K–partitioning of Signed or Weighted Bipartite Graphs

Nurettin B. Omeroglu, Ismail H. Toroslu
Middle East Technical University, Dep. of Computer Engineering, Ankara, Turkey
{omeroglu, toroslu}@ceng.metu.edu.tr

Sedat Gokalp, Hasan Davulcu
Arizona State University, Sch of Computing, Inf. and Decision Sys. Eng., Tempe, AZ, USA
{sedat.gokalp, hasan.davulcu}@asu.edu

***Abstract*— In this work, K-partitioning of signed or weighted bipartite graph problem has been introduced, which appears as a real life problem where the partitions of bipartite graph represent two different entities and the edges between the nodes of the partitions represent the relationships among them. A typical example is the set of people and their opinions, whose strength is represented as signed numerical values. Using the weights on the edges, these bipartite graphs can be partitioned into two or more clusters. In political domain, a cluster represents strong relationship among a group of people and a group of issues. In the paper, we formally define the problem and compare different heuristics, and show through both real and simulated data the effectiveness of our approaches.**

Keywords*— *Social Networks, Bipartite Graphs, Graph Partitioning

I. INTRODUCTION

Social networks became one of the hottest topics of computer science in recent years. One very common form of a social network is actually a simple bipartite graph where one partition U represents actors (e.g., people, organizations) and the other partition V represents a set of issues (e.g., political issues, beliefs). One of the earliest definitions of this problem is given in [1]. An edge between a person and an issue represents the opinion of that person on that issue. This opinion expressed with a sign, as positive or negative, (no edge between a person-issue pair expresses "no opinion"), and, a numerical value representing the strength of the opinion of person.

This work extends previously introduced idea of [2] to be able to partition bipartite graphs into k clusters (k-way partitioning) based on the opinions expressed on the edges. Notice that the clustering should produce sub-bipartite graphs such that people in a sub-bipartite graph should have strong positive opinions on the issues of that sub-bipartite graph, and they should have strong negative opinion towards the issues in other sub-bipartite graphs.

The inputs of k-way partitioning of signed bipartite graph problems are bipartite graph $G = (U \cup V, E)$, label function $\sigma : E \rightarrow R$ and partition count K. Label of an edge can be positive or negative real value. In most cases the range of the mapping is either a small subset of integers of real values. For this modeling, we assume that a positive edge from u to v where $u \in U$ and $v \in V$ means that u supports v, and a negative edge implies that u is against v. The goal of the partitioning problem is to divide the sets U and V into $(U_1, U_2, \ldots, U_k)$ and $(V_1, V_2, \ldots, V_k)$ simultaneously to form disjoint max K clusters $(U_1 \cup V_1, U_2 \cup V_2, \ldots, U_k \cup V_k)$, such that,

1. The sum of the weights of the positive edges within clusters is maximized,
2. The sum of the weights of the positive edges between clusters is minimized,
3. The sum of the weights of the negative edges within clusters is minimized,
4. The sum of the weights of the negative edges between clusters is maximized.

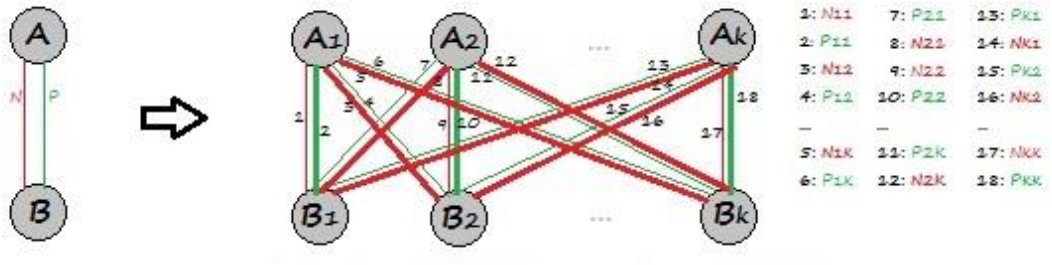

Fig. 1. Partitioning of A into A1, A2, ... Ak and B into B1, B2, ... Bk

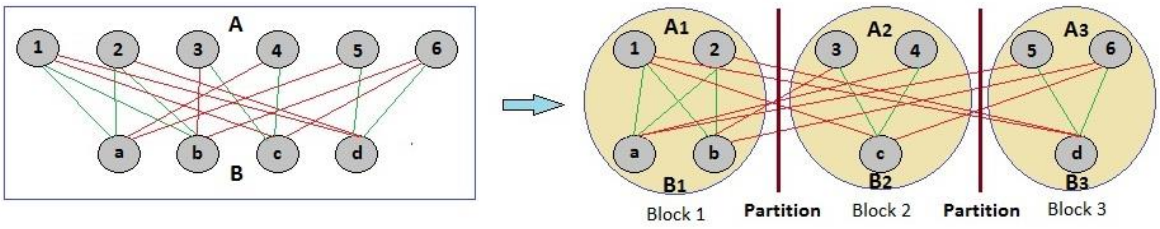

Fig. 2. Nodes are distributed among blocks

In Fig. 1, sets A and B are partitioned into K clusters from 1 to k, where clustering is done simultaneously. Simultaneous (i.e., vertical) partitioning is a little bit confusing; some may think that there are $2K$ clusters in the figure instead of K clusters. To clarify the key point of simultaneous partitioning, we can say that A_j and B_j jointly forms $Block_j$ like shown in Figure 2.

Each line in Fig. **1** represents the sum of the weights of the edges from A_i to B_j, $1 \leq i, j \leq k$ where label N (i.e., red lines) denotes negative and label P (i.e., green lines) denotes positive signs. Note that some of the lines are thicker than the others. Thick lines represent higher values than the thin ones.

