IJSEAT International Journal of Science Engineering and Advance Technology

Discovering Closely Related Peers of a Person in Social Networks

Veepuri VS Prasad Raju [1] , D.Suneetha [2]

[1]M.tech, Department of Computer Science and Systems Engineering, Andhra University, Visakhapatnam

[2]Assistant Professor, Department of Computer Science and Engineering, GITAM University, Visakhapatnam

Email: prasadraju.veepuri@gmail.com , suneethadwarapu@yahoo.com

Abstract:
In social networks, finding a group of similar people of a specified person is meaningful task in many areas like substitution/alternate recommendation system. For a given person, considering hobbies, interests etc., as base forming a group of peers from social networks. Here we propose mutual unique identification group(MUID) algorithm for identifying closely related peers.

Keywords: *social networks, node discovering, social query.*

Introduction:
Day by day the volume of data in the social networks is increasing rapidly. According to Pew Research Center on Internet Science and technology new bulletin dated 8th Oct'2015, 65% adults are using Social Networking sites which is a tenfold increase in the last decade. Since lot of data is getting accumulated in social networks, they provide a rich source for data mining researchers to extract hidden patterns and knowledge useful to various domains. In heterogeneous social networks, identifying similar peers is meaningful task. Consider 'ego' as a query node to find uniqueness of that node from its peers. each node/ entity is associated a type/label to describe its category. Consider an expertise bipartite network that relates/connects two types of nodes represents experts and their topic of expertise as depicted in figure 1.

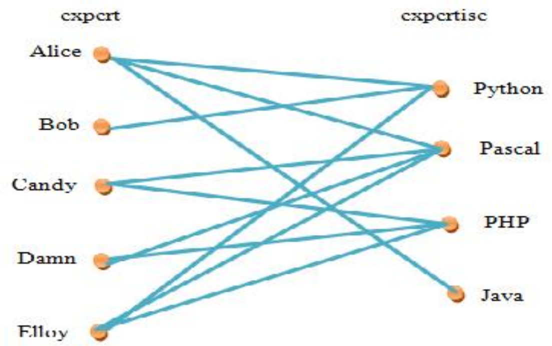


Figure1 expertise bipartite graph

In the above bipartite graph. Alice is the only one with expertise in Python, Pascal and Java. So Alice can be uniquely identified by the set {Python, Pascal, Java}. In some cases we can't distinguish one from other like Candy and Damn are expertise in both Pascal and PHP. So in such case we can define as besides Damn, Candy is a person who is expert in Pascal and PHP. In general we can represent Damn's UID as {Candy, Pascal, PHP}. If any projects comes on PHP and Pascal then we can assign Damn or Candy to accomplish that project. Consider a sub-graph Candy, Damn ,Elloy, Pascal, PHP. For any node belonging to the set, the UID of that node is available in that set. In other words, with respect to the expertise in Pascal and PHP, Candy, Damn and Elloy are indiscernible. Hence the set {candy, damn, Elloy, Pascal, PHP } is called as Mutual identification group(MUID).

Related works:

Our paper is partly inspired by data-mining work on finding unique identification sets and mutual identification groups of a ego node. Our work is related to the existing studies on social networks in three aspects namely community search queries, social network search/extraction, and social network anonymization. Some of the papers on community search queries is mainly emphasizes on selecting a set of nodes and searching a specific community depending upon the given query nodes or other constraints.

Some papers on social entity search/extraction focuses on extracting social relationship from a specific set of people from available resources on Web, authors Zhu J, Nie Z, Liu X, Zhang B, Wen J-R [13] designed an entity relationship search framework on Web data, authors Tang J, Zhang J, Yao L, Li J, Zhang L, Su Z [9] designed Arnetminer which is an academic search system that mainly aims to automatically extract the researcher's profile including the co-authorship relation graph from the Web. These studies focus on extracting some data from social networks. For a set of nodes in a graph, authors Sozio M, Gionis A [8] proposed an algorithm for finding a densely connected sub-graph in social network constituting communities. Rather than the above works, we mainly emphasize on finding the Mutually Unique Identification Groups for a given query vertex from given social network.

