Finding the Maximum Balanced Vertex Set on Complete Graphs

Chun-Hsiang Bai and Bang Ye Wu*
National Chung Cheng University, ChiaYi, Taiwan 621, R.O.C.

Abstract

A signed graph is a simple graph in which each edge is labeled by a sign either + or -. A signed graph is balanced if every cycle has even numbers of negative edges. In this paper, we study the problem how to find a maximum vertex subset of a complete signed graph such that the induced subgraph is balanced. We show that the problem can be reduced to a series of vertex cover problems and therefore admits a 2-approximation and a fixed-parameter algorithms. We also tested the practical performances of these algorithms on random graphs. Our algorithm can find optimal solution within ten seconds with 100 vertices which is much better than a trivial algorithm.

1 Introduction

A *signed graph* is a simple graph $G = (V, E, \sigma)$ in which $\sigma : E \mapsto \{1, -1\}$. That is, each edge is labeled by a sign either "+" or "-". A signed graph is *balanced* if every cycle has even numbers of negative edges. In this paper, we study the following optimization problem.

> Problem: Maximum Balanced Vertex Set (MBVS) problem
> Instance: A complete signed graph G.
> Goal: Find a maximum cardinality vertex subset $U \subseteq V$ such that subgraph of G induced by U is balanced.

The dual problem, named coMBVS, is defined similarly but the objective is to minimize the number of vertices to delete. In this paper, we shall only study the case that the input is a complete graph.

The concept of balanced signed graphs was introduced by Harary [15] for the analysis of social networks, in which a positive edge represents a positive relation (such as "like") and a negative edge is for a negative relation (such as "dislike"). Signed graphs also find other applications, for example, in mathematical analysis of large-scale biological networks [10] and in statistical physics and integrated circuit fabrication techniques [9, 25]. A bibliography of signed graphs can be found in [25].

Instead of vertex deletion, most of the previous results about finding a balanced subgraph of a signed graph focus on edge deletion, or equivalently minimum changes of edge signs. Such a problem is usually known by the name Balanced Subgraph, which is to find a balanced subgraph with maximum number of edges. Balanced Subgraph is a generalization of the NP-hard Maximum Cut problem in graphs. In the literatures, there are several results of Balanced Subgraph in the aspect of approximation algorithms and fixed-parameter algorithms [4, 10, 17]. For more details, we refer to [17].

Both the MBVS and the Balanced Subgraph problems are closely related to a general problem named "Cluster Editing" [18]. The basic problem is, given an undirected graph G, to find a minimum number of editing operations that transform G into a collection of disjoint complete subgraphs, a *cluster graph*. Each of these disjoint complete subgraphs is called a *cluster*. The editing operations may include adding edges, deleting edges, and deleting vertices. The difference between the clustering problem and the problem studied here is that the number of clusters is specified or not. The Balanced Subgraph problem is also known by the name "correlation clustering" in the field of *graph clustering*. The bridge between a subgraph problem to a clustering problem is due to Harary's theorem: a signed graph is balanced if and only if it can be partitioned into two clusters (vertex subsets) such that every positive edge is within one of the clusters and every negative edge crosses the two clusters. In the general correlation clustering problem, each item is represented by a vertex, a positive or negative edge means that the

*Corresponding author, E-mail:bangye@ccu.edu.tw

two vertices are similar or dissimilar, respectively. Given a signed graph (1 for similar and -1 for dissimilar), the objective is to produce a partitioning into clusters that places similar objects in the same cluster and dissimilar objects in different clusters, to the extent possible. In general, the number of clusters is not specified.

Shamir et al. [22] studied the computational complexity of the Cluster Editing problem and showed the NP-hardness, even for the case that the number of clusters is any fixed $k \geq 2$. There are several other results about approximation of the above and related problems in [1, 2, 3, 6, 13, 20]. For more details, see [13]. There are some results from the field of fixed-parameter algorithms [11, 12, 21]. A problem is called fixed-parameter tractable with respect to a parameter k if an instance of size n can be solved in $f(k) * n^{O(1)}$ time, where f is an arbitrary function depending only on k.

Whereas the Cluster Editing problem attracts intensive research, the vertex deletion version CVD (short for Cluster Vertex Deletion) had been widely neglected until the work by Hüffner et al [18]. For this version, the aim is to find a vertex set of minimum weight such that its deletion transforms a given graph into a disjoint of cliques. In [18], some fixed-parameter results are given for both cases that the number of clusters is pre-specified or not. We use d-CVD for the case that the number of clusters is restricted to d. Let G be a complete signed graph which is an instance of the coMBVS problem. Let G' be the unsigned graph obtained from G by removing all negative edges. We can see that any vertex-induced balanced subgraph of G corresponds to a vertex-induced subgraph of G' which consists of two disjoint cliques. Thus the coMBVS problem is equivalent to the 2-CVD problem. In [18], three fixed-parameter algorithms for weighted 2-CVD were developed whose running times are $O(2^k \cdot k^9 + nm)$, $O(1.40^k \cdot k^{3d} + nm)$, and $O(1.84^{k+d} + nm)$, in which k is the number of vertices to delete. They also show that the unweighted case can be solved in $O(1.28^k \cdot k^{2d} + nm)$ time which uses an $O(1.28^k + kn)$-time algorithm for unweighted vertex cover.

