similar communities or across different communities given some existing community assignments. We termed the as former as intra edge and the latter as inter edge. With characterization of the network changes, assignment schemes have been designed for quickly updating the communities.

We briefly review two incremental community updating techniques, which form the basis for our subsequent discussion. Both techniques consider each network change individually and use prior community information together with a mixture of pre-formulated assignment strategies and adaptive search to quickly update the communities. The Nguyen [11] technique handles both addition and deletion of nodes and edges while the Shang [10] technique handles only additions. Both techniques have similar procedures in certain simple scenarios. For example when an edge is added between node pairs in the same community no community updates are done since both techniques assumed that existing community structures can only be reinforced by such changes. For more complex updates, the techniques differ. For addition of a new node with some edge(s) to existing communities, the Shang technique first considers only one edge of the new node and assigns the node to the community which the edge connects to. It then iterates through the node’s other edges in some order and makes community assignments depending on whether each edge is an intra or inter community edge. For the Nguyen technique, all edges of the new node are considered at the same time such that the node is assigned to the community which will result in the largest increase in modularity. For addition of an edge between different communities, the Shang technique considers whether to merge the two affected communities or to leave them unchanged. Taking a different approach, the Nguyen technique conducts membership tests on nodes that are adjacent to the new edge, as well as nodes in the next hop. Tested nodes can switch communities or remain in their old communities.

Table 1 summarizes both techniques in terms of how they handle various network changes.

<table>
<thead>
<tr>
<th>Changes</th>
<th>Shang Technique</th>
<th>Nguyen Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>New node addition with edge(s)</td>
<td>New node joins community linked by one of its edges. Subsequent edges processed as intra or inter edge</td>
<td>New node joins ‘best’ adjacent community to maximize modularity</td>
</tr>
<tr>
<td>Intra edge addition</td>
<td>No change</td>
<td>No change</td>
</tr>
<tr>
<td>Inter edge addition</td>
<td>No change /merge communities</td>
<td>No change /reassign nodes within one hop of the added edge</td>
</tr>
<tr>
<td>Intra edge deletion</td>
<td>Not handled</td>
<td>No change/ divide affected community into maximal quasi-cliques and singletons, which then search for best communities to join</td>
</tr>
<tr>
<td>Inter edge deletion</td>
<td>Not handled</td>
<td>No change</td>
</tr>
<tr>
<td>Node deletion (along with its edges)</td>
<td>Not handled</td>
<td>No change/ divide affected community into singletons and nodes found by 3-clique percolation, which then search for best communities to join</td>
</tr>
</tbody>
</table>

There are several possible scenarios or motivating factors as follows:

• **Unavailability of fine grained time data:** In some networks, there may not be very fine-grained data detailing when each edge/node appears or disappears. One straightforward example is that of a co-authorship network whereby edges are placed between co-authors of a conference or journal publication. Since conferences may be held annually etc, while journals may be published monthly, annually or in other time intervals, the evolution for co-authorship networks are coarse-grained in nature with edges and nodes appearing and disappearing in batches. Within each batch, there is no clear ordering information. The same issue applies for any other networks where detailed timing information may not have been captured and only periodic snap-shots are available.

• **Information overload:** When a technique generates very fine-grained results, this will in itself generate a huge result set, especially if the network is undergoing significant changes or being tracked over a prolong period of time. For community tracking, information overload can occur in the sense that the analyst will face challenges in interpreting or summarizing the overly fine-grained community snapshots. If the
application does not require the reporting of communities after addition/deletion of every new edge or node, then one will be apt to consider incremental batch techniques.

- **Evolution timescale:** This is both linked to application requirements as well as the semantics underlying the networks being studied. One should consider if the selected time resolution makes sense for the network that is being studied. For social networks where edges are indicative of human relationships, it may be sensible, or suffice to track communities and relationships on larger timescales such as daily, weekly or monthly rather than in terms of seconds and minutes. Another example is that in tracking voting network across different elections, one may want to relate edge formation to the election period as a whole, rather than down to the time where each vote is casted.