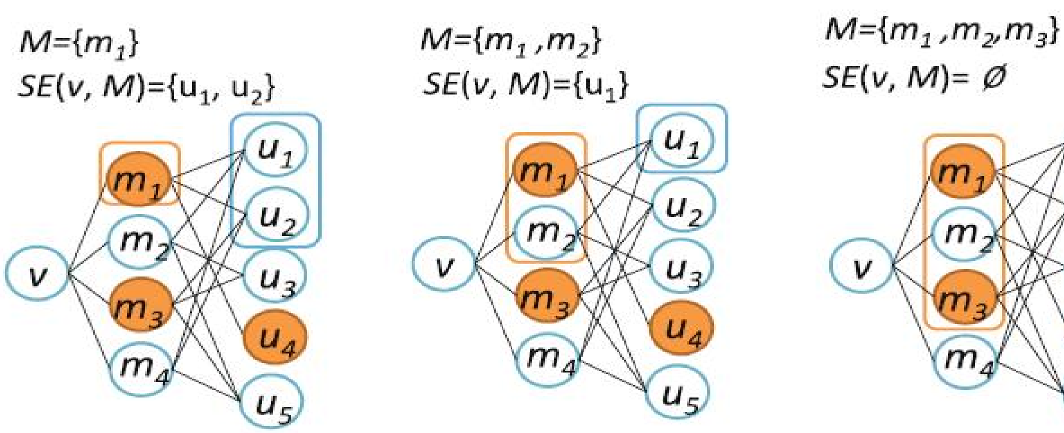
Preliminaries:

we provide some formal definitions which are taken from reference paper [1] .

Problem Definition:

Consider undirected labeled graph *G(V, E)* and a query node $v \in V$ whose first and second order neighbor set is denoted by *N(v)* and *N2(v)*, respectively. Every vertex in graph *G* is labeled with a type (ex: color, award_type, place etc.) denoted by *type(v)*.

Definition 1: Consider a graph *G(V, E)*, given a query vertex $v \in V$ and a set of nodes $M \subseteq N(v)$, a node $u \in V$ is said to be *structure equivalent* (SE) to *v* given *M*, denoted by $u \in SE(v, M)$, if $type(u) = type(v)$ and $M \in N(v)$. *SE(v, M)* is the set of nodes structure equivalent to *v* given *M*. if $SE(v, M) = \phi$ then, there does not exist any node structure equivalent to *v* given *M*



Consider above example, In Fig.(a), let $M = \{m1\}$, then $SE(v, M) = \{u1, u2\}$. here *u*4 is not inculcated in the *SE(v, M)* because u4 is different type from *v*. Figure (b), another vertex *m*2 is added into *M* (note that the nodes in *M* do not need to be of the same type), here we observed that the set *SE(v, M)* becomes smaller since $u2 \notin N(m2)$ and has to be removed from *SE(v, M)*. from the two example we can depicts that adding nodes into *M* ,the number of nodes in *SE(v, M)* gradually decreasing. In Fig. (c), when we add another node i.e, $M = \{m1, m2, m3\}$, no vertex is connected to every element in *M*. Therefore, $SE(v, M) = \phi$.

Definition 2 consider a graph *G(V,E)*, query vertex *v* and a non-empty set $M \in N(v)$, we define that uniqueness of node 'v' can be identified by the 2-tuples set *[M, SE(v, M)]*, which set is called a UID of *v*.

In some cases *SE(v,M)* may be empty. In such case we can say that "*v* is unique since there is no other vertices in *M* commonly connected one vertex as *v* does".

When *SE(v, M)* is non-empty, then the uniqueness of *v* can be interpreted as "*v* should be unique because, besides the vertices in *SE(v, M)*, the only vertex that connects to *M* is *v*". Next we introduce an interesting and useful property of UIDs.

Property 1: UIDs of a given query vertex *v* can be always be found within two-hops *v*.

