Restart and Random Walk in Local Search for Maximum Vertex Weight Cliques with Evaluations in Clustering Aggregation

Yi Fan\textsuperscript{1,2*}, Nan Li\textsuperscript{3}, Chengqian Li\textsuperscript{1}, Zongjie Ma\textsuperscript{2}, Longin Jan Latecki\textsuperscript{3}, Kaile Su\textsuperscript{1,2}

\textsuperscript{1}Department of Computer Science, Jinan University, Guangzhou, China
\textsuperscript{2}Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
\textsuperscript{3}Department of Computer and Information Sciences, Temple University, Philadelphia, USA
\textsuperscript{4}Department of Computer Science, Sun Yat-sen University, Guangzhou, China

\{yi.fan4, zongjie.ma\}@griffithuni.edu.au; \{nan.li, latecki\}@temple.edu; k.su@griffith.edu.au

Abstract

The Maximum Vertex Weight Clique (MVWC) problem is NP-hard and also important in real-world applications. In this paper we propose to use the restart and the random walk strategies to improve local search for MVWC. 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. Experimental results show that our solver outperforms state-of-the-art solvers in DIMACS and finds a new best-known solution. Moreover it is the unique solver which is comparable with state-of-the-art methods on both BHOSLIB and large crafted graphs. Also it achieves 100% success rates over all the winner determination graphs within 500s. Furthermore we evaluated our solver in clustering aggregation. Experimental results on a number of real data sets demonstrate that our solver outperforms the state-of-the-art for solving the derived MVWC problem and helps improve the final clustering results.

1 Introduction

We deal with a simple undirected graph $G = (V, E, w)$ where $V$ is the vertex set, an edge $e \in E$ is a 2-element subset of $V$, and $w : V \rightarrow R_{\geq 0}$ is a weighting function on $V$. A clique $C$ is a subset of $V$ such that each pair of vertices in $C$ is mutually adjacent. The Maximum Vertex Weight Clique (MVWC) problem is to find a clique with the greatest total vertex weight. This problem exists in many real-world applications like [Brendel and Todorovic, 2010; Brendel et al., 2011]. In this paper we are concerned in finding a clique with total vertex weight as great as possible.

However, the MVWC problem is difficult to approximate [Feige, 2005]. Up to now, there are two types of algorithms: complete algorithms [Östergård, 2001; Östergård, 2002; Kumlander, 2004; Yamaguchi and Masuda, 2008; Shimizu et al., 2012; Jiang et al., 2017] and incomplete ones [Pullan, 2008; Wu et al., 2012]. Furthermore, algorithms for the maximum weight independent set and minimum weight vertex cover problems can also work e.g., [Xu et al., 2016].

1.1 Local Search for MVWC

Local search often suffers from the cycling problem, i.e., the search may spend too much time visiting a small area, thus various tabu strategies have been proposed to deal with this problem. Such examples can be found in two recent MVWC solvers MN/TS [Wu et al., 2012] and LSCC [Wang et al., 2016]. The former adopted a randomized tabu strategy to escape from local optima, while the latter utilized the strong configuration checking strategy to do so. Nevertheless, such tabu strategies still fail to prevent the local search from moving in cycles, thus both MN/TS and LSCC need to restart periodically. More specifically they both adopted a parameter $L$ to control how often the local search restarts. However, this parameter needs to be trained before the solver is being run. Furthermore it is unreasonable to use a fixed value for $L$ during the search, since some parts of the search space can be more restrictive while some other parts can be less restrictive. Also it is difficult to understand why a particular value for $L$ is suitable. Hence, we will propose a novel heuristic to restart at a good time point. It might be the first time to address this fundamental problem in local search for MVWC.

LSCC consists in two phases: (1) randomly generating a maximal clique $C$ and then (2) improving $C$ in a deterministic way. In each local move, LSCC selects the neighboring clique with the greatest weight according to the strong configuration checking criterion. However, using a deterministic heuristic to select a neighboring solution may be problematic, since a sequence of steps would be simply repeated without improving the best $C$ found so far. Hence, we will adopt random walk to guide the search to move in different directions.

1.2 Our Contributions

We develop a solver named RRWL\textsuperscript{1} (Restart and Random Walk in Local Search) based on the restart and the random walk strategies. Roughly if a solution is revisited in some particular situation, the search will restart. Also, when the search has no other options except dropping vertices, it will use random walk.

\footnere{1}https://github.com/Fan-Yi/Restart-and-Random-Walk-in-Local-Search-for-MVWC
For showing the effectiveness, we compared RRWL with LSCC\(^2\), FastWC\(l\)q [Cai and Lin, 2016] and WLMC [Jiang et al., 2017] on DIMACS [Johnson and Trick, 1996], BHOSLIB [Xu et al., 2005] and large sparse graphs\(^3\), as well as some graphs derived from the Winner Determination Problem (WDP) [Wu and Hao, 2015; Fang et al., 2016]. Experimental results show that

1. on DIMACS, RRWL outperforms LSCC and FastWC\(l\)q in terms of the average solution quality as well as the best solution quality; moreover, it finds a new best-known solution on Mann\_a81\_clq.
2. on BHOSLIB, RRWL generally outperforms LSCC and greatly outperforms FastWC\(l\)q.
3. on large sparse graphs, RRWL is comparable with FastWC\(l\)q and greatly outperforms LSCC.
4. on the WDP graphs, the solution returned by RRWL in any run was proved to be optimal by WLMC; moreover, it locates the optimal solutions in a shorter time in many cases.

