Algorithms for NP-hard Optimization Problems and Cluster Analysis

A Dissertation
Submitted to
the Temple University Graduate Board

in Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

by

Nan Li
September, 2017

Examining Committee Members:

Longin Jan Latecki, Advisory Chair, Computer and Information Sciences
Haibin Ling, Computer and Information Sciences
Slobodan Vucetic, Computer and Information Sciences
Yimin Zhang, External Member, Electrical and Computer Engineering

ABSTRACT

Algorithms for NP-hard Optimization Problems and Cluster Analysis

by

Nan Li

The set cover problem, weighted set cover problem, minimum dominating set problem and minimum weighted dominating set problem are all classical NP-hard optimization problems of great importance in both theory and real applications. Since the exact algorithms, which require exhaustive exploration of exponentially many options, are infeasible in practice, approximation algorithms and heuristic algorithms are widely used to find reasonably good solutions in polynomial time. I propose novel algorithms for these four problems. My algorithms for the weighted set cover and minimum weighted dominating set problems are based on a three-step strategy. For the weighted set cover problem, in the first step, we reserve the sets indispensable for the optimal solution and reduce the problem size. In the second step, we build a robust solution with a novel greedy heuristic. Sets are iteratively selected according to a measure which integrates the weight, the coverage gain for the current iteration and the global coverage capacity of each set. It favors the sets that have smaller weights and better extend or consolidate the coverage, especially on the items that are contained in less sets. Since the obtained solution tends to have a robust coverage, in the third step, we further improve it by removing the redundant sets in an efficient way. For the minimum weighted dominating set problem, we first reserve the indispensable vertices for the optimal solution. Then we convert it into a weighted set cover problem to solve it. These two algorithms can be used to solve the set cover problem and minimum dominating set problem by simply considering all the sets or vertices as having the same weights. Extensive experimental evaluations on a large number of synthetic and real-world set cover instances and graphs from many domains demonstrate the

superiority of my algorithms over state-of-the-art.

Cluster analysis is a fundamental problem in data analysis, and has extensive applications in artificial intelligence, statistics and even in social sciences. The goal is to partition the data objects into a set of groups (clusters) such that objects in the same group are similar, while objects in different groups are dissimilar.

Most of the existing algorithms for clustering are designed to handle data with only one type of attributes, e.g. continuous, categorical or ordinal. Mixed data clustering has received relatively less attention, despite the fact that data with mixed types of attributes are common in real applications. I propose a novel affinity learning based framework for mixed data clustering, which includes: how to process data with mixed-type attributes, how to learn affinities between data points, and how to exploit the learned affinities for clustering. In the proposed framework, each original data attribute is represented with several abstract objects defined according to the specific data type and values. Each attribute value is transformed into the initial affinities between the data point and the abstract objects of attribute. I refine these affinities and infer the unknown affinities between data points by taking into account the interconnections among the attribute values of all data points. The inferred affinities between data points can be exploited for clustering. Alternatively, the refined affinities between data points and the abstract objects of attributes can be transformed into new data features for clustering. Experimental results on many real world data sets demonstrate that the proposed framework is effective for mixed data clustering. This work was published in our IJCAI 2017 paper Li & Latecki (2017).

Clustering aggregation, also known as consensus clustering or clustering ensemble, aims to find a single superior clustering from a number of input clusterings obtained by different algorithms with different parameters. I formulate clustering aggregation as a special instance of the maximum-weight independent set (MWIS) problem. For a given data set, an attributed graph is constructed from the union of the input clusterings. The vertices, which represent the distinct clusters, are weighted by an internal index measuring both cohesion and separation. The edges connect the vertices whose corresponding clusters overlap. Intuitively, an optimal aggregated clustering can be obtained by selecting an optimal subset of non-overlapping clusters partitioning the data set together. I formalize this intuition as the MWIS problem on the attributed graph, i.e., finding the heaviest subset of mutually non-adjacent vertices. This MWIS problem exhibits a special structure. Since the clusters of each input clustering form a partition of the dataset, the vertices corresponding to each clustering form a maximal independent set (MIS) in the attributed graph. I propose a variant of simulated annealing method that takes advantage of this special structure. My algorithm starts from each MIS, which is close to a distinct local optimum of the MWIS problem, and utilizes a local search heuristic to explore its neighborhood in order to find the MWIS. Extensive experiments on many challenging data sets show that both my algorithm for the maximum-weight independent set problem and my approach to the application of clustering aggregation achieve good performance. This work was published in our NIPS 2012 paper Li & Latecki (2012). Some new results were published in our IJCAI 2017 paper Fan et al. (2017).

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my advisor, Dr. Longin Jan Latecki. He has always been supportive and patient. Without his guidance and encouragement, none of my research works would have been possible.

I would also like to thank the other members in my dissertation committee, Dr. Haibin Ling, Dr. Slobodan Vucetic, and Dr. Yimin Zhang, for their insightful comments and encouragement.

The discussions with other group members, Zhuo Deng, David Dobor, Meng Yi, Tianyang Ma, Le Shu, Chen Shen, Ren-Hau Howard Liu and Cong Rao have always been inspiring. I've learned a lot from them. I'm very grateful for that.

Last but not the least, I would like to thank my wife and my parents for their love and support.

To My Wife - Pei Qiu, and My Parents

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER 1

Algorithms for Weighted Set Cover and Minimum Weighted Dominating Set Problems

1.1 Introduction

Given a set of items, called the universe U, and a collection $\mathcal{F}$ of sets whose union equals U, the set cover problem is to find the smallest sub-collection of $\mathcal{F}$ whose union equals the universe U. The weighted set cover problem, in which each set is associated with a positive weight, is to find the sub-collection of $\mathcal{F}$ whose union equals U with the minimum sum of weights.

Given an undirected graph $G = (V, E)$, a dominating set is a subset $D \subseteq V$ such that for every vertex $v \in V$, either $v \in D$ or at least one neighbor of v is in D. The minimum dominating set problem is to find a dominating set of the minimum size. If each vertex is associated with a positive weight, the minimum weighted dominating set problem is to find a dominating set for which the sum of weights is minimized.

The set cover and minimum dominating set problems are both classical NP-hard problems of great importance in theory. They are equivalent under L-reductions Kann (1992). It means given an instance of one problem, we can construct an equivalent instance of the other problem. Therefore, algorithms for one problem can be applied to the other problem with minor modifications. The weighted set cover and minimum dominating set problems are also closely related NP-hard problems.

The unweighted and weighted set cover and minimum dominating set problems arise in a great number of applications, including artificial intelligence Reiter (1987);

Zhao & Ouyang (2007); Saha & Getoor (2009); Shen & Li (2010); Yao & Fei-Fei (2012); Cao & Snavely (2013); Magri & Fusiello (2016), operations research Caprara et al. (1999), computer networking Stojmenovic et al. (2002); El Houmaidi & Bassiouni (2003); Subhadrabandhu et al. (2004); Aoun et al. (2006); Samuel et al. (2009), web technology Cooper et al. (2005); Wu et al. (2006); Stergiou & Tsioutsiouliklis (2015), social networks Kelleher & Cozzens (1988); Eubank et al. (2004); F. Wang et al. (2011), bioinformatics Nacher & Akutsu (2016), planning Mihail (1999), database Sellis (1988); Golab et al. (2008) and so on.

However, since these problems are all NP-hard, the exact algorithms, which require exhaustive exploration of exponentially many options, are infeasible in practice. Approximation algorithms, which are able to find reasonably good solutions in polynomial time, are widely used to solve these problems in real applications.

The classic greedy algorithm for the set cover problem repeatedly picks a set that contains the largest number of uncovered items until all the items in the universe are covered. This simple greedy heuristic achieves an approximation ratio of $\ln\delta + 1$, where $\delta = max\{|S| : S \in \mathcal{F}\}$ is the maximum cardinality of the sets in $\mathcal{F}$. In fact, no algorithm can guarantee to improve this approximation ratio by much Feige (1998). Moreover, it has been observed that this algorithm is very effective in practice, especially when compared to other approximation algorithms Grossman & Wool (1997); Gomes et al. (2006). It often finds only a small percentage ($< 10\%$) more sets than the optimal solution. The standard greedy algorithm for the minimum dominating set problem is based on the same heuristic. It repeatedly picks a vertex which covers the maximum number of previously uncovered vertices until a dominating set is obtained. Its approximation ratio is $\ln\delta' + 2$, where δ' is the maximum degree of G. For the weighted set cover problem, the classic greedy algorithm iteratively selects a set by the number of uncovered items it contains per unit weight or the inverse. It achieves an approximation ratio of $\ln|U| + 1$, where $|U|$ is the cardinality of universe

U. The standard greedy algorithm for the minimum weighted dominating set problem is based on the same heuristic and achieves an approximation ratio of $\ln|V| + 1$ on graph $G = (V, E)$.

Besides the standard algorithms described above, there are many other approximation algorithms based on greedy heuristics for the weighted or unweighted set cover and minimum dominating set problems. Sanchis (2002) described four different approximation algorithms for the minimum dominating set problem and performed extensive experimental evaluations to compare them with the standard algorithm. Ablanedo-Rosas & Rego (2010) introduced a number of normalization rules to generate surrogate constraints for the weighted set cover problem. Although it's impossible to prove the approximation ratios for these algorithms, experimental evaluations showed that their performance was usually better than those of the standard greedy algorithms. Some other algorithms Cormode et al. (2010); Stergiou & Tsioutsiouliklis (2015); Eubank et al. (2004); Campan et al. (2015) are designed with the focus on efficiency. With more or less sacrifice on the performance, these algorithms can significantly reduce the processing time, especially for large instances.

In recent years, a great number of local search based approximation algorithms were proposed for the weighted or unweighted set cover problem Yagiura et al. (2006); Kinney et al. (2007); Caserta (2007); Lan et al. (2007); Bautista & Pereira (2007); Sundar & Singh (2012); Crawford et al. (2014); Mulati & Constantino (2011); Ren et al. (2010); Beasley & Chu (1996); Naji-Azimi et al. (2010); Y. Wang, Ouyang, et al. (2017), minimum weighted or unweighted dominating set problem Raka et al. (2010); Potluri & Singh (2013); Nitash & Singh (2014); Chaurasia & Singh (2015); Bouamama & Blum (2016); Y. Wang, Cai, & Yin (2017); Hedar & Ismail (2012, 2010); Ho et al. (2006). These algorithms usually achieve good results on small or medium sparse instances. However, they are still too complex to process medium dense or large instances. In fact, within a reasonable amount of time, most of them cannot achieve better results than some of the greedy approximation algorithms on medium dense or large instances.

I propose novel algorithms for the weighted set cover and minimum weighted dominating set problems.

The proposed algorithms are based on a three-step strategy. For the weighted set cover problem, in the first step, we reserve the sets indispensable for the optimal solution. A set is defined as indispensable if it covers at least one item on its own. Then we remove the items covered by the reserved sets and the sets whose items are all covered (include but not limited to the reserved sets) to get a smaller set cover instance. In the second step, we build a robust solution with a novel greedy heuristic. Specifically, we seek the set covering by iteratively selecting a set according to a measure which integrates the related weights, the coverage gain for the current iteration and the global coverage capacity of each set. It favors the sets that have smaller weights and better extend or consolidate the coverage, especially on the items that are contained in less sets. Since the obtained solution tends to have a robust coverage, in the third step, we further improve it by removing the redundant sets in an efficient way. The remaining sets and those reserved in the first step together constitute the final approximate solution.

For the minimum weighted dominating set problem, we reserve the indispensable vertices in the first step. A vertex is defined as indispensable if it is isolated or adjacent to at least one vertex of degree exactly one. Then we convert it into a weighted set cover problem by considering the vertex set V as the universe U, and $\mathcal{F} = \{S_1, S_2, ..., S_n\}$ where set S_v, with the same weight as vertex v, consists of v and all its adjacent vertices in G, as the set family. The items covered by the sets corresponding to the reserved vertices and the sets whose items are all covered are removed. We then solve the reduced weighted set cover problem. The vertices corresponding to the sets of the final set covering and the vertices reserved in the first step constitute the final dominating set.

Extensive experimental evaluations on a large number of synthetic and real world instances from many domains demonstrate the superiority of my algorithms over state-of-the-art.

1.2 Related Work

The classic greedy algorithm for set cover problem is due to Johnson (1974); Lovász (1975); Chvatal (1979). It repeatedly picks the set that contains the largest number of uncovered items until all items in the universe are covered. The same heuristic is then applied to the minimum dominating set problem, because the two problems are equivalent under L-reductions.

Ablanedo-Rosas & Rego (2010) introduced a number of normalization rules to improve the classic greedy algorithm for the weighted set cover problem, which iteratively selects a set by the number of uncovered items it contains per unit weight or the inverse. The essential idea of these rules is to take into account the number of sets each item is contained in when computing the coverage gain of each set, instead of simply using the number of uncovered items it contains. For example, *Rule A* assigns each item a weight which is defined as the inverse of the number of sets containing it. The coverage gain of each set is defined as the sum of such weights of the uncovered items it contains. The rest rules introduced in Ablanedo-Rosas & Rego (2010) are simple variants of *Rule A*. Experimental evaluations demonstrate the effectiveness of these normalization rules, especially when solving large scale and real-world instances. However, some of these rules, such as the "Adjusted" versions and *Rule C*, not only lack theoretical justifications, but also make very little difference on performance in comparison to the other rules.

Cormode et al. (2010) proposed an efficient set cover algorithm for processing very large data sets, especially those resident on disk. All the sets are first partitioned into sub-collections based on their sizes. Then starting from the sub-collection with the largest sets, each set is iteratively processed. In each iteration, if the set contains more uncovered items than the size threshold of current sub-collection, it is selected and its uncovered items are covered. Otherwise, the set is moved to the appropriate sub-collection.

Most practical approximation algorithms for the minimum weighted dominating set problem on general graphs are based on the local search techniques. Among them, the CC^2FS algorithm proposed in Y. Wang, Cai, & Yin (2017) is probably the best one in terms of both performance and efficiency. CC^2FS is based on two new ideas. The first idea is a new variant of the Configuration Checking Cai et al. (2011) strategy, which has been widely applied to many combinatorial optimization problems in order to reduce the cycling phenomenon in local search. The new variant defines the configuration of a vertex v to be its two-level neighborhood, which is the union of the neighborhood $N(v)$ and the neighborhood of each vertex in $N(v)$. The second idea is a frequency based scoring function for vertices, according to which the score of each vertex is calculated. Another state-of-the-art local search algorithm is *ACO-PP-LS* proposed in Potluri & Singh (2013). It uses an ant colony optimization method by considering the pheromone deposit on the node and a preprocessing step immediately after pheromone initialization. Both CC^2FS and *ACO-PP-LS* can achieve good results on small or medium graphs. However, as the other local search algorithms, they are still too complex to process large graphs.

For solving the minimum dominating set problem, Sanchis (2002) described five different approximation algorithms and performed extensive experimental analyses on them. The first algorithm, "Greedy", is the well-known and standard approximation algorithm. It starts with an empty set D and iteratively adds a vertex to D which covers the maximum number of previously uncovered vertices. The second algorithm, "Greedy_Rev", works in the opposite way. Initially, D contains every vertex in the graph. At each iteration, the vertex, which is of the smallest degree and does not uniquely cover any vertex, is removed from D. The third algorithm, "Greedy_Ran", is similar to the "Greedy" algorithm, with the only difference being that in each iteration it selects the vertex to be added probabilistically according to the number of additional vertices it would cover. The fourth algorithm, "Greedy_Vote", defines a measure "vote" for each vertex v as $vote(v) = 1/(1 + degree(v))$. At each iteration, instead of selecting the vertex which covers the maximum number of uncovered vertices as the "Greedy" algorithm does, it selects the vertex which is of the maximum sum of votes from the uncovered vertices. The fifth algorithm, "Greedy_Vote_Gr", performs an exhaustive search after running "Greedy_Vote" to determine whether it is possible to remove any two vertices from D, and replace them with either one or no vertices, while still retaining a dominating set. Experimental evaluations on many small synthetic graphs demonstrate that, in comparison to the standard "Greedy" algorithm, "Greedy_Vote" exhibits superiority on some graphs, while "Greedy_Ran" and "Greedy_Rev" are found not to be worthwhile. The exhaustive local search step of "Greedy_Vote_Gr" is very time-consuming and the improvements on results are limited. Eubank et al. (2004) proposed a "FastGreedy" heuristic for determining the minimum dominating set in the context of studying the algorithmic and structural properties of very large realistic social contact networks. It also defined a "VRegularGreedy" approximation algorithm, which adds the location neighbors of the people vertices of degree one into the dominating set before applying the standard greedy algorithm. Campan et al. (2015) introduced two efficient approximation algorithms for the minimum dominating set problem. The first algorithm attempts to improve the running time of the standard greedy algorithm by removing all the covered vertices from the remaining graph in each iteration. The second algorithm first adds neighbors of the vertices of degree one into the dominating set. After removing all the covered vertices, the first algorithm is applied to the remaining graph. There are a few works searching for the minimum dominating set based on meta-heuristics, such as simulated annealing Hedar & Ismail (2012), genetic algorithm Hedar & Ismail (2010) and ant colony optimization Ho et al. (2006). As a typical example, Hedar & Ismail (2012) proposed a simulated annealing based local search algorithm for solving the minimum dominating set problem. At each step, depending on whether the current solution is a dominating set, a trial solution is generated by removing, adding or replacing a vertex in a probabilistic manner according to the degree of vertices. Simulated annealing technique is used to avoid entrapments in poor local optima.

1.3 Our Work

In this section, I present my three-step framework of approximation algorithms for the weighted set cover and the minimum weighted dominating set problems.

1.3.1 Weighted Set Cover Problem

Given the item universe $U = \{i_1, i_2, ..., i_m\}$, the set family $\mathcal{F} = \{S_1, S_2, ..., S_n\}$ whose union equals U and the positive weight $w(S)$ on each set S, we aim to find the sub-collection of $\mathcal{F}$ whose union equals U and the sum of set weights is minimized.

In the first step, we identify every set in $\mathcal{F}$ which covers at least one item in U on its own. Obviously, these sets are indispensable for the optimal set covering. Let $\mathcal{F}_R$ denote the set of these indispensable sets. We remove the items covered by the sets in $\mathcal{F}_R$. Let U' denote the remaining uncovered items. Then we remove the empty sets, which do not contain any items in U'. Let $\mathcal{F}'$ denote the remaining sets. Obviously, $\mathcal{F}_R \subseteq \mathcal{F} \setminus \mathcal{F}'$. Now we have a smaller weighted set cover problem: to find the sub-collection of $\mathcal{F}'$ whose union equals the universe U' and the sum of set weights is minimized.

In the second step, we start with an empty set $\mathcal{F}'_C = \emptyset$ and iteratively select a set from $\mathcal{F}'$ into $\mathcal{F}'_C$ until the union of sets in $\mathcal{F}'_C$ equals U'.

Let $wi(S)$ denote the inverse of weight $w(S)$ of set S. That is,

$$wi(S) = \frac{1}{w(S)} \tag{1.3-1}$$

For each item i, we compute the sum of weight inverses of the sets containing i

$$SIW_i = \sum_{S \in C_i} wi(S) \tag{1.3-2}$$

where C_i denotes the set of sets in $\mathcal{F}'$ which contain item i.

Then for each set S in $\mathcal{F}'$, we define its local coverage efficiency on each item $i \in S$ as

$$LCE_{S,i} = \frac{wi(S)}{SIW_i} \tag{1.3-3}$$

Obviously, LCE is in range $(0, 1]$

Based on the local coverage efficiencies, we define the global coverage capacity of each set S as

$$GCC_S = \sum_{i \in S} (LCE_{S,i})^{\beta_g} \tag{1.3-4}$$

where $\beta_g \in [0, +\infty)$ is a parameter for adjusting the difference of local coverage efficiencies. $\beta_g > 1$ increases the relative weight of larger LCE, while $\beta_g < 1$ works in the opposite way.

We define the local coverage gain of each set S in each iteration as

$$LCG_S = \sum_{i \in S \cap U'_u} (LCE_{S,i})^{\beta_l} \tag{1.3-5}$$

where U'_u denotes the set of uncovered items as of the current iteration, the parameter $\beta_l \in [0, +\infty)$ is similar to β_g for adjusting the difference of local coverage efficiencies.

Based on the local coverage gain and global coverage capacity, we define the

Figure 1.1: An illustration of effects of the proposed greedy heuristic, in comparison to other greedy heuristics

	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	......	LCG	GCC	CB	IW	RA'
S_1	1									2/3	2/3	**4/3**	**1**	**1/2**
S_2	2									1/3	1/3	**2/3**	**1/2**	**1/4**
S_3		1								3/4	3/4	**3/2**	**1**	**1/2**
S_4			1							2/5	2/5	**4/5**	**1**	**1/3**
S_5				1	1					7/6	7/6	**7/3**	**2**	**1**
S_6						1	1	1		2/3	157/105	**227/105**	**1**	**1/2**
S_7			1		1					9/10	9/10	**9/5**	**2**	**5/6**
S_8			2	2		2	2			13/15	16/15	**29/15**	**3/2**	**2/3**
S_9		3						3		1/4	11/28	**9/14**	**1/3**	**1/6**
S_{10}							1	1						
......														
SIW	3/2	4/3	5/2	3/2	2	3/2	5/2	7/3						

measure of coverage benefit for each set S as

$$CB_S = LCG_S + \gamma \times GCC_S \tag{1.3-6}$$

where the parameter $\gamma \in [0, +\infty)$ is for adjusting the relative weight of GCC.

We iteratively select sets according to their coverage benefits. Specifically, in each iteration, we select the set S^* which has the maximum coverage benefit (with ties broken randomly) and covers at least one uncovered item.

$$S^* = argmax_S CB_S \quad s.t. \ |S \cap U'_u| > 0 \tag{1.3-7}$$

Obviously, this greedy heuristic favors the sets that have smaller weights and better extend or consolidate the coverage, especially on the items that are contained in less sets. Its specific effects are illustrated in Figure 1.1.

Figure 1.1 shows a state during the iterative process of solving a weighted set cover problem, in which $|\mathcal{F}| > 10$ and $|U| > 8$. Suppose the first 10 sets S_1 to S_{10} only contain some of the first 8 items, i.e. i_1 to i_8. S_{10}, which is marked in gray, is already selected, while S_1 to S_9 are not. The 8 items are only contained in some of these 10 sets. i_7 and i_8, which are marked in gray, are already covered by S_{10}, while i_1 to i_6 are not covered yet. A cell (S, i) is empty if the set S does not contain the item i. Otherwise, S contains i and cell (S, i) shows the set weight $w(S)$. Suppose $\beta_l = 1, \beta_g = 1, \gamma = 1$, we calculate SIW, LCG, GCC and CB for our greedy heuristic. In addition, we calculate the measures for selecting sets according to the classic greedy heuristic and the *Rule A* proposed in Ablanedo-Rosas & Rego (2010). The other rules proposed in Ablanedo-Rosas & Rego (2010) , i.e. *Rule B* to *Rule I*, have essentially the same effects as *Rule A*.

The classic greedy heuristic selects the set with the maximum number of uncovered items per unit weight (IW) or the minimum of its inverse. The IW is defined as

$$IW_S = \frac{|S \cap U'_u|}{w(S)} \tag{1.3-8}$$

where U'_u denotes the set of uncovered items as of the current iteration.

Let RA denote the measure derived from *Rule A*. It is defined as

$$RA_S = \frac{w(S)}{\sum_{i \in S \cap U'_u} \frac{1}{|C_i|}} \tag{1.3-9}$$

where C_i denotes the set of sets which contain item i.

In Ablanedo-Rosas & Rego (2010), *Rule A* selects the set with the minimum RA. We implement *Rule A* to select the set with the maximum $RA' = 1/RA$ for consistency.

Figure 1.1 illustrates five intuitive rules for selecting sets. **(1)** S_1 and S_2 contain the same uncovered item i_1 and $w(S_1) < w(S_2)$. Apparently, between them, S_1 should be selected. We call this *Rule* 1. The three measures CB, IW and RA' all implement *Rule* 1. **(2)** S_1 and S_3 each contains 1 uncovered item and both have the same weights. Besides S_1, selecting S_2 can cover the item i_1. Similarly, besides S_3, selecting S_9 can cover i_2. Since $w(S_2) < w(S_9)$, between S_1 and S_3, intuitively, we

Table 1.1: Relationships between measures and rules

	Rule 1	*Rule* 2	*Rule* 3	*Rule* 4	*Rule* 5
CB	✓	✓	✓	✓	✓
IW	✓			✓	
RA'	✓		✓	✓	

should favor S_3. Otherwise, it is more risky that S_9 may be selected to cover i_2 later. We call this *Rule* 2. Obviously, only the measure CB implements *Rule* 2. **(3)** S_1 and S_4 each contain 1 uncovered item and both have the same weights. Unlike S_9 on i_2, there are no sets that contain i_3 and have larger weights than the sets containing i_1. Apparently, *Rule* 2 does not apply here. However, in comparison to i_1, there are more sets containing i_3. Intuitively, i_3 is easier to be covered. Therefore, we should favor S_1 over S_4 this time, because S_1 covers a more "difficult" item. We call this *Rule* 3. Both CB and RA' implement *Rule* 3. **(4)** S_1 and S_5 have the same weights. But, since S_5 contains 2 uncovered items, while S_1 only contains 1, it is straightforward for us to choose S_5. We call this *Rule* 4. CB, IW and RA' all implement *Rule* 4. **(5)** S_1 and S_6 have the same weights. S_1 contains 1 uncovered item i_1, while S_6 also contains 1 uncovered item i_6. Moreover, i_1 and i_6 are both contained in 2 sets and the set weight patterns are exactly the same. However, S_6 contains 2 covered items, i.e. i_7 and i_8. Although selecting S_6 does not bring more coverage than selecting S_1, it can consolidate the existing coverage on i_7 and i_8. It is possible that such consolidations will help release some previously selected sets. Therefore, we favor S_6 over S_1 this time. We call this *Rule* 5. Obviously, only the measure CB implements *Rule* 5.

The relationships between the three measures CB, IW and RA', and the five rules are summarized in Table 1.1. Our measure CB implements all the five rules, while IW and RA' implement only 2 and 3 rules respectively. The individual effect of these abstract rules may not affect the final result in the simple example above, but combining their effects together can make a difference when solving the real problems.

With the proposed greedy heuristic, the obtained set covering tends to be robust, which means each item is likely to be covered by more sets. Therefore, it has potential to be further improved.

