Efficient Algorithms for Social Network Coverage and Reach

Deepak Puthal[*†], Surya Nepal[†], Cecile Paris[†], Rajiv Ranjan[†] and Jinjun Chen[*]
[*]Faculty of Engineering and Information Technology
University of Technology, Sydney
Australia

[†]Digital Productivity Flagship
Commonwealth Scientific and Industrial Research Organisation
Australia

E-mail: {firstname.lastname}@csiro.au, jinjun.chen@gmail.com

***Abstract*—Social networks, though started as a software tool enabling people to connect with each other, have emerged in recent times as platforms for businesses, individuals and government agencies to conduct a number of activities ranging from marketing to emergency situation management. As a result, a large number of social network analytics tools have been developed for a variety of applications. A snapshot of social networks at any particular time, called a social graph, represents the connectivity of nodes and potentially the flow of information amongst the nodes (or vertices) in the graph. Understanding the flow of information in a social graph plays an important role in social network applications. Two specific problems related to information flow have implications in many social network applications: (a) finding a minimum set of nodes one has to know to recover the whole graph (also known as the vertex cover problem) and (b) determining the minimum set of nodes one required to reach all nodes in the graph within a specific number of hops (we refer this as the vertex reach problem). Finding an optimal solution to these problems is NP-Hard. In this paper, we propose approximation based approaches and show that our approaches outperform existing approaches using both a theoretical analysis and experimental results.**

Keywords— Approximation Algorithm; Complexity; Network Coverage; Network Reach; Social Networks.

I. INTRODUCTION

Big data is an emerging multi-disciplinary research area with the aim of analysing vast amount of data to extract actionable information. The more precise meaning of big data is given in [24] as: "*Big Data is a term applied to data sets whose size is beyond the ability of commonly used software tools to capture, manage, and process the data within a tolerable elapsed time. Big data size are constantly moving target currently ranging from a few dozen terabytes to many petabytes of data in a single data set*". An IDC report [28] predicts that, from 2005 to 2020, the global data volume will grow by a factor of 300, from 130 exabytes to 40,000 exabytes, representing a double growth in every two years. It is clear that we are entering in a data explosive era, evidenced by the volume of data from various sources and its growing generation rate. Social networks are amongst the major sources of big data. For example, Facebook stored, accessed, and analysed more than 30 petabytes of user-generated data. Over 32 billion searches were performed per month on Twitter [27]. It is thus important to find efficient and effective ways of analysing social network data.

A number of social network analytics tools have been developed and used in different applications [25] [26]. A snapshot of social networks at any particular time is called a social graph. Social graphs derived from social network data traces can be classified into two categories: articulated and behavioural [25]. Articulated graphs are those that result from people specifying their contacts through referencing [30]. There are three common reasons for which people articulate their connections: to have a list of contacts for personal use; to publicly display their connections to others; and to filter content on social media. The motivation for people to add someone to their explicit connections varies widely, but the result is that connections can include friends, colleagues, acquaintances, celebrities, friends-of-friends, public figures, and interesting strangers. In contrast, behavioural graphs are derived from communication patterns, cell coordinates, and social media interactions [31]. These might include people who send text messages to each other; those who are tagged in photos together on Facebook; people who email each other; and people who are physically in the same space, at least according to their cell phone. The approach proposed in this paper is agnostics with respect to how the graphs were obtained.

One of the benefits of online social networks is their ability to provide fast and extensive information dissemination. For example, the news of Michael Jackson's death spread faster in social networks than it did in traditional mass media [22]. Social networks thus provide fast access to large scale news data. Social networks are in fact a major source of news for many people today [30][32]. While the ease with which information flows in social networks can be very beneficial, it can also have disruptive effects, in particular with respect to misinformation – see, for example, the spread misinformation on swine flu on Twitter [23]. The questions of how to spread information widely

and how to limit the flow of misinformation in social networks thus arise. Budak *et al.* (2011) have shown that this *eventual influence limitation* problem is NP-hard for an exact solution in large scale social networks [29]. There are two specific problems related to information flow that have implications in many social network applications: (a) finding a minimum set of nodes, or vertices, (with their accompanying/incident edges) one has to know to recover the whole social graph, a problem also known as the vertex cover problem, and (b) determining a minimum set of nodes required to reach all nodes in the social graph within a specific number of hops, a problem also known as reachability (we refer it as the vertex reach problem). While finding an optimal solution to these problems is NP-Hard, Budak *et al.* (2011) have shown that algorithms based on heuristics (including even a simple heuristic based on the degree of centrality) work better than the traditional greedy algorithm [29]. Motivated by this heuristic based approach and traditional approximations based graph processing methods, this paper aims to address these two problems by proposing the approximation based algorithms.

In order for social networks to serve as reliable platforms for disseminating critical information (and preventing misinformation), it is necessary to be able to identify the minimum number of (information) transmissions required to reach each individual node in the network. In some applications, it is important that all nodes and paths in the graph are considered, and an appropriate forwarding path is selected based the properties of the nodes, links, and paths correlation [33]. However, as data sets grow and applications require real time processing, it becomes difficult to cover the entire graph or reach each individual node in a social graph. One potential way to solve this problem is to provide data to *selected nodes*, and let the data flow to other nodes through the natural communication behaviour. Towards this goal, this paper proposes a near optimal way of finding nodes that can play a role in social network applications to provide an efficient way of disseminating information. In particular, we propose approximation based approaches and compare their performance with standard approximation algorithms. The basic idea behind our approach is to introduce weights in the nodes of a social graph based on their connectivity, to process the nodes in the order of the weights and update these weights based again on connectivity. The results of our experiments show that our algorithms perform better than standard approximation algorithms.