In Figure 2, six nodes from A and four nodes from B are distributed into three blocks. In this example, separators (i.e., thick vertical red lines) cut negatively weighted edges between blocks and divide nodes into 3 partitions. For illustration, all nodes are distributed ideally so that edges within blocks are all has positive and edges between blocks are all has negative sign. As one would predict, most of the time, partitioning may not be perfect, meaning that there can be negative edges within clusters and positive edges between clusters.

To our knowledge, no efficient algorithm was presented for k-way partitioning of signed bipartite graph problem before our study, as stated in [2]. This study is the extension of the work in [2] in two folds: One of the extensions is k-way partitioning of bipartite graphs, and the other one is the reduction of the execution time almost by half using a simple observation about the hill-climbing algorithm.

Although clustering is a very well-known problem, there are not many works on bipartite graph clustering. In [6] and [8], two sets of entities (represented by two sets of nodes in the bipartite graph) were clustered. These works were related to document clustering, where one set of entities was set of words, and the other one was set of documents. In these works there was no information on the edges. In [3] and [5], signed arbitrary graphs were considered, but they were not focused to bipartite graphs. In [1] and [7], similar problems were introduced; however, no effective algorithm has been introduced.

Kernighan-Lin (KL) [10] and Fiduccia-Mattheyses (FM) [9] algorithms are two fundamental move-based heuristic algorithms used for graph partitioning from which several algorithms such as [2] have been inspired. While the first one works locally, the second one considers global connectivity. KL algorithm is an efficient heuristic method which finds effective optimal solutions for arbitrary unsigned-weighted graphs. The objective of the algorithm is to divide the graph into subsets of no larger than a given maximum size in order to minimize the sum of the weights on all edges cut. FM is a mincut heuristic algorithm which iteratively partitions networks.

The rest of the paper is organized as follows; mathematical model of the problem given in Section 2. Generic and move-based heuristic are presented in Section 3 and 4 respectively. The results obtained from real and randomly generated datasets are presented at the 4th Section. Finally, the last Section contains the conclusions and the future work.

II. MATHEMATICAL MODEL

Let $U = U_1 \cup U_2 \cup .. U_k$ and $V = V_1 \cup V_2 \cup .. V_k$ be partitioning of the nodes of bipartite graph $G = (U \cup V, E)$, and let

$$\boldsymbol{b_j} = (b_{j1}, b_{j2}, \dots, b_{j|U|})^{\boldsymbol{T}}, \quad \boldsymbol{p_j} = (p_{j1}, p_{j2}, \dots, p_{j|V|})^{\boldsymbol{T}}$$

be indicator vectors for U_j and V_j respectively, $1 \leq j \leq k$. Thus,

$$b_{ju} = \begin{cases} 1, & if\ node\ u \in U_j \\ 0, & otherwise \end{cases},$$

$$p_{jv} = \begin{cases} 1, & if\ node\ v \in V_j \\ 0, & otherwise \end{cases}$$

An example of partitioned bipartite graph is given in Figure 3, where $U = \{1, 2, 3\}, V = \{a, b, c, d\}$ and the vertices of two partitions are connected by weighted edges. In the example, vertices of U and V are divided into three blocks ($K = 3$), where $U_1 = \{1\}, U_2 = \{3\}, U_3 = \{2\}, V_1 = \{b, d\}, \quad V_2 = \{c\}$ and $V_3 = \{a\}$. The indicator vectors for this graph are as following:

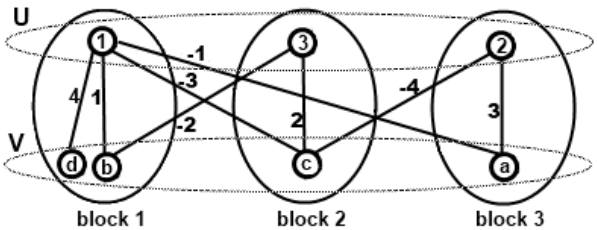

Fig. 3. Illustrative Example

$$\boldsymbol{b_1} = \begin{bmatrix}1\\0\\0\end{bmatrix} \boldsymbol{b_2} = \begin{bmatrix}0\\0\\1\end{bmatrix} \boldsymbol{b_3} = \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \boldsymbol{p_1} = \begin{bmatrix}0\\1\\0\\1\end{bmatrix} \boldsymbol{p_2} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \boldsymbol{p_3} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

Clearly,

$$\sum_{j=1}^{k}\sum_{u=1}^{|U|} b_{ju} = |U| \text{ and } \sum_{u=1}^{|U|} b_{ju} = \left|U_j\right|,\ 1 \le j \le k$$

$$\sum_{j=1}^{k}\sum_{v=1}^{|V|} p_{jv} = |V| \text{ and } \sum_{v=1}^{|V|} p_{jv} = \left|V_j\right|,\ 1 \le j \le k$$

$$\sum_{j=1}^{k} b_{ju} = 1,\ 1 \le u \le |U| \text{ and } \sum_{j=1}^{k} p_{jv} = 1,\ 1 \le v \le |V|.$$

Let $\boldsymbol{A} = (a_{uv})$ represents the adjacency matrix for the bipartite graph $G = (U \cup V, E)$. The sum of all edges in the clusters is given by [5].

$$\sum_{j=1}^{k}\sum_{u\in U_j}\sum_{v\in V_j} a_{uv} = \sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{jv} = \sum_{j=1}^{k} \boldsymbol{b_j}^{\boldsymbol{T}} \boldsymbol{A}\, \boldsymbol{p_j}$$

