Balancing a Complete Signed Graph by Changing Minimum Number of Edge Signs

Po-Sun Wei and Bang Ye Wu*
Computer Science and Information Engineering
National Chung Cheng University, ChiaYi, Taiwan 621, R.O.C.

Abstract

A signed graph *is a simple undirected graph $G = (V, E)$ in which each edge is labeled by a sign either +1 or -1. A signed graph is balanced if every cycle has even numbers of negative edges. In this paper we study the problem of balancing a complete signed graph by changing minimum number of edge signs. We give a simple algorithm for finding a solution agreeing one half of the edges. We also design a branch-and-bound algorithm and show the worst-case time complexity is $O(n \cdot 2^{\min\{n,k\}})$, in which $n = |V|$ and k is the number of changing edges. By experiments on random graphs, we show that our branch-and-bound algorithm is much faster than a trivial one.*

1 Introduction

A *signed graph* is a simple undirected graph $G = (V, E)$ in which each edge is labeled by a sign either +1 or -1, i.e., there is a function $\sigma : E \mapsto \{1, -1\}$. A signed graph is *balanced* if every cycle has even numbers of negative edges. By the assumption that a social network usually evolves toward to a balanced system via an efficient way, computing the minimum number of sign changes to balance a network may be helpful for predicting how the system evolves or mining the fake relations. In this paper, we study the following optimization problem, named Sign Change, how to balance a complete signed graph by changing minimum number of edge signs.

> Problem: Minimum sign-changes problem (Sign Change)
> Instance: A signed complete signed graph $G = (V, E = V \times V, \sigma)$ in which $\sigma : E \mapsto \{1, -1\}$.
> Goal: Minimize the number of changes on edge-signs which makes the graph balanced.

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

The concept of balanced signed graphs was introduced by Harary [13] 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"). A bibliography of signed graphs can be found in [18]. Signed graphs also find other applications and attract many researchers. In the literatures, it also appeared in other forms. For example, motivated by the mathematical analysis of large-scale biological networks, DasGupta et al. [8] formulated their problem as Balanced Subgraph, called Undirected Labeling Problem, and this problem also finds numerous other applications, e.g., in statistical physics and integrated circuit fabrication techniques [7, 18].

Given a signed graph $G = (V, E)$, Balanced Subgraph is to find a balanced subgraph with maximum number of edges. Balanced Subgraph is a generalization of the NP-hard Maximum Cut problem. DasGupta et al. developed an approximation algorithm that guarantees a solution with at least 87.9% of the number of edges of an optimal solution. By their original definitions, it may be not obvious to realize Balanced Subgraph is equivalent to Sign Change with only difference on their objectives, maximizing number of the remaining edges or minimizing the number of changed edges. We shall give the explaination later.

When all edges are labeled by -1, Balanced Subgraph is reduced to the Edge Bipartization problem, which asks for the minimum number of edges to delete to make a graph bipartite. We can note that the Edge Bipartization and the Maximum Cut problems are dual. The only difference is their objective functions: minimizing the deleted edges or maximizing the remaining edges. The Edge Bipartization problem, also known as (unweighted) Minimum Uncut prob-

lem, is Max SNP-hard. The best known approximation algorithm finds in polynomial time a solution of size $O(k \log k)$, where k is the size of an optimal solution [4].

Balanced Subgraph 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 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. The maximization version, call it MaxAgree, seeks to maximize the number of agreements: the number of 1 edges inside clusters plus the number of -1 edges across clusters. The minimization version, denoted MinDisAgree, aims to minimize the number of disagreements: the number of -1 edges within clusters plus the number of 1 edges between clusters. A natural variant of this problem is to restrict the number of clusters to a fixed k, and we shall denote this version by MaxAgree$[k]$. It can be easily realized that the problem is equivalent to Balanced Subgraph when $k = 2$.