The rest of this paper is organised as follows. Section II provides related work, on the vertex cover and vertex reach problem in particular, as it is a well-known and well-studied problem in computer science. Section III describes our proposed approaches with their algorithms and corresponding theoretical analysis. Section IV evaluates the performance and efficiency of our approaches through experimental results. Section V describes the potential applications of the proposed approaches. The final section concludes our work and points to future work.

II. Related Work

In this paper, we focus on two different problems: graph coverage (aka vertex cover) and graph reach (aka reachability; we refer as vertex reach). We use the terms *vertex* and *node* interchangeably throughout the paper. The vertex cover problem is to identify the set of vertices from which the whole graph can be regenerated by knowing the edges (in and out) of those vertices. The vertex reach problem is to identify the set of vertices from which all other vertices of the graph can be reached within a specified number of hops. Both these problems are NP-Hard problem [1]. To address them, researchers have used approximation algorithms that give a near optimal solution in polynomial time complexity.

To explain the problem further, let's consider the connected undirected graph shown in Figure 1. In order to cover (or regenerate) this graph, the set of vertices required is given by the set {B, D, E}. Vertex B covers vertices B, A, C and edges BC and BA; vertex D covers vertices D, C, E, F, G and edges DC, DE, DF and DG; Finally, vertex E covers vertices C, D, E, F, and more importantly, it covered edge EF (missing thus far), as well as edges EC and ED. In the vertex cover problem, it is important to note that all vertices and edges need to be covered. In a similar way, the set of vertices required to have a solution for the vertex reach problem is given by a set of vertices {A, D} for a single hop. With a single hop, we can reach B through A, and we can reach C, E, F, G through D. Note that this solution is not unique. Another solution would be the set {B, D}. We now describe existing approaches proposed in the literature to address the vertex cover and reach problem.

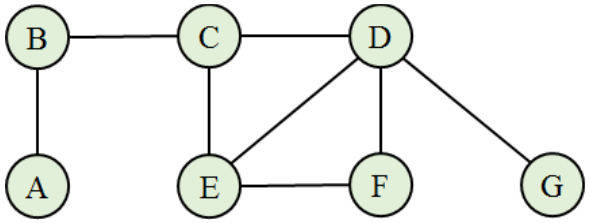

Figure 1: A simple undirected graph used to illustrate the vertex cover and reach problems.

Several approximation approaches have been proposed to solve the vertex cover problem. Bar-Yehuda *et al.* [10] in 1985 proposed an algorithm based on associating weights to the vertices, that is scores based on the number of edges that are incident to the vertex. Their approach works by reducing the weights of vertices in certain subgraph(s). This has the effect of local approximations. Monien *et al.* (1985) [11] ran a vertex cover algorithm for computing an independent set of vertices and showed that its time complexity was bounded to $O(|V| \times |E|)$. The vertex approximation factor of the vertex cover problem for both Bar-Yehuda and Monien approaches is: $2 - \Theta\left(\frac{\log\log n}{\log n}\right)$. Arora *et al.* proposed an algorithm in [12], which gave an $O\left(\sqrt{\log n}\right)$ -approximation algorithm for the Sparsest Cut, Edge Expansion, Balanced Separator, and Graph Conductance problems. Karakostas [8] in 2005 reduced the approximation factor of vertex cover problem to $2 - \Theta\left(\frac{1}{\sqrt{\log n}}\right)$. Further details

of Karakostas' work is published in 2009 [9] with the same complexity, as an improvement of the work done described in [12], which depends on the approximation factor of the sparsest cut and balanced cut problems and uses the existence of two large and well-separated sets of nodes. In 2008, Khot *et al.* [13] showed that the vertex cover is hard to approximate within any constant factor better than 2, i.e., 2− ε. Kuhn and Mastrolilli [15] in 2013 investigated the vertex cover problem for weighted graphs when a locally bounded coloring is given. Finally, in 2015, Shah *et al.* [14] proposed an approximation algorithm to solve the vertex cover problem by using Depth First Search (DFS) algorithm. It takes O (2 (V + E)) time complexity, where O (V + E) for DFS and O (V + E) for finding vertex cover.

The vertex reach problem has been studied under the "reachability" problem, where the focus is to find an optimal path to reach from one vertex to another vertex within a graph. He et al. proposed an innovative approach, HLSS(Hierarchical Labeling of SubStructures), for reachability. Their approach identifies different types of substructures within a graph and encodes them [34]. It works in two phases. The first phase identifies and encodes strongly connected components, and the second phase encodes the remaining reachability relationships. Hwang et al. [35] computed a finite vertex graph for discrete event system specification (DEVS) network. They use a subclass of DEVS, called finite and deterministic DEVS (FD-DEVS) to obtain the finite vertex reachability graph of a DEVS network. Jin et al. [36] introduced a novel tree based index framework which utilizes the directed maximal weighted spanning tree algorithm and sampling techniques to maximally compress the generalized transitive closure for the labeled graphs. They demonstrated their approach by finding edges for several graph data such as social networks, biological networks, and the semantic web. It is important to note the difference between reachability and vertex reach problem. The reachability problem is to find out whether it is possible to reach from one vertex within a graph to another, given a pair of vertex. The vertex reach problem is to identify the set of vertices in the graph to reach all vertices in the graph within a given number of hops. Our focus in this paper is on the vertex reach problem.