Since these graphs vary greatly in the sizes as well as the densities, our solver is much more robust than previous ones.

We also evaluate our solver on a number of real data sets in the application of clustering aggregation. Clustering is the task to partition a set of data objects into groups 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. Clustering aggregation, also known as ensemble clustering or consensus clustering, aims to combine multiple base clusterings into a probably better one. Formally, given a set of \( n \) data objects \( \mathcal{O} = \{ o_1, o_2, ..., o_n \} \), a base clustering \( C_i \) on \( \mathcal{O} \) is obtained by applying an exclusive clustering algorithm with a specific set of input parameters. \( C_i \) consists of \( k \) disjoint clusters \( c_{i1}, c_{i2}, ..., c_{ik} \). These clusters form a partitioning of \( \mathcal{O} \), i.e., \( \bigcup_{j=1}^{k} c_{ij} = \mathcal{O} \) and \( c_{ip} \cap c_{iq} = \emptyset \) for all \( p \neq q \). Hence a clustering is a partitioning of the set \( \mathcal{O} \), and a cluster is a subset of \( \mathcal{O} \). Multiple base clusterings can be generated by different clustering algorithms with different parameters. Their qualities are generally very different. The goal of clustering aggregation is to pursue a better final clustering based on the information revealed from these base clusterings.

We follow the idea of [Li and Latecki, 2012] to formulate clustering aggregation as an MVWC problem, and evaluate our solver in this scenario. Experimental results on a number of real data sets demonstrate that our solver outperforms the other state-of-the-art for solving the derived MVWC problem and helps improve the final clustering results.

### 2 Preliminaries

Given an edge \( e = \{ u, v \} \), we say that \( u \) and \( v \) are neighbors, and \( u \) and \( v \) are adjacent to each other. Also we use \( \mathcal{N}(v) = \{ u | u \) and \( v \) are neighbors \} \) to denote the set of \( v \)‘s neighbors. A maximal clique is a clique which is not a subset of any other clique. Given a weighting function \( w : V \to R_{\geq 0} \), the weight of a clique \( C \), denoted by \( w(C) \), is defined to be \( \sum_{v \in C} w(v) \).

We use \( age(v) \) to denote the number of steps since last time \( v \) changed its state (inside or outside the candidate clique).

#### 2.1 The Benchmark

The DIMACS, BHOSLIB and large graphs were originally unweighted. To obtain the corresponding MVWC instances, researchers use the method in [Pullan, 2008]. For the \( i \)-th vertex \( v_i \), \( w(v_i) = (i \mod 200) + 1 \). Also we compare state-of-the-art MVWC solvers in winner determination and clustering aggregation.

#### 2.2 Multi-neighborhood Search

Usually for finding a good weighted clique, the local search moves from one clique to another until the cutoff arrives, then it returns the best clique that has been found. There are three operators: \( \text{add} \), \( \text{swap} \) and \( \text{drop} \), which guide the local search to move in the clique space. In [Fan et al., 2016], two sets \( S_{\text{add}} \) and \( S_{\text{swap}} \) were defined as below which ensures that the clique property is preserved.

\[
S_{\text{add}} = \begin{cases} 
\{ v | v \notin C, v \in N(u) \text{ for all } u \in C \} & \text{if } |C| > 0; \\
\emptyset & \text{if } |C| = 0.
\end{cases}
\]

\[
S_{\text{swap}} = \begin{cases} 
\{ (u, v) | u \in C, v \notin C, \{u, v\} \notin E, \text{ and } v \in N(w) \text{ for all } w \in C \backslash \{u\} \} & \text{if } |C| > 1; \\
\emptyset & \text{if } |C| \leq 1.
\end{cases}
\]

We use \( \Delta_{\text{add}} \), \( \Delta_{\text{swap}} \) and \( \Delta_{\text{drop}} \) to denote the increase of \( w(C) \) for the operation \( \text{add} \), \( \text{swap} \) and \( \text{drop} \) respectively. Obviously, we have (1) for a vertex \( v \in S_{\text{add}} \), \( \Delta_{\text{add}}(v) = w(v) \); (2) for a vertex \( u \in C \), \( \Delta_{\text{drop}}(u) = -w(u) \); (3) for a vertex pair \( (u, v) \in S_{\text{swap}} \), \( \Delta_{\text{swap}}(u, v) = w(v) - w(u) \).

#### 2.3 The Strong Configuration Checking Strategy

Recently, [Cai et al., 2011] proposed the configuration checking (CC) strategy to reduce cycling. The CC strategy works as follows. If a vertex is removed out of the candidate set, it is forbidden to be added back into the candidate set until its configuration has been changed. Typically, the configuration of a vertex refers to the state of its neighboring vertices.

The CC strategy is usually implemented with a Boolean array named \( \text{confChange} \), where \( \text{confChange}(v) = 1 \) means that \( v \)’s configuration has changed since last time it was removed, and \( \text{confChange}(v) = 0 \) otherwise.

Later [Wang et al., 2016] modified CC into a more restrictive version, which is called strong configuration checking (SCC), to deal with the MVWC problem. The main idea of the SCC strategy is as follows: after a vertex \( v \) is dropped from or swapped from \( C \), it can be added or swapped back into \( C \) only if one of its neighbors is added into \( C \).

In details, the SCC strategy works as follows. (1) Initially \( \text{confChange}(v) \) is set to 1 for each vertex \( v \); (2) When \( v \) is added, \( \text{confChange}(n) \) is set to 1 for all \( n \in N(v) \); (3) When \( v \) is dropped, \( \text{confChange}(v) \) is set to 0; (4) When \( (u, v) \in S_{\text{swap}} \) are swapped, \( \text{confChange}(u) \) is set to 0.