The mathematical programming formulation can be written as follows:

$$\max L = \sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{jv} - \sum_{\substack{j,n=1\\ j\neq n}}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{nv} \quad (1)$$

Subject to

$$\sum_{j=1}^{k} b_{ju} = 1,\ 1 \le u \le |U| \text{ and}$$
$$\sum_{j=1}^{k} p_{jv} = 1,\ 1 \le v \le |V| \quad (2)$$

The objective function (1) gives the maximized L value as the objective value, by the help of constraints (2). b_{ju}'s and p_{jv}'s expressed in (1) and (2) are the variables of these equations. As seen in the above formulation (1), L value can be obtained by subtracting *the sum of edges across clusters* (let's say "**O"-out**) (i.e., right part) from *the sum of edges within clusters* (let's say "**I"-in**) (i.e., left part). Clearly, we can find **O** by subtracting **I** from *the total sum of edges* **(**let's say **T)**, since **T = I + O**. Thus, the above formulation (**max L= I - O**) can be rewritten as follows (**max L= 2I - T**):

$$\max L = 2\sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{jv} - \sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv}$$

As the right part of the formulation (**T**) is constant, to maximize L we need to maximize the left part of the formulation (let's say L_0);

$$\max L_0 = \sum_{j=1}^{k}\sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} b_{ju} p_{jv}$$

$$\max L_0 = \sum_{j=1}^{k} \boldsymbol{b_j}^{\boldsymbol{T}} \boldsymbol{A}\, \boldsymbol{p_j}$$

$$\max L_0 = \sum_{u=1}^{|U|}\sum_{v=1}^{|V|} a_{uv} \sum_{j=1}^{k} b_{ju} p_{jv} \quad (3)$$

Using values of Figure 3 for equation (5), we can find L_0 as

$$L_0 = (-1)(0+0+0) + (1)(1+0+0) + (-3)(0+0+0) + (4)(1+0+0) + (3)(0+0+1) + (0)(0+0+0) + (-4)(0+0+0) + (0)(0+0+0) + (0)(0+0+0) + (-2)(0+0+0) + (2)(0+1+0) + (0)(0+0+0)$$

$L_0 = 10$. Thus $L = 2L_0 - T = 2x10 - 0 = 20$

It is not possible to convert this problem into linear programming (LP) problem as it is and solve with a LP solver, since there are nonlinear terms (i.e., $b_{ju}p_{jv}$) in (3).

III. GENERIC ALGORITHMS

Two well-known generic heuristics, namely genetic algorithms and simulated annealing, have been applied to solve k-way partitioning of signed bipartite graphs problem.

A genetic algorithm (GA) [11] is a heuristic algorithm, inspired by evolutionary processes of ecological systems, that finds optimal (or near-optimal) solutions to complex optimization problems. In GAs, possible solutions to the problem are coded in chromosomes. A "chromosome" (or "individual") can be designed as a string, binary digit or other symbols that corresponds to a solution of the problem at hand. The fitness function of GA analyzes "genes" in the chromosomes, makes some qualitative assessment and provides a meaningful and comparable fitness value for that solution. Basically, thanks to the fitness function, candidate solutions pass to the next generation of solutions by discarding solutions with a "poor" fitness and accepting any with a "good" fitness value.

A typical GA works as follows:

- Construct an initial population of chromosomes by generating randomly attempted solutions to a problem
- Do the following until a satisfactory fitness level has been reached or run out of time:
  - Evaluate each fitness of the solutions
  - Keep a subset of these solutions (using different heuristics)
  - Use these solutions to generate a new population by using the crossover and mutation operators.

There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation [11].

There are two basic reproduction strategies, which are crossover and mutation. Crossover is a reproduction technique to generate two offspring from two selected parents. The chromosomes of the two parents are recombined according to some techniques to form offspring. Mutation is a reproduction mechanism, which generates new offspring from single parent. Each binary digit of the chromosome is subject to inversion under a given probability (most of the time small).

Simulated annealing (SA) is a generic probabilistic meta-algorithm used to find an approximate solution to global optimization problems, which was introduced by Kirkpatrick [12]. It is inspired by annealing in metallurgy which is a technique of controlled cooling of material to reduce defects.

A typical SA algorithm works as follows:

- Initialize temperature T, epsilon ε, alpha α
- Generate a random initial solution as current solution S_c
- Do the following till $T < \varepsilon$ or run out of time
- While stopping criteria not met do

- Find the neighbor of the current solution S_n
  - Compute $\Delta = f(S_n) - f(S_c)$ (i.e., f: fitness function)
  - Randomly generate a real number r from 0 to 1
  - If $\Delta < 0$ or $(e^{-\Delta/T} > r)$ then $S_c = S_n$
- Reduce T by multiplying with α

SA starts with some solution that is totally random, and changes it to another solution that is similar to the previous one. Newly generated solutions are generally chosen randomly, though more sophisticated methods can be applied. If this solution is a better solution, it will replace the current solution. If it is a worse one, it may be chosen to replace the current solution with a probability that depends on the temperature (i.e., cooling process, T decreases with time) and the distance Δ (i.e., difference between new (worse) solution and the old one) parameters. As the algorithm progresses, the temperature parameter decreases by multiplying α, giving worse solutions a lesser chance of replacing the current solution.

