Fast Solving Maximum Weight Clique Problem in Massive Graphs

Shaowei Cai[1*] and **Jinkun Lin**[2]
[1]State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China
[2]Key Laboratory of High Confidence Software Technologies, Peking University, Beijing, China
caisw@ios.ac.cn; jkunlin@gmail.com

Abstract

This paper explores techniques for fast solving the maximum weight clique problem (MWCP) in very large scale real-world graphs. Because of the size of such graphs and the intractability of MWCP, previously developed algorithms may not be applicable. Although recent heuristic algorithms make progress in solving MWCP in massive graphs, they still need considerable time to get a good solution. In this work, we propose a new method for MWCP which interleaves between clique construction and graph reduction. We also propose three novel ideas to make it efficient, and develop an algorithm called FastWClq. Experiments on massive graphs from various applications show that, FastWClq finds better solutions than state of the art algorithms while the run time is much less. Further, FastWClq proves the optimal solution for about half of the graphs in an averaged time less than one second.

1 Introduction

The proliferation of massive data sets brings with it a series of special computational challenges. Many data sets can be modeled as graphs, and the research of massive real-world graphs grew enormously in last decade. A *clique* of a graph is a subset of the vertices that are all pairwise adjacent. Clique is an important graph-theoretic concept, and is often used to represent dense clusters. The maximum clique problem (MCP) is a long-standing problem in graph theory, for which the task is to find a clique with the maximum number of vertices in the given graph. An important generalization of MCP is the maximum weight clique problem (MWCP), in which each vertex is associated with a positive integer, and the goal is to find a clique with the largest weight. MWCP has valuable applications in many fields [Ballard and Brown, 1982; Balasundaram and Butenko, 2006; Gomez Ravetti and Moscato, 2008].

The decision version of MCP (and thus MWCP) is one of Karp's prominent 21 NP-complete problems [Karp, 1972], and is complete for the class W[1], the parameterized analog of NP [Fellows and Downey, 1998]. Moreover, MCP (and thus MWCP) is not approximable within $n^{1-\epsilon}$ for any $\epsilon > 0$ unless NP=P [Zuckerman, 2007]. Nevertheless, these negative theoretical results have been established for "worst case", which does not often happen in practice. We still have hope of solving MWCP problems which arise in specific problem domains.

*Corresponding author

1.1 Related Work

Given their theoretical importance and practical relevance, considerable effort has been devoted to the development of various methods for MCP and MWCP, mainly including exact algorithms and heuristic algorithms. Exact algorithms can prove the optimality of their solutions, but they may fail to solve large graphs within reasonable time. On the other hand, various heuristic algorithms have been devised with the purpose of providing sub-optimal solutions within an acceptable time.

Almost all existing exact algorithms for MCP are branch-and-bound (BnB) algorithms, and they differ from each other mainly by their techniques to determine the upper bounds and their branching strategies. A large family of BnB algorithms use coloring to compute upper bounds [Tomita and Seki, 2003; Tomita and Kameda, 2007; Konc and Janezic, 2007; Tomita *et al.*, 2010; Segundo *et al.*, 2013]. Another paradigm encodes MCP into MaxSAT and then applies MaxSAT reasoning to improve the upper bound [Li and Quan, 2010; Li *et al.*, 2013]. There are also numerous works on heuristic algorithms for MCP, most of which are local search algorithms [Singh and Gupta, 2006b; Pullan and Hoos, 2006; Pullan, 2006; Guturu and Dantu, 2008; Benlic and Hao, 2013].

MWCP is more complicated than MCP and some powerful techniques for MCP are not applicable or ineffective for solving MWCP due to the vertex weights. This partly explains the fact that there are relatively fewer algorithms for MWCP. Some exact algorithms for MWCP come from and generalize previous BnB methods designed for MCP [Östergård, 1999; Kumlander, 2004]. The MaxSAT-based method is also generalized to MWCP, resulting in a state of the art exact MWCP algorithm named MaxWClq [Fang *et al.*, 2014]. More efforts are devoted to heuristic algorithms for MWCP. Massaro et al. propose a complementary pivoting algorithm based on the corresponding linear complementarity problem [Massaro *et al.*, 2002]. Busygin presents a heuristic method using a nonlinear programming formulation for MWCP [Busygin, 2006]. A hybrid evolutionary approach is offered in [Singh and Gupta, 2006a]. The Phased Local Search (PLS) algorithm is extended to MWCP [Pullan, 2008]. In [Wu *et al.*, 2012], a local search algorithm called MN/NT integrates a combined neighborhood and a dedicated tabu mechanism, and shows better performance than previous heuristic algorithms. A recent local search algorithm based on the configuration checking strategy [Cai *et al.*, 2011] called LSCC further improves MN/NT on a wide range of benchmarks [Wang *et al.*, 2016].

Traditional algorithms usually become futile on massive graphs, due to their high space complexity and time complexity. For example, most traditional algorithms utilize adjacency matrix to facilitate fast computation of some operations such as the query of whether two vertices are adjacent. But the space requirement of this data structure is prohibitive for massive graphs. Also, most commonly used strategies do not have sufficiently low time complexity, which severely limits their ability to handle massive graphs.