III. Proposed Approach

A snapshot of nodes and relationships in social networks is represented as a social graph (referred to as a graph or a social network hereafter depending on the context). We represent the graph as G = (V, E), where V is the number of vertices in the graph and E the number of edges. Vertices and edges represent the graph nodes and links between them. These sets of vertices and edges can be implemented as an adjacency list or as an adjacency matrix for both directed and undirected graphs. In this paper, we consider two types of graphs: (a) sparse graphs, i.e., those for which $|E|$ is much less than $|V|^2$, i.e,. ($E << V^2$), and (b) dense graphs, that is those graphs for which $|E|$ is close to $|V|^2$, i.e., ($E \simeq V^2$).

We next define the vertex cover and vertex reach formally.

Definition of vertex cover: A vertex cover of a graph G = (V, E) is a subset of vertices V` of V such that if edge (u, v) is an edge of the graph G, then either u is in V or v is in V`. In a more generic way, the vertex cover of a graph is a set of vertices such that each edge in the graph is incident to at least one vertex of the set.

Definition of vertex reach: A vertex reach of a graph G = (V, E) is a subset of V such that if edge (u, v) is an edge of G then one of the vertices (u or v) is in V. More generically, the vertex reach of a graph is a set of vertices such that all vertices in the graph are connected to at least one vertex of the set.

Finding the minimum set of nodes to cover or reach a complete social graph (i.e., to cover all the nodes and edges or reach all the nodes in the graph) is a NP-Hard problem, as already mentioned. However, it is possible to find a near optimal solution with reasonable time complexity, namely polynomial time complexity. Our aim in this paper is to propose a method that gives very near optimal solution for both the vertex cover and the vertex reach problems.

There are two major factors to compute the efficiency of an algorithm, i.e., its time complexity and its space complexity. There are three different ways of representing complexity: worst case, best case and average case, where the worst case is always needed to evaluate the effectiveness of an algorithm. In what follows, after we present our algorithms for both the vertex cover and reach problems, we compute their complexity and show that our approach outperforms the traditional vertex cover algorithm, covering the entire network in less than double of time for an optimal solution [1].

In the descriptions and evaluation described here, we have considered simple undirected graphs [2]. We use an adjacency list to represent the graph G, in order to reduce the space complexity. We introduce a new field weight in the list which is initialised at the beginning to store the degree of each individual vertex, i.e., our representation is the list: `{vertex; weight; *next; *ref}`. The connected nodes are represented as linked-list.

A. An algorithm for the Approximate Vertex Cover Problem

Our proposed approximation vertex cover algorithm and its workings are described in Algorithm I, and Figures 2 and 3, respectively. The algorithm takes a graph *G* as input and returns a set of vertices to cover the complete graph. The algorithm works as follows:

As mentioned above, we represent the graph as a *list* instead of a *matrix* to reduce the space complexity. We introduce one more field weight for every node. In Algorithm I, we initialise the weight as L[w], which is calculated using the number of reference nodes from each individual nodes in the *list* and kept as an array. We update the weight by decreasing the value as shown in step 9 of Algorithm I. Let C^+ be the vertex cover set, which is empty at the beginning. The algorithm selects the vertex with the highest weight, adds it C^+ and updates the weights of the remaining vertices as follows: the weight of the selected vertex is set to 0, and the weights of all its one hop neighbours are decreased by 1. The selection and weight update steps are repeated until all nodes have a weight equal to 0. The final set C^+ gives a solution for the vertex cover problem.

Algorithm I: Approximate Vertex Cover Algorithm for Social Network Coverage

Required: In the *List* we introduce another field *weight*
The *weight* is number of nodes in reference of the node (*ref*)

1: $C^+ \leftarrow \phi$
2: L=*List*
3: L[w]= weight
4: (h, v) = highest weight of the list and respective vertex
5: **if** $h \neq 0$ **then**
6: $C^+ \leftarrow C^+ \cup v$
7: v[w] $\leftarrow 0$
8: **for** all vertices of *List* L[ref] $\in \{v\}$ **do**
9: L[w] $\leftarrow$ L[w]-1
10: **end for**
11: **go to** 4
12: **else**
13: **return** C^+
14: **end if**

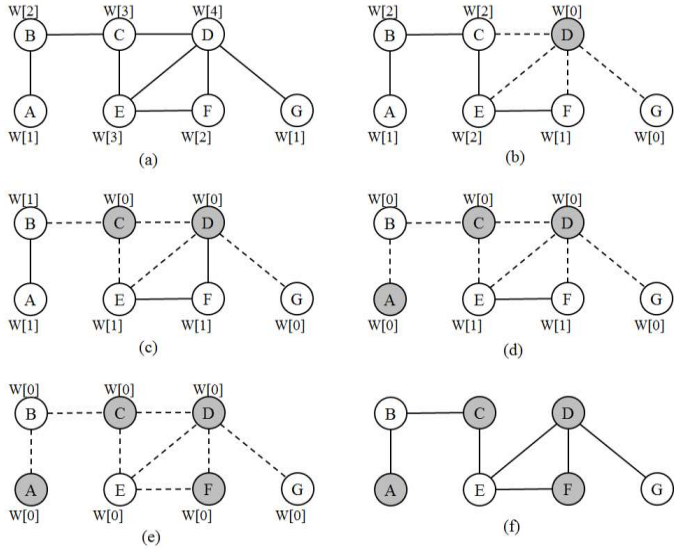

Figure 2: Illustration of the steps of the approximate vertex cover algorithm, as performed on a sample graph. *(a)* The input graph G containing 7 vertices and 8 edges. *(b)* D is the first vertex (highest weight) chosen by algorithm. It is added to C^+ (shown in grey), and its weight is now set to zero. The weights of its one hop neighbours are decreased by 1. *(c)* There are three vertices with the now highest weight (B, C, E); An arbitrary vertex, C, is chosen and added to the set C^+. *(d)* The process is repeated and now vertex A is chosen and added to C^+. *(e)* The process continues; Vertex F is chosen and added to C^+. The process is complete are all weights are now 0. *(f)* The resultant vertex cover set with shaded color.