In the third step, we remove the redundant sets in an efficient way. Specifically, we first identify every set in ${\mathcal{F}'}_C$ which covers at least one item in U' on its own. The other sets are considered to be potentially redundant and removed from ${\mathcal{F}'}_C$. If the remaining sets in ${\mathcal{F}'}_C$ do not cover the entire U' any more, we iteratively select a set with the maximum number of uncovered items per unit weight, i.e. IW, from those potentially redundant sets and add it back to ${\mathcal{F}'}_C$, until the union of sets in ${\mathcal{F}'}_C$ equals to U'. Finally, the sets in ${\mathcal{F}'}_C$ and those reserved in $\mathcal{F}_R$ together constitute the final solution.

$$\mathcal{F}^* = {\mathcal{F}'}_C \cup \mathcal{F}_R \tag{1.3-10}$$

1.3.2 Minimum Weighted Dominating Set

Given an undirected, vertex-weighted graph $G = (V, E, w)$, where V is the set of vertices, E is the set of edges, and $w : V \to \mathbb{R}^+$ is a function that associates a positive weight $w(v)$ to each vertex $v \in V$, we aim to find a subset $D \subseteq V$ of vertices such that each vertex $v \in V$ is either in D or has at least one neighbor in D, and the sum of weights is minimized.

In the first step, we identify and reserve a subset $V_R \subseteq V$ of vertices indispensable for the optimal solution. If G contains isolated vertices with degree zero, obviously, these vertices should belong to V_R. In addition, we find all vertices with degree one. Each such vertex v has only one direct neighbor u. If $w(v) \geq w(u)$, u should be added into V_R. Otherwise, v itself must be added into V_R. If so, in the end, we can get a better dominating set with smaller or equal weight by simply replacing v with u. Note that if u is also with degree one and $w(v) = w(u)$, we need to make sure only one of v and u is added into V_R. By considering the vertex set V as the universe U, and

$\mathcal{F} = \{S_1, S_2, ..., S_{|V|}\}$, where set S_v consists of the vertex v and all its adjacent vertices in G, as the set family, and $w'(S_v) = w(v)$ as the set weight function, we convert the minimum weighted dominating set problem into a weighted set cover problem. Let $\mathcal{F}_R$ denote the sets corresponding to the reserved vertices in V_R. We follow the same procedures as in our set cover algorithm to reduce the obtained weighted set cover problem and then solve it. In the end, the vertices corresponding to the sets in the final $\mathcal{F}'_C$ and the vertices reserved in V_R together constitute the approximate solution to the minimum weighted dominating set problem.

1.3.3 Theoretic Analysis

Our algorithm for the weighted set cover problem first reserves the sets indispensable for the optimal solution, and then reduces the problem size by removing the items covered by the reserved sets and the sets that do not contain any of the remaining items. This step can be finished in $\mathcal{O}(|\mathcal{F}||U|)$, where $|\mathcal{F}|$ is the set number and $|U|$ is the item number. For sparse set cover instances, it can significantly reduce the time consumed in the subsequent procedures. In the second step, we iteratively select sets according to their coverage benefits. The related values, including LCE, GCC and LCE^{β_l} can be calculated in $\mathcal{O}(|\mathcal{F}'||U'|)$ before the iterative procedure. Therefore, the complexity of each iteration of our algorithm is the same as that of the classic greedy algorithm. However, since our algorithm does not pursue the coverage on items directly as the classic greedy algorithm does, it needs more iterations to reach the full coverage. In the third step, our algorithm first removes the potentially redundant sets from the obtained solution and then uses the same heuristic of the classic greedy algorithm to select sets from those removed to restore the full coverage. In the worst case, it is to run the classic greedy algorithm on the instance with sets in $\mathcal{F}'_C$ and items in U'. However, since the number of potentially redundant sets is usually much smaller, this step is fast. The efficiency analysis on our minimum weighted dominating set algorithm is similar.

It is very difficult to give the approximation ratios for our weighted set cover and minimum weighted dominating set algorithms. However, when applied to the unweighted set cover and minimum dominating set problems, the approximation ratios of our algorithms can be guaranteed in a simple way. Specifically, our algorithm for the unweighted set cover problem has three parameters: β_l, β_g and γ. If we set $\beta_l = 0$ and $\gamma = 0$, it becomes a variant of the classic greedy approximation algorithm. The only difference is that this variant has a preprocessing and a post processing: the first step to reserve the indispensable sets and the third step to remove the redundant sets. Obviously, these two steps do not harm the final approximate solution. We know the approximation ratio of the classic greedy algorithm is $\ln \delta + 1$, where $\delta = max\{|S| : S \in \mathcal{F}\}$ is the maximum cardinality of sets in $\mathcal{F}$. Therefore, the approximation ratio of this variant is also $\ln \delta + 1$. Since our algorithm can try any parameters and return the best result, as long as it tries $\beta_l = 0$ and $\gamma = 0$, we can guarantee the approximation ratio of $\ln \delta + 1$. Similarly, our algorithm for the minimum dominating set problem can guarantee the approximation ratio of $\ln \delta' + 2$, where δ' is the maximum degree of G.

1.4 Experimental Evaluation

We evaluate the performance of our algorithms on a large number of synthetic and real world instances from many domains. There are 4 groups of experiments on weighted set cover problem, set cover problem, minimum weighted dominating set problem and minimum dominating set problem.

In these experiments, the three parameters β_l, β_g and γ of our algorithms are varied with grid search. In order to give full play to our algorithms and also generate sufficient data for the subsequent analyses on parameters, we try a large number of parameter combinations. In Section 1.4.5, we give some guidelines for choosing values of β_l, β_g and γ.

Our algorithms and most of the comparison algorithms break ties randomly. The other comparison algorithms also have more or less random processing. Therefore, we run each experiment 10 times and report the average results. If we need to report the best or average results across different parameters, we first run 10 experiments with each distinct set of parameters and compute the average results. Then we select the best or compute the average across different parameters.

In each of the 4 groups of experiments, we compare our algorithm with the classic or standard greedy algorithm for that problem and its improved version that first reserves the indispensable sets or vertices. We name them Gr and GrR uniformly in Sections 1.4.1, 1.4.2,1.4.3 and 1.4.4. It is easy to distinguish among them from the context.

Our algorithms and the comparison algorithms are all implemented in C++. All the experiments are performed on a workstation with 4x AMD Opteron 6174 2.2GHz processors and 64GB RAM.

1.4.1 Weighted Set Cover Problem

We evaluate the performance of our weighted set cover algorithm on 70 synthetic instances from the OR-Library Beasley (1990). The details of these test instances are summarized in Table 1.2. For example, the first instance "scp41" consists of 1000 sets and 200 items.

The optimal solutions of these instances are known. Their weights are given in the second columns of Table 1.3 and Table 1.4. For example, the weight of the optimal solution of the first instance "scp41" is 429.

The three parameters β_l, β_g and γ of our algorithm are varied with grid search. Specifically, we have 320 combinations derived from $\beta_l \in \{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\beta_g \in \{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\gamma \in \{0, 0.01, 0.1, 1, 10\}$. However,

Table 1.2: Test Instances for Weighted Set Cover Problem

Instance	#Set	#Item	Instance	#Set	#Item	Instance	#Set	#Item
scp41	1000	200	scpa1	3000	300	scpnre1	5000	500
scp42	1000	200	scpa2	3000	300	scpnre2	5000	500
scp43	1000	200	scpa3	3000	300	scpnre3	5000	500
scp44	1000	200	scpa4	3000	300	scpnre4	5000	500
scp45	1000	200	scpa5	3000	300	scpnre5	5000	500
scp46	1000	200	scpb1	3000	300	scpnrf1	5000	500
scp47	1000	200	scpb2	3000	300	scpnrf2	5000	500
scp48	1000	200	scpb3	3000	300	scpnrf3	5000	500
scp49	1000	200	scpb4	3000	300	scpnrf4	5000	500
scp410	1000	200	scpb5	3000	300	scpnrf5	5000	500
scp51	2000	200	scpc1	4000	400	scpnrg1	10000	1000
scp52	2000	200	scpc2	4000	400	scpnrg2	10000	1000
scp53	2000	200	scpc3	4000	400	scpnrg3	10000	1000
scp54	2000	200	scpc4	4000	400	scpnrg4	10000	1000
scp55	2000	200	scpc5	4000	400	scpnrg5	10000	1000
scp56	2000	200	scpd1	4000	400	scpnrh1	10000	1000
scp57	2000	200	scpd2	4000	400	scpnrh2	10000	1000
scp58	2000	200	scpd3	4000	400	scpnrh3	10000	1000
scp59	2000	200	scpd4	4000	400	scpnrh4	10000	1000
scp510	2000	200	scpd5	4000	400	scpnrh5	10000	1000
scp61	1000	200	scpe1	500	50			
scp62	1000	200	scpe2	500	50			
scp63	1000	200	scpe3	500	50			
scp64	1000	200	scpe4	500	50			
scp65	1000	200	scpe5	500	50			

when $\gamma = 0$, different β_g make no difference. Therefore, there are a total of 264 really distinct combinations. The reason that we do not try $\beta_l < 0.5$ or $\beta_l > 4$, and $\beta_g < 0.5$ or $\beta_g > 4$, is because such extreme values minify or magnify the difference of set weights and the difference of set populations covering each item too much, which adversely affects the solution quality. We analyze the effects of β_l, β_g and γ in Section 1.4.5.

The comparison algorithms include the classic greedy algorithm Gr and its improved version GrR, which first reserves the indispensable sets, and the variants $GrRA$, $GrRB$, $GrRD$ and $GrRE$ based on the rules *Rule A*, *Rule B*, *Rule D* and *Rule E* introduced in Ablanedo-Rosas & Rego (2010) respectively, which also first reserve the indispensable sets. We do not report the results of the other rules introduced in Ablanedo-Rosas & Rego (2010), including *Rule C*, *Rule F*, *Rule G*, *Rule H* and *Rule I*, because, (1) according to our experiments, these rules make very little difference on performance in comparison to the other rules; and (2) these rules are simple variants of the other rules, but lack theoretical justifications.

The experimental results of weighted set cover problem are given in Table 1.3 and Table 1.4. Since the results of Gr and GrR are identical on all the 70 instances, we only report those of GrR. Our results are the minimum sum of set weights of the

Table 1.3: Weighted Set Cover Results (Solution Weight), Part 1

Instance	Optimal	*GrR*	*GrRA*	*GrRB*	*GrRD*	*GrRE*	Ours
scp41	***429***	465.2	473	477	461	477	**434**
scp42	***512***	590	564	572	578	566	**528.3**
scp43	***516***	594.8	564	559	582	560	**527.2**
scp44	***494***	553.8	541	556	541	553	**503**
scp45	***512***	571	575	573	580	573	**514**
scp46	***560***	606	593	586	606	586	**567.6**
scp47	***430***	474.6	482	461	481	475	**437**
scp48	***492***	544.7	542	542	538	543	**496**
scp49	***641***	748.8	747	731	755	732	**664**
scp410	***514***	554.6	545	545	553	545	**521**
scp51	***253***	290.5	290	291	288	291	**262.2**
scp52	***302***	345.5	341	344	343	341	**314**
scp53	***226***	244.4	246	252	246	245	**229**
scp54	***242***	266.5	265	265	266	265	**245.5**
scp55	***211***	235.1	234	232	234	234	**212**
scp56	***213***	248.2	250	244	250	250	**221**
scp57	***293***	319.5	317	311	315	311	**299**
scp58	***288***	314.6	313	314	313	314	**294**
scp59	***279***	306.9	307	307	308	307	**280**
scp510	***265***	287.7	286	286	286	286	**273**
scp61	***138***	158.4	159	164	159	159	**140.8**
scp62	***146***	170.1	171	172	171	171	**151.5**
scp63	***145***	161	161	159	163	158	**149**
scp64	***131***	142	149	149	149	150	**132**
scp65	***161***	191.8	195	194	195	194	**171**
scpa1	***253***	285.3	279	282	279	284	**258**
scpa2	***252***	285.7	284	278	284	281	**259**
scpa3	***232***	263.3	264	270	264	262	**236**
scpa4	***234***	277.5	274	273	274	277	**237**
scpa5	***236***	269.2	262	261	264	261	**239.1**
scpb1	***69***	75.8	77	77	75	77	**71**
scpb2	***76***	86.8	84	86	91	86	**76**
scpb3	***80***	87	85	85	85	85	**81**
scpb4	***79***	90	89	89	89	89	**80**
scpb5	***72***	80.3	80	80	80	80	**72.4**

solutions found by our algorithm with all distinct sets of parameters each averaged over 10 runs.

As we can see, our algorithm achieves significantly better results than the comparison algorithms. Our results across different sets of parameters are much better than those of the comparison algorithms on 65 instances and tie with those on the other 5 instances. Furthermore, on many instances, our results are equal to or very close to the optimal. This demonstrates the capability of our algorithm to find superior solutions.

1.4.2 Set Cover Problem

Our algorithm can solve the (unweighted) set cover problem by simply considering all sets as having the same weights, e.g. 1. We evaluate its performance on 8 real world unweighted instances from the Frequent Itemset Mining Dataset Repository [1], which were used in Cormode et al. (2010). The details of these test instances are

[1] http://fimi.ua.ac.be/data/

Table 1.4: Weighted Set Cover Results (Solution Weight), Part 2

Instance	Optimal	*GrR*	*GrRA*	*GrRB*	*GrRD*	*GrRE*	Ours
scpc1	***227***	256.6	256	253	256	253	**235.3**
scpc2	***219***	254.1	253	253	253	250	**226**
scpc3	***243***	271.8	271	272	271	272	**251**
scpc4	***219***	259.3	258	257	258	257	**228**
scpc5	***215***	233.6	232	231	232	232	**218.1**
scpd1	***60***	70.3	71	69	71	71	**61.3**
scpd2	***66***	72.4	71	71	71	71	**67.5**
scpd3	***72***	81.1	82	82	82	82	**74**
scpd4	***62***	68.3	67	67	67	67	**62**
scpd5	***61***	70.4	69	70	69	70	**63**
scpe1	***5***	5.9	5	5	5	5	**5**
scpe2	***5***	5.1	5	6	5	6	**5**
scpe3	***5***	5	5	5	5	5	**5**
scpe4	***5***	5.2	5	5	5	5	**5**
scpe5	***5***	5	5	5	5	5	**5**
scpnre1	***29***	31.3	30	30	30	30	**29**
scpnre2	***30***	35.4	34	34	34	34	**32**
scpnre3	***27***	29.6	30	30	30	30	**28**
scpnre4	***28***	32.7	33	33	33	33	**29.3**
scpnre5	***28***	32.1	33	33	33	33	**29.1**
scpnrf1	***14***	16.3	16	16	16	16	**15**
scpnrf2	***15***	16	16	16	16	16	**15**
scpnrf3	***14***	15.2	16	16	16	16	**15**
scpnrf4	***14***	16.2	16	16	16	16	**15**
scpnrf5	***13***	15.7	15	15	15	15	**14**
scpnrg1	***176***	200.5	199	199	199	199	**183**
scpnrg2	***154***	173.8	174	171	174	176	**160**
scpnrg3	***166***	188.9	186	186	185	186	**174**
scpnrg4	***168***	192.4	194	186	194	194	**179**
scpnrg5	***168***	189.2	195	196	195	196	**175.7**
scpnrh1	***63***	73.9	73	73	73	73	**68.9**
scpnrh2	***63***	74.7	73	73	73	73	**67.1**
scpnrh3	***59***	68.9	68	66	68	68	**63.9**
scpnrh4	***58***	66	67	67	67	67	**62.6**
scpnrh5	***55***	62.7	61	61	61	61	**58**

Table 1.5: Test Instances for Set Cover Problem

Instance	#Set	#Item	Instance	#Set	#Item
chess	3196	75	retail	88162	16469
mushroom	8124	119	accidents	340183	468
pumsbStar	49046	7116	kosarak	990002	41270
pumsb	49046	7116	webdocs	1692082	5267656

summarized in Table 1.5. For example, the first instance "chess" consists of 3196 sets and 75 items.

For our algorithm, different from the experiment on weighted set cover problem, we now also consider $\beta_l < 0.5$ and $\beta_g < 0.5$ for two reasons. First, there are no differences among set weights. The adverse effect of extreme values of β_l and β_g, which is discussed in Section 1.4.1, is smaller. Second, as discussed in Section 1.3.3, by trying $\beta_l = 0$ and $\gamma = 0$, our algorithm can guarantee an approximation ratio. Therefore, we have 500 parameter combinations derived from $\beta_l \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\beta_g \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\gamma \in \{0, 0.01, 0.1, 1, 10\}$. When $\gamma = 0$, different β_g make no difference. Therefore,

Table 1.6: Set Cover Results (Set Number)

Instance	Gr	GrR	$GrRA$	$GrRB$	$GrRD$	$GrRE$	DFG	Ours
chess	8.2	8	**6**	**6**	7	**6**	8.2	**6**
mushroom	24.6	24.6	**22**	**22**	**22**	**22**	23.1	**22**
pumsbStar	749.8	711.6	649.6	655.6	650.8	652.8	745.8	**644.5**
pumsb	749.9	708	650	654.9	650.4	652.1	748.3	**644.3**
retail	5126.4	4951.6	4779.3	4786.1	4811.8	4784.6	5113.2	**4763.1**
accidents	181.2	169.7	**160**	161	**160**	161	179.7	**160**
kosarak	17761.4	17691	17584.7	17588.4	17585.7	17589.4	17735.9	**17555.9**
webdocs	406429	405556	405516	405510	405522	405515	406337	**405475.6**

there are a total of 410 really distinct combinations.

The comparison algorithms include the classic greedy algorithm Gr for the set cover problem and its improved version GrR, which first reserves the indispensable sets, $GrRA$, $GrRB$, $GrRD$ and $GrRE$ based on the rules *Rule A*, *Rule B*, *Rule D* and *Rule E* Ablanedo-Rosas & Rego (2010), which also first reserve the indispensable sets, and DFG proposed in Cormode et al. (2010). DFG algorithm has a parameter p, which governs both the theoretical approximation factor and running time. In Cormode et al. (2010), p was set to be 1.001 and 1.05 when testing DFG on the same 8 instances. For fair comparison, in addition to 1.001 and 1.05, we vary p with grid search in the range of $[1.005, 1.1]$ with a step size of 0.005, and report the best results.

As shown in Table 1.6, the performance of our algorithm is better than those of the comparison algorithms. Specifically, our algorithm achieves the best results on all the 8 instances. On chess, mushroom and accidents instances, which contain fewer items, some of the algorithms based on the normalization rules of Ablanedo-Rosas & Rego (2010) can also achieve the best results. Their performance on the other instances are also very good in comparison to Gr, GrR and DFG.

1.4.3 Minimum Weighted Dominating Set Problem

We evaluate the performance of our algorithm for the minimum weighted dominating set problem on three benchmarks. The first one is the BHOSLIB benchmark ("mis" version) Xu et al. (2005), which consists of 41 graphs. The second one is the DIMACS complementary benchmark, which consists of 37 graphs. The third one

Table 1.7: BHOSLIB Benchmark ("mis" version)

Graph	#Vertex	#Edge	Graph	#Vertex	#Edge	Graph	#Vertex	#Edge
frb30-15-1	450	17827	frb45-21-1	945	59186	frb56-25-1	1400	109676
frb30-15-2	450	17874	frb45-21-2	945	58624	frb56-25-2	1400	109401
frb30-15-3	450	17809	frb45-21-3	945	58245	frb56-25-3	1400	109379
frb30-15-4	450	17831	frb45-21-4	945	58549	frb56-25-4	1400	110038
frb30-15-5	450	17794	frb45-21-5	945	58579	frb56-25-5	1400	109601
frb35-17-1	595	27856	frb50-23-1	1150	80072	frb59-26-1	1534	126555
frb35-17-2	595	27847	frb50-23-2	1150	80851	frb59-26-2	1534	126163
frb35-17-3	595	27931	frb50-23-3	1150	81068	frb59-26-3	1534	126082
frb35-17-4	595	27842	frb50-23-4	1150	80258	frb59-26-4	1534	127011
frb35-17-5	595	28143	frb50-23-5	1150	80035	frb59-26-5	1534	125982
frb40-19-1	760	41314	frb53-24-1	1272	94227	frb100-40	4000	572774
frb40-19-2	760	41263	frb53-24-2	1272	94289			
frb40-19-3	760	41095	frb53-24-3	1272	94127			
frb40-19-4	760	41605	frb53-24-4	1272	94308			
frb40-19-5	760	41619	frb53-24-5	1272	94226			

Table 1.8: DIMACS Complementary Benchmark

Graph	#Vertex	#Edge	Graph	#Vertex	#Edge	Graph	#Vertex	#Edge
C1000.9	1000	49421	brock200_4	200	6811	keller5	776	74710
C125.9	125	787	brock400_2	400	20014	keller6	3361	1026582
C2000.5	2000	999164	brock400_4	400	20035	p_hat1500-1	1500	839327
C2000.9	2000	199468	brock800_2	800	111434	p_hat1500-2	1500	555290
C250.9	250	3141	brock800_4	800	111957	p_hat1500-3	1500	277006
C4000.5	4000	3997732	gen200_p0.9_44	200	1990	p_hat300-1	300	33917
C500.9	500	12418	gen200_p0.9_55	200	1990	p_hat300-2	300	22922
DSJC1000.5	1000	249674	gen400_p0.9_55	400	7980	p_hat300-3	300	11460
DSJC500.5	500	62126	gen400_p0.9_65	400	7980	p_hat700-1	700	183651
MANN_a27	378	702	gen400_p0.9_75	400	7980	p_hat700-2	700	122922
MANN_a45	1035	1980	hamming10-4	1024	89600	p_hat700-3	700	61640
MANN_a81	3321	6480	hamming8-4	256	11776			
brock200_2	200	10024	keller4	171	5100			

consists of all the 139 undirected simple graphs in the Network Data Repository [2]. These real world graphs are from 12 different domains, including biological networks, collaboration networks, facebook networks, infrastructure networks, interaction networks, recommendation networks, retweet networks, scientific computing networks, social networks, technological networks, temporal networks and web link networks. The details of the graphs from BHOSLIB benchmark and DIMACS complementary benchmark are summarized in Table 1.7 and Table 1.8, while those of the graphs from Network Data Repository are given in Table A.1 in Appendix A. For example, the first graph "frb30-15-1" in the BHOSLIB benchmark consists of 450 vertices and 17827 edges.

All these graphs are originally unweighted. We follow the method of Y. Wang, Cai, & Yin (2017) to assign weights to the vertices. The weighting function is defined as $w(v_k) = (k \mod 200) + 1$, where k is the vertex index.

In the experiments of minimum weighted dominating set problem, as in Section

[2]http://networkrepository.com

1.4.1, we consider 320 parameter combinations derived from $\beta_l \in \{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\beta_g \in \{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\gamma \in \{0, 0.01, 0.1, 1, 10\}$. When $\gamma = 0$, different β_g make no difference. Therefore, there are a total of 264 really distinct combinations.

We compare our minimum weighted dominating set algorithm with the standard greedy algorithm *Gr* and its improved version *GrR*, which first reserves the indispensable vertices. Furthermore, in order to evaluate the quality of our results, we compare them to those of two state-of-the-art local search algorithms, CC^2FS Y. Wang, Cai, & Yin (2017) and *ACO-PP-LS* Potluri & Singh (2013). We are not able to obtain the source code of these two algorithms. However, since the settings of our experiments are the same as Y. Wang, Cai, & Yin (2017), we can compare our results to those reported in Y. Wang, Cai, & Yin (2017) directly. Note that in Y. Wang, Cai, & Yin (2017), the "MIN" and "AVG" results are the minimal and average solution values of the 10 runs of experiment with different random seeds. Here our results correspond to the "AVG" results in Y. Wang, Cai, & Yin (2017). On some graphs, there are no results reported in Y. Wang, Cai, & Yin (2017). We mark them as "N/A". In the experiments of Y. Wang, Cai, & Yin (2017), the time limit for CC^2FS and *ACO-PP-LS* was 1000 seconds. On some graphs, *ACO-PP-LS* failed to find a dominating set within the time limit. The results were marked as "n/a". We keep these marks here.

The experimental results of minimum weighted dominating set problem are given in Table 1.9 and Table 1.10 in this section, and Table A.2 and Table A.3 in Appendix A. On BHOSLIB and DIMACS complementary benchmarks, since the results of *Gr* and *GrR* are identical on all the graphs, we only report those of *GrR*. Our results are the minimum sum of vertex weights of the solutions found by our algorithm with all distinct sets of parameters each averaged over 10 runs.

As we can see, our results are significantly better than those of *GrR*. Moreover,

Table 1.9: Minimum Weighted Dominating Set Results on BHOSLIB Benchmark (Solution Weight)

Graph	*GrR*	*ACO-PP-LS*	CC^2FS	Ours	Graph	*GrR*	*ACO-PP-LS*	CC^2FS	Ours
frb30-15-1	259	223.5	214	214	frb50-23-2	341	302.9	277	287.8
frb30-15-2	308	244	242	246	frb50-23-3	374	315.6	298.1	284
frb30-15-3	189	175	175	177	frb50-23-4	297	279	265	270.8
frb30-15-4	210	182.7	167	170	frb50-23-5	493	445.4	421.4	416
frb30-15-5	206	177.4	160	173.9	frb53-24-1	291	244	229	239.8
frb35-17-1	330	285.8	274	277	frb53-24-2	344.1	318.8	300.3	318
frb35-17-2	232	220.4	208	217	frb53-24-3	207	188.7	182	182
frb35-17-3	248	207	201	211	frb53-24-4	246	202.4	189	193
frb35-17-4	342	328.5	287	300	frb53-24-5	240	225.8	204	208.2
frb35-17-5	348	302.5	296.5	300	frb56-25-1	248	231.9	229	231
frb40-19-1	305	274.6	262	282	frb56-25-2	360	336	319	326
frb40-19-2	276	250.6	243.5	250	frb56-25-3	395	351.5	343.1	352
frb40-19-3	284	276.7	252	257	frb56-25-4	317.4	277.2	268	275
frb40-19-4	281	266.3	250	254.6	frb56-25-5	527.5	498.9	429.7	440
frb40-19-5	332	288.8	282.5	281	frb59-26-1	300	288.4	263.2	271
frb45-21-1	426	376.2	333.7	348	frb59-26-2	472	426.1	388.8	413.7
frb45-21-2	323	278.1	259.3	271	frb59-26-3	287	273.5	248	256
frb45-21-3	295	254.6	233.9	245	frb59-26-4	288	265.3	248.1	259
frb45-21-4	531	475.2	399	412.3	frb59-26-5	350.5	307.8	291.3	299
frb45-21-5	397	369.6	318.2	334	frb100-40	406	384.2	350	364
frb50-23-1	336	298.9	267.8	264					

Table 1.10: Minimum Weighted Dominating Set Results on DIMACS Complementary Benchmark (Solution Weight)

Graph	*GrR*	*ACO-PP-LS*	CC^2FS	Ours	Graph	*GrR*	*ACO-PP-LS*	CC^2FS	Ours
C1000.9	220	197	194.8	198	gen200_p0.9_55	462	439.7	433	434
C125.9	497	N/A	N/A	413	gen400_p0.9_55	307	303.6	288	286
C2000.5	11.6	10	10	10	gen400_p0.9_65	314	291.2	287	297
C2000.9	149.6	139.3	130	131	gen400_p0.9_75	382	307	307	307
C250.9	288	235	235	240	hamming10-4	101	88	86	86
C4000.5	10	9	9	9	hamming8-4	83	76.5	71	71
C500.9	245	226	228	226	keller4	253	233.1	220	229
DSJC1000.5	17	14.2	14	14	keller5	210	196.7	182	185
DSJC500.5	17	15	15	15	keller6	84	82.4	80	81
MANN_a27	406	405	405	405	p_hat1500-1	4	N/A	N/A	4
MANN_a45	1090	1080	1080	1080	p_hat1500-2	15.2	N/A	N/A	13.4
MANN_a81	3438.9	3402	3402	3402	p_hat1500-3	58	N/A	N/A	52
brock200_2	23	23	23	23	p_hat300-1	8	N/A	N/A	7
brock200_4	73	70.4	68	68	p_hat300-2	15	N/A	N/A	14
brock400_2	74	65	65	65	p_hat300-3	65	N/A	N/A	63
brock400_4	83	75.7	75	75	p_hat700-1	7	N/A	N/A	6
brock800_2	29	28.4	28	28	p_hat700-2	18	N/A	N/A	17
brock800_4	35	32.8	31	32	p_hat700-3	82	N/A	N/A	70
gen200_p0.9_44	492	458	470	461					

as an approximation algorithm based on greedy heuristic, our algorithm outperforms state-of-the-art local search based algorithms, which are much more complex and time-consuming. Specifically, our results are better than those of *ACO-PP-LS* on most graphs. On the small or medium graphs, our results are very close to those of CC^2FS. On large graphs, our algorithm exhibits significant advantage over CC^2FS. Our algorithm's superiority on efficiency is evaluated in Section 1.4.5.