Lastly, whether a vertex is allowed to be added or swapped into the candidate clique is also referred to as its tabu status.
3 A Revisiting based Restart Strategy

We first discuss the drawback of local search for MVWC, and propose a novel condition in which the search will restart.

3.1 Drawback in Local Search for MVWC

Usually in local search for propositional satisfiability, vertex cover, etc., the search will periodically pick a random unsatisfied constraint, and then selects a local move to satisfy this constraint. Thus the search is able to traverse the search space by such steps. In contrast, in the case of MVWC, the hard constraints are always satisfied, and each local move tries to maximize the clique weight. Hence, for MVWC the number of neighboring solutions is relatively small, i.e., there are relatively few possible local moves for the next step. Considering the definitions of $S_{add}$ and $S_{swap}$, the search is not extensive. It may be restricted in a cycle even though we have intelligent tabu strategies, so restart is needed to improve extensive search in local search for MVWC.

Now the problem is how to determine whether the search is restricted in a cycle. Since local search method does not allow us to memorize the whole search history, we have to develop various heuristics to estimate this condition.

3.2 First Growing Step

Usually in local search there is an objective function value which needs to be maximized. This function value will be increased and decreased periodically during the search. In current state-of-the-art MVWC solvers, the objective function value refers to the clique weight.

In each period, after the first step which increases the clique weight, we detect revisiting (Line 27 in Algorithm 2).

Revisiting is not a sufficient condition for restart

One may think that the search should restart once a solution is revisited, yet this is not the case. The local search sometimes revisits a solution due to the drop operations, and such revisiting does not mean that the search is restricted in a cycle. Hence, when a solution is revisited, we should consider whether the clique is being extended or shrunk.

Even though a solution is revisited, we have little sense whether the local search will choose to repeat a previous step. If a candidate solution is revisited, with the tabu status of the vertices nearby the same as before, then the selected local move would simply repeat a previous step, i.e., the search is probably restricted in a cycle, so restart is needed. Otherwise, if a solution is revisited, with the tabu status of the vertices nearby different from that before, the search can get rid of the cycle easily, so restart is unreasonable.

Extending and shrinking periods

In our algorithm, we employ a predicate $lastStepImproved$ such that $lastStepImproved = true$ iff the clique weight was increased in the last step. Then we use this predicate to check whether the clique is being extended or shrunk. Obviously when a vertex is added, the search is in the extending period; when a vertex is dropped, the search is in the shrinking period. However, when a pair of vertices, namely $(u, v)$, are swapped, the clique may be extending or shrinking. More specifically, if $w(u) < w(v)$, the clique is extending; otherwise the clique is shrinking.

If a solution is revisited together with the same $lastStepImproved$ value as before, then the search is quite probably restricted in a cycle. In other words if we consider revisiting only when $lastStepImproved = true$ (or $lastStepImproved = false$), the restart strategy may become more reasonable.

Why only consider extending periods

Some intuitive analyses suggest that considering revisiting in the extending period is more reasonable.

In the shrinking period, vertices are probably being dropped. In both MN/TS and LSCC, when a vertex is selected to be dropped, no tabu status is considered. So even though a solution is revisited in this period, we are not certain whether the tabu status of the vertices nearby is close to that before, therefore we are unsure whether the search is restricted in a cycle.

In the extending period, vertices are being added or swapped. In both MN/TS and LSCC, when a vertex is selected to be added or swapped into the candidate clique, it has to satisfy the tabu requirements. In this sense if a solution is revisited in this period, it seems that the tabu requirements fail to prevent the search from being restricted, so restart is reasonable.

Why choose the first growing step

Suppose that a solution is visited in the first growing step and later it is revisited in the second growing step. Obviously these two situations have different recent histories, therefore the tabu status of the latter situation may be different from the that of the previous situation. So we are unsure whether the search is restricted in a cycle. However, if both situations occur in the first growing step, we are quite confident that the search is restricted in a cycle.

Readers may think that our restart strategy may be too cautious, i.e., the search may spend too much time in a non-promising area. Yet experiments still show that our solver restarts more frequently than previous solvers.

3.3 Implementations of Detecting Revisiting

We use a hash table to approximately detect revisiting. Since the collisions, i.e., different solutions may share the same hash entry, are rare in our settings, we do not resolve them.

Definition 1 Given a clique $C$ and a prime number $p$, we define the hash value of $C$, denoted by $hash(C)$, as $(\sum_{v \in C}2^i) \mod p$, which maps a clique $C$ to its hash entry $hash(C)$.

At the beginning, we calculate $(2^i \mod p)$ iteratively with different values of $i$, based on the proposition below.

Proposition 1 $2^i \mod p = 2^{(2^{i-1} \mod p)} \mod p$.

Our solver will save these values in an array for later references. Hence, in Theorem 2 below, the subformulas $(2^i \mod p)$ can be computed in constant complexity. So the hash value of the current clique can be updated in $O(1)$ complexity as well.