Shamir et al. [16] showed that MaxAgree(MinDisAgree), as well as MaxAgree$[k]$ (MinDisAgree$[k]$) for each $k \geq 2$ is NP-hard. They used the term "Cluster Editing" to refer to this problem. The problem was also independently formulated and considered in [5], and a PTAS for MaxAgree and a constant factor approximation algorithm for MinDisAgree were given. There are several other results about approximation of the above and related problems in [1, 2, 3, 6, 11, 14]. For more details, see [11]. Giotis and Guruswami [11] provide a NP-hardness proof and prove that both MaxAgree$[k]$ and MinDisAgree$[k]$ admit a polynomial time approximation scheme for every fixed $k \geq 2$ when the input is a complete graph. The running time for MaxAgree$[k]$ is $n \cdot k^{O(\varepsilon^{-3} \log(k/(\varepsilon\delta)))}$ and $n^{O(9^k/\varepsilon^2)} \log n$ for MinDisAgree$[k]$.

There are some results from the field of fixed-parameter algorithms [9, 10, 15]. A problem is called fixed-parameter tractable with respect to a parameter k if an instance of size n can be solved in $f(k) \cdot n^{O(1)}$ time, where f is an arbitrary function depending only on k. It is known that Edge Bipartization is fixed-parameter tractable with respect to the parameter k as the number of edges to delete. More specifically, there is an algorithm exactly solving Edge Bipartization in $O(2^k \cdot m^2)$ time (m denoting the number of graph edges) [12]. In Optimal Edge Deletions for Signed Graph Balancing, they observe that Balanced Subgraph easily reduces to Edge Bipartization and can thus be solved with the same time complexity.

In this paper, we study some properties on balanced signed graphs and develop algorithms for Sign Change. We give a simple algorithm for finding a solution agreeing one half of the edges. We also design a branch-and-bound algorithm and show the worst-case time complexity is $O(n \cdot 2^{\min\{n,k\}})$, in which $n = |V|$ and k is the number of changing edges. Compared with the previous result $O(2^k \cdot m^2)$, our algorithm is slightly better but restricted to only complete graphs. By experiments on random graphs, we show that our branch-and-bound algorithm is much faster than a trivial one.

The paper is organized as follows. In Section 2, we give some notation and study some properties. We present the algorithms in Section 3, and the experiment results are given in Section 4. Finally, some concluding remarks are given in Section 5.

2 Preliminaries and some properties

2.1 Notation and definitions

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.

A clique is a 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. A cycle is positive if it contains an even number of negative edges. In the remaining paragraphs, the input graph is always $G = (V, E, \sigma)$ and we use $n = |V|$ and $m = |E|$. Let E^+ be the set of positive edges, and E^- be the set of negative edges. A *perfect 2-clustering* of a singed graph $G = (V, E, \sigma)$ is bipartition (V_1, V_2) of V such that $\sigma(u, v) = -1$ if and only if u and v are in the different subsets.

The next theorem is due to Harary.

Theorem 1: A singed graph is balanced if and only if there is a perfect 2-clustering [13].

For a bipartition (V_1, V_2), we mark each vertex in V_1 by +1 and each vertex in V_2 by -1. That is, we define a function $r \mapsto \{+1, -1\}$ such that $r(v) = +1$ for any $v \in V_1$ and $r(v) = -1$ for any $v \in V_2$. We say that an edge $e = (u, v)$ *agrees* with the bipartition if $r(u) \times r(v) = \sigma(u, v)$ and e is an *agreement*, and for otherwise e is a *disagreement*. We also abuse the term agreement to denote the total number of agreeing edges.

By Theorem 1, any edge disagreeing with the 2-cluster must be deleted. And if we have same situation in Sign Change, then we have to change the sign of those edges, too. So we can know that Sign Change is equivalent to Balanced Subgraph. At a glance, the problem Sign Change seems to look for an edge subset. However, by Theorem 1, it is to find a bipartition with minimum disagreement and thus a brute force algorithm takes $O(2^n \cdot n^2)$ time. The next theorem appears in [17], which provides a simple method to check if a complete graph is balanced.

Theorem 2: A complete signed graph is balanced if and only if all triangles are positive [17].