Figure 2 shows the steps of vertex cover algorithm using our example graph (from Figure 1). Figure 2(a) shows the sample input graph with the weights of the vertices initialised to the vertex' number of edges. The highest weight vertex of the graph is D, so it is selected to be added to the set C^+ as shown in Figure 2(b). Its weight is set to 0 (zero), and its immediate neighbours have their weight value decreased by 1. In the figure, vertex D is shaded and its incident edges are dashed. Figure 2(c) shows there are three vertices with what is now the highest weight, i.e., B, C, E. The algorithm arbitrarily chooses vertex C, adding it to the set C^+. The process iterates: Vertex A is chosen, its weights and the weights of its neighbours updated (Figure 2(d)). Finally, vertex F is chosen and the weights updated (Figure 2(e)). The resultant vertex cover is shown in Figure 2(f). Its contains four vertices *A, C, D,* and *F*. Finally, the comparison of the efficiency of our algorithm with both traditional and optimal solutions is shown in Figure 3.

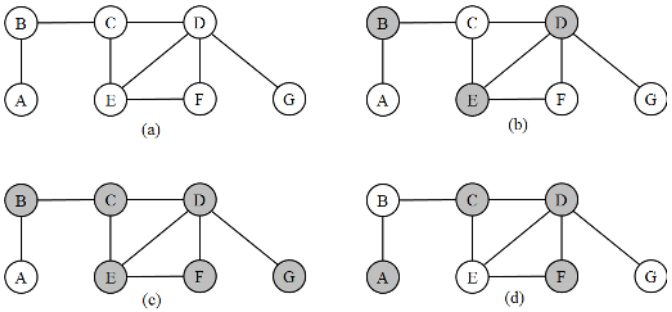

Figure 3: Comparison of Vertex-Cover results. *(a)* The original graph G. *(b)* The optimal cover set for G. *(c)* Cover set obtained with the traditional method [1]. *(d)* Cover set obtained from our algorithm.

The space complexity our algorithm is $\Theta(V + E)$ as described in [1], because we represent the graph as an adjacency list. The time complexity of our algorithm is as follows. For step 4, the complexity is O(V). In the for-loop, i.e., from steps 8 to 10, the algorithm needs to search its reference vertices for each individual vertex. The worst time complexity calculation of our approach is: $O(V * (V - 1)) = O(V^2)$. Thus, the worst time complexity of the algorithm is $\Theta(V^2)$.

We now show how our proposed approximate vertex algorithm is a polynomial-time $(2 - \varepsilon)$-approximation algorithm. Cormen *et al.*'s APPROX-VERTEX-COVER [1] is a polynomial-time 2-approximation algorithm, for C a set of vertex and C^* the optimal vertex cover, i.e., $|C| \leq 2|C^*|$.In our approach, the algorithms always picks one vertex and remove the edges connected to that vertex. This means that, most of the times, the algorithms does not consider both end points of an edge, which is what occurs in Cormen *et al.*'s algorithm. For our proposed method, we consider the resultant set of vertex as C^+, then $|C^+| = |C| - \varepsilon$.

$$\Rightarrow |C^+| = (2 - \varepsilon)\,|C^*|,\ 0 \leq \varepsilon \leq 1.$$

In some cases, C^+ will approach the optimal set of vertices when the value of ε is 1. If there is always only one highest weight vertex in each iteration, then our proposed algorithm's output set of vertices will be the optimal solution.

It is important to note that Khot *et al.* [13] show that the vertex cover might be hard to approximate to within $2- \varepsilon$, because it is a NP-Hard problem. Our proposed method computes the vertex cover in $2- \varepsilon$ approximation.

B. A Solution for the Approximate Vertex Reach Problem

We follow a similar process to solve the vertex reach problem. The vertex reach problem is to find a minimum number of vertices to reach all the vertices of a given graph within a specified number of hops – we use a single hop here.

Our approach is shown in Algorithm II and a simple illustration of the algorithm is shown in Figure 4.

Algorithm II: Approximate Vertex Reach Algorithm for a Social Network

Required: In the *List* we introduce another field *weight*
The *weight* is number of node in reference of the node (*ref*)
1: $C^{++} \leftarrow \phi$
2: L=*List*
3: L[w]= weight
4: (h, v) = highest weight of the list and respective vertex
5: **if** h ≠ -1 **then**
6: $C^{++} \leftarrow C^{++} \cup v$
7: v[w] ← -1
8: **for** all vertices of *List* L[ref]{k} ∈ {*v*} **do**
9: L[w] ← -1
10: **for** all vertex of *List* L[ref] ∈ {k} **do**
11: **if** L[w] ≠ -1 **then**
12: L[w] ← L[w] -1
13: **end if**
14: **end for**
15: **end for**
16: **go to** 4
17: **else**
18: **return** C^{++}
19: **end if**

As the vertex reach is a NP-Hard problem, our approach works on approximation. The reference weight initialization process is the same as the one described for vertex cover algorithm and it is shown in step 7 and 12 of Algorithm II. In the main step of the algorithm, the vertex with the highest weight is chosen in every iteration and all weights are updated as follows: the weight of the chosen vertex and that of all its immediate (one-hop) are set to -1; the weights of the neighbours of the selected node's neighbour are decreased by 1 if they are not already equal to -1. The selected node is added to C^{++} and the process repeats. Selected one hop neighbors from the *list* are stored in set *v* and two hop neighbors from the *list* are stored in set *k* as shown in step 8 and 10 of Algorithm II, respectively. This process continues until the weight of every node is -1; the final set C^{++}provides a solution for the vertex reach problem.