Theorem 2 Let $C$ be the current clique, then we have
1. \( \text{hash}(C \cup \{v_i\}) = [\text{hash}(C) + (2^i \mod p)] \mod p; \)
2. \( \text{hash}(C\setminus\{v_i\}) = [\text{hash}(C) + p - (2^i \mod p)] \mod p. \)

Throughout this paper, we set \( p = 10^9 + 7 \). If we use a larger prime number, the chance of collision will decrease. Thus, we set this parameter simply based on the memory capacity of our machines. With this prime number \( p \), our hash table consumes around 1 GB memory. While in our experiments, our solver performs less than \( 3 \times 10^8 \) steps in any run. Therefore, given the \( 10^9 + 7 \) hash entries, the number of collisions is negligible.

4 Random Walk for Dropping Vertices

In MN/TS and LSCC, when a vertex is selected to be added or swapped into the candidate clique, it has to satisfy the tabu requirements. Thus the tabu requirements guide the search to move in different paths. However, when a vertex is selected to be dropped, no tabu requirements exist and the one with the smallest weight is chosen. Thus if a local optimum is revisited, the candidate clique will probably be shrunk in the same way as before, and thus it may probably repeat previous steps.

In the case of adding and swapping, the clique weight may increase and lead to a new best-found clique, so best-picking is necessary. However, in the case of dropping, it is unclear why greedy search is reasonable. In local search for propositional satisfiability, vertex cover etc., it is desired to break as few hard constraints as we can. However, in local search for MVWC, the clique weight constraint is soft, so the minimum break heuristic may not be reasonable here.

In RRWL, when there are no options for the next step except dropping vertices, we randomly drop one vertex from the clique (See Line 22 in Algorithm 2). With random walk, even though a local optimum is revisited, different sub-cliques will be generated in different times, and later different add or swap operations will be applied. The search will switch to different cliques nearby and hence the local area will be searched more thoroughly.

5 The RRWL Algorithm

The top level algorithm is shown in Algorithm 1, where the localMove() procedure is shown in Algorithm 2.

```
Algorithm 1: RRWL
input : A graph \( G = (V,E,w) \) and the cutoff
output: The best clique that was found
1 \( C^* \leftarrow C \leftarrow \emptyset; \text{lastStepImproved} \leftarrow \text{true}; \)
2 step \leftarrow 1; confChange(v) \leftarrow 1 for all \( v \in V; \)
3 while elapsed time < cutoff do localMove(); \)
4 return \( C^*; \)
```

In Algorithm 2, the vertices of the operations are explicit from the context and thus omitted. All ties are broken in favor of the oldest one just like LSCC. Notice that when the conditions in Lines 9 or 18 hold, the clique weight was decreased in the last step and is increased in this step, so this is the first growing step.

```
Algorithm 2: localMove
1 if \( C = \emptyset \) then
2 add a random vertex into \( C; \)
3 while \( S_{\alpha} \neq \emptyset \) do add a random vertex from \( S_{\alpha}; \)
4 lastStepImproved \leftarrow \text{true};
5 firstGrowingStep \leftarrow \text{false};
6 \( v \leftarrow \text{a vertex in } S_{\alpha} \text{ such that } \text{confChange}(v) = 1 \)
7 \( \text{with the biggest } \Delta_{\alpha}; \text{otherwise } v \leftarrow \text{NL}; \)
8 \( (u, u') \leftarrow \text{a pair in the } S_{\alpha} \text{ such that } \text{confChange}(u') = 1 \)
9 \( \text{with the biggest } \Delta_{\alpha}; \text{otherwise } (u, u') \leftarrow (\text{NL, NL}); \)
10 if \( v \neq \text{NL} \) then
11 if lastStepImproved = \text{false} then
12 \text{firstGrowingStep} \leftarrow \text{true};
13 if \((u, u') = (\text{NL, NL}) \text{ or } \Delta_{\alpha} > \Delta_{\alpha} \text{ then } C \leftarrow C \cup \{v\}; \text{else } C \leftarrow C \setminus \{u\} \cup \{u'\}; \)
14 \text{lastStepImproved} \leftarrow \text{true};
15 else
16 if \((u, u') = (\text{NL, NL}) \text{ or } \Delta_{\alpha} < 0 \text{ then } C^* \leftarrow C; \)
17 \text{lastStepImproved} \leftarrow \text{false};
18 else
19 \text{if lastStepImproved = \text{false then}}
20 firstGrowingStep \leftarrow \text{true};
21 lastStepImproved \leftarrow \text{true};
22 \( x \leftarrow \text{a vertex in } C \text{ with the biggest } \Delta_{\alpha}; \)
23 \( \text{if } (u, u') = (\text{NL, NL}) \text{ then drop a random vertex} \)
24 \( \text{else} \)
25 \( \text{if } \Delta_{\alpha} > \Delta_{\alpha} \text{ then } C \leftarrow C \setminus \{x\}; \)
26 \( \text{else } C \leftarrow C \setminus \{u\} \cup \{u'\}; \)
27 apply SCC rules; step \leftarrow step + 1;
28 if firstGrowingStep = \text{true then}
29 mark \text{hash}(C); \)
30 if \text{hash}(C) \text{ remarked then drop all vertices; }
```

6 Evaluations on Hand-crafted Graphs

To evaluate individual impacts, we disable the random walk strategy in RRWL and develop another solver named RSL.