2.2 Some properties

In this subsection, we present some observed properties on complete balanced signed graphs.

Lemma 3: If a complete signed graph $G = (V, E, \sigma)$ is balanced, then the absolute value of the inner product of any two vectors of the adjacency matrix is $n - 2$.

Proof: Suppose that $|V_1| = k_1$ and $|V_2| = k_2$, in which V_1 and V_2 is the two clusters in Theorem 1. The matrix has the form shown in Figure 1.

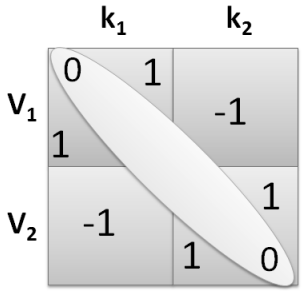

Figure 1: The adjacency matrix of G.

- If both vertices are in V_1, the inner product is $(k_1 - 2) \times (1 \times 1) + 2 \times (0 \times 1) + k_2 \times ((-1) \times (-1)) = k_1 + k_2 - 2 = n - 2$.
- The case that both vertices are in V_2 is similar.
- If one vertex is in V_1 and other vertex is in V_2, the inner product is $(k_1 - 1) \times (1 \times (-1)) + (k_2 - 1) \times ((-1) \times 1) = -(n - 2)$.

□

By definition any graph with one or two vertices is always balanced. But it is interesting to know if there is always a balanced triangle in a complete graph. Let $p(n)$ denote the maximum number of positive edges in a complete n-vertex graph such that there is no any balanced triangle. The next lemma is a necessary condition of which there must be a balanced triangle.

Lemma 4: $p(n) = n^2/4$ if n is an even number, and $p(n) = (n-1)(n+1)/4$ if n is odd.

Proof: Let G be a complete signed graph without any balanced triangle. Let $G^+ = G[E^+]$ be the subgraph induced by all the positive edges. First, we show that G^+ is a bipartite graph. Apparently, if there exists a 3-cycle in G^+, the 3-cycle is balanced triangle. Suppose that $(v_1, v_2, \ldots v_k, v_1)$ is an induced k-cycle in G^+ with k an odd number larger than three. Then, both (v_1, v_3) and (v_1, v_4) are negative edges in G, and (v_1, v_3, v_4) is a balanced triangle in G.

Since G^+ is a bipartite graph, $p(n)$ is the the maximum number of edges of any bipartite graph with n vertices, and the maximum is achieved when the cardinalities of the two vertex subsets are as equal as possible. □

3 Algorithms

3.1 An 2-approximation algorithm

By Theorem 1, the optimal balanced graph corresponds to a 2-clustering, and we can have a simple algorithm with agreement at least one half of $|E|$. Let $(v_1, v_2, \ldots, v_n)$ be an arbitrary ordering of the vertices. We first put v_1 into V_1 and then put each incoming vertex into V_1 or V_2 greedily. That is, in each iteration, we put the incoming vertex into one cluster such that the number of

agreement of its incident edges is maximized. Let $d^+(v, V_i) = |E^+ \cap \{(u, v) : u \in V_i\}|$ be the number of positive edges from v to any vertices in V_i. Similarly, let $d^-(v, V_i) = |E^- \cap \{(u, v) : u \in V_i\}|$.

Algorithm 1 Greedy algorithm

Input: A complete signed graph $G = (V, E)$.
Output: A bipartition $\mathcal{P} = (V_1, V_2)$ of V.

```
(V1, V2) ← (∅, ∅);
let V = {v1, v2, ..., vn} in which the vertices are arbitrarily labeled;
for i ← 1 to n do
    if d+(vi, V1) + d−(vi, V2) ≥ d−(vi, V1) + d+(vi, V2) then
        put vi into V1;
    else
        put vi into V2;
    end if
end for
output (V1, V2).
```

Lemma 5: Algorithm 1 finds a bipartition with agreements at least $|E|/2$.