The space complexity of the algorithm is $\Theta(V + E)$, which is the same as for the vertex cover algorithm described earlier, again because we represent the graph as an adjacency list. Also as before, the time complexity of the Algorithm II's step 4 search in the list is O(V). As there are two for-loops, one inside the other for the number of vertex computation, the worst time complexity of the algorithm will be $\Theta(V^2)$.

Figure 4 shows the steps of vertex reach algorithm for our sample graph. The weights are initialised, as shown in Figure 4(a). The vertex with the highest weight is D; it is thus first selected, as shown in Figure 4(b) – shown shaded. Its weight is set to -1, as is the weight of all the nodes directly connected to D (edges are indicated by dashes in the figure); the weight of D's neighbour's neighbours is decreased by 1; D is added to the set C^{++}. The process is repeated. At this stage, as shown in Figure 4(c), there are two vertices with the same weight, i.e., A and B. The algorithm arbitrarily chooses one, in this case vertex B. It is added to the set C^{++} and all weights are updated again. The process continues until all weights are set to -1. The resultant set is shown in Figure 4(d). It contains two vertices *B,* and *D*.

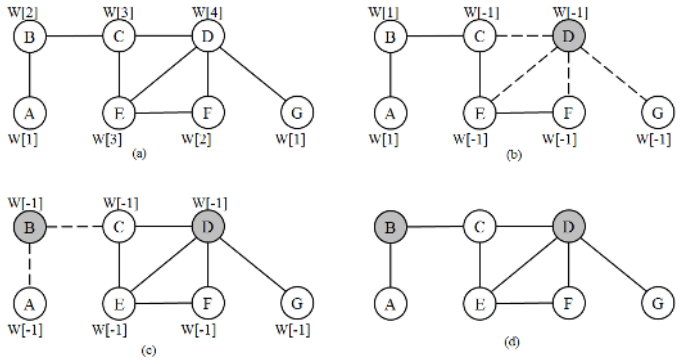

Figure 4: The operation of our approximate Vertex-Reach. *(a)* The input graph G, with the weights of the vertices initialised. *(b)* D is the vertex with the highest weight, and it is thus selected to be added to the set C^{++}. Its weight is set to -1, as is the weight of all its immediate (one hop) neighbours, and the weights of its two hop neighbours (neightbours' neighbours) decrease by 1. *(c)* The process repeats. A and B have same weight, and B is chosen arbitrarily. It is added to C^{++}. All weights are now -1, and the process stops. *(d)* The resultant Vertex-Reach set.

IV. Experiment and Evaluation

In the earlier section, we have shown theoretically that our algorithms take less space and time complexity compared to the standard traditional method [1]. In this section, our focus is to validate and demonstrate the performance of our algorithms experimentally. To this end, we conducted experiments on both dense and sparse graphs.

Consider a situation awareness application responding to the emergency scenario using Twitter data [32], where we would like to reach all members in the community to convey the current information (vertex reach problem), and we would also like to collect messages from their interactions to understand the emotional situation of the community (vertex cover problem). We thus want to identify the minimum sets of nodes required for these goals. We build a network matrix from the Twitter data interactions as the basis for the network analysis. We use UCINET software [7] for the visual demonstration of the analysis for our experiments where the nodes can be visually represented in a readable way in a two dimensional space. UCINET is a comprehensive program for analysing social networks, and it has been often used for this purpose since the early 1980s. The program contains network analysis routines (e.g., centrality measures, dynamic cohesion measures, positional analysis algorithms). The data sets given to UCINET are shown as binary networks with ties having the values of 1 and 0, where 1 represents a link between two nodes and 0 represents no links between two nodes. The experimental data set is randomly generated using the Excel software program and

imported to UCINET. We next describe the results of our experiments.

A. *Experiments with a Dense Graph*

We first considered a dense graph with 25 nodes, and ran both the vertex cover and the vertex reach algorithms. We next describe the results of our experiments.

1. Results of the Vertex Coverage

We first ran the traditional standard algorithm [1] and then our proposed algorithm. Figure 5 shows the results of the traditional vertex coverage algorithm for the 25 node randomly generated dense graph. The resultant cover set contains the nodes shown in red. It contains 22 nodes, meaning that these 22 nodes are required to cover 25 nodes of the original graph. The result of our algorithm is shown in Figure 6, and again the resultant cover set is shown in red. The set contains only 17 nodes.

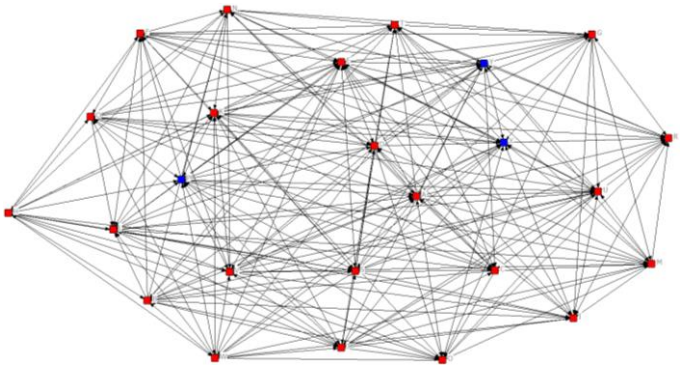

Figure 5: Experimental results for the traditional vertex cover method [1], showing the final cover set, which contains 22 nodes (shown in red).