Recently, there have been some dedicated algorithms for solving MCP in massive graphs. These MCP algorithms [Rossi *et al.*, 2014; Verma *et al.*, 2015] heavily depend on the concept of k-Core [Seidman, 1983], which is defined as a subgraph where all vertices have degree at least k, and can be computed in $O(m)$ (m is the number of edges) using bin sorting [Batagelj and Zaversnik, 2003]. However, we are not aware of any work using the k-Core concept to develop MWCP algorithms. Moreover, an analogous concept in vertex weighted graphs requires prohibitive space ($O(\overline{w} \cdot m)$, where $\overline{w}$ is the average weight of vertices) for bin sorting, and does not allow fast computation. As for MWCP, a recent progress in solving massive graphs is made in local search algorithms, by using a probabilistic heuristic called Best from Multiple Selection (BMS) [Cai, 2015]. BMS was first applied to minimum vertex cover problem [Cai, 2015], and then to MWCP, resulting in two efficient local search algorithms for MWCP called MN/TS+BMS and LSCC+BMS [Wang *et al.*, 2016]. Seen from the literatures, LSCC+BMS is currently the best algorithm for solving MWCP in massive graphs.

1.2 Contributions and Paper Organization

Although recent works made progress in solving MWCP in massive graphs, the improvements are limited to local search and the performance is still not satisfactory. In many applications the time limit is very short, or the time resource is very valuable. This calls for more practical algorithms for solving MWCP in real-world massive graphs.

In this work, we propose an efficient method for solving MWCP in massive graphs, which interleaves between clique construction and graph reduction. In a graph reduction procedure, we reduce the size of the graph by removing some vertices that are impossible to be in any clique of the optimal weight. Most real-world massive graphs are power law graphs [Eubank *et al.*, 2004; Lu and Chung, 2006], and can be reduced considerably by using a clique of certain quality in hand as a lower bound. On the other hand, a smaller graph presents smaller search space and the algorithm may find better cliques more easily, which can then be used to further reduce the graph. As far as we know, this is the first algorithm that interleaves between construction and reduction, although some previous MCP algorithms reduce the graph before calling an exact algorithm [Rossi *et al.*, 2014; Verma *et al.*, 2015].

Moreover, we propose three ideas to make the method effective and efficient. The fist one is a function for estimating the benefit of adding a vertex, which considers both the weight of the vertex and the weight of its effective neighborhood w.r.t the current clique. We also propose a dynamic BMS heuristic, which is used in choosing the adding vertex. Lastly but very importantly, we propose a fast and effective graph reduction algorithm, which relies on two reduction rules, including a novel branching-based reduction rule.

Based on these ideas, we develop an algorithm called FastWClq. Experiments on a wide range of real-world massive graphs show that, FastWClq finds better solutions than state of the art algorithms (including LSCC+BMS and MaxWClq) for most of the graphs with less run time. More encouragingly, FastWClq finds at least same-quality, sometimes better-quality solutions than its competitors even when the time limit for the competitors are 10 times and 36 times that for FastWClq. Further, FastWClq finds and proves the optimal solution for about half of the graphs in one second on average.

In the next section, we introduce some necessary background knowledge. Then, we describe our method in Section 3, including the FastWClq algorithm and its important components. Experimental evaluations of our algorithm FastWClq are presented in Section 4. Finally, we give some concluding remarks and outline the future work.

2 Preliminaries

Let G=(V,E) be an undirected graph where V=$\{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E is the set of edges in G. We use $V(G)$ and $E(G)$ to denote the vertex set and the edge set of graph G. A vertex weighted undirected graph is an undirected graph $G = (V, E)$ combined with a weighting function w so that each vertex $v \in V$ is associated with a positive integer number $w(v)$ as its weight. We use a triple to denote a vertex weighted graph, i.e., $G = (V, E, w)$. For a subset $S \subseteq V$, we let $G[S]$ denote the subgraph induced by S, which is formed from S and all of the edges connecting pairs of vertices in S. The weight of S is $w(S)$=$\sum_{v \in S} w(v)$. The neighborhood of a vertex v is $N(v)$=$\{u \in V | \{u, v\} \in E\}$, and we denote $N[v] = N(v) \cup \{v\}$. The degree of v is $d(v) = |N(v)|$.

A graph G=(V,E) is complete if its vertices are pairwise adjacent, i.e. $\forall u, v \in V, \{u, v\} \in E$. A clique C is a subset of V such that the induced graph $G[C]$ is complete. The maximum clique problem (MCP) is to find a clique of maximum cardinality in a graph, and the maximum weight clique problem (MWCP) is to find a clique of the maximum weight in a vertex weighted graph. A clique C is called a maximal clique in G if there exists no clique C' in G such that $C' \supset C$.

3 A Novel Method for MWCP

In this section, we propose an algorithm for solving MWCP called FastWClq, which interleaves between clique construction and graph reduction. We first describe the algorithm, and then introduce the important components of the algorithm.

The pseudo code of FastWClq is shown in Algorithm 1. On a top level, the algorithm works as follows. After some initializations, the algorithm executes a main loop until a limited time is reached, or an exact solution is found and proved. In each iteration of the loop, a clique is constructed by extending from an empty set (lines 3-15). To avoid ineffective construction procedures, we use pruning techniques to stop construction procedures that are known not to form a better clique than the best found clique. After the construction, if a better clique is obtained, the best found clique C^* is updated, and then the graph is reduced (if possible) by iteratively applying reduction rules. Additionally, if the graph becomes empty after reduction, then the best found solution C^* is proved to be optimal (as will be discussed in Section 3.2).