By Definition 2, For the given query vertex v, there exists many possible UIDs. From those we have to choose optimal UIDs. So we use comparison function to test the quality of UIDs.

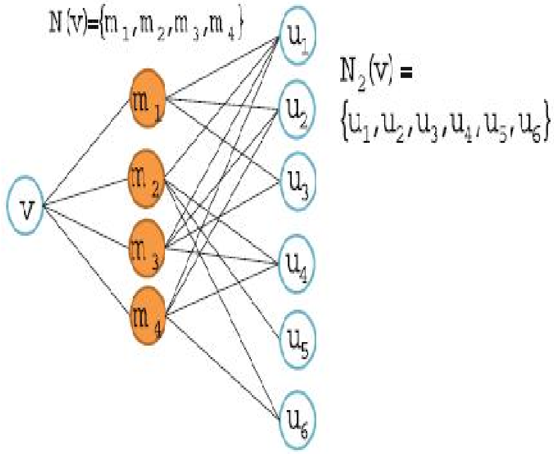
Definition 3 for the given two UIDs of *v*, $D1 = [M1, SE(v, M1)]$ and $D2 = [M2, SE(v, M2)]$, we define a comparison function *Q(v, M1, M2)* as follows.

(a) $Q(v, D1, D2) = 1$, i.e., the quality of *D*1 is better than that of *D*2, if (i) $|SE(v, M1)| < |SE(v, M2)|$ or (ii) $|SE(v, M1)| = |SE(v, M2)|$ and $|M1| < |M2|$.

(b) $Q(v, D1, D2) = 0$, i.e., the quality of *D*1 equals to that of *D*2, if $|SE(v, M1)| = |SE(v, M2)|$ and $|M1| = |M2|$.

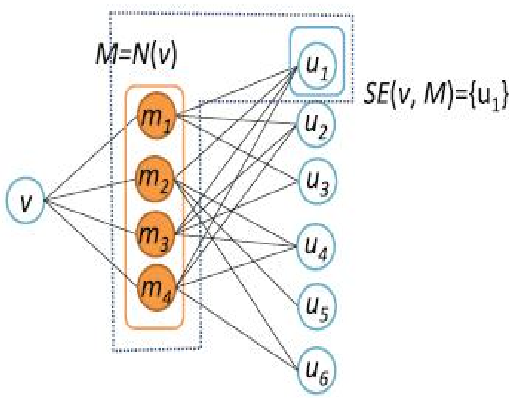
Finding UIDs:

Here we would compute UIDs using *one Hop+* and *Multiple neighbor* algorithms and tabulate the results. In *one Hop+* method we have to add all the vertices to *M* which are connected to query vertex *v*. Consider the below example.



Consider the above graph, *'v'* is query vertex,*N(v)* is first neighbors of *v* and $N_2(v)$ is second neighbors of *v*.now by considering *one Hop+* method we can find UID as follows.

Query vertex *'v'* consists of four neighbors those are added into base set *M*. for these vertices, commonly connected nodes are u_1 only which is structure equalent to node *'v'*. Here advantage is size of structure equalent(SE) is less but produces largest *M* set.



Second method to find UIDs is *multiple Neighbour* method. In this method we have to dynamically add vertices to *M* to produce minimal SE.for this we defined a formula as follows,

Definition 5 Given UID of a query vertex *v* as $D = [M, SE(v, M)]$, and for $M' = M \cup \{n\}$, where $n \in N(v) - M$, then $SEgain(v, n, M) = SE(v, M) - SE(v, M')$.

In the below Theorem we prove that the set size of *SEgain(v, n, M)* is monotonic with the size of M. In this property, $\forall u \in SE(v, N(v))$, u cannot be in *SEgain(v, n, M)* for any v, n, M since there is no larger subset $M \subseteq N(v)$ than N(v) itself.