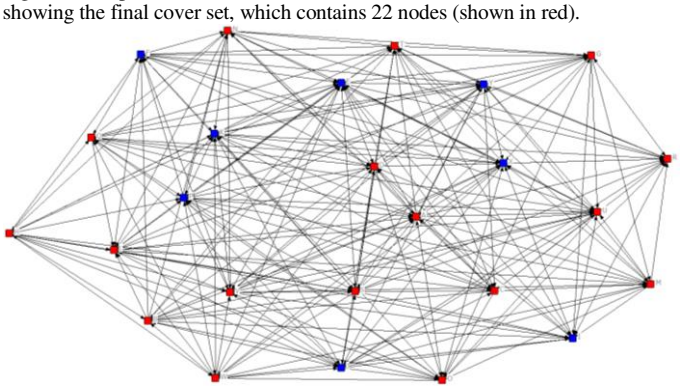

Figure 6: Experimental results for our proposed method. The resultant cover set contains 17 nodes (shown in red).

2. Results of the Vertex Reach

Figure 7 shows the dense graph used for vertex reach experiment. Similar to the experiment for the vertex cover problem, this graph contains 25 nodes. The result of our vertex reach algorithm is also shown in Figure 7, with the selected nodes in red. The resultant set contains 4 nodes, meaning that we can reach all nodes of the graph through these four nodes. We note that we cannot compare our algorithm with other methods as there are none available (note the difference between reachability and vertex reach problem).

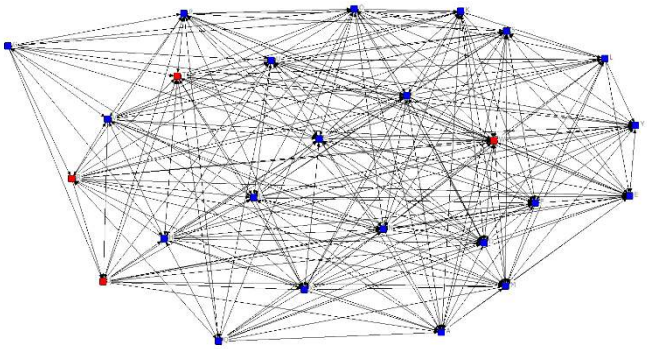

Figure 7: Experimental results for our proposed method: the reach set contains 4 nodes (shown in red).

B. *Experiments with a Sparse Graph*

We now consider a sparse graph with 25 nodes but only 71 edges, as shown in Figure 8. Similar to previous sub-section, we next describe the results for both the vertex cover and vertex reach algorithms.

1. Results of the Vertex Coverage Algorithm

We ran the experiment on the random sparse graph of Figure 8, first using the traditional standard vertex cover algorithm and then with our proposed algorithm. The results of the traditional approach is shown in Figure 8(a), where the red nodes represent the resultant set. It took 16 nodes to cover this sparse 25 node graph. In comparison, our algorithm found a set with fewer nodes (14) to cover the same network, as shown in red in Figure 8(b).

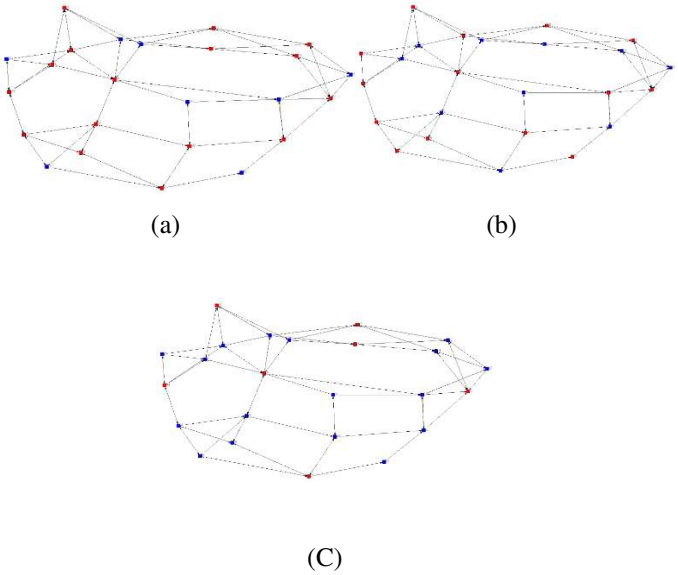

Figure 8: (a) Experimental results of the traditional vertex cover method over the sparse graph. (b) Experimental results for our proposed vertex cover algorithm. (c) Experimental results for our proposed vertex reach algorithm.

2. Results of the Vertex Reach

The input graph for the vertex reach experiment is the same as vertex cover as shown in Figure 8: a sparse graph with 25 nodes and 71 edges. The result of the proposed vertex reach algorithm is shown in Figure 8(c). The result set contain 7 nodes to reach all the nodes of the graph that are shown in red.

C. Scalability

We have illustrated the performance of our proposed vertex cover and vertex reach algorithms random graphs of 25 nodes, with both dense and sparse connectivity. In a real life scenario, considering the more than one billion Facebook users or 288 million active users in Twitter, we need to deal graphs with a much larger number of nodes. In order to show that our proposed approach is scalable to graphs with a large number of nodes, we ran our algorithms (both vertex cover and vertex reach) as well as the traditional method for vertex cover with graphs of varying number of nodes up to 1,000,000 with intervals of 5000 nodes. The result is shown in Figure 9. We see that our proposed method always resulted in a vertex cover set with fewer nodes than that of the traditional approach. We also see that the result of vertex reach algorithm is consistently about 10% of the total number of nodes in the graph. As the vertex reach algorithm is significant for information flow, it means we only need to target 10% of all people in a social network in order to reach the entire population of the network.