Now we describe the clique construction procedure. Let us first introduce some notation and definitions. We use C to denote the current clique under construction, and $StartSet$ is the set containing vertices candidate as a starting vertex to construct a clique. $CandSet = \{v | v \in N(u)\ for\ \forall u \in C\}$, i.e., each vertex in $CandSet$ is adjacent to all vertices in C; this set consists of candidate vertices eligible to extend the current clique. The $effective\ neighborhood$ of vertex v is defined as $N(v) \cap CandSet$. The concept is very important, as $w(N(v) \cap CandSet)$ is used in both pruning a procedure and evaluating the quality of candidate vertices.

In a clique construction procedure, the algorithm first pops a random vertex from $StartSet$ to serve as the starting vertex from which a clique will be extended, if $StartSet$ is not empty (line 6). If $StartSet$ becomes empty, which means all vertices have been used as the starting vertex, then another round of clique constructions begins by resetting $StartSet$ to $G(V)$, and we adjust our strategy parameter (lines 3-5). After the starting vertex u is chosen, the clique is initialized with the vertex, and $CandSet$ is initialized as $N(u)$ (lines 7-8). Then, the clique is extended iteratively by each time adding a vertex $v \in CandSet$ (lines 9-15), until $CandSet$ becomes empty. Also, we use a cost-effective upper bound to prune the procedure (lines 11-12). Obviously, $w(C) + w(v) + w(N(v) \cap CandSet)$ is an upper bound on weight of any clique extended from C by adding v and more vertices.

Although tighter upper bounds can be obtained by using more advanced techniques such as coloring, these bounds require much more time for computation. Our aim in this paper is to compute a high-quality solution in short time, so we do not use these techniques.

3.1 Choosing the Adding Vertex

An important component in FastWClq is the function $ChooseAddVertex$ (Algorithm 2), which selects a vertex from $CandSet$ to extend the current clique. To this end, we propose a novel function to estimate the benefit of vertices, and a dynamic BMS heuristic to choose the adding vertex.

Benefit Estimation Function

```
Algorithm 1: FastWClq (G, cutoff)
Input: vertex weighted graph $G = (V, E, w)$, the cutoff time
Output: A clique of $G$
1  $StartSet = V(G), C^* := \emptyset, k := k_0$;
2  while elapsed time < cutoff do
3      if $StartSet = \emptyset$ then
4          $StartSet = V(G)$;
5          $AdjustBMSnumber(k)$;
6      $u$ := pop a random vertex from $StartSet$;
7      $C := \{u\}$;
8      $CandSet := N(u)$;
9      while $CandSet \neq \emptyset$ do
10         $v := ChooseAddVertex(CandSet, k)$;
11         if $w(C) + w(v) + w(N(v) \cap CandSet) \leq w(C^*)$
           then $Break$;
12         $C := C \cup \{v\}$;
13         $CandSet := CandSet \setminus \{v\}$;
14         $CandSet := CandSet \cap N(v)$;
15     if $w(C) > w(C^*)$ then
16         $C^* := C$;
17         $G := ReduceGraph(G, C^*)$;
18         $StartSet = V(G)$;
19         if $G$ becomes empty then
20             return $C^*$;    //exact solution
21 return $C^*$;
```

We define the benefit of adding a vertex v as $benefit(v) = w(C_f) - w(C)$, where C_f is the final maximal clique grown from $C \cup \{v\}$. Note that we do not define $benefit(v)$ as $w(C_f) - w(C \cup \{v\})$, because at this moment we do not know whether or not v will be selected to extend C.

An ideal strategy is to pick the vertex with the best benefit at each iteration to extend C. However, we cannot know the true benefit value of a vertex until we finish the construction procedure. In order to compare candidate vertices at the current iteration, we propose a function to estimate the benefit of adding a vertex. The function is based on two considerations:

1) If a candidate vertex v is added into the clique C, the weight of C is increased by $w(v)$, which is a trivial lower bound of $benefit(v)$.

2) Suppose a candidate vertex v is selected to be added into the clique C. The best possible weighted clique grown from $C \cup \{v\}$ is $C \cup \{v\} \cup (N(v) \cap CandSet)$, for which the weight is $w(C) + w(v) + w(N(v) \cap CandSet)$. Thus, an upper bound of $benefit(v)$ is $w(v) + w(N(v) \cap CandSet)$.

We consider an estimation function should take into account both the lower bound and upper bound of $benefit(v)$. A simple and intuitive function which embodies this principle is to take the average over these two bounds.

$$\hat{b}(v) = \frac{w(v) + w(v) + w(N(v) \cap CandSet)}{2} = w(v) + w(N(v) \cap CandSet)/2$$

Dynamic BMS Heuristic

We choose the adding vertex based on their $\hat{b}$ values according to a dynamic BMS (Best from Multiple Selection) heuristic. The original BMS heuristic is a probabilistic strategy which returns the best element from multiple samples. It has been theoretically shown that BMS can approximate the best-picking strategy very well in $O(1)$ time [Cai, 2015]. Another advantage of the BMS heuristic is that, we can control the greediness of the algorithm by adjusting the parameter k. However, this has not been exploited previously, and previous algorithms with BMS adopt a static BMS heuristic, that is, the number of samplings k stays the same.