- **Computation Efficiency:** Incremental fine-grained techniques are optimized to generate results for micro-changes. In scenarios where the rate of change is high or where one is generating results given a network far back in time and a long sequence of changes, such techniques can turn out to be computationally more expensive than static or incremental batch techniques. This can easily be shown. Let the fine-grained technique incur a cost of $\theta_f$ for processing each network update. Also let the static technique incur a cost of $\theta_s$ ($\theta_s > \theta_f$) for result generation on each network snapshot. Given a network at time $t$ and a sequence of $h$ changes leading to the network at time $t+\Theta$, then the fine-grained technique incurs a cost of $h\theta_f$. Therefore, if $h$ is large enough (either due to a high rate of change or a large value for $\Theta$), there exists a point where $h\theta_f > \theta_s$. We are then better off applying the static technique if we do not require all the fine-grained community snapshots starting from time $t$.

Hence, given the discussed justifications, there is a need for incremental batch techniques that can process batch changes more efficiently than fine-grained techniques. Since the smallest possible batch change is simply a micro-change, it is also possible that a well designed batch technique can perform just as efficiently as a fine-grained technique for small changes. Generally both classes of techniques augment each other and are suited to different applications.

### III. THEORETICAL ANALYSIS

In this section, we surface several properties that has not been highlighted or covered in detail previously in the literature.

#### A. Order Dependency

For simplicity, consider a network of two identical communities that are connected by some edges, and whereby multiple inter and intra-community edges are then added. We prove that the assignment scheme of the Shang and Nguyen techniques can easily result in different community structures if it iterates through the added edges in different order.

Let $l$ be the number of intra-community edges in each community and $d$ be the total degree of the nodes in each community. The total edge count is $m$ as per before. Modularity for the two community case can then be written as

$$Q(sep) = 2 \left( \frac{l}{m} - \left( \frac{d}{2m} \right)^2 \right)$$  \hspace{1cm} \text{(3)}$$

If we merge the two communities, the entire network comprises one community and modularity can be trivially shown as $Q(merged)=0$. Hence, for one to prefer the two-community solution rather than their merger, we require the condition that $Q(sep) > Q(merged)$, i.e. $l/m > (d/2m)^2$. At $l/m = (d/2m)^2$, we have the equilibrium where modularity is identical for both community assignments. Consider the addition of $w$ inter-community edges, the modularity for keeping the communities separate is then

$$Q(sep) = 2 \left( \frac{l}{m + w} - \left( \frac{d + w}{2m + 2w} \right)^2 \right) = 2 \left( \frac{l}{m + w} - \frac{1}{4} \right)$$  \hspace{1cm} \text{(4)}$$

since $m=d$ for this example. Solving for the root $w^*$, we derive the number of inter-community edges required for merging as $w^* = 4l - m = 4l - (2l+1) = 2l - 1$ where $l$ is the number of incident edges for each community and equivalent to the number of inter edges in this two-community network. Accordingly we will merge the communities to maximize modularity when the number of added inter-community edges is greater than $w^*$. Following the merger, any additional edges (in the absence of new nodes) will be regarded as intra-community edges. Under the current assignment scheme, communities are not updated when intra-community edges are added. Hence for this example, the assignment scheme will always return the one community solution if there have been addition of $w\geq w^*$ inter edges and they are processed first. The number of intra edges added after that is immaterial.

Next, consider the scenario where intra-community edges are processed first. It is obvious that $w^* \uparrow$ as $l \uparrow$. Hence as the number of intra edges increase in each community, we will require more inter-community edges to be added before the communities can be merged. The equilibrium point has shifted and the new larger value of $w^*$ can again be easily computed.