In this paper, we study the coMVBS problem, i.e., the 2-CVD problem, in the special case that the input is a complete graph. We show that the problem can be reduced to a series of vertex-cover problem and thus admits a 2-approximation algorithm. Also, by employing the fixed-parameter algorithm for vertex cover, we have an $O(1.28^k \cdot n + n^3)$-time algorithm which is asymptotically better than the previous result if $n < k^4$. Furthermore, we design a branch-and-bound algorithm and test the practical performances of these algorithms, including their running times and solution qualities. Our algorithm can find optimal solution within ten seconds with 100 vertices which is much better than a trivial algorithm.

The paper is organized as follows. In Section 2, we give some notation and definitions. In Section 3, we derive some properties which are helpful for our algorithms in Section 4. The experiment results are given in Section 5. Finally, some concluding remarks are given in Section 6.

2 Preliminaries

We shall use the following notation and terms in graph theory. For a graph G, $V(G)$ and $E(G)$ denote the vertex and edge sets, respectively. Two vertices u and v are *neighbors* of each other if $(u, v) \in E$. For a vertex subset U, the subgraph of G induced by U is denoted by $G[U]$. The degree of a vertex v in a graph G, denoted by $d(G, v)$ is the number of its neighbors in G. When there is no confusion, we shall simply use $d(v)$. A clique is complete subgraph. A k-clique is a clique of k vertices. A k-cycle is a cycle of k vertices. A 3-clique is also called as a *triangle* which is also a 3-cycle. Let $\triangle uvw$ denote a triangle consisting of three vertices u, v and w. A cycle is *positive*, or *balanced*, if it contains even numbers of negative edges. In the remaining paragraphs, the input graph is always $G = (V, E)$ and we use $n = |V|$ and $m = |E|$. And we denote $V \cup \{v\}$ by $V + v$. For a graph $G = (V, E)$, a vertex subset $S \subseteq V$ is a *vertex cover* if every edge has at least one endpoint in S.

A *2-clustering* of a singed graph $G = (V, E, \sigma)$ is bipartition (V_0, V_1) of V. A perfect 2-clustering is a 2-clustering such that $\sigma(u, v) = -1$ if and only if u and v are in the different subsets.

A *characteristic vector* of a simple cycle C is a vector in GF[2]^m, $m = |E|$, which has 1's in components corresponding to edges of C and 0's in the remaining components. For two cycles C_1 and C_2, define $C_1 \oplus C_2$ be the subgraph whose characteristic vector is the sum of the ones of C_1 and C_2 (the addition is in GF[2]). Note that $C_1 \oplus C_2$ may be not a cycle.

The next theorem is due to Harary.

Theorem 1: A singed graph is balanced if and only if there is a perfect 2-clustering [15].

In a complete signed graph, each k-cycle for $k > 3$ can be represented by the sum of the characteristic vectors of several triangles. So, if all triangles are positive, then all k-cycles are positive. The next theorem follows from the following observation: If $C = C_1 \oplus C_2 \oplus \ldots \oplus C_k$ and all C_i are positive, then C is positive too.

Theorem 2: A complete signed graph is balanced if and only if all triangles are positive [23].

3 Some properties

In this section, we develop some properties which are helpful for our algorithms. A vertex subset of a signed graph is balanced if the induced subgraph is balanced.

Lemma 3 : Let $G = (V, E, \sigma)$ be a complete signed graph and $V_1 \subset V$ be a balanced subset. If $V_1 \cup \{u_1, u_2\}$ is balanced for any two vertices $u_1, u_2 \in V - V_1$, then V is balanced.