6.1 Experimental Setup

For LSCC, the search depth \( L \) was set to 4,000. When solving large crafted graphs, LSCC employs the BMS heuristic, and the parameter \( k \) was set to 100, as in [Wang et al., 2016]. For FastWCql, the parameters \( k_0 \) and \( k_{\text{max}} \) for the dynamic BMS heuristic were set to 4 and 64 respectively, as in [Cai and Lin, 2016]. WLMC was compiled by gcc 4.4.7 and all other solvers were compiled by g++ 4.7.3. The experiments on DIMACS, BHOSLIB and WDP graphs were conducted on a cluster equipped with Intel(R) Xeon(R) CPUs X5650 @2.67GHz with 16GB RAM, running Red Hat Santiago OS. The experiments on the large crafted graphs were conducted on a cluster equipped with Intel Xeon E5-2670 v3 2.3GHz with 32GB RAM, running CentOS6.
Each solver was executed on each instance with seeds from 1 to 100 on DIMACS and BHOSLIB, from 1 to 10 on the large crafted graphs and from 1 to 10 on WDP graphs, as in [Wang et al., 2016], [Cai and Lin, 2016] and [Jiang et al., 2017] respectively. Since WLMC is an exact solver, it was executed only once on each WDP instance. The cutoff was set to 500s for RSL and RRWL on the WDP graphs, while the cutoff was set to 1,000s for all solvers on all other instances. For each algorithm (except WLMC) on each instance in DIMACS, BHOSLIB and large crafted graphs, we report the maximum weight ("w_max") and averaged weight ("w_avg") of the cliques found by the algorithm. To make the comparisons clearer, we also report the difference ("Δw") between the maximum weight of the cliques found by RRWL and that found by LSCC or FastWClq. Similarly Δavg represents the difference between the averaged weights. A positive difference indicates that RRWL performed better, while a negative value indicates the opposite. For RRWL, RSL and WLMC on each WDP instance, since they returned the same quality solutions in all runs, we report the average time needed to locate the best-found solutions in each group.

6.2 DIMACS and BHOSLIB
These graphs are dense. Experimental results show that

1. on each instance in DIMACS, RRWL performs at least as well as LSCC,
2. on several instances in DIMACS, RRWL substantially outperforms both LSCC and FastWClq;
3. in BHOSLIB, RRWL performs generally better than LSCC and greatly outperforms FastWClq.

Further experiments show that LSCC performs worse than RSL and RSL performs worse than RRWL, so both the restart and the random walk strategies have their own contributions. For the sake of space, we only list all the graphs on which RRWL and LSCC found different w_max or w_avg in Table 1. Notice that on MANN-a81.clq, we found w_max = 111,341. So far as we know, this is a new best-known solution.

Now we analyze the performances. (1) When local search is done on dense graphs, S_add and S_swap are big, so there are many options for the next step. In this situation revisiting in the first growing step is not so frequent. (2) In these graphs, the vertex degrees are usually bigger than 400 and close to each other, and the vertex weights are in uniform distribution. Therefore, even though revisiting in the first growing step occurs, it is relatively easy to get rid of the cycles. (3) Since DIMACS and BHOSLIB graphs were originally derived from hard combinatorial models, they present complicated structures. Therefore better intensive search is needed and thus random walk improves the performances. (4) Since the vertex degrees are close, the bounds in FastWClq are loose, so graph reduction is ineffective.

6.3 Large Crafted Graphs
We used the benchmark in [Cai and Lin, 2016]. Some graphs have millions of vertices and dozens of millions of edges. Also they are very sparse. The average degree is usually around 10—50. There is a recent solver named LMY-GRS

<table>
<thead>
<tr>
<th>Graph</th>
<th>FastWClq</th>
<th>LSCC</th>
<th>RRWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>w_max</td>
<td>w_max(&lt;w_avg)</td>
<td>w_max(&lt;w_avg)</td>
<td>w_max(&lt;w_avg)</td>
</tr>
<tr>
<td>fhb35-17-2.clq</td>
<td>394 (392.65)</td>
<td>398 (397.13)</td>
<td>398 (397.21)</td>
</tr>
<tr>
<td>fhb35-17-4.clq</td>
<td>3567 (3541.59)</td>
<td>3683 (3676.28)</td>
<td>3683 (3683.00)</td>
</tr>
<tr>
<td>frb40-19-1.clq</td>
<td>3885 (3885.00)</td>
<td>4063 (4061.12)</td>
<td>4063 (4062.30)</td>
</tr>
<tr>
<td>frb40-19-2.clq</td>
<td>4007 (4007.00)</td>
<td>4112 (4108.36)</td>
<td>4112 (4109.95)</td>
</tr>
<tr>
<td>frb40-19-3.clq</td>
<td>3984 (3984.00)</td>
<td>4115 (4109.81)</td>
<td>4115 (4113.20)</td>
</tr>
<tr>
<td>frb40-19-4.clq</td>
<td>3963 (3963.00)</td>
<td>4136 (4133.92)</td>
<td>4136 (4134.68)</td>
</tr>
<tr>
<td>frb40-19-5.clq</td>
<td>3914 (3914.00)</td>
<td>4118 (4116.72)</td>
<td>4118 (4117.80)</td>
</tr>
<tr>
<td>frb40-19-6.clq</td>
<td>3941 (3941.00)</td>
<td>4124 (4120.85)</td>
<td>4124 (4123.70)</td>
</tr>
<tr>
<td>frb40-19-7.clq</td>
<td>3911 (3911.00)</td>
<td>4117 (4114.93)</td>
<td>4117 (4116.29)</td>
</tr>
<tr>
<td>frb40-19-8.clq</td>
<td>3984 (3984.00)</td>
<td>4115 (4109.81)</td>
<td>4115 (4113.20)</td>
</tr>
<tr>
<td>frb40-19-9.clq</td>
<td>3964 (3964.00)</td>
<td>4137 (4134.19)</td>
<td>4137 (4135.24)</td>
</tr>
<tr>
<td>frb40-19-10.clq</td>
<td>3935 (3935.00)</td>
<td>4117 (4114.93)</td>
<td>4117 (4116.29)</td>
</tr>
</tbody>
</table>

Table 1: Results on BHOSLIB and DIMACS

[Fan et al., 2016], and we adopt its data structures. Since it falls behind RRWL, so we omit it in this section.