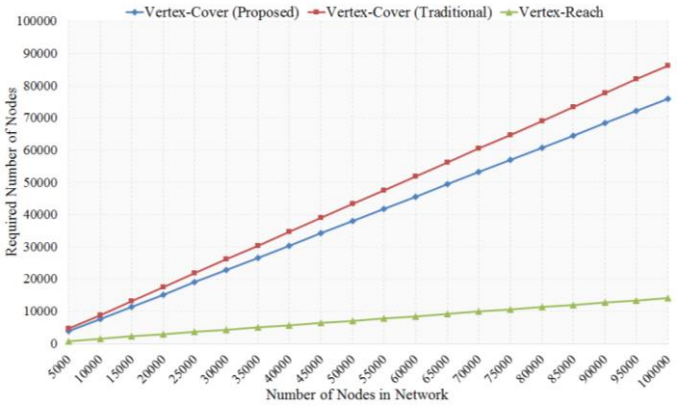

Figure 9: Efficiency of our algorithms for reach and cover sets; the cover set algorithm is also compared to the traditional method for network coverage.

V. Applications

There are many applications for both vertex cover and vertex reach solutions. In the introduction, we motivated the work through the need to stop the propagation of misinformation in Twitter. We now provide further applications beyond the limitation of misinformation, although the list is by no means an exclusive list of potential applications. We can broadly classify the applications into two groups: sensor network applications and social network applications.

A critical aspect of applications using wireless sensor networks is network lifetime. Power-constrained wireless sensor networks are usable as long as they can communicate sensed data to a processing node. Sensing and communications consume energy; therefore judicious power management and sensor scheduling can effectively extend network lifetime. The vertex cover solution can help choose a set of nodes to cover the network which prolong the network life time [16][17]. The vertex cover solution can also be used for network base routing delays in tolerance network [3]. Other potential applications include network traffic measurement, monitoring nodes for bandwidth measurement and flow monitoring [4]. Finding alternate path selections during routing is one of the major applications that can utilise solutions for the vertex cover problem [5]. Delay Tolerant Networks (DTNs) for social network significantly disable the adequacy of information scattering. To solve this, Geo *et al.* [21] formulate relay selections for multicast as a unified knapsack problem by exploiting node centrality and social community structures. Solutions to the vertex cover problem can be used to solve DTNs for social network.

The solutions to the vertex reach problem can be applied to "spread of influence" problems in social networks. For example, one such problem is determining to which k consumers a product should be marketed to ensure its widespread adoption. In [18], Kempe *et al.* show that the objective function of this problem is submodular and, as a result, a greedy algorithm finds a solution within a constant factor from the optimal. The same property is observed by Leskovec *et al.* in [19]. In this work, the authors determine which k blogs one should read to detect quickly the outbreak of an important story. The dissemination of dynamic content, such as news or traffic information, over a mobile social network is another area where the solutions to the vertex reach problem can be applied. Another application is to improve coverage and increase capacity of mobile social network by sharing the updated contents among users [20], where the vertex reach and cover problem can be applied to improve the contacts.

VI. Conclusions and Future Work

In this paper, we proposed efficient vertex cover and vertex reach algorithms for social network applications. The proposed solutions can be applied to other graph based algorithms. We also showed through a theoretical analysis and an experimental evaluation that our proposed vertex cover approach provides a $2 - \varepsilon$ approximation algorithm which results in a cover set with fewer number of nodes than the solution obtained by the standard vertex cover algorithm, with less space complexity. Our proposed vertex reach solution reach individual nodes of the network with less number of nodes. The resulting set for our vertex reach algorithm contains about 10% to 15% of the total nodes in the graph.

We plan to pursue a number of research avenues in future. The foremost is the application of our algorithms to social network applications such as limiting the flow of misinformation and targeted campaigning. We also plan to perform a study of our work on real life data sets. The proposed vertex reach algorithm works for a single hop. We plan to generalise the algorithm for a specified number of hops so that we can use the reachability to reduce the number of target nodes. The vertex reach algorithm also needs to consider the temporal aspect of edges (e.g., freshness of the interactions) so that the flow of information can be achieved within a certain time, which is important for emergency scenarios.

VII. Acknowledgement

This research is funded by Australia India Strategic Research Grant titled "Innovative Solutions for Big Data and Disaster Management Applications on Clouds (AISRF-08140) from the Department of Industry, Australia.

References

[1] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. Vol. 2. Cambridge: MIT press, 2001. ISBN 0-262-03293-7.

[2] Liu, Chung Laung. Elements of discrete mathematics. Vol. 101. New York: McGraw-Hill, 1985.

[3] Ding, Li, Bo Gu, Xiaoyan Hong, and Brandon Dixon. "Articulation node based routing in delay tolerant networks." In *Pervasive Computing and Communications, 2009. PerCom 2009. IEEE International Conference on*, pp. 1-6. IEEE, 2009.

[4] Zeng, Yongguo, Dezheng Wang, Wei Liu, and Ao Xiong. "An approximation algorithm for weak vertex cover problem in IP network traffic measurement." In*Network Infrastructure and Digital Content, 2009. IC-NIDC 2009. IEEE International Conference on*, pp. 182-186. IEEE, 2009.

[5] Akkaya, Kemal, and Mohamed Younis. "A survey on routing protocols for wireless sensor networks." *Ad hoc networks* 3, no. 3 (2005): 325-349.

[6] Puthal, Deepak. "A Near Optimal Approximation Algorithm for Vertex-Cover Problem." *arXiv preprint arXiv:1309.4953*, 2013.