Proof: Since V_1 is balanced and $V_1 \cup \{u_1, u_2\}$ is balanced for any two vertices $u_1, u_2 \in V - V_1$, by Theorem 2, it is sufficient to show that all triangles in $U = V - V_1$ are balanced. We consider a 4-clique formed by $\{u_1, u_2, u_3\} \subset V - V_1$ and $v \in V_1$. By $\triangle u_1u_2u_3 = \triangle vu_1u_2 \oplus \triangle vu_1u_3 \oplus \triangle vu_2u_3$, if $\triangle vu_1u_2$, $\triangle vu_1u_3$ and $\triangle vu_2u_3$ are all balanced, then $\triangle u_1u_2u_3$ is balanced. □

Let V_1 be balanced, and $U = V - V_1$. We want to find a maximum cardinality subset U' of U such that $U' \cup V_1$ is balanced. First, for any $u \in U$, if $V_1 + u$ is unbalanced, u should be deleted. Let H be an auxiliary graph defined by: (1) $V(H) = U$; and (2)for $u_1, u_2 \in U$, there is an edge (u_1, u_2) if and only if $V_1 \cup \{u_1, u_2\}$ is unbalanced, i.e., there exists a vertex $v \in V_1$ such that $\triangle vu_1u_2$ is not balanced. By Lemma 3, we can have the following result.

Corollary 4: If $E(H) = \emptyset$, $U \cup V_1$ is balanced.

We define MBVS(G) as an optimal solution for input graph G and MBVS(G,V_1) denote an optimal containing V_1.

Lemma 5 : If M is a matching in H, then MBVS$(G, V_1) \leq |V_1| + |U| - |M|$.

Proof: According to pigeonhole principle, for any $S \subseteq V$, if $|S| \geq |V_1| + |U| - |M|$, it must contain an edge from M. By the definition of H, it would not be balanced. □

Lemma 6: If S is an independent set in H, then $V_1 \cup S$ is balanced.

Proof: For any two vertices $u, v \in S$, there is no edge $(u, v) \in H$, i,e., $E(S) = \emptyset$. By Corollary 4, $V_1 \cup S$ is balanced. □

Let $V = \{v_i | 1 \leq i \leq n\}$ and H_i be the graph defined above with $V_1 = \{v_i\}$ and $U_i = \{v_j | i < j \leq n\}$. Let VC_i denote a minimum vertex cover in H_i.

Lemma 7: The $U_i - VC_i$ is a maximum balanced vertex subset of $G[U_i]$ containing v_i, i.e., MBVS$(G[U_i], \{v_i\})$.

Proof: It is well-known that finding minimum vertex cover is equivalent to finding maximum independent set. Since VC_i is a minimum vertex cover, $U_i - \{v_i\} - VC_i$ is an maximum independent set. Then the result follows from Lemma 6. □

Theorem 8: The MBVS problem can be solved by solving a series of $n - 1$ vertex cover problems.

Since the vertex cover problem can be solved by a fixed-parameter algorithm [8], by Theorem 8, we can have the next corollary.

Corollary 9 : The coMBVS problem can be solved in $O(1.28^k \cdot n + n^3)$ times, in which k is the number of vertices to delete.

Proof: By Theorem 8, the coMBVS problem can be solved by computing a minimum vertex cover for each H_i. Since each H_i can be constructed in $O(n^2)$ time and each of the vertex cover problems can be computed in $O(1.28^k + kn)$ time [8], the total time complexity for solving the coMBVS problem is $O(1.28^k \cdot n + n^3)$. □

Similarly, by the 2-approximation algorithm of vertex cover [24], we have the next result.

Corollary 10 : The coMBVS problem can be 2-approximated in $O(n^3)$ time.

4 Algorithms

In the previous section, we have shown some theoretical results in the aspect of fixed-parameter and approximation. We have further studied the practical performances of these algorithms by experiments. In order to compare the performances, we also designed a branch-and-bound algorithm.

4.1 Exact algorithm

We now show a branch-and-bound algorithm, and the experiment results are in the next section.

As described in the previous section, let $V = \{v_i | 1 \leq i \leq n\}$ and, for $1 \leq i < n$, H_i be the graph defined by

- $V(H_i) = \{v_j | i < j \leq n\}$; and
- $E(H_i) = \{(v_p, v_q) | \triangle v_i v_p v_q$ is not balanced and $i < p < q\}$.

Our goal is to find a minimum vertex cover for each H_i. The algorithm is given in Algorithm 1 which employing a branch-and-bound procedure for solving the vertex cover problem. Let N_v denote the set of all neighbors of v and the reduction rules will be explained later.

Algorithm 1 B&B

Input: A graph G with $V(G) = \{v_i | 1 \leq i \leq n\}$;
Output: MBVS(G);

```
B ← ∅;                                  ▷ Currently best solution
for i ← 1 to n − 1 do
    construct H_i;
    VC_i ← BBVC(V(H_i));
    B_i ← V(H_i) + v_i − VC_i;
    if |B_i| > |B|, B ← B_i;
end for
output B;
```

Procedure BBVC(U)
Input: U is the set of all undetermined vertices;
Output: A minimum vertex cover of $H_i[U]$;

```
S ← ∅;                                  ▷ Solution set
perform reduction rules and update S and U;
if U = ∅, return S;
choose a vertex v with maximum degree;
S_1 ← S ∪ {v} ∪ BBVC(U − v);
S_2 ← S ∪ {N_v} ∪ BBVC(U − N_v − v);
if |S_1| < |S_2| then
    return S_1;
else
    return S_2;
end if
```

The **Reduction Rules** we use in Algorithm 1 are as follows.