Algorithm 2: $ChooseAddVertex(CandSet, k)$

```
1 if $|CandSet| < k$ then
2     return a vertex $v \in CandSet$ with the greatest $\hat{b}$ value;
3 $v^* :=$ a random vertex in $CandSet$;
4 for $iteration := 1$ to $k - 1$ do
5     $v :=$ a random vertex in $CandSet$;
6     if $\hat{b}(v) > \hat{b}(v^*)$ then $v^* := v$;
7 return $v^*$;
```

In general, a greater k value indicates a greater greediness and more computation time. Based on this observation, we propose a dynamic BMS heuristic. In our algorithm, we start from a small k value (k_0), so that the algorithm works fast. Whenever $StartSet$ becomes empty, which means we do not find a better clique with this k value, we adjust k by increasing it as $k := 2k$, to make the algorithm construct cliques in a greedier way. Also, when k exceeds a predefined maximum value k_{max}, it is reset to $k := ++k_0$. This is implemented in the function $AdjustBMSnumber$ (line 5 in Algorithm 1).

3.2 Graph Reduction

By applying sound reduction rules (which usually depend on a clique in hand), a graph can be reduced to a smaller graph while keeping the optimal solution. This is desirable as algorithms can solve the original instance by solving a smaller and easier instance. In this subsection, we introduce a graph reduction algorithm, which relies on two reduction rules, including a novel branching-based reduction rule.

Definition 1 *Given a vertex weighted graph $G = (V, E, w)$, for a vertex $v \in V(G)$, an upper bound on the weight of any clique containing v is an integer, denoted as $UB(v)$, such that $UB(v) \geq max\{w(C)|C \text{ is a clique of } G, v \in C\}$.*

Now, we consider the following reduction rule.

Rule: *Given a vertex weighted graph $G = (V, E, w)$ and a clique C_0 in G, $\forall v \in V(G)$, if there is an upper bound $UB(v)$ such that $UB(v) \leq w(C_0)$, then delete v and its incident edges from G.*

The above rule indeed represents a family of reduction rules, and in order to obtain an applicable concrete rule, we need to specify the upper bound function and the input clique. We use the notation $Rule(UB, C_0)$ to denote a concrete rule where UB is the upper bound function and C_0 is the input clique.

Proposition 1 *Let G be a vertex weighted graph, G' the resulting graph by applying $Rule(UB, C_0)$ on G, and let w^* be the weight of the maximum weight clique of graph G, and $C^*_{G'}$ the maximum weight clique of G'. Then, $w^* = max\{w(C_0), w(C^*_{G'})\}$.*

Proof: If $w(C_0) = w^*$, then the proposition obviously holds. Now we consider the case $w(C_0) < w^*$. For graph G and a vertex $v \in V(G)$, let $C^*(v)$ be the clique s.t. $v \in C^*(v)$ and $w(C^*(v)) \geq w(C)$ for any clique C containing v. By Definition 1, we have $w(C^*(v)) \leq UB(v)$. On the other hand, any vertex deleted by $Rule(UB, C_0)$ satisfies $UB(v) \leq w(C_0)$, and thus $w(C^*(v)) \leq UB(v) \leq w(C_0) < w^*$, meaning that v cannot be contained in any clique with weight w^*. Thus, any vertex that is in a clique with weight w^* remains in G', so $w^* = w(C^*_{G'})$. $\square$

The above proposition shows that any rule in family Rule is sound w.r.t. keeping the optimal solution of the instance. Additionally, the proposition leads to the following corollary.

Corollary 1 *Let G' be the resulting graph by applying $Rule(UB, C_0)$ on vertex weighted graph G, if $V(G') = \emptyset$, then C_0 is the maximum weight clique of G.*

Given a clique in hand (by construction as shown in Algorithm 1), in order to apply reduction rules, the focus is how to compute an upper bound. Since any clique grown from vertex v can only contain vertices in $N(v)$, a trivial upper bound function is

$$UB_0(v) = w(N[v])$$

To get a tighter bound for vertex v, we consider its neighboring vertex with the maximum weight (denoted as n^*). The idea is that, for any clique C containing v, it either contains n^* or it does not. For either case, we can have a tighter upper bound than $UB_0(v)$, and finally we get the larger (worse) one as the upper bound. We divide the cases on n^* in order to balance the bounds of the two cases. Formally, we propose a branching-based upper bound as follows:

$$\begin{aligned} &UB_1(v) \\ =&max\{w(N[v]) - w(n^*), w(v) + w(n^*) + w(N(v) \cap N(n^*))\} \end{aligned}$$

Note that we use adjacency list instead of adjacency matrix for the purpose of saving space. So, checking whether a vertex $y \in N(v)$ is in $N(n^*)$, i.e., whether y and n^* are neighbors, requires $O(min\{d(y), d(n^*)\})$ time, which indicates a square time complexity for computing $N(v) \cap N(n^*)$ by each time checking whether a vertex in $N(v)$ is in $N(n^*)$. In this work, rather than use the above implementation, we use a linear implementation to compute $N(v) \cap N(n^*)$ (two scans on the smaller set and one on the larger one), by using indicators.