1.4.4 Minimum Dominating Set Problem

Our algorithm can solve the minimum (unweighted) dominating set problem by simply considering all vertices as having the same weights, e.g. 1. We evaluate its performance on the 139 undirected simple graphs from the Network Data Repository.

In the experiments of minimum dominating set problem, as in Section 1.4.2, we consider 500 parameter combinations derived from $\beta_l \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\beta_g \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\gamma \in \{0, 0.01, 0.1, 1, 10\}$. When $\gamma = 0$, different β_g make no difference. Therefore, there are a total of 410 really distinct combinations.

The comparison algorithms include the standard greedy algorithm Gr and its improved version GrR which first reserves the indispensable vertices, the two algorithms $Alg.3$ and $Alg.4$ proposed in Campan et al. (2015), the $FastGreedy$ algorithm introduced in Eubank et al. (2004), the Greedy_Rev (Gr_Rev) and Greedy_Vote (Gr_Vote) algorithms described in Sanchis (2002), and the local search algorithm $SAMDS$ proposed in Hedar & Ismail (2012). We ignore the Greedy_Ran and Greedy_Vote_Gr algorithms described in Sanchis (2002), because according to both the experimental evaluations in Sanchis (2002) and our preliminary tests, the performance of Greedy_Ran is bad and instable, while the exhaustive local search step of "Greedy_Vote_Gr" is very time-consuming and the improvements over the Greedy_Vote algorithm are very limited. The $FastGreedy$ algorithm is implemented based on the FastGreedy heuristic proposed in Eubank et al. (2004). It first sorts the vertices in descending order by their degrees, as $\{v_1, v_2, ..., v_{|V|}\}$ with $d(v_1) \geq d(v_2) \geq ... \geq d(v_{|V|})$. Then the smallest index i is picked such that $\{v_1, v_2, ..., v_i\}$ is a dominating set. The final dominating set excludes vertices v_j which have $N[v_j] \subseteq \bigcup_{k<j} N[v_k]$, where $N[v]$ denotes vertex v and its direct neighbors. $SAMDS$ Hedar & Ismail (2012) is a local search algorithm based on the simulated annealing meta-heuristic. We follow the same strategy as in Hedar & Ismail (2012) to set the initial temperature T_{max} to be large enough to make the initial probability of accepting transition close to 1. In order to give full play to $SAMDS$, unlike in Hedar & Ismail (2012), we do not set a fixed final minimum temperature T_{min} as the termination criteria. Instead, we terminate $SAMDS$ if the

Table 1.11: Minimum Dominating Set Results on 23 Real World Graphs from Network Data Repository (Vertex Number)

Graph	#Vertex	#Edge	Gr	GrR	Gr_Rev	Gr_Vote	$SAMDS$	Ours
bio-celegans	453	2025	30.5	30.5	29	30	31	29
bio-yeast	1458	1948	359.1	353.6	356.9	355.4	359.6	353
ca-AstroPh	17903	196972	2179.2	2131.5	2153.7	2114.2	2220	2070
ca-netscience	379	914	55.8	55.8	55	56	59	55
socfb-A-anon	3097165	23667394	203464	201844	203077	201852	N/A	201698.6
socfb-uci-uni	58790782	92208195	865896.5	865676.5	865702	865684.5	58790782	865675
inf-power	4941	6594	1565.5	1507.2	1547.8	1514.7	1554.3	1487.1
inf-roadNet-PA	1087562	1541514	370808	347003	363593	346400.5	N/A	338740.6
ia-email-EU	32430	54397	755.2	755	755	755	755.8	755
ia-wiki-Talk	92117	360767	11952	11935	11952.1	11936.8	46626	11935
rec-amazon	91813	125704	30819.4	28775.9	29224.6	29064.8	57388.3	28365.7
rt-retweet	96	117	32	32	32	32	32.3	32
rt-twitter-copen	761	1029	201.3	199	199.3	200	200.9	199
sc-ldoor	952203	20770807	66709.2	66709.2	67363	65992.3	496162	65387.7
sc-shipsec5	179104	2200076	12670	12665.4	16586.5	12350.5	89572.7	12069.8
soc-BlogCatalog	88784	2093195	4899.9	4894	4901	4895	46114.3	4894
soc-youtube-snap	1134890	2987624	214184	213140	213581	213275.5	N/A	213122.1
tech-RL-caida	190914	607610	41465.6	40594.2	41559.8	40651.6	109468.6	40224.8
tech-routers-rf	2113	6632	488.6	480.7	486.7	482.1	487.1	479
scc-enron-only	151	9828	6	6	6	6	6.4	6
scc-twitter-copen	8580	473614	6413.6	6410.3	6412	6410	6416.6	6410
web-BerkStan	12305	19500	3053.6	3015.2	3052.3	3028.1	3072.7	3000
web-wikipedia2009	1864433	4507315	353065	348155	352885	348537.5	N/A	347018.1

current solution has not changed for H steps. We set $H = min(|V|, 10^5)$. Furthermore, we try 9 distinct pairs of cooling ratio λ and epoch length M derived from $\lambda \in \{0.99, 0.999, 0.9999\}$ and $M \in \{10, 100, 1000\}$. The best results achieved across different pairs of λ and M are reported. Although we implement $SAMDS$ in C++, due to the time-consuming computations when selecting vertices probabilistically according to their degrees in each iteration, $SAMDS$ is too slow to process large graphs. Therefore, in addition to the termination criteria described above, we set a cut-off time of 10^4 seconds to terminate its execution. On some graphs, $SAMDS$ fails to find a dominating set within the time limit, then the results are marked as "N/A".

We report the experimental results of minimum dominating set problem on the first and last graphs in alphabetical order from each of the 12 domains in Table 1.11 in this section. The complete results are given in Table A.4 and Table A.5 in Appendix A. $Alg.3$ Campan et al. (2015), $Alg.4$ Campan et al. (2015) and $FastGreedy$ Eubank et al. (2004) algorithms are designed with focus on the efficiency. Their results are much worse than those of the other comparison algorithms on all the graphs. Due to the limited space, we do not report their results. Our results are the minimum vertex number of the solutions found by our algorithm with all distinct sets of parameters each averaged over 10 runs.

As we can see, our algorithm achieves superior performance on all the 139 graphs. Specifically, on most graphs, our results are significantly better than those of all the comparison algorithms. On some small or sparse graphs, GrR and Gr_Vote tie with our algorithm. Gr_Rev achieves good results on a few graphs, but its overall performance is not as good as GrR, Gr_Vote and our algorithm. On some small or sparse graphs, the results of $SAMDS$ are close to those of GrR, Gr_Vote and our algorithm. On large or dense graphs, either $SAMDS$ can not return a valid dominating set within 10^4 seconds, or its results are much worse than ours and those of the other comparison algorithms.

1.4.5 Discussion

Parameter Setting

Our algorithms have three interactive parameters β_l, β_g and γ. Unlike the parameters in machine learning models, those parameters do not need to be trained. Any combinations of valid values can be tried independently. We only need to compare the final solutions and return the best one. As the cooling ratio and epoch length of simulated annealing based algorithms, there are no choices of β_l, β_g and γ that will be good for all problems, and there is no general way to find the best choices for a given problem. Therefore, in general, when using our algorithms, the three parameters β_l, β_g and γ should be set with grid search.

With modern distributed computing technologies, theoretically speaking, using our algorithms is as easy as using the classic greedy approximation algorithms. However, in practice, we hope we can achieve good results by trying as few parameter combinations as possible. In this section, we analyze the effects of the three parameters of our algorithms in order to obtain some guidelines for choosing their values.

We first analyze the effects of β_l, β_g and γ in theory. Both β_l and β_g are exponent parameters used to adjust the difference of local coverage efficiencies, i.e. LCEs. $\beta_l > 1$ and $\beta_g > 1$ increase the relative weight of larger LCE, while $\beta_l < 1$ and $\beta_g < 1$ works in the opposite way. For weighted set cover and minimum weighted dominating set problems, if β_l and β_g are too small, e.g. less than 0.5, the difference of LCEs, which actually represents the underlying difference of the set weights and the set populations covering each item, will be minified too much. When $\beta_l = 0$ and $\beta_g = 0$, those differences are completely ignored, and there are no differences among sets and no difference among items. On the contrary, if β_l and β_g are too large, the difference of LCEs will be magnified too much, which also adversely affects the solution quality. Therefore, we should avoid too small and too large values when choosing β_l and β_g. For (unweighted) set cover and minimum dominating set problems, since the set weights are identical, we can try small values for β_l and β_g, but it means the difference of set populations covering each item is more or less ignored. We should still avoid too large values for β_l and β_g. γ is used to adjust the relative weight of global coverage capacity GCC when combining it with the local coverage gain LCG to be the coverage benefit CB. Since the GCC of each set is computed over all the items, while the LCG is computed over only the currently uncovered items, with equal β_l and β_g, GCC is always greater than or equal to LCG. Their difference becomes larger in later iterations. Therefore, we should avoid too large values for γ in case GCC dominates CB.

Based on this analysis, we choose parameters with grid search in the previous experiments. Specifically, for weighted set cover and minimum weighted dominating set problems, we have 264 really distinct parameter combinations derived from $\beta_l \in \{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\beta_g \in \{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\gamma \in \{0, 0.01, 0.1, 1, 10\}$. For (unweighted) set cover and minimum dominating set problems, we have 410 really distinct parameter combinations derived from $\beta_l \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\beta_g \in \{0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$; $\gamma \in \{0, 0.01, 0.1, 1, 10\}$.

In this section, we investigate the results of the previous experiments to further analyze the effects of β_l, β_g and γ. For the weighted set cover problem, on each test instance, we obtain 264 results $r_1, r_2, r_3, ..., r_{264}$ by running our algorithm with the 264 distinct parameter combinations each averaged over 10 runs. Each result r denotes the sum of set weights of the corresponding solution. In order to aggregate such results across different test instances from multiple data sets to obtain some overall statistical measures, we first normalize each r. Let r_{max} and r_{min} denote the maximum and minimum results. Obviously, r_{max} corresponds to the worse solution, while r_{min} corresponds to the best solution. We normalize each result r to be

$$r' = \frac{r_{max} - r}{r_{max}} \times 100 \tag{1.4-11}$$

r' is actually the percentage of improvement of r over the worse result r_{max} on the same test instance. Obviously, r' is in the range of $[0, 100)$, and larger r' indicates better solution.

Since we run our algorithm with different parameter combinations and return the best results, it is worth examining the relationships between the best results and the individual parameter values. To this end, in addition to r', we normalize the result vector into an $0-1$ indicator vector, where 1 indicates the corresponding result r is equal to r_{min} and 0 indicates they are not equal.

Both the r' vector and the indicator vector can be aggregated across different test instances from different data sets. For the weighted set cover problem, we normalize the 264 results of our algorithm on each of the 70 test instances from the OR-Library data set. Then we stack the normalized 1×264 row vectors of r' and 0-1 indicators into an 70×264 matrix MR' of r' and an 70×264 indicator matrix MI.

For MR', we first calculate the mean value of each column to obtain an 1×264 row vector VR'. Then for each β_l in $\{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$, we calculate the mean value of its corresponding values in VR'. We call the obtained value as Average

Relative Solution Quality ($ARSQ$) for β_l. $ARSQ$s for β_l reflect the effects of different β_l on the solution quality. We calculate $ARSQ$ for β_g and γ in the same way.

For MI, we first calculate the mean value of each column to obtain an 1×264 row vector VI. Each element of VI is the frequency of our algorithm achieves the best results with the corresponding parameter combination. Then for each β_l in $\{0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4\}$, we calculate the mean value of its corresponding values in VI. We call the obtained value as Average Best Result Frequency ($ABRF$) for β_l. $ABRF$s reveal for which values of β_l the best results are more likely to be achieved. We calculate $ABRF$ for β_g and γ in the same way.

We analyze the effects of β_l, β_g and γ in the same way for the (unweighted) set cover, minimum weighted dominating set and minimum (unweighted) dominating set problems. For the minimum weighted dominating set problem, we aggregate the r' vector and the indicator vector across different graphs from the BHOSLIB benchmark, DIMACS benchmark and Network Data Repository. The analytical results are shown in Figure 1.2, Figure 1.3, Figure 1.4 and Figure 1.5. We use line charts to present the effects of β_l and β_g. For γ, since the trial values, which are different from those of β_l and β_g, are very unevenly distributed, we use separate bar charts to present its effects.

Apparently, the effects of β_l, β_g and γ are consistent for the 4 problems. Specifically, better performance is achieved with β_l in range $[0.5, 1]$. Beyond this range, especially with larger β_l, the performance becomes worse. For β_g, significantly better performance is achieved when its value is greater than 1. When β_g becomes even larger, the performance is improved less significantly. For γ, better performance is achieved with values less than 1. Note that the $ARSQ$ and $ABRF$ of $\gamma = 0$ are overestimated. It is because when $\gamma = 0$, different β_g make no difference. We exclude those redundant parameter combinations in the experiments. Therefore, without the less effective parameter combinations that result in worse performance, e.g. $\beta_l > 1$

Figure 1.2: Parameter Effect for Weighted Set Cover Problem

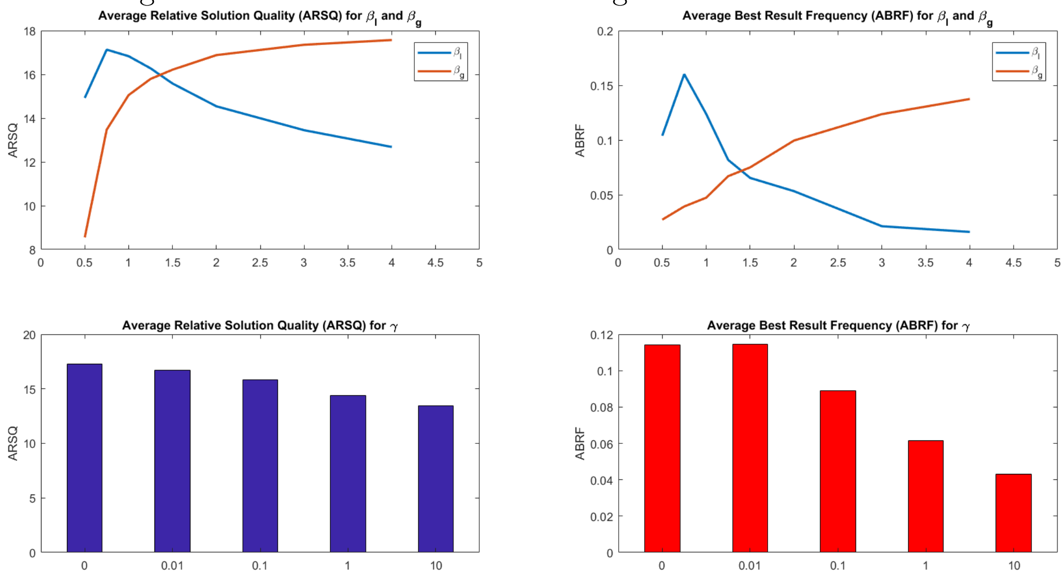


Figure 1.3: Parameter Effect for Set Cover Problem

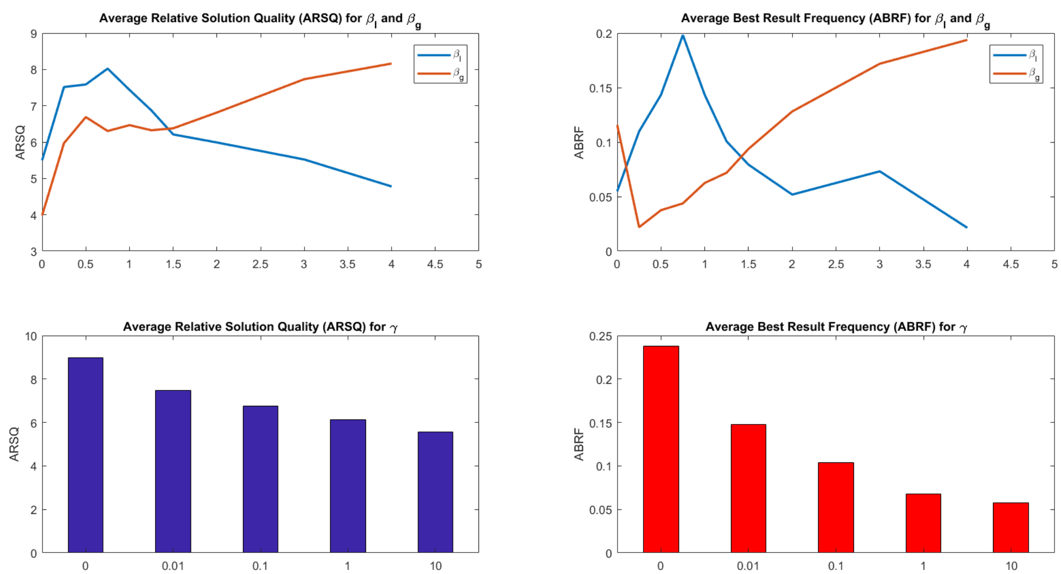


Figure 1.4: Parameter Effect for Minimum Weighted Dominating Set Problem

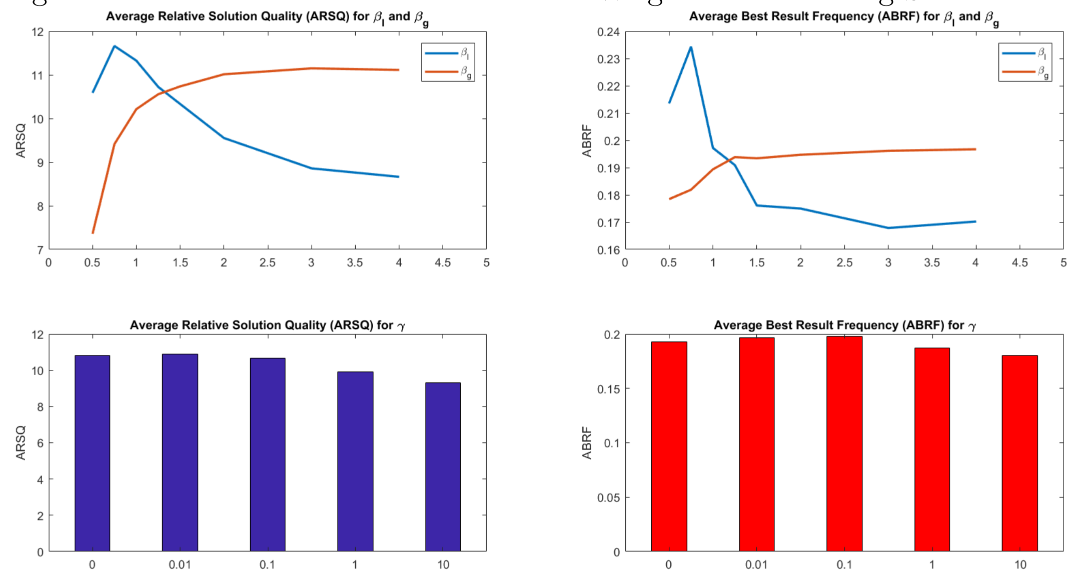
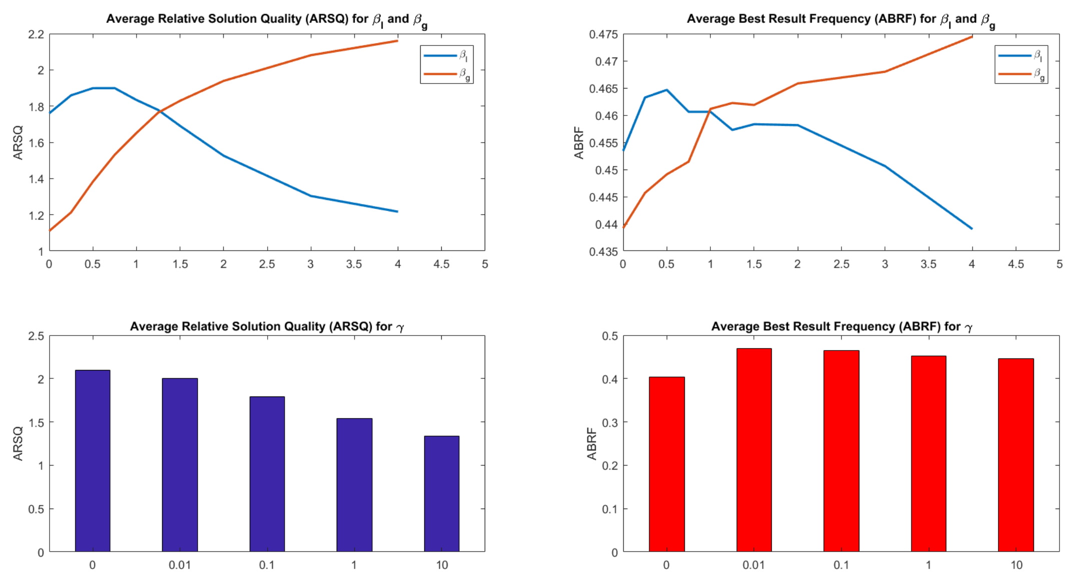


Figure 1.5: Parameter Effect for Minimum Dominating Set Problem



and $\beta_g < 1$, the $ARSQ$ and $ABRF$ of $\gamma = 0$, which represent the average performance over different parameter combinations, are higher than those of the other γ values.

Based these observations, we reduce the number of parameter combinations in our experiments. Specifically, we run our algorithms with three significantly smaller sets of parameter combinations. The first set is derived from $\beta_l \in \{0.5, 0.75, 1\}$; $\beta_g \in \{2, 3, 4\}$; $\gamma \in \{0, 0.1, 1\}$. Since different β_g make no difference when $\gamma = 0$, there are a total of 21 really distinct combinations. The second set consists of 8 combinations derived from $\beta_l \in \{0.75, 1\}$; $\beta_g \in \{2, 3\}$; $\gamma \in \{0.1, 1\}$. The third set contains only 1 combination. For weighted set cover and minimum weighted dominating set problems, it is $\beta_l = 1, \beta_g = 3, \gamma = 0.1$, while for set cover and minimum dominating set problems, it is $\beta_l = 0.75, \beta_g = 3, \gamma = 0.1$. The best results across different parameter combinations in each of these three sets are reported and compared to the results achieved with the original full sets of parameter combinations.

We run our algorithm for the weighted set cover problem on the 70 instances from OR-Library with the three smaller sets of parameter combinations. The results are given in Table 1.12.

As we can see, with 21 and 8 parameter combinations, the results of our algorithm ("Ours (21)" and "Ours (8)") are very close to those achieved with 264 parameter combinations. In comparison to the competitors' results reported in Table 1.3 and Table 1.4, "Ours (21)" and "Ours (8)" results are still much better. Specifically, "Ours (21)" and "Ours (8)" results are better than the competitors' results on 64 instances. Some competitors can only tie with our algorithm on the rest 6 instances. With the second set of only 1 parameter combination, the results of our algorithm ("Ours (1)") are still very good. On most instances, "Ours (1)" results are close to those achieved with 264 parameter combinations. Moreover, "Ours (1)" results are better than the competitors' results on 61 instances. Some competitors tie with our algorithm on 6 other instances. On the rest 3 instances, i.e. scpb2, scpnre1 and

Table 1.12: Weighted Set Cover Results with Different Sets of Parameter Combinations (Solution Weight)

Instance	Ours (264)	Ours (21)	Ours (8)	Ours (1)	Instance	Ours (264)	Ours (21)	Ours (8)	Ours (1)
scp41	434	434	434	434	scpc1	235.3	238.1	239	240
scp42	528.3	536	540	540	scpc2	226	226.1	226.1	229.6
scp43	527.2	527.2	527.2	529	scpc3	251	251	251	257
scp44	503	503	503	506	scpc4	228	233.9	233.9	234
scp45	514	518	518	518	scpc5	218.1	218.8	218.8	222
scp46	567.6	570.2	570.2	578	scpd1	61.3	62	62	62
scp47	437	439.6	439.6	447	scpd2	67.5	68	68	68
scp48	496	496	496	507.9	scpd3	74	76	76	78
scp49	664	664	664	664	scpd4	62	63	63	65
scp410	521	525	525	528	scpd5	63	63	63	67
scp51	262.2	263	263	268	scpe1	5	5	5	5
scp52	314	314	314	329	scpe2	5	5	5	5
scp53	229	229	229	231	scpe3	5	5	5	5
scp54	245.5	249	249	252	scpe4	5	5	5	5
scp55	212	213.7	213.7	215	scpe5	5	5	5	5
scp56	221	221	221	226	scpnre1	29	30	30	32
scp57	299	299	299	301	scpnre2	32	32	32	33
scp58	294	302	302.9	305	scpnre3	28	28	28	28
scp59	280	280	280	288	scpnre4	29.3	31	31	31
scp510	273	273.7	273.7	274	scpnre5	29.1	29.4	29.4	30
scp61	140.8	143	143	143	scpnrf1	15	15	15	15
scp62	151.5	154	155.3	164	scpnrf2	15	15	15	15
scp63	149	149	149	156	scpnrf3	15	15	15	15
scp64	132	132	132	136	scpnrf4	15	15	15	15
scp65	171	177.4	177.4	186	scpnrf5	14	14	14	15
scpa1	258	258	258	263	scpnrg1	183	183	183	187
scpa2	259	259	259	259	scpnrg2	160	160	160	167
scpa3	236	236	236	238	scpnrg3	174	174	177	177
scpa4	237	241	241	242	scpnrg4	179	180.9	180.9	185
scpa5	239.1	240	240	240	scpnrg5	175.7	178	178	183
scpb1	71	71	71	71	scpnrh1	68.9	69.9	69.9	70
scpb2	76	79	79	86	scpnrh2	67.1	67.4	67.4	70
scpb3	81	82	82	83	scpnrh3	63.9	63.9	63.9	67
scpb4	80	80	80	86	scpnrh4	62.6	63	63	63
scpb5	72.4	72.4	72.4	74	scpnrh5	58	59	59	60

Table 1.13: Set Cover Results with Different Sets of Parameter Combinations (Set Number)

Instance	Ours (410)	Ours (21)	Ours (8)	Ours (1)	Instance	Ours (410)	Ours (21)	Ours (8)	Ours (1)
chess	6	6	6	6	retail	4763.1	4765.4	4766.3	4770.3
mushroom	22	22	22	22	accidents	160	160	160	160
pumsbStar	644.5	644.5	644.5	644.6	kosarak	17555.9	17558.5	17558.7	17561.1
pumsb	644.3	644.3	644.3	644.3	webdocs	405475.6	405478.3	405481.5	405482.2

scpnrh3, "Ours (1)" results are slightly worse than those of some competitors, i.e. 86 vs 84, 32 vs 30 and 67 vs 66.