IV. MOVE-BASED HEURISTIC

Move based heuristic (MBH) is a typical hill-climbing algorithm. For k-partitioning of signed bipartite graphs, move-based heuristics work as follows:

- Nodes are randomly placed into the blocks at the beginning.
- Then, through iterations, the node with the highest gain value is selected and moved to the block that maximizes the gain. After each move, that node is locked. Until all the nodes are locked, the iteration continues.
- After all nodes are locked, the change in the result value L (the objective value which is defined in Section 2) is checked. If the completed iteration increases the value of L, a new iteration starts by configuring the initial state with the best state found in the previous iteration. Otherwise, iterations end and best solution is returned.

In order to measure the quality of clustering of bipartite graphs, as in [2], we have defined a gain function that recalculates the gains of all nodes as the vertices are placed into blocks. The gain calculation is done as follows:

- If both vertices are in the same block, and the edge between them is positively weighted, then moving either one will reduce the gain.
- Similarly, if the vertices are in different blocks and the edge between them is negatively weighted, then putting them into the same block will also reduce the gain.
- If the vertices are in the same block, but the edge between them is negative weighted, then, moving one of them to a different block will increase the gain.
- Finally, if two vertices are in different blocks, but the edge between them are positively weighted, then moving them into the same block will increase the gain.

Fig. 4 depicts the gain computation on a simple example. As seen from the figure, it is clear that the movement of the selected node to the 3rd block gives the largest increase for the value of L.

In MBH Algorithm, which is given below, L represents the result of the objective function, and K is the number of clusters, which is given as an input. Hill-climbing algorithms usually improve the result, but it is always possible to strike at local maximum. In order to avoid this problem we repeat the process several times. Therefore, we use one more parameter, R, to randomly start the process more than once in order to be able to avoid local minimum.

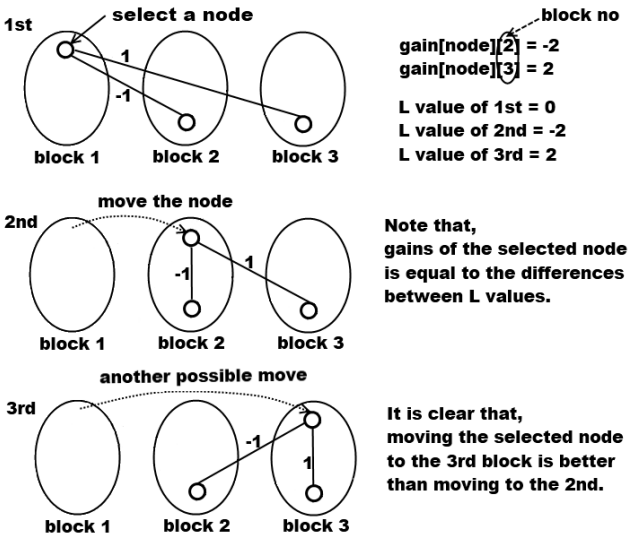

Fig. 4. Gain Computation

Move-Based Heuristic (MBH) Algorithm

Input : Graph: $G = (U \cup V, E)$
Number of clusters: K, Number of iterations: R.
Output: Maximal L value and partitions of nodes.

```
L ← −∞
while R > 0
  Initially, place each node into block 1 to K randomly
  L ← Current RESULT, L' ← L
  do
    L ← L'
    Compute gains of all nodes (Refer to Figure 4)
    do
      nod1 ← select the unlocked node with max gain
      blck1 ← select the best block for nod1
      place the nod1 into blck1
      update gains of nod1's neighbors
      L'' ← New RESULT
      lock nod1
    until all nodes are locked
  while L' < L''
  L' ← L''
end while
print L
```

While we were analyzing the outputs of MBH, we have observed that there have been some unnecessary moves in the process of MBH algorithm. These redundant moves become clear by displaying a sample run of the algorithm as in Figure 5. This figure has been obtained from questionnaire dataset (which will be explained in the following section) experiment for K=9. In this experiment, the R value is 3, and therefore the MBH algorithm has been applied 3 times with 3 random initial solutions.

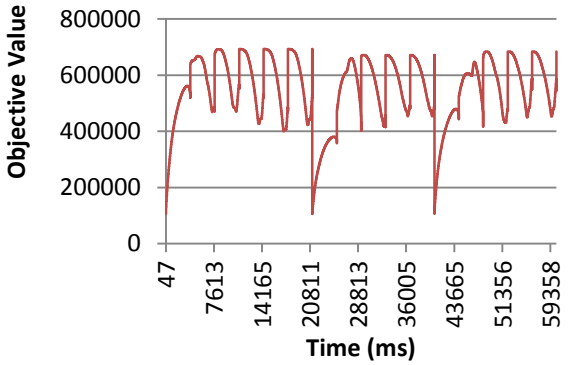

Fig. 5. Questionnaire Experiment (K=9), MBH Stats

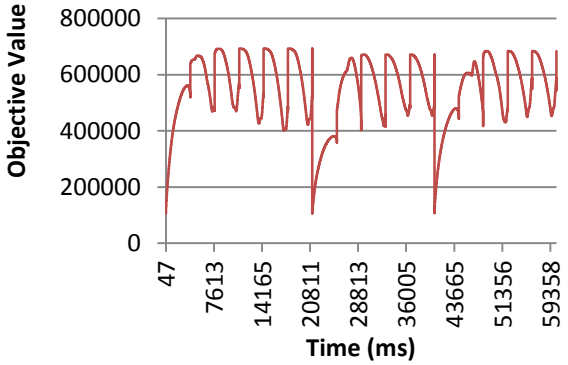

Fig**. 5** Figure 5, we see that objective values are increasing and decreasing in a systematic way. Decreasing parts are not necessary for our solution, since we have been trying to find the global maximum. Furthermore, as the chart presents, calculation of the descending values is really time consuming. Therefore, we wanted to remove the declining parts from the execution (chart) to reduce the total execution time. This is done by detecting when the values start to fall below the local maxima. In this way, we have managed to cut the unnecessary parts of the computation as shown in Figure 6. Note that the elapsed time has fallen below by half (approximately 57000 to 20000 ms).