The graph reduction algorithm is depicted in Algorithm 3. Both upper bounds are used. UB_0 requires little overhead, while UB_1 requires more computation time but is tighter. Therefore, when considering a vertex, we first use the UB_0 based reduction rule, and if this cannot delete the vertex then we apply the rule based on UB_1[1].

Our reduction algorithm works in an iterative fashion, with a queue called $RmQueue$ which contains vertices to be deleted. In the beginning, the algorithm enqueues all vertices satisfying at least one of the reduction rules into $RmQueue$. Then, a loop is carried out until $RmQueue$ becomes empty. Each iteration of the loop pops a vertex u from $RmQueue$, and deletes u and all its incident edges from G. After a vertex

[1]In practice, a trick to accelerate the procedure (slightly) for large-sized graphs is to first use UB_0 to reduce the graph to a certain size, after which UB_1 will be used.

u is deleted, we check its remained neighborhood $N_r(u)$ (the set containing all neighbors of u that have not been removed from the graph yet), and add all vertices in $N_r(u)$ that satisfy at least one of the reduction rules into $RmQueue$.

Algorithm 3: ReduceGraph (G, C_0)

```
   Input: vertex weighted graph $G = (V, E, w)$, a clique $C_0$
   Output: A simplified graph of $G$
1  foreach $v \in V(G)$ do
2      if $UB_0(v) \le w(C_0)$ —— $UB_1(v) \le w(C_0)$ then
3          $RmQueue := RmQueue \cup \{v\}$;
4  while $RmQueue \ne \emptyset$ do
5      $u :=$ pop a vertex from $RmQueue$;
6      delete $u$ and its incident edges from $G$;
7      foreach $v \in N_r(u)$ do
8          if $UB_0(v) \le w(C_0)$ —— $UB_1(v) \le w(C_0)$ then
9              $RmQueue := RmQueue \cup \{v\}$;
10 return $G$;
```

According to Corollary 1, if the *ReduceGraph* algorithm returns an empty graph, that means the found clique is an optimal weighted clique of the input graph. However, there are cases that FastWClq finds an optimal weighted clique but *ReduceGraph* cannot reduce the graph to empty, because the reduction rules are incomplete.

4 Experimental Evaluation

We carry out experiments to evaluate FastWClq on a broad range of real-world massive graphs. We compare FastWClq against the currently best heuristic MWCP algorithm LSCC+BMS [Wang *et al.*, 2016], and the currently best exact algorithm MaxWClq [Fang *et al.*, 2014].

4.1 Experimental Preliminaries

The benchmarks in our experiments were originally from the Network Data Repository online [Rossi and Ahmed, 2015],[2] including biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks, amazon recommend networks, retweet networks, scientific computation networks, social networks, technological networks, and web link networks. The original benchmarks are unweighted, and we transformed them into a vertex weighted version: For the i^{th} vertex v_i, $w(v_i)$=(i mod 200)+1.There are totally 102 graphs. Many of these real-world graphs have millions of vertices and dozens of millions of edges. These benchmarks have been used in evaluating MWCP algorithms [Wang *et al.*, 2016], as well as algorithms for Maximum Clique [Rossi *et al.*, 2014], Coloring [Rossi and Ahmed, 2014] and Minimum Vertex Cover problems [Cai, 2015].

FastWClq is implemented in C++. Parameters k_0 and k_{max} for dynamic BMS heuristic are set to 4 and 64 (=2^6). LSCC+BMS and MaxWClq were implemented in C++ by their authors. All algorithms are complied with g++ version 4.7 with -O3 option.

[2]http://www.graphrepository.com/networks.php

The experiments are carried out on a workstation under Ubuntu Linux 14.04, using 2 cores of Intel i7-4710MQ CPU @ 2.50 GHz and 32 GByte RAM. We run FastWClq and LSCC+BMS 10 times on each graph. The cutoff time (”ct”) for FastWClq is 100 seconds per run. For LSCC+BMS, we test it under two cutoff time, 100 and 1000 seconds. This is to justify that the solutions found by FastWClq are sufficiently good even compared with those found by LSCC+BMS under 10 more time limit. For the exact algorithm MaxWClq, we run it once on each graph with a cutoff time of one hour.

For each graph, we report the best clique weight (”Best”) found by each algorithm, and the average clique weight over all runs (”Avg”) if a 100 percent success rate is not reached. If an algorithm fails to provide a solution for an instance, then the corresponding column is marked as ”N/A”. If an algorithm **proves** the optimal solution, the corresponding column is marked with a ”*”. Due to the limited space, we do not report the run time for each graph; instead, we report the averaged run time for each graph family (Table 3).

4.2 Experimental Results

Part 1: We first compare the algorithms in terms of solution quality. The results are presented in Tables 1 and 2. To make the comparison between FastWClq and other algorithms more clear, for a comparing algorithm, if the solution quality is worse than that found by FastWClq, then we mark it with ”$\downarrow$”, and if it is better we mark it with ”$\uparrow$”.