Theorem : Given $SE(v, M)$, $SE(v, M')$, $n \in N(v)$, $n \notin M$ and $n \notin M'$, if $M' \supseteq M$ then $SEgain(v, n, M') \subseteq SEgain(v, n, M)$.

we have the following derivation.

(a) $\because M' \supseteq M$, $\therefore SE(v, M') \subseteq SE(v, M)$, and

(b) $\because M' \cup \{n\} \supseteq M \cup \{n\}$, $\therefore SE(v, M' \cup \{n\}) \subseteq SE(v, M \cup \{n\})$.

From (a) and (b), we have

$$SEgain(v, n, M') = SE(v, M') - SE(v, M' \cup \{n\})$$

$$\subseteq SE(v, M) - SE(v, M' \cup \{n\})$$

$$\subseteq SE(v, M) - SE(v, M \cup \{n\})$$

$$= SEgain(v, n, M).$$

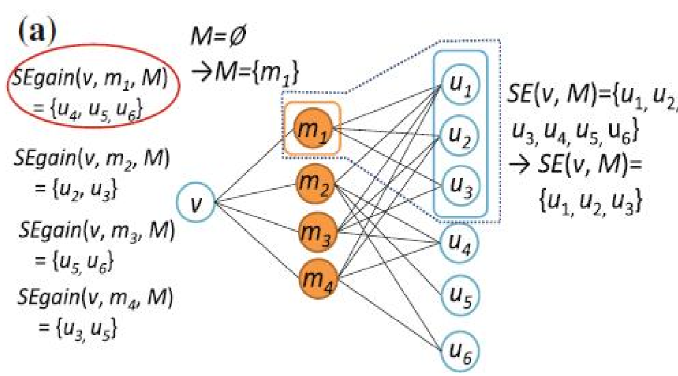
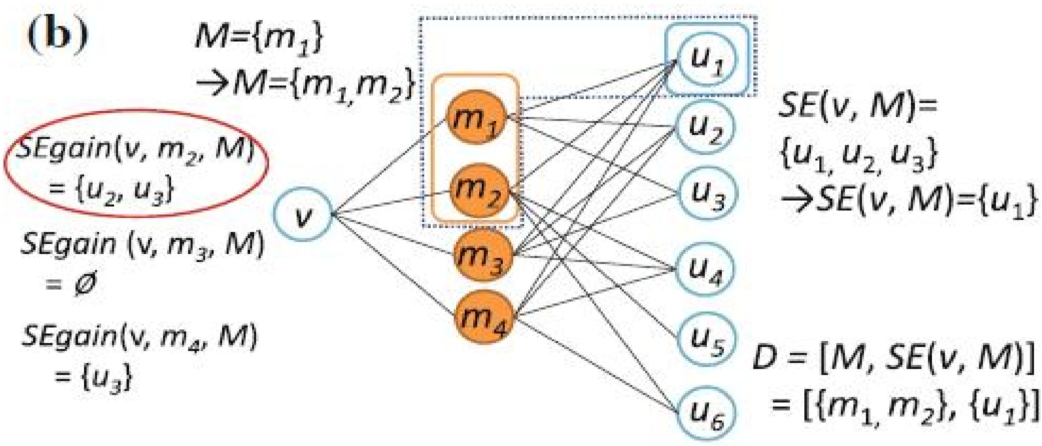


Fig. 6 The UID of *v* found by the *Multiple-Neighbor* method **a** m_1 is chosen as the first vertex to be added into *M* **b** m_2 is chosen as the second vertex to be added into *M*



In figure (a), calculated the *SEgain* for every vertex. From those we chosen *max SEgain* that is assigned to M. In figure (b),we calculated the *SEgain* for m_2,m_3,m_4 and again chosen *max SEgain* that is added to the M. This process continues until *SEgain* gets null. Consider the below pseudo code to illustrate the above example.