1. Remove any vertex with degree zero.
2. For any vertex with degree one, select its neighbor into solution and remove the both vertices.
3. If the maximum degree for any vertex is at most two, pick one vertex into the solution and repeat rules 1 and 2.

Lemma 5 gives us a lower bound (LB) of the number of vertices to delete. We can find a *maximum matching* (MM) to increase the lower bound. But finding MM is very time-consuming, and therefore we only find a *maximal matching* and we need not construct H. Algorithm 2 is our LB algorithm.

Algorithm 2 Maximal matching of H (**LB**)

```
Let U = u_1, u_2 ... u_h
mark all vertices in U as not-matched
for i = 1 to h − 1 do
    if u is not-matched then
        if there exists j > i such that (u_i, u_j) ∈ E(H) then
            put (u_i, u_j) into M
        end if
    end if
end for
return M
```

To use the lower bound function in the branch-and-bound algorithm, we always maintain the currently best solution. Whenever the lower bound of a vertex subset plus the number of vertices which is already chosen is no less than the currently best solution, we don't need to search further and return infinity.

4.2 Fixed-parameter algorithm

According to Lemma 7, we have a fixed-parameter algorithm.

Algorithm 3 Fixed-parameter algorithm

```
for i = 1 to k do
    find VC_i
    if |VC_i| < k − i + 1 then
        return yes
    end if
end for
return false
```

The currently best fixed-parameter algorithm is $O(1.28^k \cdot n + kn^2)$ [8]. However, in our implementation, we used a simpler one which uses only three reduction rules described above.

Lemma 11 : The B&B algorithm runs in $O(1.33^k \cdot n^3)$ time for solving the coMBVS problem, in which k is the number of vertices to delete.

Proof: The reduction rules 1 and 2 can be executed in $O(n)$ time. The **LB** can be computed in $O(n^2)$. Finding the maximum degree and rule 3 can also be done in $O(n^2)$ time. When we choose a vertex to branch, the degree of the chosen vertices must be greater than or equal to 3. Other cases are handled by the reduction rules. If we select this vertex into solution, then one vertex is decided; otherwise, all of its neighbors have to be selected in order to cover these edges. Let $T(k)$ denote the number of search nodes in the search tree that searching for a vertex cover of size bounded by k. We have that $T(k) \leq T(k-1) + T(k-3)$. Solving the recurrence relation, we have $T(k) \in O(1.33^k)$. Since we have to solve $O(n)$ vertex covers, the total complexity of our algorithm for the coMBVS problem is $O(1.33^k \cdot n^3)$ time. □

4.3 Approximation algorithm

By Theorem 8 and Corollary 10, we have a 2-approximation algorithm for the coMBVS problem, which is based on the 2-approximation algorithm for the vertex cover problem [24]. Let H_i' be the graph defined by $V_1 = \{v_i\}$, $U = V - V_1$. We now describe the algorithm as follows.

Algorithm 4 VERTEX-COVER

for all i **do**
  find a VC of H_i'
end for
return the maximum $V - VC_i$

Although we have a 2-approximation algorithm. We provide different algorithms, and we compare them by experiments. We approximate the MBVS problem by reducing to set cover problem.

PROBLEM: Set Cover
INSTANCE: A universe U of n elements, and a collection of subset of U, $S = \{S_1, ...S_m\}$.
QUESTION: Find a minimum cardinality subcollection of S that cover all elements of U.

When the input is a complete graph, it is just like to find the maximum acyclic subgraph in a tournament [24].

We consider each vertex as a set and each negative 3-cycle as an element. Let $G = (V, V \times V, \sigma)$ be a complete signed graph. For each vertex $v \in V$, create a set S_v which include all negative 3-cycles containing v. A subset X of S that covers all elements of U correspond to find a vertex subset of V such that $G \backslash X$ has no negative cycle.

The set-cover problem can be f-approximated in polynomial time, in which f is the *frequency*, i.e., each element occurs in at most f sets [24]. In our problem, each element at most appears in 3 sets, and therefore we have a 3-approximation solution. And we denote the algorithm SC in the following section.