[7] Borgatti, Stephen P., Martin G. Everett, and Linton C. Freeman. "Ucinet for Windows: Software for social network analysis." Harvard, MA: Analytic Technologies, 2002.

[8] Karakostas, George. "A better approximation ratio for the vertex cover problem." In *Automata, languages and programming*, pp. 1043-1050, 2005.

[9] Karakostas, George. "A better approximation ratio for the vertex cover problem." *ACM Transactions on Algorithms (TALG)* Vol. 5(4), pp. 41.1-41.8, 2009.

[10] Bar-Yehuda, Reuven, and Shimon Even. "A local-ratio theorem for approximating the weighted vertex cover problem." *North-Holland Mathematics Studies* 109, pp. 27-45, 1985.

[11] Monien, Burkhard, and Ewald Speckenmeyer. "Ramsey numbers and an approximation algorithm for the vertex cover problem." *Acta Informatica* vol. 22(1), pp. 115-123, 1985.

[12] Arora, Sanjeev, Satish Rao, and Umesh Vazirani. "Expander flows, geometric embeddings and graph partitioning." *Journal of the ACM (JACM)* Vol. 56(2), 2009.

[13] Khot, Subhash, and Oded Regev. "Vertex cover might be hard to approximate to within 2− ε." *Journal of Computer and System Sciences* vol. 74(3), pp. 335-349, 2008.

[14] Shah, Kartik, Praveenkumar Reddy, and R. Selvakumar. "Vertex Cover Problem—Revised Approximation Algorithm." In *Artificial Intelligence and Evolutionary Algorithms in Engineering Systems*, pp. 9-16. Springer India, 2015.

[15] Kuhn, Fabian, and Monaldo Mastrolilli. "Vertex cover in graphs with locally few colors." *Information and Computation* Vol. 222, pp. 265-277, 2013.

[16] Huang, Chi-Fu, and Yu-Chee Tseng. "The coverage problem in a wireless sensor network." *Mobile Networks and Applications* vol. 10(4), pp. 519-528, 2005.

[17] Cardei, Mihaela, My T. Thai, Yingshu Li, and Weili Wu. "Energy-efficient target coverage in wireless sensor networks." In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, pp. 1976-1984. IEEE, 2005.

[18] Kempe, David, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network." In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 137-146. ACM, 2003.

[19] Leskovec, Jure, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. "Cost-effective outbreak detection in networks." In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 420-429. ACM, 2007.

[20] Ioannidis, Stratis, Augustin Chaintreau, and Laurent Massoulie. "Optimal and scalable distribution of content updates over a mobile social network." In*INFOCOM 2009, IEEE*, pp. 1422-1430. IEEE, 2009.

[21] Gao, Wei, Qinghua Li, Bo Zhao, and Guohong Cao. "Multicasting in delay tolerant networks: a social network perspective." In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, pp. 299-308. ACM, 2009.

[22] Michael jackson on tmz, iran on twitter. http://www.blogher.com/spreading-news.

[23] E. Morozov. Swine flu: Twitter's power to misinform. *Foreign Policy*, April 2009.

[24] Manovich, Lev. "Trending: the promises and the challenges of big social data." 2011.

[25] Boyd, Danah, and Kate Crawford, "Six provocations for big data." A Decade in Internet Time: Symposium on the Dynamics of the Internet and Society, 2011.

[26] Hu, Han, Y. O. N. G. G. A. N. G. Wen, T. Chua, and X. U. E. L. O. N. G. Li. "Towards Scalable Systems for Big Data Analytics: A Technology Tutorial." IEEE Access, Vol. 2, pp. 652 – 687, 2014.

[27] Wikibon. *A Comprehensive List of Big Data Statistics*, 2013 [online]. Available: http://wikibon.org/blog/big-data-statistics/

[28] Gantz, John, and David Reinsel. "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east." *IDC iView: IDC Analyze the Future* 2007, pp. 1-16, 2012.

[29] Budak, Ceren, Divyakant Agrawal, and Amr El Abbadi. "Limiting the spread of misinformation in social networks." In *Proceedings of the 20th international conference on World Wide Web*, pp. 665-674. ACM, 2011.

[30] Boyd, Danah M. "Friendster and publicly articulated social networking." Conference on Human Factors and Computing Systems, pp. 1279-1282, 2004.

[31] Meiss, Mark R., Filippo Menczer, and Alessandro Vespignani. "Structural analysis of behavioral networks from the Internet." *Journal of Physics A: Mathematical and Theoretical*, vol. 41(22), pp. 224-022, 2008.

[32] Cameron, Mark A., Robert Power, Bella Robinson, and Jie Yin. "Emergency situation awareness from twitter for crisis management." In *Proceedings of the 21st international conference companion on World Wide Web*, pp. 695-698. ACM, 2012.

[33] Mtibaa, Abderrahmen, Augustin Chaintreau, Jason LeBrun, Earl Oliver, Anna-Kaisa Pietilainen, and Christophe Diot. "Are you moved by your social network application?." In *Proceedings of the first workshop on Online social networks*, pp. 67-72. ACM, 2008.

[34] He, Hao, Haixun Wang, Jun Yang, and Philip S. Yu. "Compact reachability labeling for graph-structured data." In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pp. 594-601. ACM, 2005.

[35] Hwang, Moon Ho, and Bernard P. Zeigler. "Reachability graph of finite and deterministic devs networks." *IEEE Transactions on Automation Science and Engineering*.Vol. 6 (3), 2009.

[36] Jin, Ruoming, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. "Computing label-constraint reachability in graph databases." In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 123-134, 2010.

[37] Fan, Wenfei, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. "Adding regular expressions to graph reachability and pattern queries." In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pp. 39-50, 2011.