Input: provide Graph G=(V,E) and a query vertex v
Output: UID D of query vertex v

```
Initialize SE with N2(v)
Initialize M with null(ϕ )
while | SE |  > 0 do                    // roll the loop till all nodes in SE >0
        Mmax ← argmaxM' SEgain(v,m,M) |,     m ∈ {N(v)-M} //find max of
        if  Mmax = ϕ and | M | >=1 then       //SEgain of  M and assign to  Mmax
                break;
        end
        Assign M with { M ∪ mmax }
        Assign SE with { SE – SEgain (v, mmax, M) }
end
D ← [M,SE]
Output D;
```

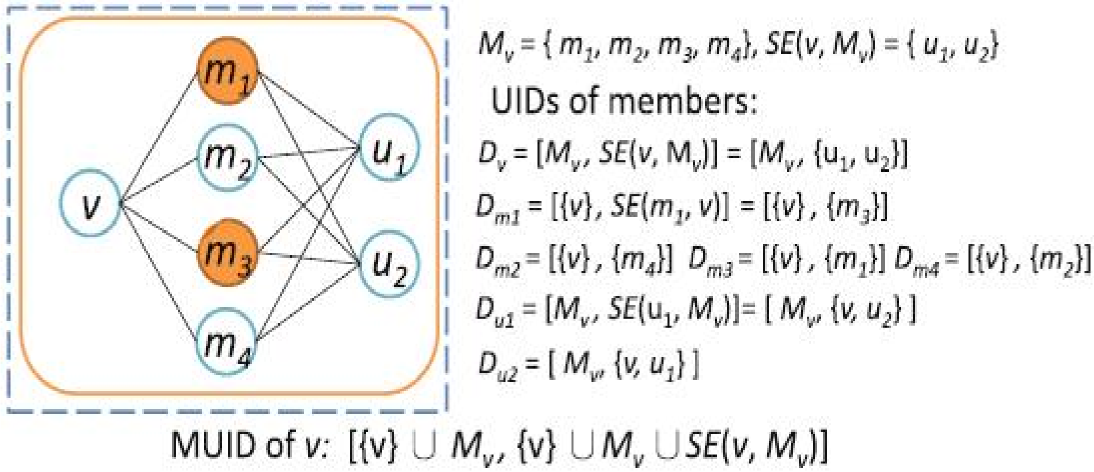
Finding MUIDs:

In this section we have to discuss about MUID briefly. Consider the MUID definition below.

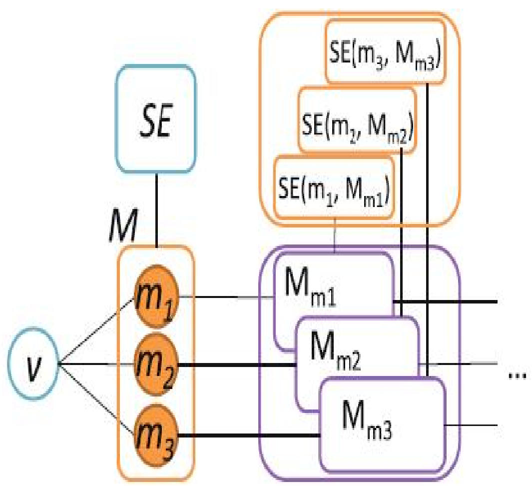
Definition 4 consider graph *G(V, E)*, query vertex *v* ∈ *V*, a set of vertices *X* ⊆ *V* is a MUID of *v* if the following two conditions satisfied.

(a) $v \in X$.
(b) $\forall u \in X, \exists D \subseteq X$, *D* is a UID of *v*

In the above graph initially we find the UID of *v* and later we find UIDs of every member in that set. At the end we performed union of all UID sets which is nothing but MUID. The main drawback here is for a given query node we have to include all neighbors to MUID set and we also find the every node UID is exists in that set. Here indirectly size



In previous section we discussed about two methods to find UIDs for a given query vertex *v*. now we extend those two algorithms to find MUID of a given vertex *v*. Let us illustrate the first algorithm *one Hop+ method*. In MUID ,we have to find the unique identification set(UID) of query node and every node's UID in that set has subset of itself. Consider the below graph. violates definition:4 rule. In order to minimize SE size we have to go for other algorithm *multiple neighbor MUID* algorithm. In *multiple neighbor MUID*, we just extend UID by finding every member's Unique ID set and union to MUID.Consider the below example,



In the above example, We have proposed to use a heuristic formula $SEgain(v, n, M) = SE(v, M) - N(n)$ as a metric to select the next neighbor to be added in the *Multiple-Neighbor* method. This criterion determines how many number of nodes should be removed from the UID if $n \in N(v)$ is added.