Algorithm 5 SC

$\bar{S} \leftarrow \emptyset$; // Let $\bar{S}$ be discarded vertices
for each $v_i, v_j, v_k \in V$ **do**
  if $\triangle v_{ijk}$ is unbalanced **then**
    move v_i, v_j, v_k from V to $\bar{S}$
  end if
end for
return $\bar{S}$

We also design a greedy algorithm which is directly applied to the coMBVS problem, instead of to set-cover or vertex-cover.

Algorithm 6 GREEDY algorithm

$\bar{S} \leftarrow \emptyset$; // Let $\bar{S}$ be discarded vertices
while coMBVS is unbalanced **do**
  Find the vertex that be contained in the most number of unbalanced triangles and remove it to $\bar{S}$
end while
return $\bar{S}$

5 Experiments

In this section, we explain the experimental environment in Section 5.1, show experimental results in Sections 5.2, 5.3 and 5.4, and we give some discussions in Section 5.5.

5.1 Environment and data

We did experiments for the exact, fixed-parameter and approximation algorithms. We used three different kinds of data as inputs.

(a) Starting from a balanced graph, we choose some random vertices and change the signs of some edges incident to the chosen vertices. The number of chosen vertices is denoted by r.

(b) Starting from a balanced graph, we choose some random edges and change the signs. We denote the number of chosen edges by e.

(c) Starting from a graph, we decide each edge sign with a probability p.

We define some notation in the following experiments.

- n : The number of vertices.
- δ : The graph density.
- t : We use t as a time threshold and stop the program when the running time exceed t.
- ϵ : We use ϵ as a upper bound of pushes and stop the program when the push times exceed ϵ.

We tested 30 different graphs for each experiment, and recorded the average and the worst case results. The experiments were conducted on a PC with Intel Core i7 processor and 8 GB of RAM.

5.2 Experiment for the exact algorithm

In the exact-algorithm experiment, we tested five versions of the branch-and-bound algorithm to observe the effects of lower bound, reduction rules and chosen vertex. Since we use the Depth-First-Search strategy, the recursive calls in the B&B algorithm is equivalent to using a stack. And we compared their push times and running times. The five algorithms are named by F$x_1x_2x_3$, in which each x_i is either 1 or 0. The meaning of the name is as follows.

- If $x_1 = 1$, we use reduction rules in the B&B Algorithm; if $x_1 = 0$, we do not use reduction rules.
- If $x_2 = 1$, we choose a vertex with the largest degree for branching; if $x_2 = 0$, we choose a vertex for any order for branching.
- If $x_3 = 1$, we use LB as lower bound; if $x_2 = 0$, we use current best solution as lower bound.

For example, F110 means that we use reduction rules and choose a vertex of the largest degree to branch but without using LB as the lower bound. We had tested the five versions: F000, F100, F110, F101 and F111.

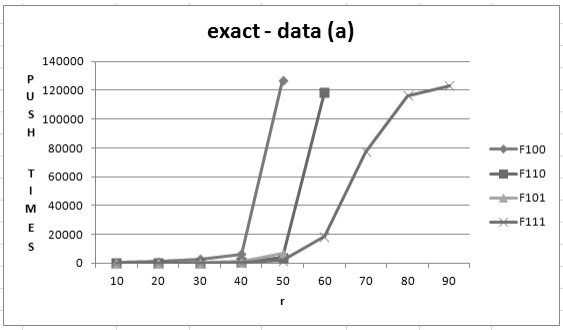

Figure 1: Comparison of push times of data (a).

We set $n = 100$ and ϵ to fifteen million for data (a). We randomly pick r vertices and change the signs of 5 edges incident to the chosen vertices. We do not show F000 in Figure 1 because its push times increase extremely fast. For example, when $r = 20$, its push times is over ϵ. Except F111, the other three versions exceed threshold $t = 15$ seconds when $m = 60, 70$ and 60, respectively.

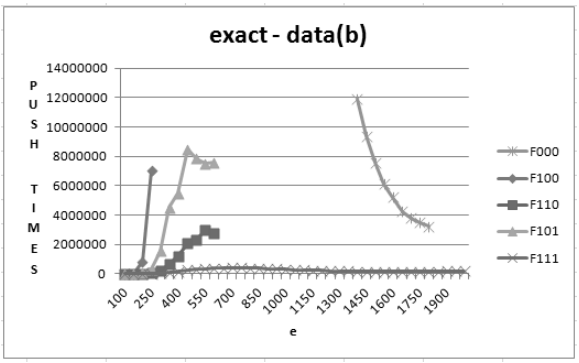

Figure 2: Comparison of push times of data (b).

For data (b), we set $n = 100$, e from 100 to 2000 and ϵ to fifteen million. We do not show F000 when e below 1400 because its push times also extremely fast. For example, when $e = 100$, its push times is only 3351, but when $e = 300$, its push times exceed ϵ. And we observed that its push times decrease significantly when e is greater than 1400.