The dynamic nature of the equilibrium point has implications when we iterate through multiple inter and intra edges using the current assignment scheme. By processing intra-community edges first, we are essentially reinforcing the current community assignments and hence more likely to retain them over subsequent network changes. There is also a lower probability of community merging with the addition of inter edges. On the other hand, processing inter-community edges first will likely lead to more community merging and a final solution with fewer communities. In the absence of implicit ordering of the new edges, one may want to shuffle them in random order to avoid biasness. However, this has
implications on the computational efficiency as will be explained in the next section.

Finally we note that there are various other scenarios concerning node and edge deletions/additions where one can illustrate order dependency of the current assignment-based schemes. We omit further discussion here for brevity.

B. Ordering and Computation Efficiency

We have seen that the assignment scheme can output different results depending on the order of iteration through the network changes. It turns out that the iteration order affects the computation efficiency as well. Consider the addition of inter and intra-community edges. The latter is much more efficient to handle since only membership checks on the two end nodes of the edge are required and no subsequent action needs to be taken. For inter-community edges, there is a need for more extensive checks. Depending on the assignment scheme, communities may either be merged or some switching of node memberships (beyond the immediate affected nodes) will be undertaken. If this occurs, further computation cost is incurred. Therefore, in the interest of efficiency maximization, one will be biased towards having fewer and larger communities such that subsequent edge additions have a larger probability of being treated as intra-community rather than inter-community edges. This can be achieved in practice by simply handling all inter-community edges before proceeding to intra-community edges.

Formally, consider the same network scenario of an initial assignment of two identical communities and the subsequent addition of \(v\) intra-community edges and \(w\) inter-community edges, where \(w=w^*+\Delta, \Delta > 0\). Hence we are assured that if the inter-community edges are processed first, the two communities will be merged. Let the computation cost of conducting checks for intra and inter-community edges be \(p\) and \(q_C\) respectively where \(q_C > p\). For inter edges, also let the cost of community merging be \(q_M\). Consider the case where inter-community edges are first processed. The first \(w^*\) edges incur a cost of \(q_Cw^*\). We can then merge the communities at either the \(w^*\)-th or \(w^*+1\)-th edge. In any case, it does not change our main result. Merging the two communities at the former, there will be \(\Delta\) edges that were inter-community with respect to the initial community assignment, that are now regarded as intra-community edges. Therefore, we process \(\Delta+v\) intra-community edges. The total computation cost is \(\beta_1 = q_Cw^* + q_M + p(\Delta+v)\). If we process all the intra-community edges first before the inter-community edges, two outcomes are possible. The first outcome is that no merging takes place at any point in time and all initial inter-community edges are processed as such. This gives a computation cost of \(\beta_2 = (w^*+\Delta)q_C + pv\). For the second outcome, merging takes place after some number of inter-community edges are processed (i.e. \(w^*+\Delta_1, \Delta_1 < \Delta\)) in which case the remaining edges will simply be processed as intra-community edges. The total computation cost is \(\beta_3 = (w^*+\Delta_1)q_C + q_M + p(\Delta+v+\Delta_1)\). It is easy to see that \(\beta_1 < \beta_3\) and that \(\beta_2 > \beta_1\) if \(\Delta q_C \neq q_M + \Delta p\). Hence we have explicitly shown that ordering has an effect on computation cost.

While it seems that in the current example, we can minimize cost by simply processing all inter-community edges first, this is not a satisfactory approach. As we have explained earlier, doing so will bias the results towards larger and fewer communities. There is an inherent trade-off. One will be prudent to incur some computation cost to obtain unbiased results or to consider efficient batch updating techniques that is less order dependent.

C. Community Spliting due to Intra Edge Addition

We assert the following.

Property 1: For modularity maximization, addition of intra edges to a community can lead to it splitting into multiple communities, especially when edges are being added in close proximity to one another.