Proof: By definition $(d^-(v_i, V_1)+d^+(v_i, V_2))+(d^+(v_i, V_1) + d^-(v_i, V_2))$ is the total number of edges from v_i to $V_1 \cup V_2$. By the greedy strategy used in the algorithm, at least one half of these edges agree with the bipartition. Summing up the agreements for all v_i, the total agreement is at least one half of the number of edges. □

A complete signed graph with even number of vertices and all negative edges is a simple tight example for the approximation algorithm (as shown in Figure 2). For this extreme example, the algorithm finds a bipartition with $\frac{n}{2}$ vertices in either side. The ratio of the number of agreements to the number of edges is $(\frac{n}{2} \times \frac{n}{2})/\binom{n}{2} = n/(2n-2)$.

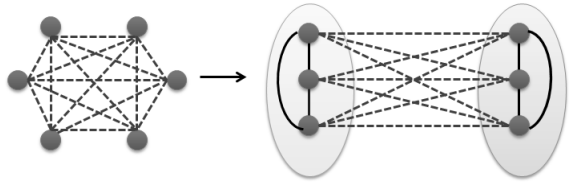

Figure 2: A tight example for the approximation algorithm.

3.2 Exact algorithms

To find an exact solution, we choose the first vertex v_1 as a basic, and then we need to determine for each vertex that it is in the same or the different cluster of v_1.

Let V_1 be the cluster which v_1 belongs to. By a $(n-1)$-vector $R =< r_2, r_3, \ldots, r_n >$, we denote a partial solution such that, for each $2 \le i \le n$, $r_i = 1$ means that v_i must be in V_1; $r_i = -1$ for $v_i \in V_2$; and $r_i = 0$ if it is not determined yet. Let U be the set of undetermined vertices.

For a partial solution, we can compute the number of sign-changes of all pairs of determined vertices. Also we can estimate a lower bound for any undetermined vertex. The algorithm uses a stack to contain all partial solutions to be explored. So it's a depth-first-search tree-searching algorithm.

Define

$$\begin{aligned} & g(V_1, V_2) \\ = \; & |E^- \cap \{(u, v) : u, v \in V_1\}| \\ & +|E^- \cap \{(u, v) : u, v \in V_2\}| \\ & +|E^+ \cap \{(u, v) : u \in V_1, v \in V_2\}| \\ = \; & d^-(V_1) + d^-(V_2) + d^+(V_1, V_2) \end{aligned} \quad (1)$$

which is the number of edges to be changed for partitioning $V_1 \cup V_2$ into V_1 and V_2.

Define

$$\begin{aligned} & h(V_1, V_2) \\ = \; & \sum_{v \in U} \min\{d^-(v, V_1) + d^+(v, V_2), \\ & d^-(v, V_2) + d^+(v, V_1)\} \end{aligned} \quad (2)$$

which is a lower bound of the number of edges to be changed to partitioning U. Therefore $f(V_1, V_2) = g(V_1, V_2) + h(V_1, V_2)$ is a lower bound function. If it is not less than the current best solution, we need not push it into the stack.

The two functions g and h can be computed from the previous values to reduce the computation cost. Precisely speaking, we have

$$\begin{aligned} & g(V_1 + x, V_2) \\ = \; & g(V_1, V_2) + d^-(x, V_1) + d^+(x, V_2) \end{aligned} \quad (3)$$

and

$$\begin{aligned} & g(V_1, V_2 + x) \\ = \; & g(V_1, V_2) + d^-(x, V_2) + d^+(x, V_1) \end{aligned} \quad (4)$$

in which $V_1 + x$ denote $V_1 \cup \{x\}$ for short. For $v \notin V_1 \cup V_2$, let $h_1(v, V_1, V_2) = d^-(v, V_1) + d^+(v, V_2)$ and $h_2(v, V_1, V_2) = d^-(v, V_2) + d^+(v, V_1)$. Then we have that $h(V_1, V_2) = \sum_{v \in U} \min\{h_1(v, V_1, V_2), h_2(v, V_1, V_2)\}$. Furthermore, for $v, x \in U$ and $v \ne x$, we have