There are 12 graphs that have less than 1000 vertices, where all the algorithms find the optimal solution within a few seconds, and thus they are not reported.For the remaining 90 instances, FastWClq always finds a better or equal-quality solution compared to its competitors, with only one exception. FastWClq performs better on 47 instances than LSCC+BMS under the same cutoff time (100s), and performs better on 27 instances when the cutoff time for LSCC+BMS extends to 1000s. The exact solver MaxWClq fails on most of these graphs, yet it proves the optimal solution for 29 instances (including the 12 small instances), all of which have less than 12 thousand vertices. FastWClq proves the optimal solution for 46 instances, including 7 instances with millions of vertices, and the largest one that has 24 million vertices (inf-road-usa). The local search algorithm LSCC+BMS is essentially unable to prove the optimality of the solution.

Part 2: We now compare run time of the algorithms, which is summarized in Table 3. For each family, we calculate average run time over all runs for each instance, and report the average value of these average run time. If an algorithm fails to find a solution in all runs (marked with ”N/A”), its run time is considered to be the cutoff time on that instance.

FastWClq is usually orders of magnitude faster than the other two algorithms. Indeed, if we run LSCC+BMS with longer time to get the same quality solution by FastWClq (if possible), the run time of LSCC+BMS would be much longer. Moreover, FastWClq proves the optimal solution for 46 instances with the averaged time of 0.902 second, and exactly solves the largest instance with 24 million vertices (inf-road-usa) in 5.67 seconds. To summarize, FastWClq finds optimal or sub-optimal solutions for theses graphs within a few seconds on average, and solves many graphs in one second.

Table 1: Comparison of solution quality (I)

Graph	FastWClq ct=100s Best (Avg)	LSCC+BMS ct=100s Best (Avg)	LSCC+BMS ct=1000s Best (Avg)	MaxWClq ct=3600s Best
bio-dmela	805	805	805	805*
Bio-yeast	629*	629	629	629*
ca-AstroPh	5338*	5338	5338	5338
ca-citeseer	8838*	8838(8502.5) ↓	8838	N/A ↓
ca-coauthors-dblp	37884*	37884(26987.9) ↓	37884(37003.5) ↓	N/A ↓
ca-CondMat	2887*	2887	2887	N/A ↓
ca-CSphd	489*	N/A ↓	N/A ↓	489*
ca-dblp-2010	7575*	7456(7031.4) ↓	7575(7491.7) ↓	N/A ↓
ca-dblp-2012	14108*	14108(9305.4) ↓	14108	N/A ↓
ca-Erdos992	958*	958	958	958*
ca-GrQc	4279*	4279	4279	4279*
ca-HepPh	24533*	24533	24533	24533*
ca-hollywood-2009	222720*	222720(122957.6) ↓	222720	N/A ↓
ca-MathSciNet	2792*	2611(2257) ↓	2611(2556.2) ↓	N/A ↓
socfb-A-anon	2872	2602(1902.5) ↓	2728(2429.7) ↓	N/A ↓
socfb-B-anon	2662	2058(1789.4) ↓	2513(2035.2) ↓	N/A ↓
socfb-Berkeley13	4906	4906(4839.6) ↓	4906	N/A ↓
socfb-CMU	4141	4141	4141	4141*
socfb-Duke14	3694	3694	3694	3694*
socfb-Indiana	5412	5412(5274.8) ↓	5412	N/A ↓
socfb-MIT	3658	3658	3658	3658*
socfb-OR	3523	3523(3459.8) ↓	3523	N/A ↓
socfb-Penn94	4738	4738(4668.6) ↓	4738	N/A ↓
socfb-Stanford3	5769	5769	5769	5769*
socfb-Texas84	5546	5546(5545.2) ↓	5546	N/A ↓
socfb-uci-uni	1045	N/A ↓	N/A ↓	N/A ↓
socfb-UCLA	5595	5595	5595	N/A ↓
socfb-UConn	5733	5733	5733	N/A ↓
socfb-UCSB37	5669	5669	5669	4621 ↓
socfb-UF	6043	6043	6043	N/A ↓
socfb-UIllinois	5730	5730(5685.6) ↓	5730	N/A ↓
socfb-Wisconsin87	4239	4239	4239	N/A ↓
inf-power	888*	888	888	N/A ↓
inf-roadNet-CA	752*	668(604.5) ↓	668(640.5) ↓	N/A ↓
inf-roadNet-PA	669	599(598.2) ↓	599 ↓	N/A ↓
inf-road-usa	766*	N/A ↓	N/A ↓	N/A ↓
ia-email-EU	1350	1350	1350	N/A ↓
ia-email-univ	1473*	1473	1473	1473
ia-enron-large	2490	2490	2490	N/A ↓
ia-fb-messages	791	791	791	791*
ia-reality	374*	374	374	374*
ia-wiki-Talk	1884	1884	1884	N/A ↓
rec-amazon	942*	942	942	N/A ↓
rt-retweet-crawl	1367	1367(1349.8) ↓	1367	N/A ↓

Table 2: Comparison of solution quality (II)