Experimental results show that RRWL and FastWClq returned the same w_max and w_avg on all instances except 5 instances (See Table 2). Moreover, both of them outperform LSCC+BMS greatly. Further experiments show that RSL and RRWL returned the same values on all instances except ca-hollywood-2005. On this instance, RSL and FastWClq returned the same values. For the sake of space, in Table 2, we simply list some largest graphs in this benchmark.

Thanks to the power-distribution law, graph reduction is powerful. Therefore, FastWClq finds good solutions, and is able to confirm the optimality of the returned solution on many graphs. In contrast, the local search in LSCC+BMS is seriously restricted, because both S_add and S_swap are very small. Hence, LSCC+BMS falls behind FastWClq.

RRWL is a pure local search solver, so it must be able to traverse the search space quite comprehensively. Here, the restart strategy guides the search to different parts of the search space. Since the graph structures are relatively simple in this benchmark, so better intensive search is not needed and thus the contribution of random walk is not observed.

### 6.4 Winner Determination Problem Graphs

An important application of MVWC is to solve the winner determination problem (WDP), because WDP can naturally be formulated as MVWC.

We compared RRWL and RSL with WLMC on the WDP benchmark\(^5\) provided in [Lau and Goh, 2002], which has been used to test MVWC algorithms. The benchmark contains 499 instances with up to 1,500 items and 1,500 bids, and can be divided into 5 groups by the item number and the bid number. Each group is labeled as REL-m-n, where \(m\) is the number of items and \(n\) is the number of bids. The REL-1000-1500 group contains 99 instances, while all other groups contain 100 instances. Thus with 10 seeds, both RSL and RRWL performed a total of 4,990 runs in this benchmark.

When formulated as MVWC, the graphs contain up to 1,500 vertices with density from 0.06 to 0.33. Therefore the average density of WDP graphs is significantly smaller than that of the DIMACS and BHOSLIB graphs, and bigger than that of the large crafted graphs.

Experimental results show that the solutions returned by RSL and RRWL were all proved to be optimal by WLMC, i.e., both solvers found the optimal solutions in all runs. In Table 3, we present the averaged time (seconds) needed to locate the respective solutions for each solver in each group ("LocateTime"). Since WLMC only deals with integer weights, we do not include it in this section.

Further experiments show that sparse graphs lead to high restart frequencies. For example on frb59-26-5.clq whose density is 0.89, our solver restarts every 1,300 steps on average; on in143.wclq whose density is 0.31, it restarts every 150 steps; and on soc-livejournal whose density is 3.4 \(\times\) 10^{-7}, it restarts every 60 steps.

### 7 Evaluations in Clustering Aggregation

We follow the essential idea of [Li and Latecki, 2012] to formulate clustering aggregation as an MVWC problem. In the first experiment, we compare RRWL with state-of-the-art MVWC solvers on those graphs derived from clustering aggregation on real data sets. Experimental results show that our solver RRWL outperforms all other solvers\(^6\). In the second experiment, we embed our solver into the algorithm framework of [Li and Latecki, 2012] (CA+RRWL)\(^7\) to evaluate the overall performance, and compare it to other recent clustering aggregation methods. Experiments show that CA+RRWL is competitive.

### 7.1 Recent Clustering Aggregation Algorithms

The COMUSA algorithm proposed in [Mimaroğlu and 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. [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. [Huang et al., 2016a] proposed to sparsify the co-association matrix of “microcluster” with the k-nearest neighbors strategy and learn new similarities based on weighted evidence accumulation clustering (WEAC) and graph partitioning with multi-granularity link analysis (GP-MGLA).

---

\(^5\)All the vertex weights are rounded down to an integer just like [Jiang et al., 2017].

\(^6\)Since WLMC only deals with integer weights, we do not include it in this section.

\(^7\)https://github.com/Fan-Yi/Restart-and-Random-Walk-in-Local-Search-for-MVWC-in-Clustering-Aggregation
Table 4: Data Sets for Experimental Evaluation

<table>
<thead>
<tr>
<th>Data set</th>
<th>#Instance</th>
<th>#Attribute</th>
<th>#Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris</td>
<td>150</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Zoo</td>
<td>101</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>Semeion</td>
<td>1593</td>
<td>256</td>
<td>10</td>
</tr>
<tr>
<td>PD</td>
<td>10992</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>Vowel</td>
<td>990</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>ISOLET</td>
<td>7797</td>
<td>517</td>
<td>26</td>
</tr>
<tr>
<td>Letter</td>
<td>20000</td>
<td>16</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 5: Base Clusterings and Graph Information

| Data set   | k | #Clustering | #Cluster | |V| | |E| |davg |
|------------|---|-------------|----------|---------------|--------|--------|
| Iris       | 2:1:10 | 18 | 108 | 108 | 4355.5 | 80.7 |
| Zoo        | 3:1:11 | 18 | 126 | 126 | 6084 | 96.6 |
| Semeion    | 6:1:14 | 18 | 180 | 180 | 10643 | 118.3 |
| PD         | 6:1:14 | 18 | 180 | 180 | 11707 | 130.1 |
| Vowel      | 7:1:15 | 18 | 198 | 198 | 14869.1 | 150.2 |
| ISOLET     | 22:1:30 | 18 | 468 | 468 | 89085 | 380.4 |
| Letter     | 22:1:30 | 18 | 468 | 468 | 86418.3 | 369.3 |