For the set cover problem, we run our algorithm on the 8 real world instances introduced in Table 1.5 with the three sets of parameter combinations.The results are given in Table 1.13. As we can see, with 21 and 8, or even just 1 parameter combination, the results of our algorithm are almost identical to those achieved with the full set of 410 parameter combinations.

For the minimum weighted dominating set problem and minimum dominating set problem, we run our algorithms on the 139 undirected simple graphs described in Table A.1 in Appendix A with three sets of parameter combinations. We report the

Table 1.14: Minimum Weighted Dominating Set Results on 23 Real World Graphs with Different Sets of Parameter Combinations (Solution Weight)

Graph	Ours (264)	Ours (21)	Ours (8)	Ours (1)	Graph	Ours (264)	Ours (21)	Ours (8)	Ours (1)
bio-celegans	1792.8	1828	1838	1838	rt-twitter-copen	15412	15420.1	15435	15639.5
bio-yeast	26305.6	26312	26312	26343.3	sc-ldoor	5443677	5444511	5444511	5448435
ca-AstroPh	135247.9	135416.9	135416.9	136131.1	sc-shipsec5	530423.7	532945.7	532945.7	534082.6
ca-netscience	4264.1	4264.3	4265	4319.3	soc-BlogCatalog	383529.3	383710.6	383776.7	384229.4
socfb-A-anon	17061460	17070800	17070800	17095100	soc-youtube-snap	16773990	16786150	16786370	16805080
socfb-uci-uni	84069030	84084270	84084270	84153010	tech-RL-caida	3143612	3144663	3144663	3153399
inf-power	122513.8	122800.5	122800.5	122973.4	tech-routers-rf	35652	35668.5	35668.5	35702
inf-roadNet-PA	28979500	28979500	28979500	29050040	scc-enron-only	761	761	761	761
ia-email-EU	72359	72367	72367	72408.1	scc-twitter-copen	629220.7	629247.9	629252.8	629274.6
ia-wiki-Talk	973320	974313.5	974326.9	975305.7	web-BerkStan	290274.8	290274.8	290368	290458.1
rec-amazon	2102511	2103468	2104804	2113931	web-wikipedia2009	26954720	26959580	26959580	27046630
rt-retweet	1162	1162	1162	1162					

Table 1.15: Minimum Dominating Set Results on 23 Real World Graphs with Different Sets of Parameter Combinations (Vertex Number)

Graph	Ours (264)	Ours (21)	Ours (8)	Ours (1)	Graph	Ours (264)	Ours (21)	Ours (8)	Ours (1)
bio-celegans	29	29	29	29	rt-twitter-copen	199	199	199	199
bio-yeast	353	353	353	353.1	sc-ldoor	65387.7	65556.6	65587.9	65610.9
ca-AstroPh	2070	2073	2073	2076	sc-shipsec5	12069.8	12262.7	12262.7	12273.1
ca-netscience	55	55	55	55	soc-BlogCatalog	4894	4894	4894	4894
socfb-A-anon	201698.6	201699.1	201699.1	201700.7	soc-youtube-snap	213122.1	213122.1	213122.1	213122.2
socfb-uci-uni	865675	865676	865676	865676	tech-RL-caida	40224.8	40234.3	40234.3	40246.5
inf-power	1487.1	1488.5	1488.5	1488.6	tech-routers-rf	479	479	479	479
inf-roadNet-PA	338740.6	339897.8	339897.8	340075.5	scc-enron-only	6	6	6	6
ia-email-EU	755	755	755	755	scc-twitter-copen	6410	6410	6410	6410
ia-wiki-Talk	11935	11935	11935	11935	web-BerkStan	3000	3000	3000	3001
rec-amazon	28365.7	28393.4	28393.4	28407	web-wikipedia2009	347018.1	347052.7	347068	347097.2
rt-retweet	32	32	32	32					

results on the first and last graphs in alphabetical order from each of the 12 domains in Table 1.14 and Table 1.15 in this section. The complete results are given in Table A.6, Table A.7 and Table A.8 in Appendix A.

As we can see, with much smaller sets of parameter combination, our algorithms for minimum weighted dominating set problem and minimum dominating set problem can still achieve very good results.

In practice, we can consider the observations introduced above as guidelines for choosing appropriate parameter values for our algorithms, when computing time is important.

Efficiency

In this section, we evaluate the efficiencies of our algorithms for the weighted set cover and minimum weighted dominating set problems by comparing their running time with those of the classic greedy algorithms .

Table 1.16: Evaluation of Efficiency on 70 Instances of Weighted Set Cover Problem (consumed CPU time in seconds)

Instance	Gr	Ours	Instance	Gr	Ours	Instance	Gr	Ours
scp41	0.005	0.006	scp65	0.003	0.009	scpe4	0.001	0.004
scp42	0.005	0.006	scpa1	0.007	0.017	scpe5	0.001	0.004
scp43	0.002	0.004	scpa2	0.007	0.018	scpnre1	0.071	0.185
scp44	0.002	0.004	scpa3	0.007	0.017	scpnre2	0.071	0.185
scp45	0.001	0.004	scpa4	0.007	0.017	scpnre3	0.071	0.21
scp46	0.001	0.004	scpa5	0.007	0.017	scpnre4	0.071	0.192
scp47	0.001	0.004	scpb1	0.014	0.037	scpnre5	0.071	0.184
scp48	0.001	0.004	scpb2	0.014	0.036	scpnrf1	0.137	0.371
scp49	0.001	0.004	scpb3	0.014	0.037	scpnrf2	0.137	0.359
scp410	0.001	0.004	scpb4	0.014	0.037	scpnrf3	0.137	0.359
scp51	0.003	0.008	scpb5	0.014	0.037	scpnrf4	0.137	0.359
scp52	0.003	0.009	scpc1	0.012	0.029	scpnrf5	0.137	0.359
scp53	0.003	0.008	scpc2	0.012	0.03	scpnrg1	0.063	0.159
scp54	0.003	0.008	scpc3	0.012	0.03	scpnrg2	0.063	0.158
scp55	0.003	0.008	scpc4	0.012	0.029	scpnrg3	0.064	0.158
scp56	0.003	0.008	scpc5	0.012	0.029	scpnrg4	0.064	0.157
scp57	0.003	0.009	scpd1	0.025	0.071	scpnrg5	0.064	0.158
scp58	0.003	0.008	scpd2	0.025	0.065	scpnrh1	0.145	0.381
scp59	0.003	0.008	scpd3	0.025	0.063	scpnrh2	0.146	0.375
scp510	0.003	0.009	scpd4	0.025	0.065	scpnrh3	0.145	0.373
scp61	0.003	0.009	scpd5	0.025	0.063	scpnrh4	0.145	0.393
scp62	0.003	0.009	scpe1	0.001	0.004	scpnrh5	0.145	0.372
scp63	0.003	0.008	scpe2	0.001	0.004			
scp64	0.003	0.008	scpe3	0.001	0.005			

We run our algorithms and the corresponding classic greedy algorithms, which are all implemented with the same subroutines and data structures, e.g. max-heap, on the 70 instances from OR-Library and the 139 graphs from the Network Data Repository, and record the consumed CPU time. For our algorithms, since different parameters may affect the number of iterations needed to reach the full coverage in the second step, we run them with 8 different parameter combinations derived from $\beta_l \in \{0.75, 1\}$; $\beta_g \in \{2, 3\}$; $\gamma \in \{0.1, 1\}$, which are chosen based on the guidelines introduced in Section 1.4.5, and report the average consumed CPU time.

The evaluation results of efficiency are given in Table 1.16 and Table 1.17. "Gr" column contains the consumed CPU time of the classic greedy algorithms, while "Ours" column contains those of our algorithms. As we can see, on all the 70 weighted set cover instances, our algorithm is slower than the classic greedy algorithm, while on 69 out of the 139 graphs for minimum weighted dominating set problem, our algorithm is significantly faster. The main reason is that, as we discuss in Section 1.3.3, our algorithms reduce the problem size after reserving the indispensable sets or vertices in this first step. This processing does not work for the 70 weighted set cover instances because they do not contain any indispensable sets. On the 69

Table 1.17: Evaluation of Efficiency on 139 Graphs for Minimum Weighted Dominating Set Problem (consumed CPU time in seconds)

Graph	Gr	Ours	Graph	Gr	Ours	Graph	Gr	Ours
bio-celegans	0.004	0.004	ia-wiki-Talk	0.457	0.191	scc-infect-hyper	0.005	0.012
bio-diseasome	0.001	0.002	rec-amazon	0.287	0.464	scc-reality	2.566	0.114
bio-dmela	0.028	0.014	rt-retweet	0	0.002	scc-retweet	0.048	0.011
bio-yeast	0.003	0.002	rt-retweet-crawl	4.739	1.212	scc-retweet-crawl	0.631	0.653
ca-AstroPh	0.147	0.136	rt-twitter-copen	0.006	0.006	scc-rt-alwefaq	0.002	0.01
ca-CSphd	0.003	0.002	sc-ldoor	13.76	38.186	scc-rt-assad	0	0.005
ca-CondMat	0.09	0.125	sc-msdoor	6.103	17.152	scc-rt-bahrain	0.002	0.002
ca-Erdos992	0.011	0.004	sc-nasasrb	0.784	2.275	scc-rt-barackobama	0.004	0.003
ca-GrQc	0.014	0.017	sc-pkustk11	1.56	4.348	scc-rt-damascus	0.001	0.001
ca-HepPh	0.088	0.056	sc-pkustk13	1.975	5.511	scc-rt-dash	0.002	0.002
ca-MathSciNet	1.509	0.943	sc-pwtk	3.564	9.976	scc-rt-gmanews	0.004	0.004
ca-citeseer	1.054	1.213	sc-shipsec1	1.35	3.573	scc-rt-gop	0.001	0.001
ca-coauthors-dblp	10.649	19.136	sc-shipsec5	1.762	4.695	scc-rt-http	0.002	0.002
ca-dblp-2010	1.036	1.064	soc-BlogCatalog	1.692	0.379	scc-rt-israel	0.001	0.001
ca-dblp-2012	1.565	1.394	soc-FourSquare	4.575	1.774	scc-rt-justinbieber	0.004	0.004
ca-hollywood-2009	46.994	21.175	soc-LiveMocha	1.688	0.331	scc-rt-ksa	0.002	0.002
ca-netscience	0.001	0.007	soc-brightkite	0.262	0.153	scc-rt-lebanon	0.001	0.001
socfb-A-anon	31.776	6.74	soc-buzznet	1.975	1.3	scc-rt-libya	0.002	0.002
socfb-B-anon	28.315	5.95	soc-delicious	2.414	1.249	scc-rt-lolgop	0.006	0.004
socfb-Berkeley13	0.543	0.305	soc-digg	7.45	2.428	scc-rt-mittromney	0.003	0.003
socfb-CMU	0.152	0.087	soc-dolphins	0	0.003	scc-rt-obama	0.001	0.001
socfb-Duke14	0.306	0.149	soc-douban	0.528	0.301	scc-rt-occupy	0.001	0.001
socfb-Indiana	0.842	0.874	soc-epinions	0.114	0.056	scc-rt-occupywallstnyc	0.002	0.001
socfb-MIT	0.153	0.069	soc-flickr	4.089	1.579	scc-rt-oman	0.001	0.002
socfb-OR	0.675	0.332	soc-flixster	13.439	5.416	scc-rt-onedirection	0.003	0.003
socfb-Penn94	0.926	0.694	soc-gowalla	1.213	0.884	scc-rt-p2	0.002	0.002
socfb-Stanford3	0.347	0.098	soc-karate	0	0.002	scc-rt-qatif	0.002	0.002
socfb-Texas84	1.053	0.624	soc-lastfm	7.198	2.567	scc-rt-saudi	0.003	0.003
socfb-UCLA	0.482	0.322	soc-livejournal	40.225	20.879	scc-rt-tcot	0.001	0.002
socfb-UCSB37	0.306	0.255	soc-orkut	109.876	88.789	scc-rt-tlot	0.001	0.001
socfb-UConn	0.387	0.288	soc-pokec	27.699	10.652	scc-rt-uae	0.002	0.002
socfb-UF	0.963	0.634	soc-slashdot	0.389	0.178	scc-rt-voteonedirection	0	0.001
socfb-UIllinois	0.801	0.809	soc-twitter-follows	1.291	0.731	scc-twitter-copen	0.281	0.024
socfb-Wisconsin87	0.543	0.559	soc-wiki-Vote	0.006	0.016	web-BerkStan	0.031	0.061
socfb-uci-uni	246.064	83.257	soc-youtube	3.006	1.585	web-arabic-2005	1.325	1.069
inf-power	0.012	0.095	soc-youtube-snap	6.179	4.256	web-edu	0.007	0.013
inf-road-usa	119.168	166.914	tech-RL-caida	0.837	0.915	web-google	0.003	0.003
inf-roadNet-CA	9.264	11.425	tech-WHOIS	0.046	0.023	web-indochina-2004	0.044	0.058
inf-roadNet-PA	4.629	6.244	tech-as-caida2007	0.072	0.034	web-it-2004	5.422	9.543
ia-email-EU	0.077	0.046	tech-as-skitter	13.741	11.308	web-polblogs	0.006	0.003
ia-email-univ	0.004	0.006	tech-internet-as	0.115	0.109	web-sk-2005	0.478	0.737
ia-enron-large	0.166	0.102	tech-p2p-gnutella	0.222	0.133	web-spam	0.03	0.012
ia-enron-only	0	0.003	tech-routers-rf	0.007	0.006	web-uk-2005	6.487	7.177
ia-fb-messages	0.005	0.003	scc-enron-only	0.005	0.015	web-webbase-2001	0.034	0.028
ia-infect-dublin	0.002	0.005	scc-fb-forum	0.037	0.002	web-wikipedia2009	9.38	5.837
ia-infect-hyper	0.001	0.001	scc-fb-messages	0.271	0.012			
ia-reality	0.011	0.008	scc-infect-dublin	0.11	0.269			

graphs, however, it can significantly reduce the problem size. Therefore, although our algorithms generally have more iterations in the second step than the classic greedy algorithms, and have an extra third step, they can be very efficient when processing large real world instances.

1.5 Conclusion

I propose approximation algorithms for the weighted set cover and minimum weighted dominating set problems based on a novel greedy heuristic. Extensive experimental evaluations on a large number of synthetic and real world set cover instances and graphs from many domains demonstrate their superiority over state-of-the-art.

CHAPTER 2

Affinity Learning for Mixed Data Clustering

2.1 Introduction

Clustering is the task of partitioning the data objects into a set of groups (clusters) such that objects in the same group are similar, while objects in different groups are dissimilar. It is one of the most fundamental problems in data mining and machine learning. Numerous algorithms have been developed for clustering. Most of them are designed to handle data with only one type of attributes, e.g. continuous, categorical or ordinal. Mixed data clustering has received relatively less attention, despite the fact that data with mixed types of attributes are common in real applications.

For mixed data clustering, one of the greatest challenges is how to measure the affinities or distances between data points. One of the most straightforward methods for processing mixed data is the so-called 1-hot or 1-of-K encoding. A categorical attribute with K distinct values is encoded to K $0-1$ binary attributes. Each categorical attribute value is transformed into a 1 on its corresponding binary attribute. Then they are treated just like continuous attributes. The more formal Gower's similarity coefficient Gower (1971) and its extensions Legendre & Legendre (1998); Podani (1999) compute the partial affinity between two data points on each attribute according to the data type, and then aggregate all of them into a composite similarity measure. Such methods are widely used in practice. However, they essentially compute the affinity or distance "locally" between two data points, without considering the attribute values of other data points. This may result in missing some intrinsic information. For example, in many real world data sets, some values of a categorical attribute are inherently related. Such information would be missed by similarity measures like Gower's coefficient, which simply assume different categories are totally independent and unrelated.

We propose a novel affinity learning based framework for mixed data clustering. It includes how to process data with mixed-type attributes, how to learn affinities between data points, and how to exploit the learned affinities for clustering.

First, each original attribute is represented with several abstract objects defined according to the specific data type and values. Each attribute value is then transformed into the initial affinities between the data point and the abstract objects of attribute. For categorical attributes, each category is defined as an abstract object. Its affinities to the data points in this category are initialized to a constant value. For each continuous attribute, two abstract objects are defined to represent its minimum and maximum values. Their initial affinities to each data point are transformed from the individual continuous attribute value with a novel method. For ordinal attributes, all possible values are first ranked and then replaced by their ranks. The new ordinal attributes are processed as continuous attributes.

After the data processing, we obtain a bipartite graph consisting of the data points, the abstract objects of attributes, and the initial affinities between them. The next step is to learn new affinities, including inferring the unknown affinities and refining the known affinities. We adopt the algorithm proposed in Li & Latecki (2015), which essentially implements the von Neumann kernel Kandola et al. (2003) from the perspective of transitive inference confidence. Specifically, the new affinities are learned according to the transitive property of the affinitive relation. All the initial affinities are scaled with a common scaling factor. Any transitive inference process without self-loops is considered to be effective to reveal the two objects are affinitive. The confidence of such an inference process is quantified as the product of the related scaled affinity values. In general, there can be an infinite number of distinct transitive inference processes between two objects. The confidence of all these inference processes are added up to be the new affinity between two objects. The details of this affinity learning algorithm are presented in Section 2.3.2.

In comparison to Gower's similarity coefficient and its extensions, our affinity learning method shares the similar idea of aggregating partial affinities on individual attributes into an overall measure. But the significant difference is that our affinities are computed "globally" by taking into account the interconnections among the attribute values of all data points, not just between the two data points. This is illustrated in Figure 2.1. The numbered blue circles represent data points, i.e. $\{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \boldsymbol{x_4}\}$. The two rectangles represent categorical attributes R and Y, each of which has three distinct attribute values. If we compute the affinity S_{ij} just between the two data points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, like Gower's coefficient does, then S_{13} and S_{14} are both 0, because they don't have any common attribute values. However, because of the existence of $\boldsymbol{x_2}$, which shares one common attribute value with $\boldsymbol{x_1}$ and $\boldsymbol{x_3}$ respectively, it's intuitive to infer that $\boldsymbol{x_1}$ is more affinitive to $\boldsymbol{x_3}$ in comparison to $\boldsymbol{x_4}$. Our affinity learning method can capture such information by taking into account all transitive inference processes, including $\boldsymbol{x_1} \rightarrow R_1 \rightarrow \boldsymbol{x_2} \rightarrow Y_2 \rightarrow \boldsymbol{x_3}$.

The inferred affinities between data points can be used by many clustering algorithms. Alternatively, the refined affinities between data point and the abstract objects of attribute can be transformed into new data features. With such features, any algorithms can be used for clustering.

The mixed data clustering algorithms derived from the proposed framework achieve superior performance on many real world data sets. The details of the experimental evaluation are presented in Section 2.4.

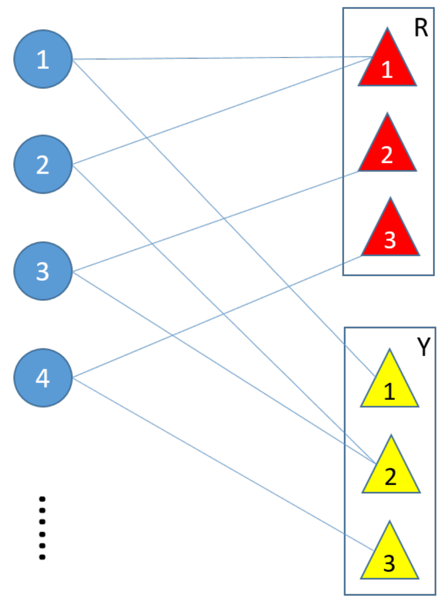


Figure 2.1: An illustration of data point connections via their attribute values. Blue circles represent data points. Rectangles represent categorical attributes, each has three distinct attribute values.

2.2 Related Work

For mixed data clustering, in addition to using 1-hot encoding to obtain continuous features or Gower's coefficient Gower (1971) and its extensions Legendre & Legendre (1998); Podani (1999) to measure the similarities between data points, as introduced in Section 2.1, there are also some specially designed clustering algorithms, including k-prototypes Z. Huang (1997, 1998), K-means-mixed Ahmad & Dey (2007), CAVE Hsu & Chen (2007), M-ART Hsu & Huang (2008), INTEGRATE Böhm et al. (2010), INCONCO Plant & Böhm (2011), SCENIC Plant (2012) and so on. K-prototypes algorithm Z. Huang (1997, 1998), which essentially follows the same idea of k-means algorithm, calculates the dissimilarity between two mixed-type objects as a combination of the squared Euclidean distance measure on the numeric attributes and the simple matching dissimilarity measure on the categorical attributes. K-means-mixed Ahmad & Dey (2007), like k-prototypes, is also based on the k-means paradigm and combines distance measures computed separately on numeric attributes and categorical attributes. Unlike k-prototypes, k-means-mixed does not assume a binary or a discrete measure between two distinct categorical attribute values but computes the distance as a function of their overall distribution and co-occurrence with other categorical attributes. This idea of computing distances ”globally” is similar to ours, but it’s only applied within categorical attributes. CAVE Hsu & Chen (2007) uses variance to measure the similarity of the numeric part of the data and computes the similarity of the categorical part based on entropy weighted by the distances in the hierarchies. Similarly, the incremental clustering algorithm M-ART Hsu & Huang (2008) also computes the distance between two data points according to distance hierarchies associated with the mixed-type attributes. INTEGRATE Böhm et al. (2010) applies ideas from information theory to implement the k-means paradigm. It models both numerical and categorical attributes with their probability distributions and minimizes a cost function based on the Minimum Description Length principle for clustering. INCONCO Plant & Böhm (2011) and SCENIC Plant (2012) process mixed-type attributes in a similar way as INTEGRATE. Their main advantage is the capability of modeling and revealing the cluster-specific dependency patterns among the attributes.

To learn affinities between heterogeneous objects of data points and attributes, we adopt the algorithm proposed in Li & Latecki (2015), which models the new affinities from the perspective of transitive inference confidence. It essentially implements the von Neumann kernel defined in Kandola et al. (2003). The idea of learning semantic similarity between terms from a corpus for measuring similarity between text documents in Kandola et al. (2003) is similar to our idea of capturing the intrinsic information between attribute values. One significant difference, besides the applications are different, is that we explicitly model the interconnections among data points and attribute values together. There are also some other algorithms can be used for affinity learning , such as Zhou et al. (2003) and Yang et al. (2013). The main reasons we do not choose them include: 1. their row or column normalizations on the initial affinity matrix change the original relationships between the heterogeneous objects; and 2. they are not as semantically intuitive and meaningful as the one Li & Latecki (2015) we adopt.

2.3 Our Framework

2.3.1 Mixed Data Processing

We first transform the data points and their mixed-type attribute values into abstract objects and initial affinities. For categorical attributes, each category is defined as an abstract object. Its affinities to the data points in this category are initialized to 1, while its initial affinities to the rest data points are 0. This is similar to the 1-hot encoding. For each continuous attribute C, two abstract objects are defined to represent its minimum and maximum values, i.e. C_{min} and C_{max}. The attribute value x_C of the data point $\boldsymbol{x}$ is transformed into two initial affinities to the abstract objects of C_{min} and C_{max}. Suppose they are $S_{\boldsymbol{x},C_{min}} = a$ and $S_{\boldsymbol{x},C_{max}} = b$, we have two requirements,

$$\begin{cases} a^2 + b^2 = 1 \\ (C_{min} \times a + C_{max} \times b)/(a + b) = x_C \end{cases} \tag{2.3-1}$$

To understand the first requirement, consider the illustration in Figure 2.2. a, b, c, d on the edges represent the initial affinities of two data points $\boldsymbol{x}$ and $\boldsymbol{x'}$ to the abstract objects of C_{min} and C_{max} respectively. The diffusion based affinity learning algorithms essentially compute the affinity $S_{\boldsymbol{x},\boldsymbol{x'}}$ as

$$S_{\boldsymbol{x},\boldsymbol{x'}} = a \times c + b \times d \tag{2.3-2}$$

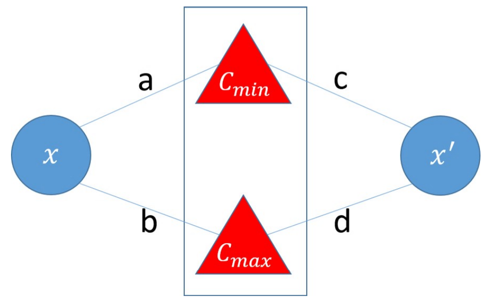


Figure 2.2: An illustration for explaining the first requirement in equation (2.3-1) for transforming a continuous attribute value into initial affinities.

If $\boldsymbol{x}$ and $\boldsymbol{x'}$ have the same attribute value x_C on C, obviously their affinities to C_{min} and C_{max} should be the same, i.e. $a = c$ and $b = d$. It's also obvious to require that $S_{\boldsymbol{x},\boldsymbol{x'}}$ to be a constant, e.g. 1, no matter what the attribute value x_C is. Therefore, we get the first requirement,

$$S_{\boldsymbol{x},\boldsymbol{x'}} = a \times c + b \times d = a \times a + b \times b = 1 \qquad (2.3\text{-}3)$$

The second requirement makes sure the original attribute value can be restored from the transformed affinities.