Graph	FastWClq ct=100s Best (Avg)	LSCC+BMS ct=100s Best (Avg)	LSCC+BMS ct=1000s Best (Avg)	MaxWClq ct=3600s Best
sc-ldoor	4081	4060(3806.8) ↓	4074(3999.8) ↓	N/A ↓
sc-msdoor	4088	4074(3947.2) ↓	4088(4059.3) ↓	N/A ↓
sc-nasasrb	4548	4548(4540.8) ↓	4548	N/A ↓
sc-pkustk11	5298	5298(4860.1) ↓	5298(5215.2) ↓	N/A ↓
sc-pkustk13	6306*	5877(5759.7) ↓	6306(5958) ↓	N/A ↓
sc-pwtk	4620	4596(4518) ↓	4620(4610.4) ↓	N/A ↓
sc-shipsec1	3540	3540(3073.7) ↓	3540(3336.7) ↓	N/A ↓
sc-shipsec5	4524*	4500(3997.2) ↓	4524(4504.8) ↓	N/A ↓
soc-BlogCatalog	4803	4803	4803	N/A ↓
soc-brightkite	3672	3650(3643.7) ↓	3672(3663.2) ↓	N/A ↓
soc-buzznet	2981	2981(2980) ↓	2981	N/A ↓
soc-delicious	1547	1547(1511.8) ↓	1547(1545.6) ↓	N/A ↓
soc-digg	5303	4675(4645.7) ↓	5303(4800.6) ↓	N/A ↓
soc-douban	1682*	1682	1682	N/A ↓
soc-epinions	1657	1657	1657	N/A ↓
soc-flickr	7083	7083(6161) ↓	7083	N/A ↓
soc-flixster	3805	3805(3036.9) ↓	3805	N/A ↓
soc-FourSquare	3064	3064(2991.9) ↓	3064(3061.4) ↓	N/A ↓
soc-gowalla	2335	2335(2193.5) ↓	2335(2291.8) ↓	N/A ↓
soc-lastfm	1773	1773(1753.6) ↓	1773	N/A ↓
soc-livejournal	21368*	3589(2046.9) ↓	15599(4640.6) ↓	N/A ↓
soc-LiveMocha	1784(1775)	1784 ↑	1784 ↑	N/A ↓
soc-orkut	5452	N/A ↓	N/A ↓	N/A ↓
soc-pokec	3191	2341(1788.1) ↓	2341(2214.8) ↓	N/A ↓
soc-slashdot	2811	2811	2811	N/A ↓
soc-twitter-follows	808	808	808	N/A ↓
soc-youtube	1961	1961	1961	N/A ↓
soc-youtube-snap	1787	1787(1711.4) ↓	1787	N/A ↓
tech-as-caida2007	1869	1869	1869	N/A ↓
tech-as-skitter	5703	5611(4033.1) ↓	5703(5540.9) ↓	N/A ↓
tech-internet-as	1692	1692	1692	N/A ↓
tech-p2p-gnutella	703*	703	703	N/A ↓
tech-RL-caida	1861	1861	1861	N/A ↓
tech-routers-rf	1460*	1460	1460	1460*
tech-WHOIS	6154	6154	6154	6154*
web-arabic-2005	10558*	10558	10558	N/A ↓
web-BerkStan	3249*	3249	3249	3249
web-edu	2077*	2077	2077	2077*
web-google	1749*	1749	1749	1749*
web-indochina-2004	6997*	6997	6997	6997
web-it-2004	45477*	45477(44373.5) ↓	45477	N/A ↓
web-sk-2005	11925*	11925(9501.4) ↓	11925	N/A ↓
web-spam	2503	2503	2503	2503*
web-uk-2005	54850*	54850	54850	N/A ↓
web-webbase-2001	3574*	3574(3339.4) ↓	3574	3574
web-wikipedia2009	3891	3455(1370) ↓	3455(2405.3) ↓	N/A ↓

Table 3: Comparison of averaged run time on graph families

Graph	FastWClq ct=100s	LSCC+BMS ct=100s	LSCC+BMS ct=1000s	MaxWClq ct=3600s
Biology	0.001	0.024	0.024	1.118
Collaboration	4.543	32.237	318.852	1804.262
Facebook	6.338	30.838	198.568	2801.899
Infrastructure	1.573	40.932	377.693	N/A
Interaction	0.135	0.616	0.616	1200.642
Recommend	0.017	3.165	3.165	N/A
Retweet	0.027	12.133	21.852	1200.014
Science	0.437	47.751	406.918	N/A
Social Network	18.281	29.216	243.498	3130.438
Technique	1.763	8.898	74.298	2572.473
Web Link	0.241	17.512	116.081	1504.651
All	3.032	21.211	160.142	2274.136

5 Conclusions and Future Work

This paper presented a novel method for Maximum Weight Clique problem (MWCP), which aims to solve massive graphs within short time. The method interleaves between clique construction and graph reduction. Three ideas were proposed to improve the algorithm, including a benefit estimation function, a dynamic BMS heuristic, and a graph reduction algorithm. The resulting algorithm is called FastWClq. Experiments on real-world massive graphs show that, FastWClq finds better solutions than state of the art algorithms while the run time is much less, even when the time limit for the competitor is much more. Also, FastWClq proves the optimal solution for about half of the tested graphs in one second, including graphs with millions of vertices.

A significant direction is to apply this "Construction and Reduction" method and the ideas to other graph problems.

Acknowledgement

This work is supported by 973 Program 2014CB340301 and National Natural Science Foundation of China 61502464.

References

[Balasundaram and Butenko, 2006] Balabhaskar Balasundaram and Sergiy Butenko. Graph domination, coloring and cliques in telecommunications. In *Handbook of Optimization in Telecommunications*, pages 865–890. 2006.

[Ballard and Brown, 1982] DH Ballard and CM Brown. Computer vision. *New Jersey: Prentice Hall*, 1982.