$$\begin{aligned} & h_1(v, V_1 + x, V_2) \\ = \; & \begin{cases} h_1(v, V_1, V_2) + 1 & \text{if } (v, x) \in E^- \\ h_1(v, V_1, V_2) & \text{if } (v, x) \in E^+ \end{cases} \end{aligned} \quad (5)$$

and

$$h_1(v, V_1, V_2 + x) = \begin{cases} h_1(v, V_1, V_2) & \text{if } (v, x) \in E^- \\ h_1(v, V_1, V_2) + 1 & \text{if } (v, x) \in E^+ \end{cases} \quad (6)$$

A similar formula for h_2 can be derived. By this way, we can update the lower bound in $O(n)$ time when a vertex is moved from U to V_1 or V_2.

Algorithm 2 Branch-and-bound algorithm

Input: A complete graph $G = (V, E)$, in which $V = \{v_i | 1 \leq i \leq n\}$.
Output: A minimum number of sign changes from complete graph G.

```
initially a stack T; Best ← ∞;
push (∅, ∅, 1) into T;
while T is not empty do
    pop (V1, V2, i) from T;
    compute g(V1 + vi, V2) and g(V1, V2 + vi)
    if i = n then
        Best ← min{Best, g(V1 + vi, V2), g(V1, V2 + vi)};
    end if
    compute h(V1 + vi, V2) and h(V1, V2 + vi)
    if f(V1 + vi, V2) < Best then
        push (V1 + vi, V2, i + 1) into T;
    end if
    if f(V1, V2 + vi) < Best then
        push (V1, V2 + vi, i + 1) into T;
    end if
end while
return Best;
```

Using the above branch-and-bound algorithm, we can also design a fixed-parameter algorithm (BaB_2). For a fixed-parameter algorithm with the number of sign-changes k as parameter, we ask if there exists a solution with sign-changes at most k. To this aim, we only need to set Best to $k+1$ in the above algorithm. We show that the worst-case time complexity is indeed polynomial for fixed k.

Lemma 6: The time complexity of Algorithm BaB_2 is $O(n \cdot 2^{\min\{n,k\}})$, and therefore it is a fixed parameter algorithm.

Proof: First, by Equations (3–6), it is easy to see that each iteration takes only $O(n)$ time. We only need to show the upper bound of the number of iterations. Since each vertex is inserted into V_1 or V_2, the number of iteration is trivially bounded by $O(2^n)$. We now show another bound $O(2^k)$ by induction.

Let $T(i, k)$ denote the worst-case number of partial solutions pushed into the stack T such that $|V_1| + |V_2| = i$ and we can change at most k edge signs. By the algorithm, we have, $T(i, k) \leq T(i + 1, k - j) + T(i + 1, k - (i - j))$ for $i < n$ and $k > 0$, in which j is the number of edges have been changed when move a vertex from U to V_1 or V_2; and $T(i, k) = 1$ for $i = n$ or $k = 0$. Suppose by the induction hypothesis that $T(i, k') \leq 2^{k'}$ for any $k' < k$.

In the case that $0 < j < k$, we have

$$\begin{aligned} & T(i, k) \\ \leq\ & T(i + 1, k - j) + T(i + 1, k - (i - j)) \\ \leq\ & 2^{k-j} + 2^{k-(i-j)} \leq 2^k. \end{aligned}$$

Otherwise, $j = 0$ or k. We show the case $j = 0$ and the other case is similar. If $j = 0$,

$$T(i, k) = T(i + 1, k) + T(i + 1, k - i). \quad (7)$$

Summing (7) for i from 1 to $n - 1$, we have

$$\begin{aligned} & T(1, k) \\ \leq\ & T(2, k - 1) + T(3, k - 2)... + T(k + 1, 0) \\ \leq\ & 2^{k-1} + 2^{k-2} + \ldots + 2^0 = 2^k - 1 < 2^k. \end{aligned}$$

□

4 Experimental results

We did experiments on algorithms for the minimum sign-changes problem. In our experiments, we used two kinds of random graphs. By these data, we tested our algorithms and observed factors which affect the solution and the time complexity.