Specifically, to transform the attribute value x_C of $\boldsymbol{x}$ into the initial affinities, x_C is first scaled with the Min-Max normalization.

$$x'_C = \frac{x_C - C_{min}}{C_{max} - C_{min}} \qquad (2.3\text{-}4)$$

The scaled attribute value x'_C is in range $[0, 1]$, i.e. $C'_{min} = 0$ and $C'_{max} = 1$. We have

$$\begin{cases} a^2 + b^2 = 1 \\ (0 \times a + 1 \times b)/(a + b) = x'_C \end{cases} \qquad (2.3\text{-}5)$$

Solve the system of equations, we get the affinity transformation formula as

$$\begin{cases} a = \sqrt{(1 - x_C')^2/(2 \times {x_C'}^2 - 2 \times x_C' + 1)} \\ b = \sqrt{{x_C'}^2/(2 \times {x_C'}^2 - 2 \times x_C' + 1)} \end{cases} \tag{2.3-6}$$

In this way, if two data points have the same value on a continuous attribute, their partial affinity inferred by the diffusion based affinity learning algorithm described below based on this agreement is always the same, no matter what the value is.

For ordinal attributes, all possible values are first ranked and then replaced by their ranks. The new ordinal attributes are processed as continuous attributes.

If an attribute value of $\boldsymbol{x}$ is missing, the related initial affinities are all set to 0.

2.3.2 Affinity Learning

Now we have a bipartite graph consisting of n data points, m abstract objects of attribute, and the initial affinities between them. We construct a nonnegative symmetric affinity matrix $A = (a_{ij})_{\alpha \times \alpha}$, where $\alpha = m + n$.

$$A = \begin{bmatrix} A_{DD} & A_{DC} \\ A_{CD} & A_{CC} \end{bmatrix} \tag{2.3-7}$$

where A_{DD} is a $n \times n$ zero matrix indicating that the affinities between data points are unknown; $A_{DC} = A_{CD}^{\intercal}$ is a $n \times m$ matrix consisting of the initial affinities between data points and the abstract objects of attributes; A_{CC} is a $m \times m$ zero matrix indicating that the affinities between abstract objects of attributes are unknown.

The next step is to scale the nonzero entries in A, i.e. the initial affinities, with a common scaling factor Δ which satisfies

$$\Delta > \max(a_{max}, \rho(A)) \tag{2.3-8}$$

where a_{max} is the maximum entry of A; $\rho(A)$ is the spectral radius of A.

Each entry a_{ij} of A is scaled with Δ to obtain another matrix $A' = (a'_{ij})_{\alpha\times\alpha}$ where

$$a'_{ij} = \frac{a_{ij}}{\Delta} \tag{2.3-9}$$

Obviously, any entry a'_{ij} of A' is less than 1. Also, the spectral radius of A' is less than 1. Therefore,

$$\lim_{l\to\infty}(A')^l = \mathbf{0} \tag{2.3-10}$$

Then we compute a matrix A^* as

$$A^* = (I - A')^{-1} \tag{2.3-11}$$

where I is the $\alpha \times \alpha$ identity matrix.

Each entry a^*_{ij} of A^* denotes a value,

$$a^*_{ij} = \sum_{l=0}^{\infty}[(A')^l]_{ij} \tag{2.3-12}$$

which is the learned affinity between objects i and j.

The inferred affinities between data points are in A^*_{DD}. The refined affinities between data points and abstract objects of attributes are in A^*_{DC}.

To get the scaling factor Δ, we need to calculate the spectral radius $\rho(A)$ of A. With iterative eigenvalue algorithms, it can be done in $\mathcal{O}(\alpha^2)$. Scaling the nonzero entries of A takes $\mathcal{O}(\alpha^2)$. The straightforward computation for inverting the matrix $I - A'$ takes $\mathcal{O}(\alpha^3)$. Advanced algorithms, such as Strassen algorithm, can further reduce the asymptotic computational complexity. Therefore, the straightforward time complexity of our affinity learning algorithm is $\mathcal{O}(\alpha^3)$. However, A and $I - A'$ are usually very sparse. Consequently, the practical efficiency should be much better.

We evaluate it on several real data sets with α up to about $30,000$. The details are presented in Section 2.4

2.3.3 Clustering with Learned Affinities

In this work, we use the complete-linkage algorithm for clustering with the inferred affinities between data points in A^*_{DD}. It is one of the agglomerative hierarchical clustering methods. Specifically, in the beginning, each data point is in a cluster of its own. Then these clusters are iteratively combined until the target cluster number is reached. At each step, the two clusters, whose two members (one in each cluster) have the minimum pair-wise affinity, are combined.

Alternatively, the refined affinities of data points to the abstract objects of attributes can be used as new data features. Specifically, in the $n \times m$ matrix A^*_{DC}, each row is considered as a m-dimensional feature vector of the corresponding data point. In this work, we choose k-means algorithm and complete-linkage algorithm for clustering with such features.

2.4 Experimental Evaluation

2.4.1 Experimental Setup

We evaluate the performance of the proposed clustering framework on several real world data sets from the UCI Machine Learning Repository, including 5 mixed-type (Acute Inflammations, Heart Disease, Credit Approval, Contraceptive Method Choice and Adult) and 2 categorical (Soybean and Tic-Tac-Toe Endgame). The detailed information of these data sets is summarized in Table 2.1.

Each record of Acute Inflammation data set corresponds to the *yes* or *no* diagnoses of two diseases of the urinary system. We transform the two diagnoses into four classes, i.e. *(yes,yes), (yes,no), (no,yes) and (no,no)*. For Adult data set, we only

Table 2.1: Data Sets for Experimental Evaluation (number of different types of attributes, number of instances and number of classes)

Data set	Continuous	Categorical	Ordinal	#Instance	#Class
Acute Inflammations	1	5	-	120	4
Heart Disease	6	6	1	270	2
Credit Approval	6	9	-	690	2
Contraceptive Method Choice	2	7	-	1,473	3
Adult	6	8	-	48,842	2
Soybean	-	35	-	47	4
Tic-Tac-Toe Endgame	-	9	-	958	2

use the training set, which contains $32,561$ records. For fair comparison, we remove the records with missing attribute values. The final data set contains $30,162$ records. We skip the attribute ”education”, because it is fully expressed by another attribute ”education-num”. In Credit Approval data set, 37 (about %5) records have one or more missing values. We simply remove them.

The clustering algorithms derived from the proposed framework include: 1. **IA+CL** (**I**nferred **A**ffinities between data points + **C**omplete-**L**inkage algorithm); 2. **FRA+CL** (**F**eature from **R**efined **A**ffinities of the data point to the abstract objects of attributes + **C**omplete-**L**inkage algorithm); 3. **FRA+KM** (**F**eature from **R**efined **A**ffinities of the data point to the abstract objects of attributes + **K**-**M**eans algorithm).

For the three derived clustering algorithms, we vary the scaling factor Δ in equation 2.3-9 in the range of $(\max(a_{max}, \rho(A)), 4 \times \max(a_{max}, \rho(A))]$ (see equation (2.3-8)) with a step size of 10. The best results achieved by each algorithm in this process are reported. For FRA+CL and FRA+KM, we use the squared Euclidean distance measure.

The comparison algorithms include: 1. **OH+CL** (Feature from **O**ne-**H**ot encoding + **C**omplete-**L**inkage algorithm); 2. **OH+KM** (Feature from **O**ne-**H**ot encoding + **K**-**M**eans algorithm); 3. **GC+CL** (**G**ower's **C**oefficient + **C**omplete-**L**inkage algorithm); 4. **KP** (k-prototypes) Z. Huang (1997, 1998); 5. **KMM** (K-means-mixed)

Ahmad & Dey (2007). These algorithms are widely used in practice for mixed data clustering. Some of them are still state-of-the-art in performance. Many recent algorithms, such as Plant & Böhm (2011); Plant (2012) are very complex to be implemented. We are not able to obtain the source code from the authors.

The Gower's coefficient in GC+CL processes ordinal attributes according to Eqs. 2a-b of Podani (1999). For KP (k-prototypes), we scale all numeric attributes to the range of $[0, 1]$ with Min-Max normalization and randomly select k data points without missing values to be the initial prototypes. The parameter γ is varied from 0.5 to 1.5 with a step size of 0.1 for the 5 mixed-type data sets. When using k-means technique, including k-prototypes and K-means-mixed, the maximum number of iterations is set to be 1000 for the Adult data set, which contains much more data, and 100 for the other 6 data sets. Moreover, all the tests are run for 100 times and the average results are reported.

For all the clustering algorithms above, we set the target number of clusters to be the number of classes in each data set. The clustering quality is measured in terms of Jaccard Coefficient, Fowlkes and Mallows Index, and FScore Jing et al. (2007). The results are consistent, so only FScore is reported. Suppose k is the class and cluster number, n is the number of data points, n_i and n_j are the numbers of data points in class CLA_i and cluster CLU_j respectively, n_{ij} is the number of data points in both CLA_i and CLU_j, FScore is defined as

$$FScore = \sum_{i=1}^{k} \left(\frac{n_i}{n} \times \max_{1 \le j \le k} \frac{2 \times R_{ij} \times P_{ij}}{R_{ij} + P_{ij}}\right) \tag{2.4-13}$$

where $R_{ij} = n_{ij}/n_i$ and $P_{ij} = n_{ij}/n_j$.

All the experiments are implemented in MATLAB R2016a and conducted on a PC with Intel(R) Core(TM) i7 processor up to 3.4 GHz and 16GB RAM.

Table 2.2: Clustering Results (FScore on AI: Acute Inflammations; HD: Heart Disease; CA: Credit Approval; CMC: Contraceptive Method Choice; Adult; Soybean; TTT: Tic-Tac-Toe Endgame)

	AI	HD	CA	CMC	Adult	Soybean	TTT
IA+CL	**0.92**	0.78	**0.78**	**0.52**	**0.75**	**1**	0.71
FRA+CL	**0.92**	**0.79**	0.75	0.51	0.73	**1**	**0.76**
FRA+KM	0.80	**0.79**	0.70	0.44	0.73	0.89	0.58
GC+CL	**0.92**	0.71	0.63	0.47	0.58	**1**	0.68
OH+CL	0.76	0.63	0.64	0.48	0.69	**1**	0.68
OH+KM	0.72	0.76	0.69	0.44	0.73	0.88	0.57
KP	0.51	0.76	0.62	0.42	0.68	0.84	0.58
KMM	0.79	0.78	0.77	0.43	0.73	0.91	0.60

2.4.2 Experimental Results

As shown in Table 2.2, the three clustering algorithms derived from the proposed framework achieve superior performance (with ties) on all the 7 data sets. Apparently, among these three algorithms, IA+CL is the best. It achieves the best performance on 5 data sets. In comparison to GC+CL, the performance of IA+CL is consistently better (with ties). Since they use the same clustering algorithm, it proves that our inferred affinities between data points, which are computed "globally", capture more useful information than the "locally" computed similarities. We can see FRA+CL is consistently better (with ties) than OH+CL, and FRA+KM is consistently better (with ties) than OH+KM. It means the feature derived from the refined affinities of the data point to the abstract objects of attributes, which is also computed "globally" in our framework, is more effective than the 1-hot encoding feature. On some data sets, the performance of KP and KMM are competitive. But overall, our IA+CL and FRA+CL are superior. Obviously, the algorithms derived from the proposed framework are effective for mixed data clustering.

In order to further prove that it is beneficial to take into account the interconnections among the attribute values of all data points, we compare the performance of IA+CL, FRA+CL and FRA+KM, which are reported in Table 2.2, with those

Table 2.3: Clustering Results with Locally and Globally Learned Affinities (FScore on AI: Acute Inflammations; HD: Heart Disease; CA: Credit Approval; CMC: Contraceptive Method Choice; Adult; Soybean; TTT: Tic-Tac-Toe Endgame)

	LIA+CL	IA+CL	FNRA+CL	FRA+CL	FNRA+KM	FRA+KM
AI	0.92	0.92	0.92	0.92	0.79	0.80
HD	0.71	0.78	0.71	0.79	0.78	0.79
CA	0.60	0.78	0.60	0.75	0.69	0.70
CMC	0.49	0.52	0.49	0.51	0.44	0.44
Adult	0.67	0.75	0.67	0.73	0.69	0.73
Soybean	1	1	1	1	0.88	0.89
TTT	0.68	0.71	0.68	0.76	0.57	0.58

achieved with ”locally” inferred affinities and features from non-refined affinities. Specifically, after scaling the initial affinities with equation 2.3-9, we obtain the matrix A'. A'_{DC} contains the non-refined affinities of data points to the abstract objects of attributes. We use them as data feature (**FNRA**: **F**eature from **N**on-**R**efined **A**ffinities) for clustering with the complete-linkage (**CL**) and k-means (**KM**) algorithms. To obtain the ”local” affinities between data points, we compute $A^* = A' \times A'$. The affinities in A^*_{DD} are inferred ”locally” just between each pair of data points. We call them **LIA** (**L**ocally **I**nferred **A**ffinities) and use the complete-linkage (**CL**) algorithm for clustering. When comparing LIA+CL versus IA+CL, FNRA+CL versus FRA+CL and FNRA+KM versus FRA+KM, on each data set, LIA+CL, FNRA+CL and FNRA+KM use the same scaling factors Δ as IA+CL, FRA+CL and FRA+KM respectively. Table 2.3 shows the performance comparisons. As we can see, the performance of using ”globally” inferred or refined affinities are always better or equal to those of using ”locally” inferred or non-refined affinities. It demonstrates that the proposed framework is effective for modeling and exploiting the interconnections among the attribute values of all data points to improve clustering performance.

In order to show that the proposed framework for mixed data clustering is applicable in practice, we evaluate its efficiency on real world data sets. The proposed

Table 2.4: Time Consumed on Affinity Learning (sec.)

Data set	#Instance	#Attribute	#Object	Time Consumed
Acute Inflammations	120	6	132	0.005
Heart Disease	270	13	300	0.01
Credit Approval	653	15	705	0.03
Contraceptive Method Choice	1473	9	1493	0.09
Adult	30,162	13	30,256	40
Soybean	47	35	105	0.005
Tic-Tac-Toe Endgame	958	9	985	0.04

framework consists of three main components: 1. processing mixed data; 2. learning affinities; 3. clustering with the learned affinities. As introduced in Section 2.3.1, it takes linear time to process the mixed data. When clustering with the learned affinities, the time complexity totally depends on the selected clustering algorithm. Therefore, we only evaluate the efficiency of affinity learning. The average time consumed on this step in the clustering experiments are reported in Table 2.4.

As shown in Table 2.4, for small data sets, which contain at most thousands of objects, the time consumed on affinity learning is negligible. For medium data sets, such as Adult, it may take a few minutes. Since these results are obtained on an ordinary PC, we can say, with modern computation technologies and computing power, the proposed framework is applicable in practice.

2.5 Conclusions

The main contributions of this work include: 1. we develop a novel framework for mixed data clustering; 2. our approach to mixed data processing, especially the way we transform continuous attribute values into initial affinities, is novel; 3. it's novel to transform the refined affinities between data points and the abstract objects of attributes into new data features. Experimental results on several real world data sets demonstrate the proposed framework is effective.

CHAPTER 3

Clustering Aggregation as Maximum-Weight Independent Set

3.1 Introduction

Clustering is a fundamental problem in data analysis, and has extensive applications in artificial intelligence, statistics and even in social sciences. The goal is to partition the data objects into a set of groups (clusters) such that objects in the same group are similar, while objects in different groups are dissimilar.

In the past few decades, many different clustering algorithms have been developed. Some popular ones include K-means, DBSCAN, Ward's algorithm, EM-clustering and so on. However, there are potential shortcomings for each of the known clustering algorithms. For instance, K-means Lloyd (1982) and its variations have difficulty detecting the "natural" clusters, which have non-spherical shapes or widely different sizes or densities. Furthermore, in order to achieve good performance, they require an appropriate number of clusters as the input parameter, which is usually very hard to specify. DBSCAN Ester et al. (1996), a density-based clustering algorithm, can detect clusters of arbitrary shapes and sizes. However, it has trouble with data which have widely varying densities. Also, DBSCAN requires two input parameters specified by the user: the radius, Eps, to define the neighborhood of each data object, and the minimum number, $minPts$, of data objects required to form a cluster.

Clustering aggregation, also known as consensus clustering or clustering ensemble, refers to a kind of methods which try to find a single (consensus) superior clustering from a number of input clusterings obtained by different algorithms with different parameters. The basic motivation of these methods is to combine the advantages of different clustering algorithms and overcome their respective shortcomings. Besides generating stable and robust clusterings, consensus clustering methods can be applied in many other scenarios, such as categorical data clustering, "privacy-preserving" clustering and so on. Some representative methods include Gionis et al. (2007); Strehl & Ghosh (2002a); Fred & Jain (2002); Singh et al. (2008); Nguyen & Caruana (2007); Fern & Brodley (2004); Topchy et al. (2003); Mimaroglu & Erdil (2011); D. Huang et al. (2015, 2016b,a). Strehl & Ghosh (2002a) formulates clustering ensemble as a combinatorial optimization problem in terms of shared mutual information. That is, the relationship between each pair of data objects is measured based on their cluster labels from the multiple input clusterings, rather than the original features. Then a graph representation is constructed according to these relationships, and finding a single consolidated clustering is reduced to a graph partitioning problem. Similarly, in Gionis et al. (2007), a number of deterministic approximation algorithms are proposed to find an "aggregated" clustering which agrees as much as possible with the input clusterings. Fred & Jain (2002) also applies a similar idea to combine multiple runs of K-means algorithm. Singh et al. (2008) proposes to capture the notion of agreement using an measure based on a 2D string encoding. They derive a nonlinear optimization model to maximize the new agreement measure and transform it into a strict 0-1 Semidefinite Program. Nguyen & Caruana (2007) presents three iterative EM-like algorithms for the consensus clustering problem. The COMUSA algorithm proposed in Mimaroglu & Erdil (2011) first constructs a similarity graph based on the co-association matrix. Then it identifies new clusters by iteratively selecting a pivot data object and expanding the cluster with its immediate free neighbors which are most similar to the pivot. D. Huang et al. (2015) proposed two algorithms termed weighted evidence accumulation clustering (WEAC) and graph partitioning with multi-granularity link analysis (GP-MGLA). WEAC integrates the reliability of each base clustering into the co-association matrix and uses agglomerative algorithms to obtain the final clustering. GP-MGLA models the three levels of granularity in clustering aggregation, i.e., data objects, clusters and base clusterings, in a single bipartite graph, and partitions it to divide data objects into the final clusters. D. Huang et al. (2016b) proposed to sparsify the co-association matrix of "microcluster" with the k-nearest neighbors strategy and learn new similarities based on random walks. Two algorithms, probability trajectory accumulation (PTA) and probability trajectory based graph partitioning (PTGP) were proposed to obtain the final clustering with the learned similarities. PTA is based on agglomerative algorithms, while PTGP is based on the graph partitioning technique. D. Huang et al. (2016a) formulated clustering aggregation as a binary linear programming problem and proposed a solver based on max-product belief propagation on a factor graph.

A common feature of these consensus clustering methods is that they usually do not access to the original features of the data objects. They utilize the cluster labels in different input clusterings as the new features of each data object to find an optimal clustering. Consequently, the success of these consensus clustering methods heavily relies on a premise that the majority of the input clusterings are reasonably good and consistent, which is not often the case in practice. For example, given a new challenging dataset, it is probable that only some few of the chosen underlying clustering algorithms can generate good clusterings. Many moderate or even bad input clustering can mislead the final "consensus". Furthermore, even if we choose the appropriate underlying clustering algorithms, in order to obtain good input clusterings, we still have to specify the appropriate input parameters. Therefore, it is desired to devise new consensus clustering methods which are more robust and do not need the optimal input parameters to be specified.

Our definition of "clustering aggregation" is different. Informally, for each of the clusters in the input clusterings, we evaluate its quality with some internal indices measuring both the cohesion and separation. Then we select an optimal subset of clusters, which partition the dataset together and have the best overall quality, as the "aggregated clustering". (We give a formal statement of our "clustering aggregation" problem in Section 3.2). In this framework, ideally, we can find the optimal "aggregated clustering" even if only a minority of the input clusterings are good enough. Therefore, we only need to specify an appropriate range of the input parameters, rather than the optimal values, for the underlying clustering algorithms.

We formulate this "clustering aggregation" problem as a special instance of Maximum-Weight Independent Set (MWIS) problem. An attributed graph is constructed from the union of the input clusterings. The vertices, which represent the distinct clusters, are weighted by an internal index measuring both cohesion and separation. The edges connect the vertices whose corresponding clusters overlap (In practice, we may tolerate a relatively small amount of overlap for robustness). Then selecting an optimal subset of non-overlapping clusters partitioning the dataset together can be formulated as seeking the MWIS of the attributed graph, which is the heaviest subset of mutually non-adjacent vertices. Moreover, this MWIS problem exhibits a special structure. Since the clusters of each input clustering form a partition of the dataset, the vertices corresponding to each clustering form a maximal independent set (MIS) in the attributed graph.

The most important source of motivation for this work is Brendel & Todorovic (2010). In Brendel & Todorovic (2010), image segmentation is formulated as a MWIS problem. Specifically, given an image, they first segment it with different bottom-up segmentation schemes to get an ensemble of distinct superpixels. Then they select a subset of the most "meaningful" non-overlapping superpixels to partition the image. This selection procedure is formulated as solving a MWIS problem. In this respect, our work is very similar to Brendel & Todorovic (2010). The only difference is that our work applies the MWIS formulation to a more general problem, clustering aggregation.

The maximum-weight independent set problem, which is complementary to the maximum-weight clique problem, is known to be NP-hard. Many heuristic or local search algorithms are proposed to find the approximate solutions. Some of the most effective algorithms include LSCC and LSCC+BMS Y. Wang et al. (2016), FastWClq Cai & Lin (2016), WLMC Jiang et al. (2017) and RRWL Fan et al. (2017). They are all proposed for solving the maximum-weight clique problem. Obviously, they can also be used for solving the maximum-weight independent set problem. LSCC and LSCC+BMS Y. Wang et al. (2016) consist of two phases: (1) randomly generating a maximal clique C and then (2) improving C in a deterministic way. In each local move, they select the neighboring clique with the greatest weight according to the strong configuration checking criterion. FastWClq Cai & Lin (2016) interleaves between clique construction and graph reduction. WLMC Jiang et al. (2017) is an exact branch-and-bound algorithm. It exploits a novel preprocessing to derive an initial vertex ordering and to reduce the size of the graph, and incremental vertex-weight splitting to reduce the number of branches in the search space. RRWL Fan et al. (2017) uses the restart and the random walk strategies to improve local search. If a solution is revisited in some particular situation, the search will restart. In addition, when the local search has no other options except dropping vertices, it will use random walk.

As we mentioned before, in the context of clustering aggregation, the formulated MWIS problem exhibits a special structure. That is, the vertices corresponding to each clustering form a maximal independent set (MIS) in the attributed graph. This special structure is valuable for finding good approximations to the MWIS because, although these MISs may not be the global optimum of the MWIS, they are close to distinct local optimums. We propose a variant of simulated annealing method that takes advantage of this special structure. Our algorithm, simulated annealing based on maximal independent set (SAMIS), starts from each MIS and utilizes a local search heuristic to explore its neighborhood in order to find better approximations to the MWIS. The best solution found in this process is returned as the final approximate MWIS. Since the exploration for each MIS is independent, our algorithm is suitable for parallel computation.

Finally, since the selected clusters may not be able to cover the entire dataset, our approach performs a post-processing to assign the missing data objects to their nearest clusters.

3.2 Our Work

Consider a set of n data objects $D = \{d_1, d_2, ..., d_n\}$. A clustering C_i of D is obtained by applying an exclusive clustering algorithm with a specific set of input parameters on D. The disjoint clusters $c_{i1}, c_{i2}, ..., c_{ik}$ of C_i are a partition of D, i.e. $\bigcup_{j=1}^{k} c_{ij} = D$ and $c_{ip} \cap c_{iq} = \emptyset$ for all $p \neq q$.

With different clustering algorithms and different parameters, we can obtain a set of m different clusterings of D: $C_1, C_2, ..., C_m$. For each cluster c_{ij} in the union of these m clusterings, we evaluate its quality with an internal index measuring both cohesion and separation.

We use the average silhouette coefficient of a cluster as such an internal index in this work. The silhouette coefficient is defined for an individual data object. It is a measure of how similar that data object is to data objects in its own cluster compared to data objects in other clusters. Formally, the silhouette coefficient for the t^{th} data object, S_t, is defined as

$$S_t = \frac{b_t - a_t}{\max(a_t, b_t)} \tag{3.2-1}$$

where a_t is the average distance from the t^{th} data object to the other data objects in the same cluster as t, and b_t is the minimum average distance from the t^{th} data object to data objects in a different cluster, minimized over clusters.

Silhouette coefficient ranges from -1 to +1 and a positive value is desirable. The quality of a particular cluster c_{ij} can be evaluated with the average of the silhouette coefficients of the data objects belonging to it.

$$ASC_{c_{ij}} = \frac{\sum_{t \in c_{ij}} S_t}{|c_{ij}|} \tag{3.2-2}$$

where S_t is the silhouette coefficient of the t^{th} data object in cluster c_{ij}, $|c_{ij}|$ is the cardinality of cluster c_{ij}.

We select an optimal subset of non-overlapping clusters from the union of all the clusterings, which partition the dataset together and have the best overall quality, as the "aggregated clustering". The selection of clusters is formulated as a special instance of the Maximum-Weight Independent Set (MWIS) problem.

Formally, consider an undirected and weighted graph $G = (V, E)$, where $V = \{1, 2, ..., n\}$ is the vertex set and $E \subseteq V \times V$ is the edge set. For each vertex $i \in V$, a positive weight w_i is associated with i. $A = (a_{ij})_{n \times n}$ is the adjacency matrix of G, where $a_{ij} = 1$ if $(i, j) \in E$ is an edge of G, and $a_{ij} = 0$ if $(i, j) \notin E$. A subset of V can be represented by an indicator vector $\mathbf{x} = (x_i) \in \{0, 1\}^n$, where $x_i = 1$ means that i is in the subset, and $x_i = 0$ means that i is not in the subset. An independent set is a subset of V, whose elements are pairwise nonadjacent. Then finding a maximum-weight independent set, denoted as $\mathbf{x}^*$ can be posed as the following:

$$\begin{aligned} &\mathbf{x}^* = argmax_{\mathbf{x}} \mathbf{w}^{\mathbf{T}} \mathbf{x}, \\ &s.t. \quad \forall i \in V : x_i \in \{0, 1\}, \quad \mathbf{x}^T A \mathbf{x} = 0 \end{aligned} \tag{3.2-3}$$

The weight w_i on vertex i is defined as:

$$w_i = ASC_{c_i} \times |c_i| \tag{3.2-4}$$

where c_i is the cluster represented by vertex i, ASC_{c_i} and $|c_i|$ are its quality measure and cardinality respectively.