[Batagelj and Zaversnik, 2003] Vladimir Batagelj and Matjaz Zaversnik. An $O(m)$ algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[Benlic and Hao, 2013] Una Benlic and Jin-Kao Hao. Breakout local search for maximum clique problems. *Computers & Operations Research*, 40(1):192–206, 2013.

[Busygin, 2006] Stanislav Busygin. A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics*, 154(15):2080–2096, 2006.

[Cai *et al.*, 2011] Shaowei Cai, Kaile Su, and Abdul Sattar. Local search with edge weighting and configuration checking heuristics for minimum vertex cover. *Artificial Intelligence*, 175(9):1672–1696, 2011.

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of IJCAI 2015*, pages 747–753, 2015.

[Eubank *et al.*, 2004] Stephen Eubank, V. S. Anil Kumar, Madhav V. Marathe, Aravind Srinivasan, and Nan Wang. Structural and algorithmic aspects of massive social networks. In *Proc. of SODA-04*, pages 718–727, 2004.

[Fang *et al.*, 2014] Zhiwen Fang, Chu-Min Li, Kan Qiao, Xu Feng, and Ke Xu. Solving maximum weight clique using maximum satisfiability reasoning. In *Proc. of ECAI 2014*, pages 303–308, 2014.

[Fellows and Downey, 1998] M. R. Fellows and R.G. Downey. *Parameterized Complexity*. Springer, 1998.

[Gomez Ravetti and Moscato, 2008] Martín Gomez Ravetti and Pablo Moscato. Identification of a 5-protein biomarker molecular signature for predicting alzheimer's disease. *PloS one*, 3(9):e3111, 2008.

[Guturu and Dantu, 2008] Parthasarathy Guturu and Ram Dantu. An impatient evolutionary algorithm with probabilistic tabu search for unified solution of some NP-hard problems in graph and set theory via clique finding. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 38(3):645–666, 2008.

[Karp, 1972] RM Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[Konc and Janezic, 2007] Janez Konc and Dušanka Janezic. An improved branch and bound algorithm for the maximum clique problem. *Communications in Mathematical and in Computer Chemistry*, 58:569–590, 2007.

[Kumlander, 2004] D. Kumlander. Fast maximum clique algorithms for large graphs. In *Proceedings of the fourth conference on engineering computational technology*, pages 202–208, 2004.

[Li and Quan, 2010] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *Proc. of AAAI*, pages 128–133, 2010.

[Li *et al.*, 2013] Chu-Min Li, Zhiwen Fang, and Ke Xu. Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *Proc. of ICTAI 2013*, pages 939–946, 2013.

[Lu and Chung, 2006] L. Lu and F. Chung. *Complex Graphs and Networks*. American Math. Society, New York, USA, 2006.

[Massaro *et al.*, 2002] Alessio Massaro, Marcello Pelillo, and Immanuel M. Bomze. A complementary pivoting approach to the maximum weight clique problem. *SIAM Journal on Optimization*, 12(4):928–948, 2002.

[Östergård, 1999] Patric R. J. Östergård. A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, 3:153–156, 1999.

[Pullan and Hoos, 2006] Wayne Pullan and Holger H Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, pages 159–185, 2006.

[Pullan, 2006] Wayne Pullan. Phased local search for the maximum clique problem. *J. Comb. Optim.*, 12(3):303–323, 2006.

[Pullan, 2008] Wayne Pullan. Approximating the maximum vertex/edge weighted clique using local search. *J. Heuristics*, 14(2):117–134, 2008.

[Rossi and Ahmed, 2014] Ryan A Rossi and Nesreen K Ahmed. Coloring large complex networks. *Social Network Analysis and Mining (SNAM)*, pages 1–52, 2014.

[Rossi and Ahmed, 2015] Ryan A Rossi and Nesreen K Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of AAAI 2015*, 2015.

[Rossi *et al.*, 2014] Ryan A Rossi, David F Gleich, Assefaw H Gebremedhin, and M. Patwary. Fast maximum clique algorithms for large graphs. In *Proc. of WWW*, pages 365–366, 2014.

[Segundo *et al.*, 2013] Pablo San Segundo, Fernando Matía, Diego Rodríguez-Losada, and Miguel Hernando. An improved bit parallel exact maximum clique algorithm. *Optimization Letters*, 7(3):467–479, 2013.

[Seidman, 1983] S. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[Singh and Gupta, 2006a] Alok Singh and Ashok Kumar Gupta. A hybrid evolutionary approach to maximum weight clique problem. *International Journal of Computational Intelligence Research*, 2(4):349–355, 2006.

[Singh and Gupta, 2006b] Alok Singh and Ashok Kumar Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12(1-2):5–22, 2006.

[Tomita and Kameda, 2007] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optimization*, 37(1):95–111, 2007.

[Tomita and Seki, 2003] Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete mathematics and theoretical computer science*, pages 278–289. 2003.

[Tomita *et al.*, 2010] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *Proceedings of WALCOM 2010*, pages 191–203, 2010.

[Verma *et al.*, 2015] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015.

[Wang *et al.*, 2016] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Two efficient local search algorithms for maximum weight clique problem. In *Proceedings of AAAI 2016*, 2016.

[Wu *et al.*, 2012] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, 2012.

[Zuckerman, 2007] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.