We shall use the following terms to describe the experiments. By g and $g + h$, we denote the two versions of the branch-and-bound algorithm which use g and $g+h$ as the lower bounds as described in the previous section. Let $Push$ denote the number of pushes, OPT denote the number of sign changes in an optimal solution, and k denote the upper bound of sign-changes in the fixed-parameter algorithm which reports if there exists a solution with at most k sign-changes. We set a bound ten millions of the number of pushes (recursive calls) in the experiments. The program was aborted if no solution was found before exceeding the bound.

4.1 Test data

- First-type
  A complete signed graph is constructed by randomly choosing e negative edges and the remaining edges are positive.

- Second-type
  We first randomly generated a balanced complete graph with specified two cluster sizes. Then we randomly pick k edges and change their signs to make the graph unbalanced. By this way we can control the optimal solutions.

4.2 Experiments

4.2.1 Lower bound

The first experiment is used to test the effect of our lower bound function. We fix n=40 and compare the numbers of pushes used by g and $g+h$ to find an optimal. In Figure 3, we can clearly know that $g+h$ needs much less number of pushes.

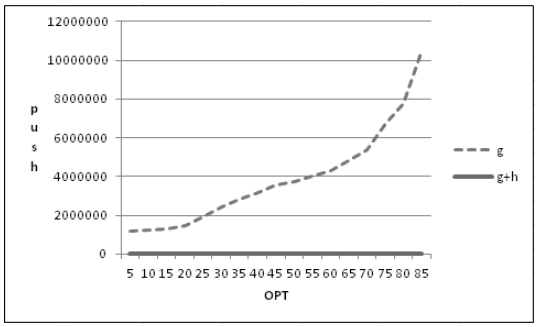

Figure 3: Comparison between lower bounds.

We tested lower bound by finding out the hard cases. For a specified n, the hard case is the value of k such that the number of pushes exceeds 10 millions. For different numbers of n and k, in our experiments, we tested g and $g+h$ with the same inputs, and the results are shown in Figure 4 and Table 1. The pushes of g are much more than those of $g+h$, so we can know that our lower bound is useful.

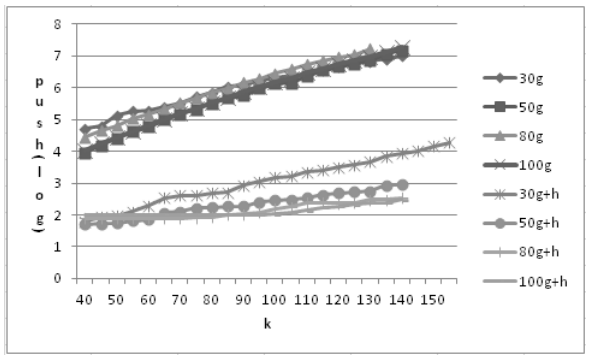

Figure 4: Efficiency comparison between lower bounds: number of pushes in log-scale.

Table 1: Hard case (number k) for algorithm with g or $g+h$ as lower bound.

n	30	50	80	100
g	155	135	135	135
$g+h$		500	700	830

4.2.2 Vertex selection method for branching

In the branch-and-bound algorithm, we need to choose a vertex v in each iteration and make two branches according to putting v into V_1 or V_2. In this experiment, we would like to know if the method of selecting vertex can affect the time complexity. We fixed n=100 and used Second-type random graphs. We tested three selection methods: with-order, largest-gap and smallest-gap.

- with-order
  When choosing the next vertex to branch, we simply choose the next undetermined one.
- largest-gap
  Choosing $v = \arg\max_v |h(V_1 + v, V_2) - h(V_1, V_2 + v)|$.
- smallest-gap
  Choosing $v = \arg\min_v |h(V_1 + v, V_2) - h(V_1, V_2 + v)|$.

We computed the numbers of pushes to compare the three methods and show the result in Figure 5.