We start with the same example network of two identical communities \(C_i\) and \(C_j\) with node sets \(\text{set}(C_i)\) and \(\text{set}(C_j)\) respectively. We then add sufficient inter edges (basically any number > \(w^*\)) between the communities until they are merged to maximize modularity. From this point on, we can add any edges between existing nodes and they will be regarded as intra edges. Intuitively, one will expect the existing community structure to be strengthened and the optimal assignment to remain intact. However let us examine a scenario where we restrict the placement of new edges. Let \(e_{ij}\) be a new edge connecting a pair of nodes in \(\text{set}(C_i)\). For simplicity and without loss of generality, assume equal additions of \(e_{ij}\) and \(e_{ji}\). With \(\delta\) additions to each of \(\text{set}(C_i)\) and \(\text{set}(C_j)\), modularity for reverting to the previous community assignments \(C_i\) and \(C_j\) is then

\[
Q(\text{sep}) = 2\left(\frac{l + \delta}{m + 2\delta + w} - \frac{1}{4}\right)
\]

As the number of intra additions increase, \(\frac{l + \delta}{m + 2\delta + w} \to 0.5\) such that the bracketed term can reach a point where it is positive which leads to \(Q(\text{sep})\) being greater than \(Q(\text{merged})\). Hence to maximize modularity, it is necessary to split the community despite the addition of intra edges!

While the result appears to be surprising, it can be intuitive if we consider it in the context of the formation of sub-communities. Within a larger community, short range edges can form over time as a network evolves. This is especially true when the transitive property applies, e.g. becoming friends with a friend of a friend. Hence our illustrated scenario of restricted edge addition can easily occur in real world networks and is not contrived.

The effects of such edge additions can result in sub-communities forming within a larger community. An alternative interpretation is that communities are hierarchical in nature and the added intra edges only strengthen the larger, coarser communities at the higher level, while at the lower
level, one can derive finer sub-communities such that higher values of modularity are attained. In any case, the current assignment schemes are highly inadequate due to its simplistic assumption that intra edge additions only strengthen existing communities at no change to the maximum attainable modularity. Hence sub-community formation is undetectable.

D. Community Merging due to Intra Edge Deletion

The following property follows.

Property 2: For modularity maximization, removal of intra edges in a community can lead it to merge with one or more adjacent communities.

This is another property which appears surprising, but is in fact the dual version of the intra edge addition scenario and highly evident when one work backwards from the previous example. Given two connected identical communities, we now consider the deletion of δ intra edges from each community where δ ≤ l. We then have

\[
Q_{\text{sep}} = 2 \left( \frac{1 - \delta}{m - 2\delta} - \frac{1}{4} \right) = 2 \left( \frac{l - \delta}{2(l - \delta) + 1} - \frac{1}{4} \right)
\]  \tag{6}

As more intra edges are deleted, the bracketed term decreases and will eventually reach a point where it becomes negative. This implies \( Q_{\text{sep}} < Q_{\text{merged}} = 0 \) and hence we are apt to merge the communities together to achieve a higher modularity value of 0. Essentially, small communities with higher density of connections merge to form a larger community with a lower connection density. If we consider hierarchical communities, we can interpret the scenario as one where sub-communities at the lower level have disappeared. The maximal achievable modularity value is also lowered and can now only be satisfied by coarser community assignments. We also note that for more complicated scenarios, it may be required to conduct node reassignments rather than just pure community merging to achieve optimal modularity values.

IV. PROPOSED INCREMENTAL BATCH TECHNIQUE

We now describe our proposed incremental batch technique which is motivated by the earlier discussed use cases and takes into account the newly surfaced properties. While the technique is designed to handle batch changes, it is equally adept at handling small or micro network changes. It relies on several strategies such as singleton community initialization, exploiting prior information and a thorough node movement phase where all nodes are iterated through.

Firstly, singleton communities are initialized for nodes directly affected by network changes before the node movement phase. Since batch changes are considered, each node may be affected by multiple changes. Nonetheless the affected node will constitute only one singleton community at the start. Such an initialization followed by node movements constitutes a much more general scheme than any assignment based scheme which updates node memberships based on the nature of changes. In particular assignment schemes are very difficult to design when there are numerous and complex changes. For example, a node can simultaneously add edges to both existing and new nodes as well as lose edges to multiple other nodes. The assignment possibilities are endless.