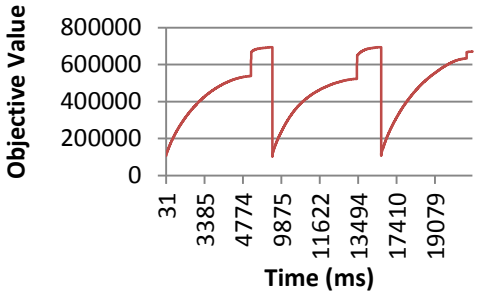

Fig. 6. Questionnaire Experiment (K=9), Opt-MBH Stats

The standard version of MBH locks all of the nodes (traverses all of the nodes) in each iteration even after a local maximum has been reached, which causes the decline of the total gain. In order to cut this wasted time we have modified the algorithm just to detect whether the local maximum has been reached. This is done simply by comparing the current objective value, L'', with the one that is already been obtained, L'. If the trend of the current value is a decrease, then, the rest of the traversing the nodes has been abandoned. Below is the extension on MBH Algorithm for this addition. This addition should be made to the end of the inner loop between the lines 8-15 of MBH Algorithm**.**

Optimized MBH Algorithm

At each iteration, we set the $declineCount \leftarrow 0$

…

1: **if** $L''(current\ result) < L'(local\ maximum)$ **then**
2: $declineCount \mathrel{+}= 1$
3: **else if** $L'' \geq L'$ **then**
4: ; // Do Nothing
5: **else**
6: **if** $declineCount > 0$ **then** $declineCount \mathrel{-}= 1$
7: **else** $declineCount = 0$
8: **end if**
9: **if** $declineCount > \alpha$ **then**
10: **exit loop**
11: **end if**

…

V. Experimental Results

The methods expressed in this work are all implemented in C++, using the Visual Studio 2005 development environment. Tests are done on a commodity computer having Windows 7 x86 OS, Intel Core 2 Duo 2.00 GHz CPU, and 3 GB RAM.

In this work, the algorithms have been tested with both real datasets and with randomly generated data. We have generated random datasets with dimensions 10x10, 20x20, 40x40 and 80x80. For each size, sparse (less than 20% of the edges have non-zero values) vs. dense (more than 55% of the edges have non-zero values), signed (i.e., -1, 0, 1) vs. ranged (-10, …, 0, …, 10) versions are also generated.

In Table 1, first 16 rows correspond to randomly generated datasets. 17th and 18th rows show the characteristics of the real

world data sets. 17th dataset contains questionnaire with 48 questions, which are applied to 7572 people. The questions were ranked between -5 and +5. 18th dataset corresponds to US Congress (SENATE) dataset which is published publicly in www.govrack.us. From this site, we have used the roll call votes for the 111th US Congress Senate that covers the years 2009-2010. The 111th Senate data contains information about 108 senators and their votes on 696 bills. We have constructed a signed bipartite graph as in [3] based on the votes of the senators on the bills.

In our experiments, we have run each algorithm 10 times on 18 datasets. In these executions, the parameters used in algorithms and their values are as follows:

- **GA:** Iteration Count: 50, Population Size: 500, Elitist Selection: 5%, Roulette-Wheel Selection: 90%, Random Generation: 5%, Uniform Crossover Rate: 0.6, Mutation Rate for Each Chromosome: 0,0001
- **SA:** Alpha: 0.99999, Temperature: 400.0, Epsilon: 0.001
- **MBH:** R (Randomly Restart Number): 25

Table 1. Characteristics of Datasets

	K=2				K=3				K=9			
	GA	**SA**	**MBH**	**OptMBH**	**GA**	**SA**	**MBH**	**OptMBH**	**GA**	**SA**	**MBH**	**OptMBH**
1	1281	962	2	0	1249	1229	2	6	1264	1524	5	5
2	1403	713	3	3	1316	853	0	5	1243	1198	2	6
3	1262	961	0	5	1094	1131	0	2	1042	1366	3	3
4	1150	805	3	0	1069	1022	3	24	1090	1295	8	0
5	2320	858	6	5	2225	1048	8	14	2198	1552	19	13
6	2846	749	6	3	2814	989	8	2	2587	1535	17	8
7	1526	1134	8	2	1546	1449	5	3	1520	2042	11	5
8	1657	881	6	3	1724	1142	6	5	1686	1668	11	6
9	6289	984	17	8	6185	1452	28	14	5956	2315	69	30
10	7020	842	22	8	6922	1203	30	14	6401	2133	66	38
11	3268	1206	20	8	3295	1647	27	13	3159	2702	47	28
12	4523	975	25	10	4116	1317	30	14	3849	2288	50	24
13	20458	1301	98	39	20132	1863	133	58	19453	3537	289	155
14	20892	1103	119	41	20171	1691	148	58	20263	3309	348	148
15	8802	1306	97	25	8574	1972	19	59	8590	3697	265	126
16	9201	1315	106	20	8917	1952	128	55	9343	3906	259	114
17	1115816	12321	41116	11023	1105766	14602	57415	19580	1090272	23489	143526	54711
18	271807	2136	878	289	271807	3451	1033	401	269835	7193	2002	948

A. Senator Experiment (Dataset Number 18)

Opt-MBH algorithm had clustered 108 senators and 696 bills into 3 clusters. That is, the gain has increased when the cluster size is increased from 2 to 3, but, there were no increase afterwards. Figure 7 shows the 2-way and the 3-way partitioning of the bipartite graphs. In these figures columns correspond to the bills and the rows correspond to the senators. The colors (green and red) correspond to the votes of senators on the bills (favor or against). Notice that blue lines have been inserted into these figures in order to make clusters more visible.