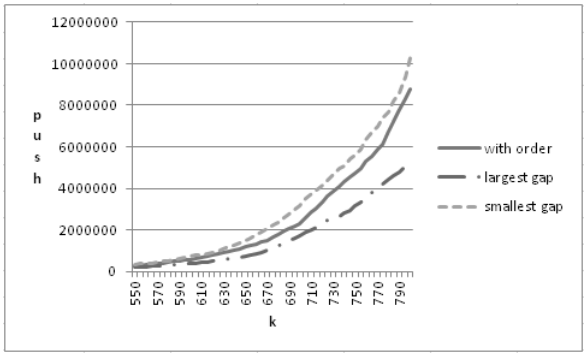

Figure 5: Different selection methods.

4.2.3 Cluster size and positive/negative ratio

By positive/negative ratio we mean that the ratio of the number of the positive edges to the number of negative edges in an input graph. In this experiment, we want to know whether the cluster

size and the positive/negative ratio affect the time complexity or not.

- Cluster size
  We tested three kinds of cluster ratios: 1:1, 1:2, 1:3 with n=100 and show the result in Figure 6. By Figure 6, we can clearly know cluster size does not affect the number of pushes.

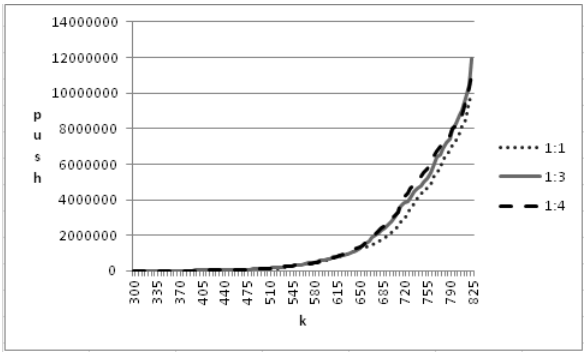

Figure 6: Cluster size affect.

- Positive/negative ratio
  In this experiment we tested different positive/negative ratios with n=40. We show the results in Table 2, which include ratio of positive and negative edges, number of pushes, number of sign changes and execution time (in sec.).

Table 2: Positive/negative ratio

ratio	push	change	time
1:1	28410510	301.7	46.02
1:2	34667422	304.2	56.20
1:3	49778203	309.3	78.72
1:4	74347274	313.9	116.85
2:1	943689	258.5	1.69
3:1	23398	195.0	0.048
4:1	3706	156.0	0.0063

4.3 Discussion

By the experiments, we make the following discussions.

- In Figure 3 and Figure 4, it's clearly to see the better lower bound $g+h$ reduces the running time significantly. And by Table 1, we can know, for g, hard case occurs in nearly k=135 and the number of vertices does not affect it. But for g+h, the hard case of value k increases as n increases.
- By Figure 5, we can know that the largest-gap is the best of the three methods because the number of pushes is the smallest.
- In our experimental results, we can find out cluster size does not affect the number of pushes. By Table 2, we can observe the more negative edges, the more number of pushes, changes, and taking more time. The reason is that the minimum sign-changes increases when the number of initial negative edges is more. So we can know positive/negative ratio makes a great impact on the number of pushes.
- We had also compared time complexities of instances of which the fixed-parameter algorithm returns "yes" or "no". We found that there is no significant difference.
- Finally, from the experiments we observe a phenomenon that the number of pushes is equal to n when $k \leq n$.

5 Conclusion

In this paper, we study some properties of a complete balanced graph. And we design a 2-approximation algorithm and an exact algorithm for the minimum sign change problem with time complexity $O(n \cdot 2^{\min\{n,k\}})$.

The experimental results show that our lower bound can significantly reduce the running time, select method of next vertex and positive/negative ratio can also affect the number of pushes. But cluster size does not affect the number of pushes.

We have observed that the algorithm runs very fast when $k \leq n$. We shall try to give a formal proof in the future. Another interesting future work is the weighted version of the minimum sign change problem.

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] 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.

[8] 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.

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

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

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

[12] 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.

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

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

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

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

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

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