For addition of intra edges, directly affected nodes are also initialized to singleton communities in contrast to the Shang and Nguyen techniques. Depending on the subsequent node movement stage, singleton communities may be absorbed into other communities or take in nodes from other communities to grow in size. In principal, this means that it is possible to detect the formation of sub-communities. Hence we overcome the inadequacies discussed in property 1.

Our initialization scheme also means that the proposed technique is consistent with the static Louvain technique described in section II. For the limiting case whereby the network undergoes huge changes such that all nodes are affected and initialized to singleton communities, both techniques are equivalent. This also means that the computation cost of the proposed technique approaches that of the Louvain technique for very large changes whereas the cost of assignment techniques grows with the number of changes.

Assuming that community memberships have previously been derived by some technique, this can be exploited by our technique. The existing detected communities are used as the starting point for the algorithm together with newly initialized community singletons due to network changes. If the changes in community structure are not drastic, then most nodes are already partitioned correctly and this approach helps to speed up convergence in the algorithm during the node movement stage. Once again, the process is highly similar to the Louvain technique except that we are starting with much fewer singletons and avoiding much re-computation.

Lastly, we highlight that all nodes in the network are iterated through by the technique during the node movement phase regardless of their geodesic distance from the network changes. While it is computationally cheaper to consider only nodes within one or two hops from the changes, such a heuristic will limit the solution space with the potential for poorer modularity values in certain scenarios. In the context of modularity maximization, a network change can have effects that propagate outwards beyond one or two hops such that the community memberships of nodes further away are affected. The scenario of adjacent communities merging described in property 2 is one such example. Our strategy of iterating through all nodes means that in principle, the solution space is not limited and one may still achieve community merging.

The incremental batch technique is summarized as follows:

1. For new nodes and existing nodes adjacent to edge/node addition or deletion, initialize to singleton communities. For all other nodes, retain their previous community memberships. This completes the initialization phase.
2. Repeat steps 1 to 3 as per the Louvain technique until no further increase in modularity is possible.

V. EXPERIMENT RESULTS

In this section, we compare the performance of the proposed incremental batch technique against a modified
incremental fine-grained technique and the Louvain technique. The modified incremental fine-grained technique largely follows Nguyen technique except that for greater consistency with the Louvain technique, we use singleton initialization for intra edge and node deletion, followed by a node movement phase through the affected communities. A superior technique will achieve higher modularity values than others. We also tabulate the computation time each technique takes to initialize and derive communities.

A. Artificial Networks

First we conduct experiments with artificially generated networks whereby the network is specified to undergo a series of changes. We utilize the popular Lancichinetti-Fortunato-Radicchi (LRF) model [12] which is representative of real world networks with power law distributions of degree and community size. For LRF networks, the strength of community structures is adjusted by varying the mixing parameter \( \mu \) which in turn sets the intra and inter connection probabilities. \( \mu \) is simply the ratio of the external degree of a node (with respect to its community) to its total degree. Increasing the value of \( \mu \) decreases the community strength. In our experiments, we generate initial LRF networks of 1000 nodes with \( \mu \) set at 0.5. We set the average degree to 20, maximum degree at 50, exponent of the degree distribution at -2 and exponent of the community size distribution at -1. Each community is specified to have between 10 to 50 nodes. Starting from the initial networks, we then generate a sequence of 5 batch changes. At each batch, we apply a mixture of 100 random network changes, whereby changes can be micro: edge addition or deletion; or macro: addition of a node with accompanying edges or deletion of a node and its edges. For node additions, the degree of the new node is sampled from the existing network degree distribution. Communities are derived from the previous snapshot. The solution space around the current optimal is then explored more extensively, resulting in better modularity values. At the same time, computation for node movements from the much less optimal point of singleton communities are eliminated, hence resulting in lower run times.