Table 6: Average Performance in Terms of MVWC Weight

<table>
<thead>
<tr>
<th>Method</th>
<th>Iris</th>
<th>Zoo</th>
<th>Semeion</th>
<th>PD</th>
<th>Vowel</th>
<th>ISOLET</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>MWBC</td>
<td>226.1</td>
<td>74.9</td>
<td>192.7</td>
<td>626.7</td>
<td>319.5</td>
<td>1353.4</td>
<td>2014.8</td>
</tr>
<tr>
<td>SAMC</td>
<td>725.9</td>
<td>73.1</td>
<td>205.2</td>
<td>5518.3</td>
<td>347.9</td>
<td>1497.6</td>
<td>5322.7</td>
</tr>
<tr>
<td>FastWLq</td>
<td>332.9</td>
<td>73.1</td>
<td>205.8</td>
<td>5532.7</td>
<td>347.7</td>
<td>1422.4</td>
<td>1818.4</td>
</tr>
<tr>
<td>LSCC</td>
<td>332.9</td>
<td>73.1</td>
<td>205.8</td>
<td>5535.2</td>
<td>348.1</td>
<td>1498</td>
<td>5364.2</td>
</tr>
<tr>
<td>LSCC+BMS</td>
<td>332.9</td>
<td>73.1</td>
<td>205.8</td>
<td>5535.2</td>
<td>348.1</td>
<td>1497.2</td>
<td>5364.5</td>
</tr>
<tr>
<td>RRLW</td>
<td>332.9</td>
<td>73.1</td>
<td>205.8</td>
<td>5535.2</td>
<td>348.1</td>
<td>1501.1</td>
<td>5364.5</td>
</tr>
</tbody>
</table>

7.2 Experimental Setup

We now introduce the data sets and the MVWC formulation.

The Data Sets

The evaluations are 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 4. For instance, Iris has 150 data objects; each of them has 4 attributes; and the data objects are from 3 classes.

The MVWC Formulation

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 5. 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 represents a cluster. If two clusters c_i and c_j, which are from two different clusterings, contain some common data objects, we say that they are overlapping. For any two non-overlapping clusters, there is an edge connecting the vertices representing them. 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. The basic statistics of the derived graph of each data set are given in Table 5. 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 d_{avg}. Obviously, the derived graphs are all very dense.

The weight of each vertex is defined as the sum of the silhouette coefficients of the data objects belonging to the corresponding cluster. Specifically, in a certain clustering, 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)} 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. The weight w_i on vertex i, which represents the cluster c_i is defined as: w_i = \sum_{t \in c_i} S_t.

7.3 Comparisons of Different MVWC Solvers

In the first experiment, we compare RRWL with state-of-the-art MVWC solvers on the derived 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 FastWLq, LSCC, LSCC+BMS, simulated annealing based on maximal clique (SAMC) [Li and Latecki, 2012], and MWBC, which serves as the baseline, just returns the set of vertices belonging to the same base clustering and having the maximum sum of weights. For SAMC, we use the same parameters as in [Li and Latecki, 2012], i.e., q = 0.3, \beta = 0.999, and iteration number n = 100. The parameters of FastWLq, LSCC and LSCC+BMS are the same as before.

FastWLq, LSCC, LSCC+BMS and our solver are implemented in C++ and invoked from MATLAB on a PC with Intel(R) Core(TM) i7 processor up to 3.4 GHz and 16 GB RAM. We set the cut off to be 10 minutes and used one seed.

Results Table 6 shows that RRWL is the unique solver which finds best cliques in all data sets, which shows the robustness and superiority of our solver. Comparing Tables 5 and 6 we find that the bigger the graph is, the more difficult it is to find an MVWC. For example, most solvers work well on Iris and Zoo while very few solvers work well on ISOLET and Letter. This also shows that our algorithm is more scalable. We also evaluate RSL in this scenario, but find no difference from RRWL. In fact the clustering aggregation graphs are simple, so random walk for better intensive search is not needed.

7.4 Comparisons of Final Clustering Results

In the second experiment, we replace the SAMC solver with ours in the algorithm framework of [Li and Latecki,
Table 7: Average Performance in Terms of Time (milliseconds) Consumed to Find the Best Solution

<table>
<thead>
<tr>
<th>Method</th>
<th>Iris</th>
<th>Zoo</th>
<th>Semenon</th>
<th>PD</th>
<th>Vowel</th>
<th>ISOLET</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastWClq</td>
<td>142.204</td>
<td>220.224</td>
<td>137.905</td>
<td>151.859</td>
<td>155.125</td>
<td>331.169</td>
<td>286.414</td>
</tr>
<tr>
<td>LSQC</td>
<td>80.370</td>
<td>225.374</td>
<td>168.369</td>
<td>118.152</td>
<td>115.388</td>
<td>941.115</td>
<td></td>
</tr>
<tr>
<td>LSQC-BMS</td>
<td>79.1248</td>
<td>227.101</td>
<td>101.312</td>
<td>101.1126</td>
<td>223.74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRWL</td>
<td>352.390</td>
<td>238.225</td>
<td>225.250</td>
<td>124.931</td>
<td>390.87</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Average Performance of Clustering Aggregation in Terms of NMI