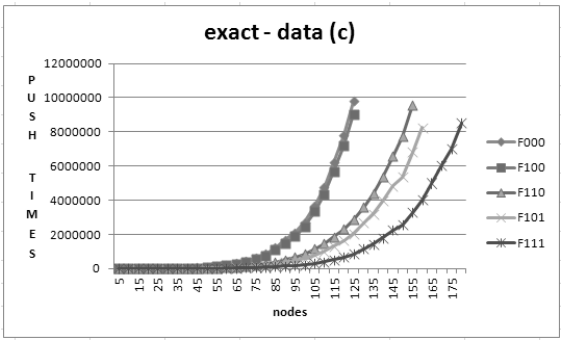

Figure 3: Comparison of push times of data (c).

We set n from 5 to 200, $p = 0.5$, and a push upper bound ϵ to ten million for data (c). It is expectable that the push times of each algorithm get more when the number of vertices increases. F000, F100, F110, F101 and F111 exceed ϵ when the number of vertices are 130, 130, 160, 165 and 185 respectively.

5.3 Experiment for the fixed-parameter algorithm

For a parameter k, the goal is to find if there is a feasible solution of deleting at most k vertices. We set $n = 120$ and $r = 100$ for data (a). We iteratively decrease the value of k to find which is the optimal solution. We run 30 different graphs for each k. The two curves of Figures 4 and 5 are explained as follows. And let the hard case be the push times reach at the peak of the curve.

- *yes* case: There exists a solution of deleting at most k vertices. We use h_y to denote the hard case of *yes*-instances.
- *no* case: There does not exist a solution of deleting k vertices. We use h_n to denote the hard case of *no*-instances.

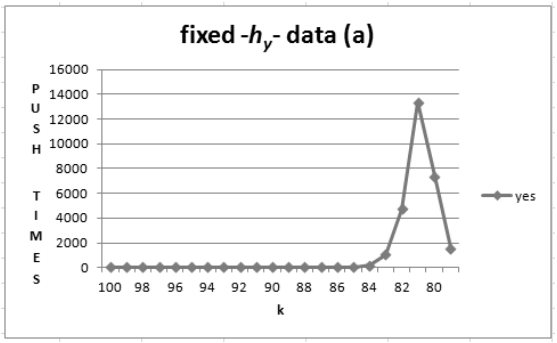

Figure 4: The hard case of *yes*-instances of data (a).

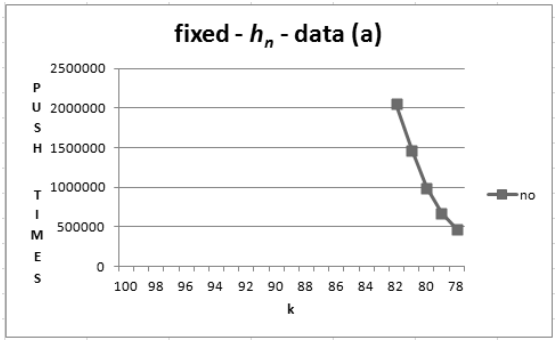

Figure 5: The hard case of *no*-instances of data (a).

5.4 Experiment for the approximation algorithms

In these experiments, we tested three approximation algorithms that we describe in section 4.1. We further slightly improve the *SC* algorithm: we check all vertices that had be deleted after *SC* algorithm with any order. If it could be added back to the solution without causing any unbalance, we added it. We denote the algorithm by *SC-s*. Note that VERTEX-COVER also uses this method.

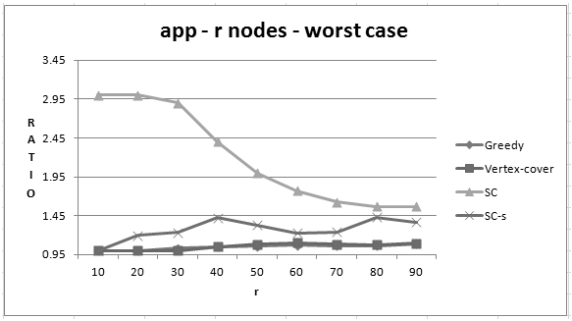

Figure 6: Comparison of worst ratio of data (a).

Table 1: The running time of $r \geq 60$.

m	60	70	80	90
GREEDY	0.088	0.091	0.102	0.099
VERTEX-COVER	0.075	0.062	0.065	0.061
SC	0.009	0.009	0.010	0.009
SC-s	0.010	0.010	0.011	0.010

We set $n = 100$ for data (a), and the result shows in Figure 6. Note that the curves of GREEDY and VERTEX-COVER overlap. We randomly pick r vertices and change the signs of 5 edges incident to the chosen vertices.