We can explain the modularity results by interpreting the node movement phase of the incremental techniques as a search for some better optimal solution around an existing local optimal. Compared to the Louvain technique which has to start with singleton communities, the incremental techniques enjoy the advantage of starting from a much more optimal point with communities derived from the previous snapshot. The solution space around the current optimal is then explored more intensively, resulting in better modularity values. At the same time, computation for node movements from the much less optimal point of singleton communities are eliminated, hence resulting in lower run times.

B. Real World Networks

We further compared the techniques on real world networks. The tested networks are

- **Enron email**: Enron emails [13] are used to construct a network of email users for the year 2001. Nodes represent email users and are linked if the corresponding users have communicated by email at least once. 12 network snapshots are generated. Edges and nodes are added across snapshots as new email communications are observed. The initial network consists of 6281 nodes and 15,787 edges while the final network contains 52,599 nodes and 193,074 edges.

- **Internet Autonomous Systems (AS)**: An AS is effectively a set of routers under a single administration. Routing in a network of ASes is coordinated by the Border Gateway Protocol. There are several types of ASes, e.g. ISPs, end-users; and relationships between ASes, e.g. provider-customer. AS networks can be obtained from [14]. We have generated monthly network snapshots for the year 2003. The initial network consists of 14,806 nodes and 27,012 edges while the final network contains 16,654 nodes and 30,808 edges.

For both the Enron and AS networks, it can be seen that both incremental techniques once again outperforms the Louvain technique in terms of achieving higher modularity values. Differing from the previous experiments on artificial networks, the incremental fine-grained technique now consistently attains the highest modularity values throughout all snapshots. However, this is achieved at extremely high cost.
in computation. For example, for the second snapshot of the Enron network, the incremental fine-grained technique incurs a runtime that is around 50 times that of the Louvain and incremental batch techniques. For the AS networks, the cost of the fine-grained technique ranges from 12 to 15 times higher for each snapshot. To explain this, we note that both networks are undergoing large changes across snapshots. As explained previously in section II, the computation cost of any incremental fine-grained techniques scales with the amount of changes. Hence the technique is handicapped by having to iterate sequentially through each of these changes, e.g. from snapshot 1 to 2, the Enron network undergoes an addition of around 3500 nodes and close to 6000 edges. On the other hand, the run time of the incremental batch technique is much less sensitive to the amount of changes and is able to provide computation savings over the Louvain technique. For the Enron and AS networks, the former requires on average only 48% and 83.5% respectively of the runtime incurred by the latter, while achieving higher modularity values.

Fig. 2. Modularity values and ln(run time) across monthly snapshots of the Enron email network. For clearer comparisons, a logarithmic vertical scale is used for the runtime plot.

For the real world networks here, the incremental fine-grained technique outperforms the batch technique in accuracy. This can be explained by the fact that in iterating and updating communities after each change and due to the large amount of changes, the fine-grained technique is on the whole conducting a much more intensive exploration of the solution space across network snapshots. This is also manifested in the much higher run time. Hence if maximal accuracy is desired and run time is not an issue, the fine-grained technique here does provide an alternative. On the other hand, the proposed batch technique is a more natural choice in applications where resources are limited. It is able to update communities fairly accurately at a low cost.

Finally, while the networks tested here are not overly huge, the incremental technique is easily applicable on networks with millions of nodes, since it simply extends the Louvain technique. The latter had already been demonstrated on such networks.

VI. CONCLUSION

We have discussed various application needs and justification for an incremental technique that is able to process network changes in batches to update communities. We then proposed one such technique for updating communities as a network evolves. In the process, we have also highlighted properties related to the evolution of sub-communities that any reasonable incremental technique should take into account. Results on both artificial and real world networks have shown our proposed technique to be both accurate and efficient.

REFERENCES