The US Senate has 2-party system (with 2 independents, mostly inclined to Democrats), with 100 members. During the 2 years of 111th Senate, the numbers of the members of both parties have changed due to different circumstances. Therefore, the total number of senators has also increased to 108. In two clustering, the clusters were roughly representing the party lines. During 111th Senate, the number of Republicans was 39 in its minimum level, and one of the clusters our system has obtained exactly had that many senators. Of course, there are several Senators voting quite independently from their respective parties. However, even in 3-cluster structure, it has been observed that senators were not clustered forming a 3rd group. Only, a small number of bills have been discovered, which are mostly been rejected by the senators of both parties. The structures of 2 and 3 clusters are as follows:

- in 2-way, 39 senators and 257 bills formed one cluster and 69 senators and 439 bills formed the other one,
- in 3-way, the number of senators were the same for the first two clusters, and the third cluster had 0 senators, however, 7 bills from the first cluster, and 4 bills from the second cluster had been moved into the third one making it with 0 senators and 11 bills.

B. Questionnaire Dataset (Dataset Number 17

In the experiment corresponding to the 17th dataset since the data size was very large and dimensions were disproportional, we could not print the results in a figure similar to the one that we have done for the Senate experiment. The clusters are shown in Table 5.

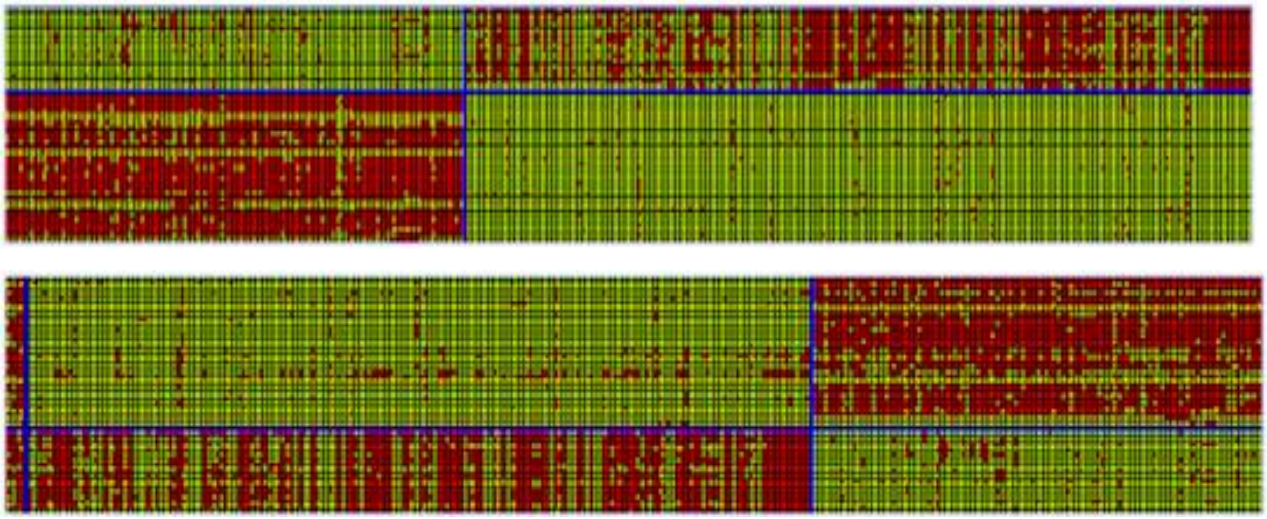

Fig. 7. Partitioning of Senators and Bills with K = 2 (top) and K = 3 (bottom)

Table 2. Clusters for Questionnaire Experiment (P: # of persons, Q: # of questions in a cluster)

Clusters	K=2		K=3		K=4		K=5		K=6		K=7		K=8		K=9	
	P	Q	P	Q	P	Q	P	Q	P	Q	P	Q	P	Q	P	Q
1	7458	33	6320	34	5864	33	6083	34	5941	33	6088	34	6080	34	6074	34
2	114	15	1252	0	981	3	637	1	883	3	611	1	572	1	531	1
3	NA	NA	0	14	727	0	447	0	463	0	417	1	423	1	498	1
4	NA	NA	NA	NA	0	12	405	1	203	1	287	0	162	0	198	0
5	NA	NA	NA	NA	NA	NA	0	12	82	2	130	1	126	1	114	1
6	NA	NA	NA	NA	NA	NA	NA	NA	0	9	39	0	123	0	64	0
7	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0	11	86	1	52	0
8	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0	10	41	2
9	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0	9
10	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
RESULT	642291		^ 686261		^ 693867		^ 694499		^ 694551		^ 694759		^ 694843		^ 694851	

Table 2. Comparison of All Results (K=2, K=3, K=9)