In Multiple-Neighbor–MUID, all the newly inculcated nodes in *M* and *SE(v, M)* also uniquely identified. Similar to the *Multiple-Neighbor* method for the UID problem, here we define a heuristic function to estimate the quality of neighbors as candidates to be added into *M*. The idea is to execute the UID *Multiple-Neighbor* method for one pass on all nodes and record the *M* and *SE(v, M)* sets of each node *v* as *UIDSE(v)* and *UIDM(v)*. When choosing the neighbors of *v* into *M* in *Multiple-Neighbor MUID*, we give higher priority for neighbor *n* with smaller $|SE(v, M)|$ in UID, or $|UIDSE(n)|$. Given equal-sized $|UIDSE(n)|$, we then define the secondary criteria as minimizing $|UIDM(n) - M^{'}|$. This is because that exploiting neighbors already in *M* could potentially introduce fewer new vertices. Given equal-sized $|UIDSE(n)|$ and $|UIDM(n) - M^{'}|$ the tertiary criteria is larger $SEgain(v,m, M)$. Note that for *n* to be chosen into *M*, $|SEgain(v, n, M)|$ must be larger than 0. Algorithm provided to illustrate MUID as follows

---

input: graph G=(V,E), a query vertex v, pre computed SE(v , M) and M in UID
output: MUID set X of v

```
Take first neighbors N(v)={m_1,m_2,m_3,....m_d}
Assign M_x with ϕ
Assign SE_x with ϕ
Initialize UID_SE(v) with SE(v,m) of v's UID
Initialize UID_m(v) with M of v's UID
W is the stack and initially empty
// stack W stores not yet uniquely identified vertices in it
push vertex v into W
while | W | > 0 do
        assign W with the stack top element popped from W
        Initialize SE_w with SE(w,M_x) , and M_w with ϕ
        N_x ← N(v) - X //here X is final output set initially its empty.
          while | SE_w | > 0 do
                  for n ∈ N_x do
                              if | SEgain(w,n,M_w) | =0 then
                        N_x ← N_x –{n}
                              end                    // if end
                  end                          // for end
                  if N_x = ϕ then
                        break;
                  end                          // if end
                  minUID_SE ← max integer
                  minUID_M ← max integer
                  maxSEG ← min integer
                  n_opt ← null
                  for n ∈ N_x do
                              if ( UID_SE(n) < minUID_SE) or
                              ( UID_SE(n) = minUID_SE and UID_M(n) < minUID_M) or
                        ( UID_SE(n) < minUID_SE and UID_M(n) = minUID_M and
                              ( SE_gain(w, n, M_w) > maxSEG) then
                                minUID_SE ← UID_SE (n)
                                minUID_M ← UID_M(n)
                                maxSEG ← SE_gain(w, n, M_w)
                                n_opt ← n
                              end                          //if end
                    end                        // for end
                          M_w ← M_w ∪ { n_opt }
                        SE_w ← SE_w - N (n_opt)
                        push n_opt into W
          end                                  // inner while end
          for u ∈ SE_w do
                        if N(u) ⊃ N(w) then
                              push u into W // ∃M_u that w ∉ SE(u,M_u)
                        end                          // if end
                  end                                // for end
          M_x ← M_x ∪ M_w
          SE_x ← SE_x ∪ SE_w
        end                                    // outer while
  Output X= [M_x , SE_x]
```

The quality measure of MUIDs is conceptually similar to UIDs. For each MUID, the primary goal is to minimize $|SE(v, M)|$ and secondary goal is to minimize $|M|$ of the UIDs. we can formally define the metric for MUID: Given an MUID X, for all $v \in X$, the UID $D_v = [M_v, SE(v, M_v)]$. For the metric of union size, we define size of union of SE set (USE) and size union of M set (UM):

U S E = $|\bigcup_{v \in X} SE(v, M_v)|$, and

U M = $|\bigcup_{v \in X} M_v|$.