<table>
<thead>
<tr>
<th>Method</th>
<th>Iris</th>
<th>Zoo</th>
<th>Semenon</th>
<th>PD</th>
<th>Vowel</th>
<th>ISOLET</th>
<th>Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMUSA</td>
<td>0.346</td>
<td>0.577</td>
<td>0.395</td>
<td>0.509</td>
<td>0.409</td>
<td>0.334</td>
<td>0.360</td>
</tr>
<tr>
<td>WEAC-SL</td>
<td>0.688</td>
<td>0.687</td>
<td>0.419</td>
<td>0.496</td>
<td>0.404</td>
<td>0.375</td>
<td>0.274</td>
</tr>
<tr>
<td>WEAC-CL</td>
<td>0.700</td>
<td>0.688</td>
<td>0.434</td>
<td>0.516</td>
<td>0.412</td>
<td>0.588</td>
<td>0.280</td>
</tr>
<tr>
<td>WEAC-RWL</td>
<td>0.900</td>
<td>0.706</td>
<td>0.534</td>
<td>0.511</td>
<td>0.396</td>
<td>0.381</td>
<td></td>
</tr>
<tr>
<td>GP-MGLA</td>
<td>0.906</td>
<td>0.692</td>
<td>0.445</td>
<td>0.548</td>
<td>0.411</td>
<td>0.602</td>
<td>0.291</td>
</tr>
<tr>
<td>ECFG</td>
<td>0.533</td>
<td>0.698</td>
<td>0.487</td>
<td>0.575</td>
<td>0.409</td>
<td>0.652</td>
<td>0.282</td>
</tr>
<tr>
<td>PTA-SL</td>
<td>0.345</td>
<td>0.606</td>
<td>0.431</td>
<td>0.463</td>
<td>0.375</td>
<td>0.563</td>
<td>0.249</td>
</tr>
<tr>
<td>PTA-CL</td>
<td>0.331</td>
<td>0.644</td>
<td>0.475</td>
<td>0.536</td>
<td>0.402</td>
<td>0.640</td>
<td>0.301</td>
</tr>
<tr>
<td>PTA-RRWL</td>
<td>0.348</td>
<td>0.660</td>
<td>0.473</td>
<td>0.541</td>
<td>0.399</td>
<td>0.639</td>
<td>0.301</td>
</tr>
<tr>
<td>PTGP</td>
<td>0.754</td>
<td>0.689</td>
<td>0.469</td>
<td>0.554</td>
<td>0.403</td>
<td>0.616</td>
<td>0.274</td>
</tr>
<tr>
<td>CA-SAMC</td>
<td>0.700</td>
<td>0.712</td>
<td>0.552</td>
<td>0.677</td>
<td>0.427</td>
<td>0.698</td>
<td>0.359</td>
</tr>
<tr>
<td>CA-RRWL</td>
<td>0.700</td>
<td>0.716</td>
<td>0.555</td>
<td>0.685</td>
<td>0.429</td>
<td>0.702</td>
<td>0.366</td>
</tr>
</tbody>
</table>

2012] to evaluate its performance for clustering aggregation (CA+RRWL). The cut off time is set to be 5 minutes. The comparison clustering aggregation algorithms include COMUSA [Mimaroegl and Erdil, 2011], WEAC+SL [Huang et al., 2015], WEAC+CL [Huang et al., 2015], WEAC+AL [Huang et al., 2015], GP-MGLA [Huang et al., 2015], ECFG [Huang et al., 2016b], PTA-SL [Huang et al., 2016a], PTA-CL [Huang et al., 2016a], PTA-AL [Huang et al., 2016a], PTGP [Huang et al., 2016a] and CA+SAMC [Li and Latecki, 2012]. For these algorithms, we follow the author-recommended or default settings and parameters.

Note that COMUSA, ECFG, CA+SAMC and our CA+RRWL 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 [Huang et al., 2016b] and specify the cluster number for those “non-automatic” algorithms to be the one automatically estimated by CA+SAMC. For CA+SAMC and our CA+RRWL, 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 [Li and Latecki, 2012] 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 and Ghosh, 2002]. 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 with one seed and report the average NMI.

8 Conclusions and Future Work

In this paper, we developed a local search MVWC solver. Experimental results show that RRWL outperforms state-of-the-art solvers on DIMACS and reports a new best-known solution. It is the unique solver which achieves state-of-the-art performances in both standard and large crafted graphs, as well as the graphs derived from the winner determination problem. Also it helps improve the final clustering results on real data sets.

The main contributions include: (1) a restart strategy to improve extensive search; (2) a random walk strategy to improve intensive search; (3) a fast approximate hash table to detect revisiting.

As for future works we will study variants of the restart strategy in the context of MVWC. Also, we will apply these methods to solve other problems.

Acknowledgments

We thank all anonymous reviewers for their valuable comments. We thank Yanyan Xu for proofreading the submission of this paper. This work is supported by ARC Grant FT0991785, NSF Grant No. 61463044, NSFC Grant No. 61572234, Grant No. [2014]7421 from the Joint Fund of the NSF of Guizhou province of China, and the US NSF under Grant IIS-1302164.

We gratefully acknowledge the support of the Griffith University eResearch Services Team and the use of the High Performance Computing Cluster “Gowonda” to complete this research. This research was also supported by use of the NeCTAR Research Cloud and by QCI[http://www.qcif.edu.au]. The NeCTAR Research Cloud is a collaborative Australian research platform supported by the National Collaborative Research Infrastructure Strategy.

References