Our problem (3.2-3) is a special instance of MWIS problem, since graph G exhibits an additional structure, which we will unitize in the proposed algorithm. The vertex set V can be partitioned into disjoint subsets $\mathbf{P} = \{P_1, P_2, ..., P_m\}$, where P_i corresponds to the clustering C_i, such that each P_i is also a maximal independent set (MIS), which means it is not a subset of any other independent set. This follows from the fact that each clustering C_i is a partition of the dataset D. Formally,

$$\bigcup_{i=1}^{m} P_i = V, \quad P_i \cap P_j = \emptyset, \quad i \neq j, \quad \text{and} \quad P_i \ \text{ is MIS}, \quad \forall\, i, j \in \{1, 2, ..., m\} \tag{3.2-5}$$

The basic idea of our maximum-weight independent set algorithm is to explore the neighborhood of each known MIS P_i independently with a local search heuristic in order to find better solutions. The proposed algorithm is an instance of simulated annealing methods Kirkpatrick et al. (1983) with multiple initializations.

Our algorithm starts with a particular MIS P_i, denoted by x_0. x_{t+1}, which is a neighbor of x_t, is obtained by replacing some lower-weight vertices in x_t with higher-weight vertices under the constraint of always being an independent set. Specifically, we first reduce x_t by removing a proportion q of lower-weight vertices. Here we remove a proportion, rather than a fixed number, of vertices in order to make the reduction adaptive with respect to the number s of vertices in x_t. In practice, we use $ceil(s \times q)$ to make sure at least one vertex will be removed. Note that this step is probabilistic, rather than deterministic. The probability that a vertex i will be retained is proportional to its WD value, which is defined as follows.

$$WD_i = \frac{w_i}{\sum_{j \in N_i} w_j} \tag{3.2-6}$$

where N_i is the set of vertices which are connected with vertex i in G.

Intuitively, larger WD value indicates larger weight, less conflict with other vertices or both. Therefore, the obtained x'_t is likely to contain vertices with large weights and have large potential room for improvement. The parameter of proportion q is used to control the "radius" of the neighborhood to be explored.

Then our algorithm iteratively improves x'_t by adding compatible vertices one by one. In each iteration, it first identifies all the vertices compatible with the existing ones in current x'_t, called candidates. Then a "local" measure WD' is calculated to evaluate each of these candidates:

$$WD'_i = \frac{w_i}{\sum_{j \in N'_i} w_j} \tag{3.2-7}$$

where N'_i is the set of *candidate* vertices which are connected with vertex i.

The large value of WD'_i indicates that candidate i either can bring large improvement this time (numerator) or has small conflict with further improvements (denominator) or both.

The candidate with the largest WD' value is added to x'_t. In next iteration, this new x'_t will be further improved. This iterative procedure continues until x'_t cannot be further improved. We obtain x'_t as a randomized neighbor of x_t.

Now our algorithm calculates the acceptance ratio $\alpha = e^{(W(x'_t)-W(x_t))/\beta^t}$, where $W(x) = w^T x$; $0 < \beta < 1$ is a constant which is usually picked to be close to 1. If $\alpha \geq 1$, then x'_t is accepted as x_{t+1}. Otherwise, it is accepted with probability α.

This exploration starting from P_i continues for a number of iterations, or until x_t converges. The best solution encountered in this process is recorded. After exploring

Algorithm 1: Simulated Annealing based on Maximal Independent Set (SAMIS)

Input: Graph G, weights $\mathbf{w}$, adjacency matrix $\mathbf{A}$, the known MIS $P = \{P_1, P_2, ..., P_m\}$

Output: An approximate solution to MWIS

1 Calculate WD for each vertex;
2 **for** *Each MIS* P_i **do**
3   Initialize x_0 with P_i;
4   **for** $t = 1, 2, ..., n$ **do**
5   Reduce x_t to x'_t probabilistically by removing a proportion q of vertices with relatively lower WD values;
6   **repeat**
7   Identify candidate vertices compatible with current x'_t;
8   Calculate WD' for each candidate;
9   Update x'_t by adding the candidate with the largest WD';
10   **until** x'_t *cannot be further improved*;
11   Calculate $\alpha = min[1, e^{(W(x'_t)-W(x_t))/\beta^t}]$;
12   Update x_{t+1} as x'_t with probability α, otherwise $x_{t+1} = x_t$;
13   **end**
14 **end**
15 return the best solution found in the process;

the neighborhood for all the known MISs, the best solution is returned. A formal description can be found in Algorithm 1.

Our algorithm is essentially a variant of simulated annealing method Kirkpatrick et al. (1983), since the maximization of $W(x) = w^T x$ is equivalent to the minimization of the energy function $E(x) = -W(x) = -w^T x$. Lines 5 to 10 in Alg. 1 define a randomized "moving" procedure of making a transition from x_t to its neighbor x'_t. When calculating the acceptance ratio $\alpha = e^{(W(x'_t)-W(x_t))/\beta^t}$, suppose $T_0 = 1$ (initial temperature), then it is equivalent to $\alpha = e^{(-(W(x_t)-W(x'_t)))/(\beta^t)} = e^{(-(E(x'_t)-E(x_t)))/(\beta^t)}$. Hence Algorithm 1 is a variant of simulated annealing. Therefore, our algorithm converges in theory.

In practice, the convergence of our algorithm is fast. In all the experiments presented in next section, our algorithm converges in less than 100 iterations. The reason is that our algorithm takes advantage of that the known MISs are close to distinct local maximum. Also, the local search heuristic of our algorithm is effective to find better candidate in the neighborhood.

The parameter q controls the "radius" of the neighborhood to be explored in each iteration. Small q means small "radius" and results in more iterations to converge. On the other side, using large q will take less advantage of the known MISs. Unstable exploration also results in more iterations to converge.

Since our algorithm explores the neighborhood of each known MIS independently, its efficiency can be further improved by using parallel computation.

3.3 Experimental Evaluation

We evaluate the performance of our approach to clustering aggregation and SAMIS algorithm for MWIS problem with three experiments.

In these experiments, for the underlying clustering algorithms, including K-means, single linkage, complete linkage and Ward's clustering, we use the implementations in MATLAB. Unless specified explicitly, the parameters are MATLAB's defaults. For example, when using K-means, we only specify the number K of desired clusters. The default "Squared Euclidean distance" is used as the distance measure. When calculating silhouette coefficients, we use MATLAB's function "silhouette(X,clust)" and the default metric "Squared Euclidean distance". For robustness in our experiments, we tolerate slight overlap between clusters. That is, for the adjacency matrix $A = (a_{ij})_{n\times n}$, $a_{ij} = 1$ if $\frac{|c_i \cap c_j|}{min(|c_i|,|c_j|)} > 0.05$, and $a_{ij} = 0$ otherwise. In these experiments, the parameters of our local search algorithm are: $q = 0.3$; $\beta = 0.999$; iteration number $n = 100$. We test different combinations of $q = 0.1 : 0.1 : 0.5$ and $n = 100 : 100 : 1000$. The results are almost the same.

In the first experiment, we evaluate our approach's ability to achieve good performance without specifying the optimal input parameters for the underlying clustering algorithms. We use the data set from Fränti & Virmajoki (2006). This data set con-

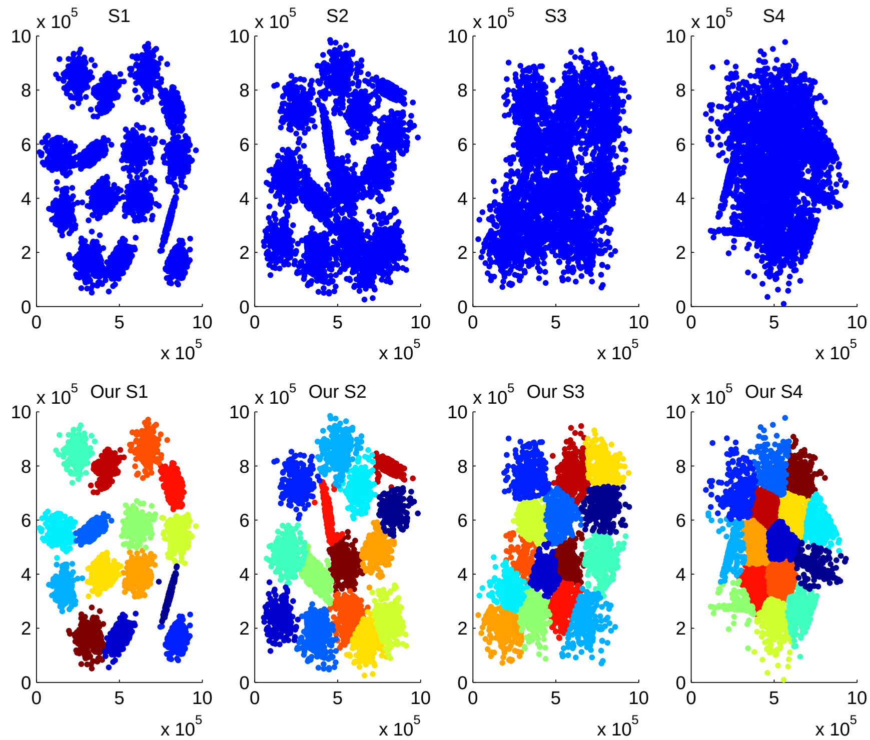


Figure 3.1: Clustering aggregation without parameter tuning. (top row) Original data. (bottom row) Clustering results of our approach. Best viewed in color.

sists of 4 subsets (S1, S2, S3, S4) of synthetic 2-d data points. Each subset contains 5000 vectors in 15 Gaussian clusters, but with different degree of cluster overlapping. We choose K-means as the underlying clustering algorithm and vary the parameter $K = 5 : 1 : 25$, which is the desired number of clusters. Since different runs of K-means starting from random initialization of centroids typically produce different clustering results, we run K-means 5 times for each value of K. That is, there are a total of $21 \times 5 = 105$ different input clusterings. Note that, in order to show the performance of our approach clearly, we do not perform the post-processing of assigning the missing data points to their nearest clusters.

As shown in Fig. 3.1, on each of the four subsets, the aggregated clustering obtained by our approach has the correct number (15) of clusters and near-perfect

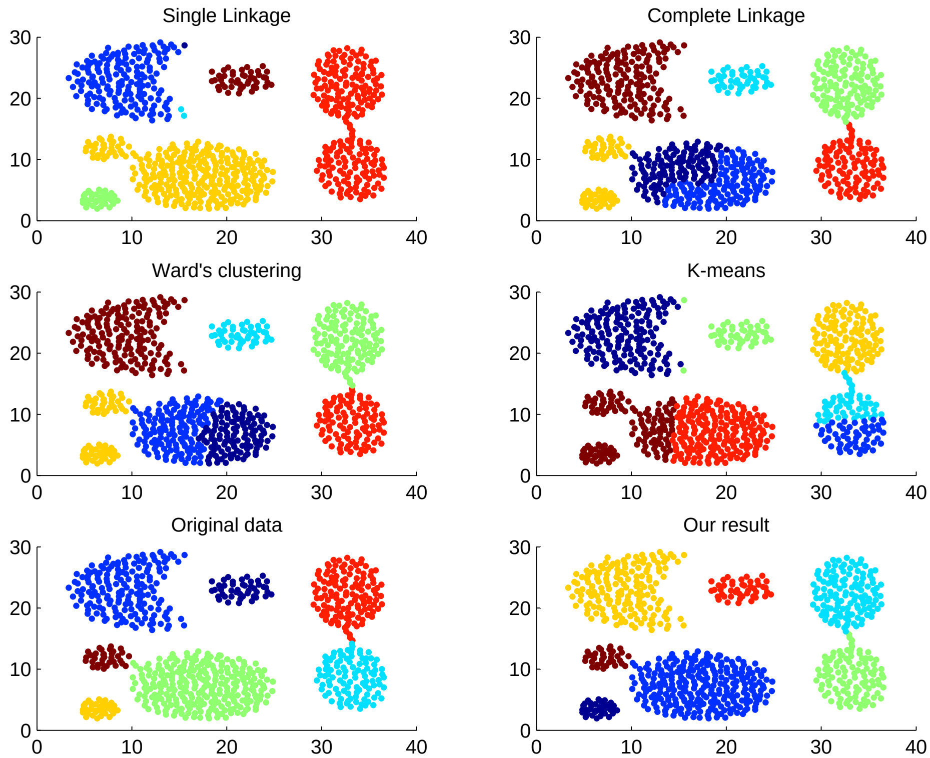


Figure 3.2: Clustering aggregation on four different input clusterings. Best viewed in color.

structure. Only a very small portion of data points is not assigned to any cluster. These results confirm that our approach can automatically decide the optimal number of clusters without any parameter tuning for the underlying clustering algorithms.

In the second experiment, we evaluate our approach's ability of combining the advantages of different underlying clustering algorithms and canceling out the errors introduced by them. The data set is from Gionis et al. (2007). As shown in the fifth panel of Fig. 3.2, this synthetic data set consists of 7 distinct groups of 2-d data points, which have significantly different shapes and sizes. There are also some "bridges" between different groups of data points. Consequently, this data set is very challenging for any single clustering algorithm. In this experiment, we use four different underlying clustering algorithms implemented in MATLAB: single linkage, complete linkage, Ward's clustering and K-means. The first two are both agglomerative bottom-up algorithms. The only difference between them is that when merging pairs of clusters, single linkage is based on the minimum distance, while complete linkage is based on maximum distance. The third one, Ward's clustering algorithm, is also an agglomerative bottom-up algorithm. In each merging step, it chooses the pair of clusters which minimize the sum of the square of distances from each point to the mean of the two clusters. The fourth algorithm is K-means.

For each of the underlying clustering algorithms, we vary the input parameter of desired number of clusters as $4:1:10$. That is, we have a total of $7 \times 4 = 28$ input clusterings.

Note that, unlike Gionis et al. (2007), we do not use the average linkage clustering algorithm, because by specifying the correct number of clusters, it can generate near-perfect clustering by itself. We abandon the best algorithm here in order to show the performance of our approach clearly. But, in practice, by utilizing good underlying clustering algorithms, it can significantly increase the chance for our approach to obtain superior aggregated clusterings. Like the first experiment, we do not perform the post-processing in this experiment.

In the first four panels of Fig. 3.2, we show the clustering results obtained by the four underlying clustering algorithms with the number of clusters set to be 7. Obviously, even with the optimal input parameters, the results of these algorithms are far from being correct. The ground truth and the result of our approach are shown in the fifth and sixth panels, respectively. As we can see, our aggregated clustering is almost perfect, except for the three green data points in the "bridge" between the cyan and green "balls". These results confirm that our approach can effectively combine the advantages of different clustering algorithms and cancel out the errors introduced by them. Also, in contrast to the other consensus clustering algorithms,

Table 3.1: Data Sets for Experimental Evaluation

Data set	#Instance	#Attribute	#Class
Iris	150	4	3
Zoo	101	16	7
Semeion	1593	256	10
PD	10992	16	10
Vowel	990	10	11
ISOLET	7797	617	26
Letter	20000	16	26

Table 3.2: Base Clusterings and Graph Information

Data set	k	#Clustering	#Cluster	$\lvert V\rvert$	$\lvert E\rvert$	d_{avg}
Iris	2:1:10	18	108	108	1422.5	26.3
Zoo	3:1:11	18	126	126	1791	28.4
Semeion	6:1:14	18	180	180	5467	60.7
PD	6:1:14	18	180	180	4403	48.9
Vowel	7:1:15	18	198	198	4633.9	46.8
ISOLET	22:1:30	18	468	468	20273	86.6
Letter	22:1:30	18	468	468	22859.3	97.7

such as Gionis et al. (2007), our aggregated clustering is obtained without specifying the optimal input parameters for any of the underlying clustering algorithm. This is a very desirable feature in practice.

The third experiment is performed on 7 real data sets from the UCI machine learning repository Lichman (2013), including Iris, Zoo, Semeion Handwritten Digit (Semeion), Pen Digits (PD), Vowel, ISOLET and Letter Image Recognition (Letter). The detailed information of these data sets are given in Table 3.1. For instance, Iris has 150 data objects; each of them has 4 attributes; and the data objects are from 3 classes.

To generate multiple base clusterings for each data set, we use two classic clustering algorithms, k-means and complete-linkage, and vary the desired cluster number k in the range shown in Table 3.2. For instance, on Iris data set, we vary k from 2 to 10 with a step size of 1 for both k-means and complete-linkage algorithms. As a result, we obtain 18 base clusterings with a total of 108 clusters.

Then a simple undirected and vertex-weighted graph is constructed. Each vertex

Table 3.3: Average Performance in Terms of MWIS Weight

Method	Iris	Zoo	Semeion	PD	Vowel	ISOLET	Letter
MWBC	127.5	64.9	192	5265.7	319.5	1353.4	5014.8
FastWClq	**132.9**	**73.1**	**205.8**	5532.7	347.7	1422.4	5184.4
LSCC	**132.9**	**73.1**	**205.8**	**5535.2**	**348.1**	1498	5364.2
LSCC+BMS	**132.9**	**73.1**	**205.8**	**5535.2**	**348.1**	1497.2	**5364.5**
RRWL	**132.9**	**73.1**	**205.8**	**5535.2**	**348.1**	**1500.1**	**5364.5**
SAMIS	**132.9**	**73.1**	205.2	5518.5	347.9	1497.6	5322.7

represents a cluster. If two clusters c_i and c_j, which are from two different clusterings, contain some common data objects, we say they are overlapping. For any two overlapping clusters, there is an edge connecting the vertices representing them. The basic statistics of the derived graph of each data set are given in Table 3.2. Note that since k-means may return different clusterings for the same data set and the same k due to its randomness in initialization, we construct 100 graphs for each data set and report the average edge number and average vertex degree. The weight of each vertex is defined as sum of the silhouette coefficients of the data objects belonging to the corresponding cluster.

We first compare our SAMIS algorithm with state-of-the-art maximum-weight clique solvers, which are applied on the complementary graphs. In consideration of the randomness of k-means, we generate 10 graphs for each data set and report the average performance. The algorithms for comparison include FastWClq, LSCC, LSCC+BMS, RRWL and MWBC, which serves as the baseline and just returns the set of vertices belonging to the same base clustering and having the maximum sum of weights. For LSCC, the search depth L was set to 4,000. When employing the BMS heuristic, the parameter k was set to 100, as in Y. Wang et al. (2016). For FastWClq, the parameters k_0 and k_{max} for the dynamic BMS heuristic were set to 4 and 64 respectively, as in Cai & Lin (2016). For RRWL, we set the cut off to be 10 minutes and use one seed. FastWClq, LSCC, LSCC+BMS and RRWL are implemented in C++ and invoked from MATLAB.

As shown in Table 3.3, the performance of SAMIS is very close to those of state-of-the-art.

Then we evaluate the performance of our clustering aggregation approach CA+SAMIS. The comparison algorithms include COMUSA Mimaroglu & Erdil (2011), WEAC+SL D. Huang et al. (2015), WEAC+CL D. Huang et al. (2015), WEAC+AL D. Huang et al. (2015), GP-MGLA D. Huang et al. (2015), ECFG D. Huang et al. (2016a), PTA+SL D. Huang et al. (2016b), PTA+CL D. Huang et al. (2016b), PTA+AL D. Huang et al. (2016b) and PTGP D. Huang et al. (2016b). For these algorithms, we follow the author-recommended or default settings and parameters.

Note that COMUSA, ECFG and our CA+SAMIS can automatically determine the cluster number in the aggregated clustering, while the rest algorithms need it as an input parameter. For fair comparisons, we follow the experimental protocol in D. Huang et al. (2016a) and specify the cluster number for those "non-automatic" algorithms to be the one automatically estimated by CA+SAMIS. For CA+SAMIS, there may be a couple of data objects which are not covered by the aggregated clustering or are covered by more than one cluster due to the slight overlap. In that case, we perform the post-processing to assign such data objects to their nearest clusters.

The quality of the final aggregated clustering is measured in terms of the normalized mutual information (NMI) Strehl & Ghosh (2002b). A higher NMI indicates that the aggregated clustering matches the ground-truth class memberships better. In consideration of the randomness of k-means, we run experiment on each data set 100 times and report the average NMI.

As shown in Table 3.4, CA+SAMIS is very competitive in clustering aggregation compared with other state-of-the-art techniques.

Table 3.4: Average Performance of Clustering Aggregation in Terms of NMI

Method	Iris	Zoo	Semeion	PD	Vowel	ISOLET	Letter
COMUSA	0.346	0.577	0.395	0.509	0.409	0.534	**0.360**
WEAC+SL	0.688	0.687	0.419	0.496	0.404	0.575	0.274
WEAC+CL	0.700	0.688	0.434	0.516	0.412	0.588	0.280
WEAC+AL	0.700	0.696	0.434	0.534	0.411	0.596	0.281
GP-MGLA	0.706	0.692	0.445	0.548	0.411	0.602	0.291
ECFG	0.533	0.698	0.487	0.575	0.409	0.652	0.282
PTA+SL	0.345	0.668	0.431	0.463	0.375	0.563	0.249
PTA+CL	0.331	0.644	0.475	0.556	0.402	0.640	0.301
PTA+AL	0.348	0.660	0.473	0.541	0.399	0.639	0.301
PTGP	**0.754**	0.687	0.469	0.554	0.403	0.616	0.274
CA+SAMIS	0.700	**0.712**	**0.552**	**0.676**	**0.427**	**0.698**	0.359

3.4 Conclusion

We formulate clustering aggregation as a special instance of maximum-weight independent set problem and propose a novel local search algorithm for solving it. Experimental results on many real-world data sets demonstrate that both our algorithm for the maximum-weight independent set problem and our approach to clustering aggregation achieve good performance.

BIBLIOGRAPHY

Ablanedo-Rosas, J. H., & Rego, C. (2010). Surrogate constraint normalization for the set covering problem. *European Journal of Operational Research*, *205*(3), 540–551.

Ahmad, A., & Dey, L. (2007). A k-mean clustering algorithm for mixed numeric and categorical data. *Data & Knowledge Engineering*, *63*(2), 503–527.

Aoun, B., Boutaba, R., Iraqi, Y., & Kenward, G. (2006). Gateway placement optimization in wireless mesh networks with qos constraints. *IEEE Journal on Selected Areas in Communications*, *24*(11), 2127–2136.

Bautista, J., & Pereira, J. (2007). A grasp algorithm to solve the unicost set covering problem. *Computers & Operations Research*, *34*(10), 3162–3173.

Beasley, J. E. (1990). Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, *41*(11), 1069–1072.

Beasley, J. E., & Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European journal of operational research*, *94*(2), 392–404.

Böhm, C., Goebl, S., Oswald, A., Plant, C., Plavinski, M., & Wackersreuther, B. (2010). Integrative parameter-free clustering of data with mixed type attributes. In *Pacific-asia conference on knowledge discovery and data mining* (pp. 38–47).

Bouamama, S., & Blum, C. (2016). A hybrid algorithmic model for the minimum weight dominating set problem. *Simulation Modelling Practice and Theory*, *64*, 57–68.

Brendel, W., & Todorovic, S. (2010). Segmentation as maximum-weight independent set. In *Advances in neural information processing systems* (pp. 307–315).

Cai, S., & Lin, J. (2016). Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence, IJCAI 2016, new york, ny, usa, 9-15 july 2016* (pp. 568–574). Retrieved from `http://www.ijcai.org/Abstract/16/087`

Cai, S., Su, K., & Sattar, A. (2011). Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, *175*(9-10), 1672–1696.

Campan, A., Truta, T. M., & Beckerich, M. (2015). Fast dominating set algorithms for social networks. In *Maics* (pp. 55–62).

Cao, S., & Snavely, N. (2013). Graph-based discriminative learning for location recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 700–707).

Caprara, A., Fischetti, M., & Toth, P. (1999). A heuristic method for the set covering problem. *Operations research*, *47*(5), 730–743.

Caserta, M. (2007). Tabu search-based metaheuristic algorithm for large-scale set covering problems. In *Metaheuristics* (pp. 43–63). Springer.

Chaurasia, S. N., & Singh, A. (2015). A hybrid evolutionary algorithm with guided mutation for minimum weight dominating set. *Applied Intelligence*, *43*(3), 512–529.

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of operations research*, *4*(3), 233–235.

Cooper, C., Klasing, R., & Zito, M. (2005). Lower bounds and algorithms for dominating sets in web graphs. *Internet Mathematics*, *2*(3), 275–300.

Cormode, G., Karloff, H., & Wirth, A. (2010). Set cover algorithms for very large datasets. In *Proceedings of the 19th acm international conference on information and knowledge management* (pp. 479–488).

Crawford, B., Soto, R., Cuesta, R., & Paredes, F. (2014). Application of the artificial bee colony algorithm for solving the set covering problem. *The Scientific World Journal*, *2014*.

El Houmaidi, M., & Bassiouni, M. A. (2003). k-weighted minimum dominating sets for sparse wavelength converters placement under nonuniform traffic. In *Modeling, analysis and simulation of computer telecommunications systems, 2003. mascots 2003. 11th ieee/acm international symposium on* (pp. 56–61).

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd* (Vol. 96, pp. 226–231).

Eubank, S., Kumar, V., Marathe, M. V., Srinivasan, A., & Wang, N. (2004). Structural and algorithmic aspects of massive social networks. In *Proceedings of the fifteenth annual acm-siam symposium on discrete algorithms* (pp. 718–727).

Fan, Y., Li, N., Li, C., Ma, Z., Jan Latecki, L., & Su, K. (2017, 08). Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation.

Feige, U. (1998). A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, *45*(4), 634–652.

Fern, X. Z., & Brodley, C. E. (2004). Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the twenty-first international conference on machine learning* (p. 36).

Fränti, P., & Virmajoki, O. (2006). Iterative shrinking method for clustering problems. *Pattern Recognition*, *39*(5), 761–775.

Fred, A. L., & Jain, A. K. (2002). Data clustering using evidence accumulation. In *Pattern recognition, 2002. proceedings. 16th international conference on* (Vol. 4, pp. 276–280).

Gionis, A., Mannila, H., & Tsaparas, P. (2007). Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *1*(1), 4.

Golab, L., Karloff, H., Korn, F., Srivastava, D., & Yu, B. (2008). On generating near-optimal tableaux for conditional functional dependencies. *Proceedings of the VLDB Endowment*, *1*(1), 376–390.

Gomes, F. C., Meneses, C. N., Pardalos, P. M., & Viana, G. V. R. (2006). Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Computers & Operations Research*, *33*(12), 3520–3534.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 857–871.

Grossman, T., & Wool, A. (1997). Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, *101*(1), 81–92.

Hedar, A.-R., & Ismail, R. (2010). Hybrid genetic algorithm for minimum dominating set problem. In *International conference on computational science and its applications* (pp. 457–467).

Hedar, A.-R., & Ismail, R. (2012). Simulated annealing with stochastic local search for minimum dominating set problem. *International Journal of Machine Learning and Cybernetics*, *3*(2), 97–109.