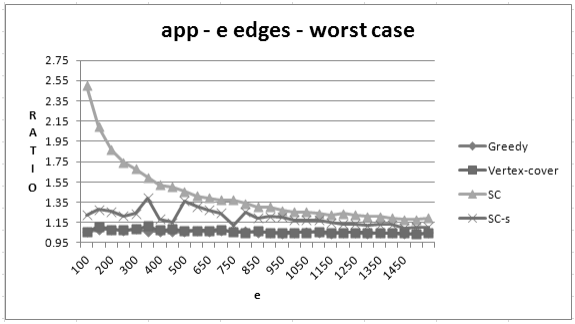

Figure 7: Comparison of worst ratio of data (b).

The result of data (b) is showed in Figure 7, we set $n = 100$, and we randomly choose e edges and change the signs.

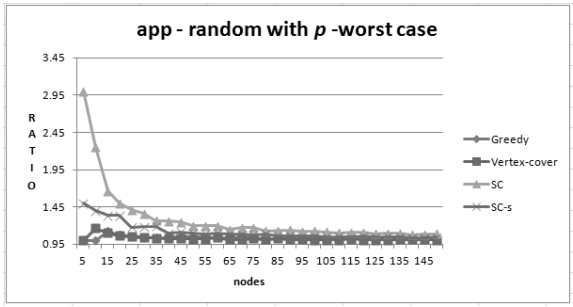

Figure 8: Comparison of worst ratio of data (c).

For data (c), we set $p = 0.5$ and n from 5 to 150 (show in Figure 8).

5.5 Discussion

In the experiment of exact-algorithm, Algorithm F111 performs the best in push times for all data. We further compared the running time of the three data, and we observed that the push times and running time depend on the density of H. The performance of F111 is good when the density is below 0.13 in data (a), see Figure 1 and Figure 9. And Algorithm F111 is still better than F000 for any value of r in data (a). We also observed that F111 is better than F000 when the density is below 0.45 in data (b), particularly good when the density is below 0.12. When the density is small, F000 performs very bad and exceeds our threshold even for small n. According to the experiment results of data (a) and (b), we know that F111 can run quickly at low density, and F000 performs very bad at low density. And F111 can deal with larger number of vertices than the others, see Figure 3.

However, when the density exceed 0.45 in data (b), F000 can run faster than F111, although its push times is greater than F111. This reason is due to that **LB** is not very helpful with higher density.

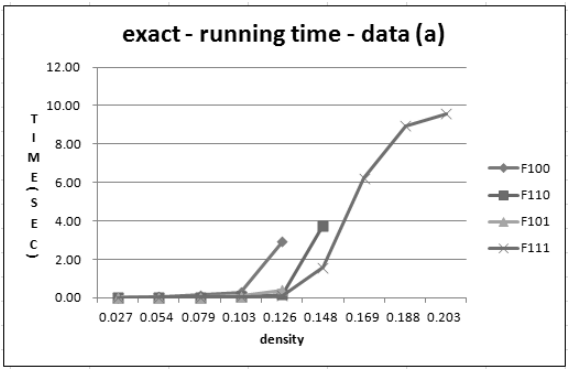

Figure 9: The density of H of data (a).

In the experiment of fixed-parameter algorithms, h_y and h_n almost happen at the same time and h_n exactly happens at the first time that *no* case appears. For example, in Figure 5, h_n happens at $k = 82$ which is the first time *no* case appears. The reason is due to that when k is close to the optimal solution, the program may takes more time to identify that it is yes or no. The curve in Figure 4 declines when k is greater than 81. The reason is due to the effect of **LB**. And the push times of *no* case is more than *yes* case, see Figures 4 and 5.

In the experiment of approximation-algorithms, we observe that Greedy and Vertex-cover are almost as good as the optimal solution in all the tested cases, see Figures 6, 7 and 8. The reason may be that the worse cases rarely happen in such random graphs we generated. All approximation algorithms have good ratios when the number of deleted vertices is large, i.e., the size of optimal solution is small. *SC-s* can easily reduce the ratio of *SC* by using slight improvement that we describe in Section 5.4. In Table 1, we compare the running times of all approximation algorithms. We can see that all approximation algorithms run efficiently. We observed that the running time of Greedy is more than Vertex-cover when $r \geq 60$ in data (a), but less than Vertex-cover when $r < 60$. The reason is due to the value of density. If we want a better running time with density ≤ 0.13, we should choose Greedy; otherwise, we should choose Vertex-cover. And the ratio of Greedy may slightly better than Vertex-cover in the

average case.

We also compare the size of solution of the three kinds of data. The solution size of data (a) is the largest. And the solution size of data (c) is the smallest.