	K=2				K=3				K=9			
	GA	SA	MBH	OptMBH	GA	SA	MBH	OptMBH	GA	SA	MBH	OptMBH
1	35	35	35	35	43	43	43	43	42	43	43	43
2	177	171	177	177	205	199	205	205	204	200	205	205
3	14	14	14	14	16	16	16	16	16	16	16	16
4	120	114	120	120	122	120	122	122	122	121	122	122
5	96	96	96	96	107	108	108	108	105	111	112	112
6	591	578	591	591	712	706	719	719	691	718	723	723
7	49	49	49	49	57	58	59	59	61	62	63	63
8	261	253	261	261	302	295	307	307	320	315	325	325
9	273	275	276	275	307	315	316	316	300	326	326	324
10	1652	1678	1683	1684	2044	2108	2115	2111	2178	2275	2274	2274
11	143	146	147	147	174	179	178	179	182	190	188	188
12	940	932	944	944	1105	1116	1119	1131	1134	1184	1178	1177
13	798	809	815	815	895	945	936	939	896	983	973	967
14	4993	5092	5108	5106	5817	6073	6041	6026	5898	6463	6382	6356
15	438	440	444	444	514	535	531	530	507	561	551	555
16	2647	2680	2697	2692	3068	3235	3202	3223	3085	3360	3276	3259
17	642282	642291	64221	642291	685281	678842	686261	686261	682912	682900	695053	694831
18	46066	46066	46066	46066	46066	46422	46422	46422	46376	46422	46422	46422

Table 3. Time to Find the Best Solutions (K=2, K=3, K=9)

	K=2				K=3				K=9			
	GA	**SA**	**MBH**	**OptMBH**	**GA**	**SA**	**MBH**	**OptMBH**	**GA**	**SA**	**MBH**	**OptMBH**
1	212	863	0	0	379	1114	0	0	796	1402	0	3
2	231	587	0	2	309	710	0	0	608	941	0	3
3	187	860	0	3	239	1025	0	2	337	1215	0	2
4	229	662	0	0	231	780	2	2	367	932	0	0
5	502	752	0	0	1006	934	2	9	1479	1357	10	3
6	643	591	0	0	1386	788	6	0	1797	1201	5	3
7	372	1019	0	0	668	1309	2	2	1021	1821	5	0
8	440	728	2	0	699	977	2	2	1017	1342	0	2
9	2573	850	2	2	2977	1260	3	13	4582	2075	22	17
10	2883	611	5	2	3728	869	9	10	5438	1643	41	19
11	1470	1073	6	3	1966	1481	6	8	2427	2390	22	6
12	1623	774	6	6	2336	1027	14	6	3287	1811	33	17
13	8707	1056	36	14	14140	1557	55	31	17056	2944	147	87
14	11179	755	50	14	16875	1178	64	32	18411	2298	152	58
15	4672	1089	20	13	6254	1685	39	33	7850	3235	128	81
16	5001	1017	48	5	6134	1482	80	40	8777	2847	136	84
17	923707	7454	746	437	1071273	10518	4855	1407	1079663	17068	61346	32776
18	33924	1406	14	11	33924	2321	16	15	191094	4607	39	34

Table 4. Execution Time of Algorithms (K=2, K=3)

	K=2				K=3				K=9			
	GA	**SA**	**MBH**	**OptMBH**	**GA**	**SA**	**MBH**	**OptMBH**	**GA**	**SA**	**MBH**	**OptMBH**
1	1281	962	2	0	1249	1229	2	6	1264	1524	5	5
2	1403	713	3	3	1316	853	0	5	1243	1198	2	6
3	1262	961	0	5	1094	1131	0	2	1042	1366	3	3
4	1150	805	3	0	1069	1022	3	24	1090	1295	8	0
5	2320	858	6	5	2225	1048	8	14	2198	1552	19	13
6	2846	749	6	3	2814	989	8	2	2587	1535	17	8
7	1526	1134	8	2	1546	1449	5	3	1520	2042	11	5
8	1657	881	6	3	1724	1142	6	5	1686	1668	11	6
9	6289	984	17	8	6185	1452	28	14	5956	2315	69	30
10	7020	842	22	8	6922	1203	30	14	6401	2133	66	38
11	3268	1206	20	8	3295	1647	27	13	3159	2702	47	28
12	4523	975	25	10	4116	1317	30	14	3849	2288	50	24
13	20458	1301	98	39	20132	1863	133	58	19453	3537	289	155
14	20892	1103	119	41	20171	1691	148	58	20263	3309	348	148
15	8802	1306	97	25	8574	1972	19	59	8590	3697	265	126
16	9201	1315	106	20	8917	1952	128	55	9343	3906	259	114
17	1115816	12321	41116	11023	1105766	14602	57415	19580	1090272	23489	143526	54711
18	271807	2136	878	289	271807	3451	1033	401	269835	7193	2002	948

Opt-MBH algorithm had partitioned this weighted bipartite graph into 9 clusters, as the best clustering structure. We tried all the cluster sizes from 2 to 9. We have discovered that the objective value increased for each cluster size as it can be seen from Figure 8. Similar to the Senate experiment, some clusters had only vertices from one of the partitions of the bipartite graphs.

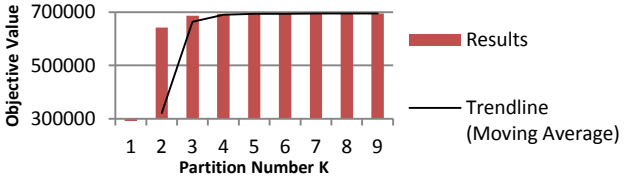

Fig. 8. Results of Questionnaire Experiment with Moving Average Trendline

C. Comparing MBH and Optimized-MBH

In the final experiment, MBH and Opt-MBH have been compared with inputs of not only the same datasets but also the same randomly generated initial solutions. "17th dataset with K=9" and "18th dataset with K=8" results are displayed in the figures below. As can be seen from the minimum values in these graphics, R=3 has been used in these experiments. Figure 9 contains two graphics which emphasize total execution time of the algorithms and the cut points (i.e., vertical lines) in Opt-MBH for the questionnaire dataset. In Figures 10 and 11, senator dataset has been used and approximate saved times are also shown.