Ho, C. K., Singh, Y. P., & Ewe, H. T. (2006). An enhanced ant colony optimization metaheuristic for the minimum dominating set problem. *Applied Artificial Intelligence*, *20*(10), 881–903.

Hsu, C.-C., & Chen, Y.-C. (2007). Mining of mixed data with application to catalog marketing. *Expert Systems with Applications*, *32*(1), 12–23.

Hsu, C.-C., & Huang, Y.-P. (2008). Incremental clustering of mixed data based on distance hierarchy. *Expert Systems with Applications*, *35*(3), 1177–1185.

Huang, D., Lai, J., & Wang, C. (2015). Combining multiple clusterings via crowd agreement estimation and multi-granularity link analysis. *Neurocomputing*, *170*, 240–250. Retrieved from `http://dx.doi.org/10.1016/j.neucom.2014.05.094` doi: 10.1016/j.neucom.2014.05.094

Huang, D., Lai, J., & Wang, C. (2016a). Ensemble clustering using factor graph. *Pattern Recognition*, *50*, 131–142. Retrieved from `http://dx.doi.org/10.1016/j.patcog.2015.08.015` doi: 10.1016/j.patcog.2015.08.015

Huang, D., Lai, J., & Wang, C. (2016b). Robust ensemble clustering using probability trajectories. *IEEE Trans. Knowl. Data Eng.*, *28*(5), 1312–1326. Retrieved from `http://dx.doi.org/10.1109/TKDE.2015.2503753` doi: 10.1109/TKDE.2015.2503753

Huang, Z. (1997). Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(pakdd)* (pp. 21–34).

Huang, Z. (1998). Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, *2*(3), 283–304.

Jiang, H., Li, C., & Manyà, F. (2017). An exact algorithm for the maximum weight clique problem in large graphs. In *Proceedings of the thirty-first AAAI conference on artificial intelligence, february 4-9, 2017, san francisco, california, USA.* (pp. 830–838). Retrieved from `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14370`

Jing, L., Ng, M. K., & Huang, J. Z. (2007). An entropy weighting k-means algorithm for subspace clustering of high-dimensional sparse data. *IEEE Transactions on Knowledge and Data Engineering*, *19*(8), 1026–1041.

Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, *9*(3), 256–278.

Kandola, J., Cristianini, N., & Shawe-taylor, J. S. (2003). Learning semantic similarity. In *Advances in neural information processing systems* (pp. 673–680).

Kann, V. (1992). *On the approximability of np-complete optimization problems* (Unpublished doctoral dissertation). Royal Institute of Technology Stockholm.

Kelleher, L. L., & Cozzens, M. B. (1988). Dominating sets in social network graphs. *Mathematical Social Sciences*, *16*(3), 267–279.

Kinney, G. W., Barnes, J. W., & Colletti, B. W. (2007). A reactive tabu search algorithm with variable clustering for the unicost set covering problem. *International Journal of Operational Research*, *2*(2), 156–172.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Lan, G., DePuy, G. W., & Whitehouse, G. E. (2007). An effective and simple heuristic for the set covering problem. *European journal of operational research*, *176*(3), 1387–1403.

Legendre, P., & Legendre, L. (1998). Numerical ecology, volume 24, (developments in environmental modelling).

Li, N., & Latecki, L. J. (2012). Clustering aggregation as maximum-weight independent set. In *Advances in neural information processing systems* (pp. 782–790).

Li, N., & Latecki, L. J. (2015). Affinity inference with application to recommender systems. In *Web intelligence and intelligent agent technology (wi-iat), 2015 ieee/wic/acm international conference on* (Vol. 1, pp. 393–400).

Li, N., & Latecki, L. J. (2017, 08). Affinity learning for mixed data clustering.

Lichman, M. (2013). *UCI machine learning repository.* Retrieved from `http://archive.ics.uci.edu/ml`

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, *28*(2), 129–137.

Lovász, L. (1975). On the ratio of optimal integral and fractional covers. *Discrete mathematics*, *13*(4), 383–390.

Magri, L., & Fusiello, A. (2016). Multiple model fitting as a set coverage problem. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3318–3326).

Mihail, M. (1999). Set cover with requirements and costs evolving over time. In *Randomization, approximation, and combinatorial optimization. algorithms and techniques* (pp. 63–72). Springer.

Mimaroglu, S., & Erdil, E. (2011). Combining multiple clusterings using similarity graph. *Pattern Recognition*, *44*(3), 694–703. Retrieved from `http://dx.doi.org/10.1016/j.patcog.2010.09.008` doi: 10.1016/j.patcog.2010.09.008

Mulati, M. H., & Constantino, A. A. (2011). Ant-line: A line-oriented aco algorithm for the set covering problem. In *Computer science society (sccc), 2011 30th international conference of the chilean* (pp. 265–274).

Nacher, J. C., & Akutsu, T. (2016). Minimum dominating set-based methods for analyzing biological networks. *Methods*, *102*, 57–63.

Naji-Azimi, Z., Toth, P., & Galli, L. (2010). An electromagnetism metaheuristic for the unicost set covering problem. *European Journal of Operational Research*, *205*(2), 290–300.

Nguyen, N., & Caruana, R. (2007). Consensus clusterings. In *Data mining, 2007. icdm 2007. seventh ieee international conference on* (pp. 607–612).

Nitash, C., & Singh, A. (2014). An artificial bee colony algorithm for minimum weight dominating set. In *Swarm intelligence (sis), 2014 ieee symposium on* (pp. 1–7).

Plant, C. (2012). Dependency clustering across measurement scales. In *Proceedings of the 18th acm sigkdd international conference on knowledge discovery and data mining* (pp. 361–369).

Plant, C., & Böhm, C. (2011). Inconco: interpretable clustering of numerical and categorical objects. In *Proceedings of the 17th acm sigkdd international conference on knowledge discovery and data mining* (pp. 1127–1135).

Podani, J. (1999). Extending gower's general coefficient of similarity to ordinal characters. *Taxon*, 331–340.

Potluri, A., & Singh, A. (2013). Hybrid metaheuristic algorithms for minimum weight dominating set. *Applied Soft Computing*, *13*(1), 76–88.

Raka, J., Milan, T., & Dana, S. (2010). Ant colony optimization applied to minimum weight dominating set problem. In *Proceedings of the 12th wseas international conference on automatic control, modelling & simulation* (pp. 322–326).

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial intelligence*, *32*(1), 57–95.

Ren, Z.-G., Feng, Z.-R., Ke, L.-J., & Zhang, Z.-J. (2010). New ideas for applying ant colony optimization to the set covering problem. *Computers & Industrial Engineering*, *58*(4), 774–784.

Saha, B., & Getoor, L. (2009). On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proceedings of the 2009 siam international conference on data mining* (pp. 697–708).

Samuel, H., Zhuang, W., & Preiss, B. (2009). Dtn based dominating set routing for manet in heterogeneous wireless networking. *Mobile Networks and Applications*, *14*(2), 154–164.

Sanchis, L. A. (2002). Experimental analysis of heuristic algorithms for the dominating set problem. *Algorithmica*, *33*(1), 3–18.

Sellis, T. K. (1988). Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, *13*(1), 23–52.

Shen, C., & Li, T. (2010). Multi-document summarization via the minimum dominating set. In *Proceedings of the 23rd international conference on computational linguistics* (pp. 984–992).

Singh, V., Mukherjee, L., Peng, J., & Xu, J. (2008). Ensemble clustering using semidefinite programming. In *Advances in neural information processing systems* (pp. 1353–1360).

Stergiou, S., & Tsioutsiouliklis, K. (2015). Set cover at web scale. In *Proceedings of the 21th acm sigkdd international conference on knowledge discovery and data mining* (pp. 1125–1133).

Stojmenovic, I., Seddigh, M., & Zunic, J. (2002). Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on parallel and distributed systems*, *13*(1), 14–25.

Strehl, A., & Ghosh, J. (2002a). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, *3*(Dec), 583–617.

Strehl, A., & Ghosh, J. (2002b). Cluster ensembles — A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, *3*, 583–617. Retrieved from `http://www.jmlr.org/papers/v3/strehl02a.html`

Subhadrabandhu, D., Sarkar, S., & Anjum, F. (2004). Efficacy of misuse detection in ad hoc networks. In *Sensor and ad hoc communications and networks, 2004. ieee secon 2004. 2004 first annual ieee communications society conference on* (pp. 97–107).

Sundar, S., & Singh, A. (2012). A hybrid heuristic for the set covering problem. *Operational Research*, *12*(3), 345–365.

Topchy, A., Jain, A. K., & Punch, W. (2003). Combining multiple weak clusterings. In *Data mining, 2003. icdm 2003. third ieee international conference on* (pp. 331–338).

Wang, F., Du, H., Camacho, E., Xu, K., Lee, W., Shi, Y., & Shan, S. (2011). On positive influence dominating sets in social networks. *Theoretical Computer Science*, *412*(3), 265–269.

Wang, Y., Cai, S., & Yin, M. (2016). Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of the thirtieth AAAI conference on artificial intelligence, february 12-17, 2016, phoenix, arizona, USA.* (pp. 805–811). Retrieved from `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11915`

Wang, Y., Cai, S., & Yin, M. (2017). Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, *58*, 267–295.

Wang, Y., Ouyang, D., Zhang, L., & Yin, M. (2017). A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. *Science China Information Sciences*, *60*(6), 062103.

Wu, P., Wen, J.-R., Liu, H., & Ma, W.-Y. (2006). Query selection techniques for efficient crawling of structured web sources. In *Data engineering, 2006. icde'06. proceedings of the 22nd international conference on* (pp. 47–47).

Xu, K., Boussemart, F., Hemery, F., & Lecoutre, C. (2005). A simple model to generate hard satisfiable instances. *arXiv preprint cs/0509032*.

Yagiura, M., Kishida, M., & Ibaraki, T. (2006). A 3-flip neighborhood local search for the set covering problem. *European Journal of Operational Research*, *172*(2), 472–499.

Yang, X., Prasad, L., & Latecki, L. J. (2013). Affinity learning with diffusion on tensor product graph. *IEEE transactions on pattern analysis and machine intelligence*, *35*(1), 28–38.

Yao, B., & Fei-Fei, L. (2012). Action recognition with exemplar based 2.5 d graph matching. In *European conference on computer vision* (pp. 173–186).

Zhao, X., & Ouyang, D. (2007). Improved algorithms for deriving all minimal conflict sets in model-based diagnosis. In *International conference on intelligent computing* (pp. 157–166).

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., & Schölkopf, B. (2003). Learning with local and global consistency. In *Nips* (Vol. 16, pp. 321–328).

APPENDIX A

Appendix A

Table A.1: Details of 139 Undirected Simple Graphs in Network Data Repository

Graph	#Vertex	#Edge	Graph	#Vertex	#Edge
bio-celegans	453	2025	soc-flickr	513969	3190452
bio-diseasome	516	1188	soc-flixster	2523386	7918801
bio-dmela	7393	25569	soc-gowalla	196591	950327
bio-yeast	1458	1948	soc-karate	34	78
ca-AstroPh	17903	196972	soc-lastfm	1191805	4519330
ca-CSphd	1882	1740	soc-livejournal	4033137	27933062
ca-CondMat	21363	91286	soc-orkut	2997166	106349209
ca-Erdos992	6100	7515	soc-pokec	1632803	22301964
ca-GrQc	4158	13422	soc-slashdot	70068	358647
ca-HepPh	11204	117619	soc-twitter-follows	404719	713319
ca-MathSciNet	332689	820644	soc-wiki-Vote	889	2914
ca-citeseer	227320	814134	soc-youtube	495957	1936748
ca-coauthors-dblp	540486	15245729	soc-youtube-snap	1134890	2987624
ca-dblp-2010	226413	716460	tech-RL-caida	190914	607610
ca-dblp-2012	317080	1049866	tech-WHOIS	7476	56943
ca-hollywood-2009	1069126	56306653	tech-as-caida2007	26475	53381
ca-netscience	379	914	tech-as-skitter	1694616	11094209
socfb-A-anon	3097165	23667394	tech-internet-as	40164	85123
socfb-B-anon	2937612	20959854	tech-p2p-gnutella	62561	147878
socfb-Berkeley13	22900	852419	tech-routers-rf	2113	6632
socfb-CMU	6621	249959	scc_enron-only	151	9828
socfb-Duke14	9885	506437	scc_fb-forum	897	71011
socfb-Indiana	29732	1305757	scc_fb-messages	1899	531893
socfb-MIT	6402	251230	scc_infect-dublin	10972	175573
socfb-OR	63392	816886	scc_infect-hyper	113	6222
socfb-Penn94	41536	1362220	scc_reality	6809	4714485
socfb-Stanford3	11586	568309	scc_retweet	18469	65990
socfb-Texas84	36364	1590651	scc_retweet-crawl	1131801	24015
socfb-UCLA	20453	747604	scc_rt_alwefaq	4157	355
socfb-UCSB37	14917	482215	scc_rt_assad	2035	96
socfb-UConn	17206	604867	scc_rt_bahrain	4659	129
socfb-UF	35111	1465654	scc_rt_barackobama	9551	226
socfb-UIllinois	30795	1264421	scc_rt_damascus	2962	41
socfb-Wisconsin87	23831	835946	scc_rt_dash	5968	39
socfb-uci-uni	58790782	92208195	scc_rt_gmanews	8330	1078
inf-power	4941	6594	scc_rt_gop	3716	7
inf-road-usa	23947347	28854312	scc_rt_http	5691	6
inf-roadNet-CA	1957027	2760388	scc_rt_israel	3686	12
inf-roadNet-PA	1087562	1541514	scc_rt_justinbieber	9364	442
ia-email-EU	32430	54397	scc_rt_ksa	5775	23
ia-email-univ	1133	5451	scc_rt_lebanon	3370	5
ia-enron-large	33696	180811	scc_rt_libya	5021	26
ia-enron-only	143	623	scc_rt_lolgop	9742	4510
ia-fb-messages	1266	6451	scc_rt_mittromney	7850	108
ia-infect-dublin	410	2765	scc_rt_obama	3040	4
ia-infect-hyper	113	2196	scc_rt_occupy	3090	60
ia-reality	6809	7680	scc_rt_occupywallstnyc	3594	931
ia-wiki-Talk	92117	360767	scc_rt_oman	4452	13
rec-amazon	91813	125704	scc_rt_onedirection	7704	368
rt-retweet	96	117	scc_rt_p2	4785	15
rt-retweet-crawl	1112702	2278852	scc_rt_qatif	6718	11
rt-twitter-copen	761	1029	scc_rt_saudi	6805	91
sc-ldoor	952203	20770807	scc_rt_tcot	4506	18
sc-msdoor	415863	9378650	scc_rt_tlot	3513	8
sc-nasasrb	54870	1311227	scc_rt_uae	4757	12
sc-pkustk11	87804	2565054	scc_rt_voteonedirection	1833	5
sc-pkustk13	94893	3260967	scc_twitter-copen	8580	473614
sc-pwtk	217891	5653221	web-BerkStan	12305	19500
sc-shipsec1	140385	1707759	web-arabic-2005	163598	1747269
sc-shipsec5	179104	2200076	web-edu	3031	6474
soc-BlogCatalog	88784	2093195	web-google	1299	2773
soc-FourSquare	639014	3214986	web-indochina-2004	11358	47606
soc-LiveMocha	104103	2193083	web-it-2004	509338	7178413
soc-brightkite	56739	212945	web-polblogs	643	2280
soc-buzznet	101163	2763066	web-sk-2005	121422	334419
soc-delicious	536108	1365961	web-spam	4767	37375
soc-digg	770799	5907132	web-uk-2005	129632	11744049
soc-dolphins	62	159	web-webbase-2001	16062	25593
soc-douban	154908	327162	web-wikipedia2009	1864433	4507315
soc-epinions	26588	100120			

Table A.2: Minimum Weighted Dominating Set Results on 139 Real World Graphs from Network Data Repository (Solution Weight), Part 1

Graph	*Gr*	*GrR*	*ACO-PP-LS*	CC^2FS	Ours
bio-celegans	1907.3	1871.1	N/A	N/A	1792.8
bio-diseasome	7183	6940.5	N/A	N/A	6601
bio-dmela	121123.5	118310	117222	113500.3	113830.9
bio-yeast	28345	27178.1	27091.5	26285	26305.6
ca-AstroPh	148904	144617.5	n/a	134418.9	135247.9
ca-CSphd	49424.9	48244.9	47194.8	46456	46487.5
ca-CondMat	228645	223809	n/a	207176.9	209028.4
ca-Erdos992	142783.5	141440	140849	140362	140378
ca-GrQc	62114.9	59647.2	60389	56035.1	56351.9
ca-HepPh	134563.5	131327.5	n/a	122729.4	123935.1
ca-MathSciNet	5648130	5456670	n/a	5326405.3	5240994
ca-citeseer	3064205	3021815	n/a	2940554.6	2889754
ca-coauthors-dblp	2749605	2700820	n/a	2595357.9	2534643
ca-dblp-2010	3535630	3496300	n/a	3472060.5	3444017
ca-dblp-2012	4013930	3900240	n/a	3757678.9	3685284
ca-hollywood-2009	3897650	3841835	N/A	N/A	3606478
ca-netscience	4645.6	4610.3	N/A	N/A	4264.1
socfb-A-anon	17861750	17400200	N/A	N/A	17061460
socfb-B-anon	16931550	16460500	N/A	N/A	16126930
socfb-Berkeley13	103100	100609.5	n/a	94297.5	94541.8
socfb-CMU	28505.2	27666.2	28349	26054.7	26360.7
socfb-Duke14	37528.4	36644.5	n/a	33994.7	34046.7
socfb-Indiana	104758	102822	n/a	94858.4	95513.5
socfb-MIT	33384	32697.1	33081	30821.4	31096.7
socfb-OR	840032	818234	n/a	4463030	785667.2
socfb-Penn94	186478	181802	n/a	168791.6	169507.2
socfb-Stanford3	64388.2	62445	n/a	58800.9	59056.7
socfb-Texas84	130883	127694.5	n/a	118419.7	118939.6
socfb-UCLA	107818	105645.5	n/a	98778.3	99423
socfb-UCSB37	66753.7	65035	n/a	60588.8	60873.3
socfb-UConn	67309.8	65413.4	n/a	60941.3	61256.5
socfb-UF	121989.5	119357.5	n/a	110979.5	111322.4
socfb-UIllinois	105872	103401	n/a	95639.1	96473.9
socfb-Wisconsin87	91720.1	89420.1	n/a	83315.2	83819
socfb-uci-uni	86992900	85142100	N/A	N/A	84069030
inf-power	129175	127528	127745	121060.1	122513.8
inf-road-usa	670320500	649191000	N/A	N/A	628835100
inf-roadNet-CA	56264000	54060800	N/A	N/A	52519970
inf-roadNet-PA	31095850	29812350	N/A	N/A	28979500
ia-email-EU	74503.5	73033.4	n/a	72359	72359
ia-email-univ	17253.7	16920.8	16723.6	15704	15862.3
ia-enron-large	152112.5	150945	n/a	147191.9	146819.2
ia-enron-only	1700.8	1704	N/A	N/A	1514
ia-fb-messages	19243	18420.6	18464.4	17915	17926
ia-infect-dublin	2866.6	2847.7	N/A	N/A	2373.9
ia-infect-hyper	99	70	N/A	N/A	70
ia-reality	3610	3601	3601	3601	3601
ia-wiki-Talk	1002125	986974.5	n/a	972951.6	973320
rec-amazon	2239510	2184010	n/a	2093432.6	2102511
rt-retweet	1190	1172	N/A	N/A	1162
rt-retweet-crawl	7540515	7283490	N/A	N/A	7130382
rt-twitter-copen	16709.3	15996.9	N/A	N/A	15412
sc-ldoor	5533210	5533210	n/a	5459928.6	5443677
sc-msdoor	1606400	1606400	n/a	1578798.7	1571300
sc-nasasrb	46542.3	46542.3	n/a	37792.3	39612.4
sc-pkustk11	103030	103030	n/a	94835	95938.9
sc-pkustk13	66665.5	66665.5	n/a	58797	57050.6
sc-pwtk	353438.5	353737.5	n/a	278306.8	291743.8
sc-shipsec1	435447.5	435447.5	n/a	390430.7	403784.6
sc-shipsec5	570797	570830	n/a	516918.3	530423.7
soc-BlogCatalog	396890	389767.5	n/a	383122.5	383529.3
soc-FourSquare	5512305	5448815	n/a	5416931.7	5391189
soc-LiveMocha	102282	99674.7	n/a	94551.9	94283.7
soc-brightkite	1012095	997156	n/a	978472	981078.9
soc-buzznet	7898.8	7699.3	n/a	7050.4	7025
soc-delicious	5070800	4964710	n/a	4929250.4	4903925
soc-digg	5683720	5557835	n/a	5502484.2	5453174
soc-dolphins	421.2	389	N/A	N/A	361
soc-douban	827531.5	814256.5	n/a	809674.8	809560.6
soc-epinions	522156	511182	n/a	499497.6	500945.6

Table A.3: Minimum Weighted Dominating Set Results on 139 Real World Graphs from Network Data Repository (Solution Weight), Part 2

Graph	Gr	GrR	$ACO\text{-}PP\text{-}LS$	CC^2FS	Ours
soc-flickr	8140620	8018110	n/a	7946198.7	7888499
soc-flixster	8754370	8657125	N/A	N/A	8604676
soc-gowalla	3185410	3134890	n/a	3098493.3	3066474
soc-karate	77	77	N/A	N/A	70
soc-lastfm	6170720	6106535	N/A	N/A	6062298
soc-livejournal	62905200	61379500	N/A	N/A	59628020
soc-orkut	7975515	7782095	N/A	N/A	7102610
soc-pokec	15754250	15306400	N/A	N/A	14600560
soc-slashdot	1090485	1078410	n/a	1064886.6	1066510
soc-twitter-follows	231233	229546.5	n/a	228759	228773.1
soc-wiki-Vote	15027.7	14526.4	N/A	N/A	14205.9
soc-youtube	7231355	7070885	n/a	6994482.9	6923370
soc-youtube-snap	17295600	17007750	N/A	N/A	16773990
tech-RL-caida	3339890	3245995	n/a	3170840.2	3143612
tech-WHOIS	50328.9	49459.4	48007	46218.1	46409.8
tech-as-caida2007	203676	199092.5	n/a	194595.9	194861.3
tech-as-skitter	13733200	13574950	N/A	N/A	13131270
tech-internet-as	312165	304226.5	n/a	297112.8	297452.3
tech-p2p-gnutella	1090755	1069930	n/a	1056833.7	1058023
tech-routers-rf	38621.6	36924.5	36443.4	35485	35652
scc-enron-only	766.2	766.2	N/A	N/A	761
scc-fb-forum	38913	38827	N/A	N/A	38793.4
scc-fb-messages	61323	61316.5	N/A	N/A	61308
scc-infect-dublin	42407.4	42033.6	N/A	N/A	39273.2
scc-infect-hyper	2	2	N/A	N/A	2
scc-reality	2288	2238	N/A	N/A	2059
scc-retweet	1741310	1740940	N/A	N/A	1740641
scc-retweet-crawl	112468000	112457000	N/A	N/A	112448000
scc-rt-alwefaq	408523	408521	N/A	N/A	408445
scc-rt-assad	199216	199163	N/A	N/A	199153
scc-rt-bahrain	458554	458551	N/A	N/A	458551
scc-rt-barackobama	949067	949039	N/A	N/A	949039
scc-rt-damascus	292184.5	292132	N/A	N/A	292132
scc-rt-dash	595126	595097	N/A	N/A	595074
scc-rt-gmanews	819630	819546	N/A	N/A	819546
scc-rt-gop	367859	367859	N/A	N/A	367859
scc-rt-http	566894	566894	N/A	N/A	566861
scc-rt-israel	364585	364585	N/A	N/A	364585
scc-rt-justinbieber	932178	932159	N/A	N/A	932075
scc-rt-ksa	576943	576943	N/A	N/A	576943
scc-rt-lebanon	335789	335789	N/A	N/A	335789
scc-rt-libya	500742	500742	N/A	N/A	500742
scc-rt-lolgop	946802	946761	N/A	N/A	946761
scc-rt-mittromney	777502.5	777350	N/A	N/A	777325
scc-rt-obama	301927	301927	N/A	N/A	301927
scc-rt-occupy	301361	301361	N/A	N/A	301276
scc-rt-occupywallstnyc	348615	348525	N/A	N/A	348513
scc-rt-oman	442537	442530	N/A	N/A	442530
scc-rt-onedirection	766678	766678	N/A	N/A	766640
scc-rt-p2	477965	477965	N/A	N/A	477965
scc-rt-qatif	669249	669238	N/A	N/A	669238
scc-rt-saudi	681221	681221	N/A	N/A	681215
scc-rt-tcot	445941	445936	N/A	N/A	445936
scc-rt-tlot	347267	347267	N/A	N/A	347267
scc-rt-uae	473533.5	473506.5	N/A	N/A	473488
scc-rt-voteonedirection	180898	180898	N/A	N/A	180898
scc-twitter-copen	630476	629775.5	N/A	N/A	629220.7
web-BerkStan	296346.5	295995	n/a	288930.1	290274.8
web-arabic-2005	1644745	1616085	n/a	1582922	1579468
web-edu	23534	23110.1	23108.6	23105	23106
web-google	16449.4	15810.2	15419	15036	15059.3
web-indochina-2004	120243.5	119038	n/a	116995.9	117075.6
web-it-2004	2799590	2676965	n/a	2622204.3	2579465
web-polblogs	7779.1	7467.9	N/A	N/A	7217
web-sk-2005	2299320	2276545	n/a	2257320.7	2253954
web-spam	66204.7	64036.2	64817	61938	62012.9
web-uk-2005	93629.4	93254.9	n/a	93183	93183
web-webbase-2001	99465	97384.6	n/a	94954	95217.8
web-wikipedia2009	28569150	27789900	N/A	N/A	26954720

Table A.4: Minimum Dominating Set Results on 139 Real World Graphs from Network Data Repository (Vertex Number), Part 1