We are now able to compare the quality of two MUIDs by replacing $|SE(v, M)|$ and $|M|$ in Definition 3 with *USE* and *UM*. Similarly, for the metric of sum of size, we define total size of SE set (TSE) and total size of M set (TM):

T S E = $\sum_{v \in X} |SE(v, M_v)|$, and

T M = $\sum_{v \in X} |M_v|$.

Experimental Results :

We performed experiments on three datasets which include KDD movie dataset includes 35311 vertices, and edges 168868 and 20 different types of nodes(means categories).Taiwan academic network consists of 63122 number of vertices, 770155 number of edges and 6 types of nodes.finally HepTh citation network has 41840 number of vertices having 4 different types of nodes and 933149 number of edges.

Algorithms applied on above datasets and results are tabulated below.

Name	|V|	|E|	Number of types
KDD movie data set	**35311**	**168868**	**20**
TW academic network	**63122**	**770155**	**6**
HepTh citation network	**41840**	**933149**	**4**

UID Experimental results:

Dataset	MN	OH
KDD movie	1.20	1.36
TW academic	1.71	1.92
HepTh	1.00	2.69

MUID Experimental results:

Dataset	MN	OH
KDD movie	1.60	1.43
TW academic	1.87	2.04
HepTh	1.91	2.32

Conclusion:

This paper shows that the uniqueness of a node can be captured by using *Multiple neighbor* method and *One Hop+* method and which are extended to find mutual identification groups. In this work we have done nodes of similar types. The future work relay on some other metrics like diversity of types, coverage, etc, and can provide effective methods to identify such sets and compare and contrast with each other

References:

1. Yi-Chen Lo, Jhao-Yin Li, Mi-Yen Yeh, Shou-De Lin, Jian Pei (2013) *What distinguish one from its peers in social networks*. In Springer.

2. Albert R, Barabási A-L (2002) *Statistical mechanics of complex networks*. Rev Modern Phys 74:47–97

3. Erd˝os P, Rényi A (1961) On the evolution of random graphs. Bull Inst Int Stat 38:343 Fortunato S (2010) *Community detection in graphs*. Phys Rep 486(3–5):75–174

4. Lappas T, Liu K, Terzi E (2009) *Finding a team of experts in social networks*. In Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining

5. Li C-T, Lin S-D (2009) *Egocentric information abstraction for heterogeneous social networks.* InASONAM,2009

6. Li C-T, Shan M-K (2010) *Team formation for generalized tasks in expertise social networks*. In IEEE SocialCom

7. Newman M (2004) *Detecting community structure in networks*. Eur Phys J B Condens Matter Complex Syst 38(2):321–330

8. Sozio M, Gionis A (2010) *The community-search problem and how to plan a successful cocktail party*. In Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining

9. Tang J, Zhang J, Yao L, Li J, Zhang L, Su Z (2008) *ArnetMiner: extraction and mining of*

ISSN 2321-6905

academic social networks. In Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining

10. Watts D, Strogatz S (1998) *Collective dynamics of small-world networks*. Nature 363:202–204

11. Yang D-N, Chen Y-L, Lee W-C, Chen M-S (2011) *Social-temporal group query with acquaintance constraint.* In Proceeding of international conference on very large data, bases

12. Zhou B, Pei J (2008) *Preserving privacy in social networks against neighborhood attacks.* In Proceedings of IEEE international conference on data, engineering

14. Zhu J, Nie Z, Liu X, Zhang B, Wen J-R (2009) *StatSnowball: a statistical approach to extracting entity relationships.* In Proceedings of international world wide web conference.