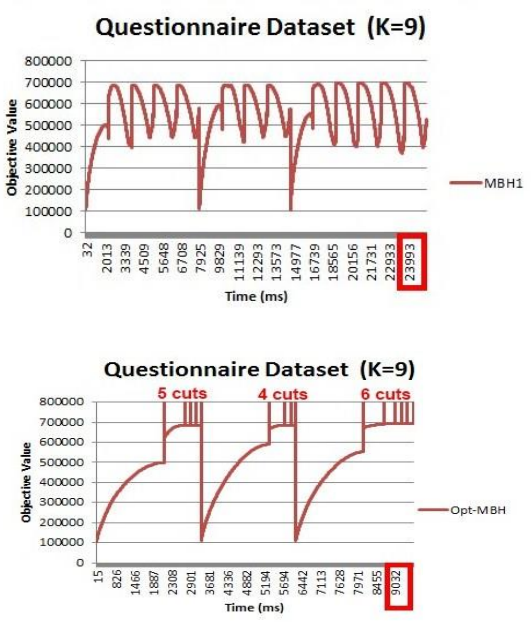

Fig. 9. Execution times of Questionnaire Dataset with MBH and Opt-MBH

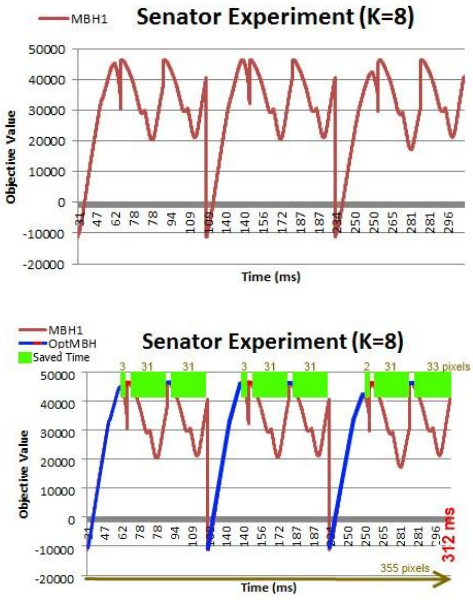

Fig. 10. Execution times of Senator Dataset with MBH

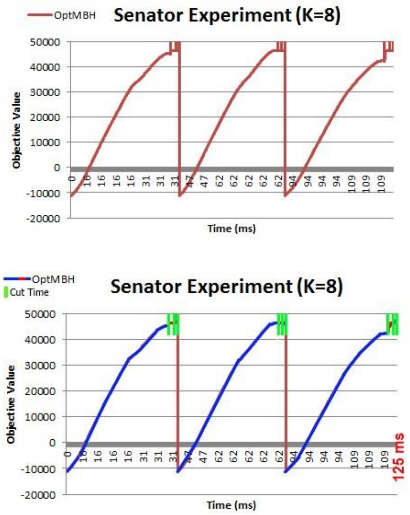

Fig. 111. Execution times of Senator Dataset with Opt-MBH

VI. Conclusion and Future Work

This work extends previous work on 2-way clustering of signed bipartite graphs [2] to k-way clustering of signed or weighted bipartite graphs. This problem appears in social networks in many different forms.

In this study, for *k-way partitioning of the signed bipartite graphs* problem, mathematical methods, generic algorithms and various move-based heuristics have been developed. We have shown that our approaches are quite effective through experiments on not only randomly generated data, but also real world data. Our experiments show that optimized move-based heuristic algorithm produces the best result and has the best execution time.

REFERENCES

[1] Andrej, M., and Doreian, P. Partitioning signed two-mode networks. Journal of Mathematical Sociology 33, pages 196–221, 2009.

[2] Banerjee, S., Sarkar, K., Gokalp, S., Sen, A., and Davulcu, H. Partitioning Signed Bipartite Graphs for Classification of Individuals and Organizations. Social Computing, Behavioral - Cultural Modeling and Prediction Lecture Notes in Computer Science Volume 7227, 2012, pp 196-204

[3] Bansal, N., Blum, A., and Chawla, S. Correlation clustering. In Machine Learning, pp. 238–247, 2002.

[4] Charikar, M., Guruswami, V., and Wirth, A. Clustering with qualitative information. Journal of Computer and System Sciences. Volume 71, Issue 3, October 2005, Pages 360–383.

[5] Sen, A., Deng, H., and Guha, S. On a graph partition problem with application to VLSI layout. Inf. Process. Lett. 43(2), 87–94, 1992.

[6] Dhillon, I.S. Co-clustering documents and word using bipartite spectral graph partitioning. In Proceedings of the KDD, 2001, pp. 269-274.

[7] Zaslavsky, T. Frustration vs. clusterability in two-mode signed networks (signed bipartite graphs), Unpublished manuscript (http://www.math.binghamton.edu/zaslav/Tpapers/fvc.pdf).

[8] Zha, H., He, X., Ding, C., Simon, H., and Gu, M. Bipartite graph partitioning and data clustering. In Proceedings of the 10th International Conference on Information and Knowledge Management, pp. 25–32, 2001. ACM.

[9] Fiduccia, C.M., and Mattheyses, R.M. A Linear-Time Heuristic for Improving Network Partitions. In Design Automation, pp. 175-181, 1982.

[10] Kernighan, B.W., and Lin, S. An Efficient Heuristic Procedure for Partitioning Graphs. In Bell System Technical, vol.49, pp. 291-307, 1970.

[11] Holland, J. H. Adaption in Natural and Artificial Systems. University of Michigan Press, 1975.

[12] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. Science 220, 671-680, 1983.