6 Conclusion

In this paper, we study the coMVBS problem on complete graphs. We show that the problem can be reduced to a series of vertex-cover problem. Thus, it admits a 2-approximation algorithm and an exact solution can be found in $O(1.28^k \cdot n + n^3)$ time algorithm, in which k is the number of vertices to delete. Furthermore, we design a branch-and-bound algorithm and test the practical performances of these algorithms, including their running times and solution qualities.

An interesting future work is how to generalize this work to general graphs. Any better result in the aspect of approximation and fixed-parameter of this problem and for the general cluster editing problem is also interesting.

Acknowledgment

This work was supported in part by NSC 98-2221-E-194-027-MY3 and NSC 100-2221-E-194-036-MY3 from the National Science Council, Taiwan, R.O.C.

References

[1] N. Ailon, M. Charikar, A. Newman, Aggregating Inconsistent Information: Ranking and Clustering, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing(STOC)*, pp. 684–693, 2005.

[2] N. Alon, K. Makarychev, Y. Makarychev, A. Naor., Quadratic forms on graphs, *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pp. 486–493, 2005.

[3] S. Arora, E. Berger, E. Hazan, G. Kindler, S. Safra, On non-approximability for quadratic programs, *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.

[4] A. Avidor, M. Langberg, The multi-multiway cut problem, *Proc. 9th SWAT, volume 3111 of LNCS, Springer*, pp. 273–284, 2004.

[5] N. Bansal, A. Blum, S. Chawla, Correlation clustering, *Machine Learning, Special Issue on Clustering*, pp.89–113, 2004.

[6] M. Charikar, V. Guruswami, A. Wirth, Clustering with qualitative information, *Journal of Computer and System Sciences*, pp. 360–383, October 2005.

[7] M. Charikar, A. Wirth, Maximizing quadratic programs: extending Grothendieck's inequality, *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 54–60, 2004.

[8] J. Chen, I.A. Kanj, G. Xia, Improved parameterized upper bounds for vertex cover, *Proc. 31st MFCS. Lecture Notes in Computer Science*, vol. 4162, pp. 238-249. Springer, Berlin (2006)

[9] C. Chiang, A. B. Kahng, S. Sinha, X. Xu, A. Z. Zelikovsky, Fast and efficient bright-field AAPSM conflict detection and correction, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 115–126, 2007.

[10] B. DasGupta, G. A. Enciso, E. D. Sontag, Y. Zhang, Algorithmic and complexity results for decompositions of biological networks into monotone subsystems, *Proc. 5th WEA, volume 4007 of LNCS, Springer*, pp. 253–264, 2006.

[11] R. G. Downey, M. R. Fellows.g, Parameterized Complexity, *Springer*, pp. 273–284, 1999.

[12] J. Flum, M. Grohe, Parameterized Complexity Theory, *Springer*, 2006.

[13] I. Giotis and V. Guruswami, Correlation clustering with a fixed number of clusters, *Theory Comput.*, 249–266, 2006.

[14] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, S. Wernicke, Compression based fixed-parameter algorithms for feedback vertex set and edge bipartitiony, *Journal of Computer and System Sciences*, pp. 1386–1396, 2006.

[15] F. Harary, On the notion of balance of a signed graph, *Michigan Mathematical Journal* , pp.143–146, 1953.

[16] F. Heider, Attitudes and cognitive organization, *Journal of Psychology* , pp. 107–112, 1946.

[17] F. Hüffner, N. Betzler and R. Niedermeier, Optimal edge deletions for signed graph balancing, *Proceedings of the 6th Workshop on Experimental Algorithms (WEA07)*, June 6–8, 2007.

[18] F. Hüffner, C. Komusiewicz, H. Moser, R. Niedermeier, Fixed-parameter algorithms for cluster vertex deletion, Theory of Computing Systems 47(1): 196–217, 2010

[19] S. Khot, On the power of unique 2-prover 1-round games, *Proc. 34th STOC*, pp. 767–775, 2002.

[20] A. Nemirovski, C. Roos, T. Terlaky, On maximization of quadratic form over intersection of ellipsoids with common center, *Mathematical Programming*, pp. 463–473, 1999.

[21] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, *Oxford University Press*, 2006.

[22] R. Shamir, R. Sharan, D. Tsur., Cluster graph modification problems, *Proceedings of 28th Workshop on Graph Theory (WG)*, pp. 379–390, 2002.

[23] S. Wasserman, K. Faust, Social Network Analysis, Cambridge University Press, Cambridge, 1994.

[24] V. Vazirani, Approximation Algorithms, *Springer*, 2001.

[25] T. Zaslavsky, Bibliography of signed and gain graphs, *lectronic Journal of Combinatorics*, 1998.