Graph	Gr	GrR	Gr_Rev	Gr_Vote	$SAMDS$	Ours
bio-celegans	30.5	30.5	29	30	31	29
bio-diseasome	99.1	96	96	97	98.4	96
bio-dmela	1479	1456.1	1481.1	1458.5	1486	1453
bio-yeast	359.1	353.6	356.9	355.4	359.6	353
ca-AstroPh	2179.2	2131.5	2153.7	2114.2	2220	2070
ca-CSphd	530.7	523.5	524.9	526.7	528.5	523
ca-CondMat	3108	3050.7	3055.5	3049	3149.6	2996.1
ca-Erdos992	1446.2	1446	1446	1446	1447.6	1446
ca-GrQc	801.6	781.1	786.9	785.1	805	776
ca-HepPh	1733.4	1686	1696.3	1690.7	1737.6	1665
ca-MathSciNet	66387.9	65701.5	65852.4	65783.5	286928.1	65577.3
ca-citeseer	33991.3	33479.5	33469.9	33519.4	120732.7	33214.8
ca-coauthors-dblp	40323.5	38863.3	38969.7	37938.4	N/A	36010
ca-dblp-2010	36079.6	35604.1	35605.1	35611.8	118643	35367.4
ca-dblp-2012	46967.9	46421.2	46464.9	46444.3	225713.5	46153.2
ca-hollywood-2009	53250.8	52147.7	53612.5	50822.9	N/A	49493.8
ca-netscience	55.8	55.8	55	56	59	55
socfb-A-anon	203464	201844	203077	201852	N/A	201698.6
socfb-B-anon	188089.5	187077.5	187774	187104	N/A	187032.8
socfb-Berkeley13	1830.4	1744.7	1882.5	1664	1935.9	1642
socfb-CMU	499.9	472.5	506.6	454	516.7	444.1
socfb-Duke14	666.2	638.7	679.1	608.1	710.1	598
socfb-Indiana	1984.1	1893.8	2074.8	1745.9	2128	1729
socfb-MIT	567.6	544.5	563	523	577.6	520.2
socfb-OR	11366.6	10919.6	11290.6	10854.2	21427.3	10728.6
socfb-Penn94	3408.6	3249.8	3502	3068.4	3605.8	3038.8
socfb-Stanford3	1006	964.6	1029.1	939	1037.8	931
socfb-Texas84	2465.7	2335.1	2587.3	2179.1	2670.7	2163
socfb-UCLA	1849.4	1756.8	1912.9	1678	1940.6	1651.1
socfb-UCSB37	1250	1186.1	1297.1	1128.2	1348.7	1109
socfb-UConn	1255.1	1201.4	1297.9	1117	1325	1098
socfb-UF	2333.7	2247	2505.9	2077.4	2570.3	2065.4
socfb-UIllinois	2022.2	1953.6	2138.7	1804.2	2201.4	1788.2
socfb-Wisconsin87	1699.5	1627.8	1769.5	1526	1832.5	1511.2
socfb-uci-uni	865896.5	865676.5	865702	865684.5	58790782	865675
inf-power	1565.5	1507.2	1547.8	1514.7	1554.3	1487.1
inf-road-usa	8628450	8147765	8431275	8184455	N/A	7974437
inf-roadNet-CA	663688	625317	655848	622936.5	N/A	609320.4
inf-roadNet-PA	370808	347003	363593	346400.5	N/A	338740.6
ia-email-EU	755.2	755	755	755	755.8	755
ia-email-univ	224.5	215.3	225.4	214	225	211
ia-enron-large	2000.9	1992.1	2020.6	1990.3	2085.6	1979.3
ia-enron-only	21.3	21.3	23	21	23.5	21
ia-fb-messages	259.9	250.8	255	254	257.3	249
ia-infect-dublin	51	50.8	54	50	56.6	47.9
ia-infect-hyper	3	3	3	3	6	3
ia-reality	81	81	81	81	81.1	81
ia-wiki-Talk	11952	11935	11952.1	11936.8	46626	11935
rec-amazon	30819.4	28775.9	29224.6	29064.8	57388.3	28365.7
rt-retweet	32	32	32	32	32.3	32
rt-retweet-crawl	75901.9	75740	75768.5	75753.4	N/A	75740
rt-twitter-copen	201.3	199	199.3	200	200.9	199
sc-ldoor	66709.2	66709.2	67363	65992.3	496162	65387.7
sc-msdoor	21592	21592	21797.2	21351.3	N/A	21073.1
sc-nasasrb	1429.7	1429.7	1518.7	1360.6	1785.7	1306.7
sc-pkustk11	2709.6	2709.6	2683.1	2671.1	N/A	2573.5
sc-pkustk13	1480.8	1480.8	1597.1	1462.5	45262	1399.1
sc-pwtk	5660.7	5659.2	6087.4	5620.4	141625.8	5455.1
sc-shipsec1	9420.9	9420.9	11544.3	9305.3	61189.1	9091.4
sc-shipsec5	12670	12665.4	16586.5	12350.5	89572.7	12069.8
soc-BlogCatalog	4899.9	4894	4901	4895	46114.3	4894
soc-FourSquare	61441.2	61017.8	61159	61050	366356	60984.9
soc-LiveMocha	1484.7	1471.8	1564.8	1434	41257	1430.1
soc-brightkite	13085.2	12951.5	13025.8	12977.4	33406.2	12940
soc-buzznet	133.4	131.5	136.4	130	42455.8	128.2
soc-delicious	56071.4	55765.2	55929.4	55770.3	358228.5	55725.2
soc-digg	66826.4	66179.7	66583	66227.6	N/A	66155
soc-dolphins	15.7	14	16	15	16	14
soc-douban	8373.1	8364	8364	8364	81199.9	8364
soc-epinions	6496.5	6437.5	6458.7	6443.5	6520.9	6435

Table A.5: Minimum Dominating Set Results on 139 Real World Graphs from Network Data Repository (Vertex Number), Part 2

Graph	Gr	GrR	Gr_Rev	Gr_Vote	$SAMDS$	Ours
soc-flickr	98758.6	98104.6	98404	98179.8	394866.5	98064.3
soc-flixster	91245.6	91019	91019.4	91043.3	N/A	91019
soc-gowalla	42554.6	41777.9	42245.6	41830.5	169392.6	41627.4
soc-karate	4	4	4	4	4.9	4
soc-lastfm	67401.6	67226.8	67233.4	67237.4	N/A	67226
soc-livejournal	816264	797399	807055.5	798791.5	N/A	794323.6
soc-orkut	125093.5	120744.5	134075	114415	N/A	113901
soc-pokec	222260	213956	222004.5	210930	N/A	209070.7
soc-slashdot	14207.7	14158.9	14208.4	14162.5	48639.9	14157
soc-twitter-follows	2269.1	2269	2269	2269	201632.3	2269
soc-wiki-Vote	216	210.2	213.4	210.4	213.6	209.2
soc-youtube	90671	89788.3	90495.7	89893.2	449984.4	89733.5
soc-youtube-snap	214184	213140	213581	213275.5	N/A	213122.1
tech-RL-caida	41465.6	40594.2	41559.8	40651.6	109468.6	40224.8
tech-WHOIS	621.8	615.1	624	617.6	635.9	611
tech-as-caida2007	2407.6	2400.2	2405.1	2400.2	2411.1	2400
tech-as-skitter	186905	184377	189407	183934	N/A	182386.6
tech-internet-as	3688.6	3679.3	3684.4	3681.3	3707.9	3679
tech-p2p-gnutella	12740.2	12572	12592	12581.3	45771.5	12571
tech-routers-rf	488.6	480.7	486.7	482.1	487.1	479
scc-enron-only	6	6	6	6	6.4	6
scc-fb-forum	436	436	436	436	436.1	436
scc-fb-messages	635	635	635	635	635	634
scc-infect-dublin	879.3	865.5	835.9	850	872.5	826.5
scc-infect-hyper	1	1	1	1	1	1
scc-reality	53.1	53	53	53	56.5	53
scc-retweet	17417.6	17415.2	17417	17416	17417.5	17415
scc-retweet-crawl	1120790	1120720	1120725	1120740	N/A	1120710
scc-rt-alwefaq	4102	4102	4102	4102	4102	4102
scc-rt-assad	2010	2010	2010	2010	2010	2010
scc-rt-bahrain	4614	4614	4614	4614	4614	4614
scc-rt-barackobama	9486	9486	9486	9486	9486.1	9486
scc-rt-damascus	2939	2939	2940	2939	2939.4	2939
scc-rt-dash	5949	5949	5949	5949	5949	5949
scc-rt-gmanews	8207	8207	8207	8207	8207	8207
scc-rt-gop	3709	3709	3709	3709	3709	3709
scc-rt-http	5687	5687	5687	5687	5687	5687
scc-rt-israel	3675	3675	3675	3675	3675	3675
scc-rt-justinbieber	9309	9309	9309	9309	9309	9309
scc-rt-ksa	5762	5762	5762	5762	5762	5762
scc-rt-lebanon	3365	3365	3365	3365	3365	3365
scc-rt-libya	5004	5004	5004	5004	5004	5004
scc-rt-lolgop	9483	9483	9483	9483	9483	9483
scc-rt-mittromney	7784	7784	7784	7784	7784.3	7784
scc-rt-obama	3036	3036	3036	3036	3036	3036
scc-rt-occupy	3054	3054	3054	3054	3054.3	3054
scc-rt-occupywallstnyc	3477	3477	3477	3477	3477	3477
scc-rt-oman	4441	4441	4441	4441	4441	4441
scc-rt-onedirection	7672	7672	7672	7672	7672.1	7672
scc-rt-p2	4771	4771	4771	4771	4771	4771
scc-rt-qatif	6708	6708	6708	6708	6708	6708
scc-rt-saudi	6783	6783	6783	6783	6783.1	6783
scc-rt-tcot	4491	4491	4491	4491	4491	4491
scc-rt-tlot	3506	3506	3506	3506	3506	3506
scc-rt-uae	4746	4746	4746	4746	4746	4746
scc-rt-voteonedirection	1829	1829	1829	1829	1829	1829
scc-twitter-copen	6413.6	6410.3	6412	6410	6416.6	6410
web-BerkStan	3053.6	3015.2	3052.3	3028.1	3072.7	3000
web-arabic-2005	17432.3	17050.9	17236.6	17114.4	81385.8	16946.9
web-edu	249	249	249	249	252.6	249
web-google	209.3	206	206.2	206	208	205
web-indochina-2004	1499.7	1489.7	1489	1495	1502.2	1489
web-it-2004	33274	33010.4	33015.2	33022.6	N/A	32997
web-polblogs	107.3	104	107	104.1	106.8	104
web-sk-2005	26760.3	26547.9	28315.2	26601.4	69232.6	26472.9
web-spam	847	834.9	837.8	835	845.4	831
web-uk-2005	1423.7	1421	1421	1421	N/A	1421
web-webbase-2001	1020.8	1006.5	1050.7	1011	1040.5	1005
web-wikipedia2009	353065	348155	352885	348537.5	N/A	347018.1

Table A.6: Minimum Weighted Dominating Set Results on 139 Real World Graphs with Different Sets of Parameter Combinations (solution weight), Part 1

Graph	Ours (264)	Ours (21)	Ours (8)	Ours (1)
bio-celegans	1792.8	1828	1838	1838
bio-diseasome	6601	6611	6611	6615.1
bio-dmela	113830.9	113830.9	113830.9	114560.2
bio-yeast	26305.6	26312	26312	26343.3
ca-AstroPh	135247.9	135416.9	135416.9	136131.1
ca-CSphd	46487.5	46523.2	46535.2	46945.2
ca-CondMat	209028.4	209037.1	209439.3	210572.1
ca-Erdos992	140378	140378	140378	140563
ca-GrQc	56351.9	56431	56568.6	56747.6
ca-HepPh	123935.1	124056.9	124056.9	124265.3
ca-MathSciNet	5240994	5244600	5244600	5269835
ca-citeseer	2889754	2894713	2896497	2914613
ca-coauthors-dblp	2534643	2546158	2546158	2555403
ca-dblp-2010	3444017	3445555	3445555	3447730
ca-dblp-2012	3685284	3689587	3689866	3712836
ca-hollywood-2009	3606478	3606478	3606478	3617160
ca-netscience	4264.1	4264.3	4265	4319.3
socfb-A-anon	17061460	17070800	17070800	17095100
socfb-B-anon	16126930	16132080	16132080	16161110
socfb-Berkeley13	94541.8	94541.8	94541.8	94924.5
socfb-CMU	26360.7	26360.7	26361	26553.3
socfb-Duke14	34046.7	34109	34131	34304
socfb-Indiana	95513.5	95577.5	95577.5	95886
socfb-MIT	31096.7	31096.7	31101	31177.1
socfb-OR	785667.2	786089.8	786089.8	788519.2
socfb-Penn94	169507.2	169559.3	169580.7	170542.9
socfb-Stanford3	59056.7	59090.5	59090.5	59267
socfb-Texas84	118939.6	119017.9	119017.9	119442.1
socfb-UCLA	99423	99423	99423	99775.5
socfb-UCSB37	60873.3	60873.3	60873.3	61014.2
socfb-UConn	61256.5	61260.4	61329.5	61610.3
socfb-UF	111322.4	111322.4	111322.4	111735.2
socfb-UIllinois	96473.9	96705.6	96745.6	97053.6
socfb-Wisconsin87	83819	83879	83879	83936.7
socfb-uci-uni	84069030	84084270	84084270	84153010
inf-power	122513.8	122800.5	122800.5	122973.4
inf-road-usa	628835100	628951500	628951500	630609900
inf-roadNet-CA	52519970	52524450	52524450	52639340
inf-roadNet-PA	28979500	28979500	28979500	29050040
ia-email-EU	72359	72367	72367	72408.1
ia-email-univ	15862.3	15862.3	15862.3	15935.1
ia-enron-large	146819.2	146819.2	146819.2	147147.8
ia-enron-only	1514	1516	1516	1522
ia-fb-messages	17926	17926	17926	18015.4
ia-infect-dublin	2373.9	2374	2374	2431
ia-infect-hyper	70	70	70	70
ia-reality	3601	3601	3601	3601
ia-wiki-Talk	973320	974313.5	974326.9	975305.7
rec-amazon	2102511	2103468	2104804	2113931
rt-retweet	1162	1162	1162	1162
rt-retweet-crawl	7130382	7131385	7131543	7147075
rt-twitter-copen	15412	15420.1	15435	15639.5
sc-ldoor	5443677	5444511	5444511	5448435
sc-msdoor	1571300	1571547	1571573	1573072
sc-nasasrb	39612.4	39964	39964	41523.8
sc-pkustk11	95938.9	96100.7	96100.7	96158.9
sc-pkustk13	57050.6	57383.4	57383.4	58036.2
sc-pwtk	291743.8	295548.7	295548.7	300450.7
sc-shipsec1	403784.6	404992.4	404992.4	406244.9
sc-shipsec5	530423.7	532945.7	532945.7	534082.6
soc-BlogCatalog	383529.3	383710.6	383776.7	384229.4
soc-FourSquare	5391189	5392623	5394607	5395300
soc-LiveMocha	94283.7	94292.9	94301.1	94737.2
soc-brightkite	981078.9	981336.1	981336.1	982608.6
soc-buzznet	7025	7070.4	7070.4	7160
soc-delicious	4903925	4904882	4906103	4908477
soc-digg	5453174	5459081	5459081	5463582
soc-dolphins	361	361	361	385
soc-douban	809560.6	809573.1	809933.4	809977.6
soc-epinions	500945.6	500961.5	500961.5	501857.6

Table A.7: Minimum Weighted Dominating Set Results on 139 Real World Graphs with Different Sets of Parameter Combinations (solution weight), Part 2

Graph	Ours (264)	Ours (21)	Ours (8)	Ours (1)
soc-flickr	7888499	7895167	7898808	7899993
soc-flixster	8604676	8605179	8608010	8608244
soc-gowalla	3066474	3067010	3067010	3073083
soc-karate	70	70	70	70
soc-lastfm	6062298	6062867	6064896	6065356
soc-livejournal	59628020	59635580	59635580	59816360
soc-orkut	7102610	7104485	7104485	7157335
soc-pokec	14600560	14601440	14601440	14659990
soc-slashdot	1066510	1067030	1067108	1067507
soc-twitter-follows	228773.1	228773.1	228774	228954
soc-wiki-Vote	14205.9	14205.9	14249.2	14272
soc-youtube	6923370	6924653	6924653	6941947
soc-youtube-snap	16773990	16786150	16786370	16805080
tech-RL-caida	3143612	3144663	3144663	3153399
tech-WHOIS	46409.8	46420.1	46441.7	46671.4
tech-as-caida2007	194861.3	195015.9	195057.2	195802
tech-as-skitter	13131270	13131270	13131270	13162260
tech-internet-as	297452.3	297509	297568.7	298265.4
tech-p2p-gnutella	1058023	1058458	1058920	1059031
tech-routers-rf	35652	35668.5	35668.5	35702
scc-enron-only	761	761	761	761
scc-fb-forum	38793.4	38793.4	38793.4	38813
scc-fb-messages	61308	61308	61308	61312.6
scc-infect-dublin	39273.2	39311.5	39398.4	39462.6
scc-infect-hyper	2	2	2	2
scc-reality	2059	2077	2077	2077.1
scc-retweet	1740641	1740641	1740670	1740702
scc-retweet-crawl	112448000	112448000	112449000	112450000
scc-rt-alwefaq	408445	408445	408445	408445
scc-rt-assad	199153	199153	199153	199153
scc-rt-bahrain	458551	458551	458551	458551
scc-rt-barackobama	949039	949039	949039	949039
scc-rt-damascus	292132	292132	292132	292132
scc-rt-dash	595074	595074	595074	595097
scc-rt-gmanews	819546	819546	819546	819546
scc-rt-gop	367859	367859	367859	367859
scc-rt-http	566861	566861	566894	566894
scc-rt-israel	364585	364585	364585	364585
scc-rt-justinbieber	932075	932098	932098	932098
scc-rt-ksa	576943	576943	576943	576943
scc-rt-lebanon	335789	335789	335789	335789
scc-rt-libya	500742	500742	500742	500742
scc-rt-lolgop	946761	946761	946761	946761
scc-rt-mittromney	777325	777325	777350	777350
scc-rt-obama	301927	301927	301927	301927
scc-rt-occupy	301276	301276	301276	301317
scc-rt-occupywallstnyc	348513	348513	348513	348513
scc-rt-oman	442530	442530	442530	442530
scc-rt-onedirection	766640	766640	766640	766677
scc-rt-p2	477965	477965	477965	477965
scc-rt-qatif	669238	669238	669238	669238
scc-rt-saudi	681215	681215	681215	681215
scc-rt-tcot	445936	445936	445936	445936
scc-rt-tlot	347267	347267	347267	347267
scc-rt-uae	473488	473488	473488	473488
scc-rt-voteonedirection	180898	180898	180898	180898
scc-twitter-copen	629220.7	629247.9	629252.8	629274.6
web-BerkStan	290274.8	290274.8	290368	290458.1
web-arabic-2005	1579468	1579468	1579468	1580014
web-edu	23106	23106	23106	23106.2
web-google	15059.3	15067.5	15069.3	15148.4
web-indochina-2004	117075.6	117075.6	117082.3	117100.1
web-it-2004	2579465	2587880	2587880	2592358
web-polblogs	7217	7245	7296.6	7331.2
web-sk-2005	2253954	2254966	2254966	2254966
web-spam	62012.9	62012.9	62012.9	62242
web-uk-2005	93183	93183.1	93183.1	93183.2
web-webbase-2001	95217.8	95266.8	95266.8	95590.2
web-wikipedia2009	26954720	26959580	26959580	27046630

Table A.8: Minimum Dominating Set Results on 139 Real World Graphs with Different Sets of Parameter Combinations (Vertex Number)

Graph	Ours (410)	Ours (21)	Ours (8)	Ours (1)	Graph	Ours (410)	Ours (21)	Ours (8)	Ours (1)
bio-celegans	29	29	29	29	soc-flickr	98064.3	98066.5	98067	98067.4
bio-diseasome	96	96	96	96	soc-flixster	91019	91019	91019	91019
bio-dmela	1453	1453	1453	1453	soc-gowalla	41627.4	41636.2	41636.9	41642.8
bio-yeast	353	353	353	353.1	soc-karate	4	4	4	4
ca-AstroPh	2070	2073	2073	2076	soc-lastfm	67226	67226	67226	67226
ca-CSphd	523	523	523	523	soc-livejournal	794323.6	794389.9	794389.9	794501.4
ca-CondMat	2996.1	3003.3	3004	3005.2	soc-orkut	113901	113901.5	113911.3	114318.5
ca-Erdos992	1446	1446	1446	1446	soc-pokec	209070.7	209201.8	209201.8	209549.3
ca-GrQc	776	777.9	777.9	778.1	soc-slashdot	14157	14157.3	14157.3	14157.3
ca-HepPh	1665	1667.8	1667.8	1668.2	soc-twitter-follows	2269	2269	2269	2269
ca-MathSciNet	65577.3	65585.1	65585.1	65588.2	soc-wiki-Vote	209.2	209.6	209.6	209.8
ca-citeseer	33214.8	33227.5	33227.5	33240.9	soc-youtube	89733.5	89733.7	89734.1	89734.4
ca-coauthors-dblp	36010	36425	36425	36637.9	soc-youtube-snap	213122.1	213122.1	213122.1	213122.2
ca-dblp-2010	35367.4	35379.6	35379.6	35391.3	tech-RL-caida	40224.8	40234.3	40234.3	40246.5
ca-dblp-2012	46153.2	46167.3	46167.3	46174.8	tech-WHOIS	611	611	611.5	612
ca-hollywood-2009	49493.8	49921	49921	50188.6	tech-as-caida2007	2400	2400	2400	2400
ca-netscience	55	55	55	55	tech-as-skitter	182386.6	182472.6	182483.3	182538.8
socfb-A-anon	201698.6	201699.1	201699.1	201700.7	tech-internet-as	3679	3679	3679	3679
socfb-B-anon	187032.8	187034.3	187034.3	187034.9	tech-p2p-gnutella	12571	12571	12571	12571
socfb-Berkeley13	1642	1644.1	1644.1	1658.1	tech-routers-rf	479	479	479	479
socfb-CMU	444.1	449	449	455	scc-enron-only	6	6	6	6
socfb-Duke14	598	602	602	605	scc-fb-forum	436	436	436	436
socfb-Indiana	1729	1731.1	1731.1	1742.8	scc-fb-messages	634	635	635	635
socfb-MIT	520.2	522	522	524.1	scc-infect-dublin	826.5	830	830	831.1
socfb-OR	10728.6	10744.2	10744.2	10757.5	scc-infect-hyper	1	1	1	1
socfb-Penn94	3038.8	3038.8	3039.1	3060.9	scc-reality	53	53	53	53
socfb-Stanford3	931	932	932	936.1	scc-retweet	17415	17415	17415	17415
socfb-Texas84	2163	2167	2167	2183.1	scc-retweet-crawl	1120710	1120710	1120710	1120710
socfb-UCLA	1651.1	1651.1	1651.1	1664.2	scc-rt-alwefaq	4102	4102	4102	4102
socfb-UCSB37	1109	1111	1111	1118.2	scc-rt-assad	2010	2010	2010	2010
socfb-UConn	1098	1106.8	1106.8	1115.1	scc-rt-bahrain	4614	4614	4614	4614
socfb-UF	2065.4	2068.1	2074	2088.1	scc-rt-barackobama	9486	9486	9486	9486
socfb-UIllinois	1788.2	1789.1	1789.1	1805	scc-rt-damascus	2939	2939	2939	2939
socfb-Wisconsin87	1511.2	1517.1	1517.1	1526.2	scc-rt-dash	5949	5949	5949	5949
socfb-uci-uni	865675	865676	865676	865676	scc-rt-gmanews	8207	8207	8207	8207
inf-power	1487.1	1488.5	1488.5	1488.6	scc-rt-gop	3709	3709	3709	3709
inf-road-usa	7974437	7987953	7987953	7999017	scc-rt-http	5687	5687	5687	5687
inf-roadNet-CA	609320.4	610960.9	610960.9	611801.4	scc-rt-israel	3675	3675	3675	3675
inf-roadNet-PA	338740.6	339897.8	339897.8	340075.5	scc-rt-justinbieber	9309	9309	9309	9309
ia-email-EU	755	755	755	755	scc-rt-ksa	5762	5762	5762	5762
ia-email-univ	211	211	211	212	scc-rt-lebanon	3365	3365	3365	3365
ia-enron-large	1979.3	1982	1982.3	1982.4	scc-rt-libya	5004	5004	5004	5004
ia-enron-only	21	21	21	21	scc-rt-lolgop	9483	9483	9483	9483
ia-fb-messages	249	249	249	249	scc-rt-mittromney	7784	7784	7784	7784
ia-infect-dublin	47.9	49.1	50	50	scc-rt-obama	3036	3036	3036	3036
ia-infect-hyper	3	3	3	3	scc-rt-occupy	3054	3054	3054	3054
ia-reality	81	81	81	81	scc-rt-occupywallstnyc	3477	3477	3477	3477
ia-wiki-Talk	11935	11935	11935	11935	scc-rt-oman	4441	4441	4441	4441
rec-amazon	28365.7	28393.4	28393.4	28407	scc-rt-onedirection	7672	7672	7672	7672
rt-retweet	32	32	32	32	scc-rt-p2	4771	4771	4771	4771
rt-retweet-crawl	75740	75740	75740	75740	scc-rt-qatif	6708	6708	6708	6708
rt-twitter-copen	199	199	199	199	scc-rt-saudi	6783	6783	6783	6783
sc-ldoor	65387.7	65556.6	65587.9	65610.9	scc-rt-tcot	4491	4491	4491	4491
sc-msdoor	21073.1	21122	21133.1	21140.3	scc-rt-tlot	3506	3506	3506	3506
sc-nasasrb	1306.7	1310.5	1316.8	1317.9	scc-rt-uae	4746	4746	4746	4746
sc-pkustk11	2573.5	2593.3	2601	2601.4	scc-rt-voteonedirection	1829	1829	1829	1829
sc-pkustk13	1399.1	1423	1428	1439.9	scc-twitter-copen	6410	6410	6410	6410
sc-pwtk	5455.1	5455.1	5461.7	5466.6	web-BerkStan	3000	3000	3000	3001
sc-shipsec1	9091.4	9126.6	9143.1	9143.8	web-arabic-2005	16946.9	16957	16957	16967.8
sc-shipsec5	12069.8	12262.7	12262.7	12273.1	web-edu	249	249	249	249
soc-BlogCatalog	4894	4894	4894	4894	web-google	205	205	205	205.1
soc-FourSquare	60984.9	60987.5	60987.5	60987.5	web-indochina-2004	1489	1489	1489	1489.1
soc-LiveMocha	1430.1	1430.1	1430.1	1434.8	web-it-2004	32997	32997	32997	32997.2
soc-brightkite	12940	12941	12941	12941	web-polblogs	104	104	104	104
soc-buzznet	128.2	129	129	129	web-sk-2005	26472.9	26476.1	26480.7	26482.3
soc-delicious	55725.2	55726.4	55726.4	55727.9	web-spam	831	832	832	832.1
soc-digg	66155	66155.1	66155.1	66156.5	web-uk-2005	1421	1421	1421	1421
soc-dolphins	14	14	14	14	web-webbase-2001	1005	1005.3	1005.8	1005.8
soc-douban	8364	8364	8364	8364	web-wikipedia2009	347018.1	347052.7	347068	347097.2
soc-epinions	6435	6435	6435	6435					