Journal of Combinatorial Optimization (2020) 39:933–954
https://doi.org/10.1007/s10878-020-00529-9

An efficient local search algorithm for solving maximum edge weight clique problem in large graphs

Yi Chu[1,2] · Boxiao Liu[1,2] · Shaowei Cai[3] · Chuan Luo[4,5] · Haihang You[1]

Published online: 4 February 2020

Abstract
Maximum vertex weight clique problem (MVWCP) and maximum edge weight clique problem (MEWCP) are two significant generalizations of maximum clique problem (MCP), and can be widely used in many real-world applications including molecular biology, broadband network design and pattern recognition. Recently, breakthroughs have been made for solving MVWCP in large graphs, resulting in several state-of-the-art algorithms, such as WLMC, FastWClq and LSCC + BMS. However, less attention has been paid to solving MEWCP in large graphs. In this paper, we present an efficient Stochastic Local Search (SLS) algorithm for MEWCP by combining clique construction, local search and graph reduction, resulting in a new algorithm named ReConSLS. We also propose a new upper bound function for edge weighted graphs which is essential for graph reduction. Extensive experiments on a wide range of large graphs demonstrate that ReConSLS surpasses state-of-the-art SLS competitors on the majority of testing graphs.

Keywords Graph reduction · Maximum edge weight clique problem · Stochastic local search

1 Introduction

Given an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a clique C is a subset of V, such that all vertices in C are connected. The maximum clique problem (MCP) is to find a clique with the maximum size in G. MCP has two significant generalizations: (1) the maximum vertex weight clique problem (MVWCP), in which each vertex is associated with a positive weight, and the goal is to find a clique with the maximum weight of vertices; (2) the maximum edge weight

✉ Haihang You
youhaihang@ict.ac.cn

Extended author information available on the last page of the article

clique problem (MEWCP), in which each edge is associated with a positive weight, and the goal is to find a clique with the maximum weight of edges.

MCP is a classical combinatorial optimization problem that was among one of the first problems proved to be NP-hard (Karp 1972). The weighted version of MCP (including MVWCP and MEWCP) has many real-world applications, including computer vision, pattern recognition, robotics (Ballard and Brown 1982), broadband network design (Park et al. 1996), wireless telecommunication (Balasundaram and Butenko 2006). Due to the theoretical significance and practical application value, considerable efforts have been made to develop algorithms for MCP, MVWCP and MEWCP. Algorithms for solving these problems are usually categorized into two classes: complete algorithms and incomplete algorithms.

On the one hand, most of the complete algorithms for MCP are based on the general branch and bound (BnB) framework. These algorithms differ from each other mainly by their techniques of determining the upper bound and their branching strategies Carraghan and Pardalos 1990; Tomita and Seki 2003; Tomita and Kameda 2007; San Segundo et al. 2011; Li and Quan 2010; Li et al. 2013; Jiang et al. 2017. On the other hand, incomplete algorithms cannot prove the optimality of the returned solutions, but these algorithms are usually able to find good quality solutions within reasonable time (Battiti and Protasi 2001; Pullan and Hoos 2006; Pullan 2006, 2008; Pullan et al. 2011; Wu et al. 2012; Benlic and Hao 2013; Wang et al. 2016; Fan et al. 2017a). Recently, numerous work focused on solving MVWCP in large-sized real-world graphs, resulting in several start-of-the-art algorithms. LSCC+BMS (Wang et al. 2016) adopts a balanced strategy called ‘Best from Multiple Selections’ (BMS) (Cai 2015) to improve the performance of a state-of-the-art algorithm named LSCC on massive graphs. FastWClq (Cai and Lin 2016) utilizes a new method for MVWCP which interleaves between clique construction and graph reduction, and achieves good performance on solving massive real-world graphs.

There are two classes of methods to obtain the exact solutions of MEWCP: (1) formulating MEWCP into mathematical programming such as integer programming (IP) (Gouveia and Martins 2015) and mixed integer programming (MIP); (2) using branch-and-bound method. The unconstrained quadratic binary program (UQP) model is used to represent the nonlinear formula of MEWCP. A tabu search heuristic designed for UQP produces a new algorithm for MEWCP named TS/UQP (Alidaee et al. 2007). Computational experiments on small instances show that the nonlinear model has computational advantages over the linear model. EWCLIQUE (Shimizu et al. 2018) is a purely combinatorial branch-and-bound algorithm and is faster than the methods based on mathematical programming on random benchmark and DIMACS benchmark. To promote the development of the methods that using branch-and-cut framework for MEWCP, some work is focused on a family of cutting planes and gives the theoretical proof that under certain conditions, one of the inequalities in this family defines a facet for MEWCP (Fomeni 2017).

Although great efforts have been made to solve MVWCP on large graphs, less attention was paid to MEWCP. Recently, two local search algorithms have been proposed for MEWCP. CERS (Fan et al. 2017b) can be used to solve both MVWCP and MEWCP, which outperforms a number of state-of-the-art algorithms on large graphs. LSMR (Li et al. 2018) is another efficient algorithm for MEWCP on massive graphs.

These local search algorithms derived from algorithms for MVWCP, although the frameworks of several algorithms for MVWCP could be adopted to solve MEWCP, many prominent strategies for MVWCP are not suitable for directly solving MEWCP.

1.1 Main contributions

In this paper, we are devoted to improving the performance over the state-of-the-art algorithms for solving MEWCP on large graphs. Our main contributions in this paper are concluded as follows.

Firstly, we construct a novel framework for solving MEWCP, which works in three different phases, i.e., clique construction, local search and graph reduction. Based on this framework, we develop an effective algorithm named ReConSLS for solving MEWCP.

Secondly, we propose an effective upper bound function for edge-weighted graphs which improves the performance of graph reduction.

Thirdly, we incorporate a number of effective techniques into this framework, including BMS strategy (Cai 2015), benefit estimation function of vertices (Cai and Lin 2016), resulting in a considerable performance improvement in the phases of clique construction and local search.

Finally, we conduct extensive experiments to compare ReConSLS against four state-of-the-art SLS algorithms including LSMR, CERS, LSCC and LSCC+BMS on a broad range of practical benchmarks, including 139 graphs from Network Data Repository and 18 graphs from KONECT (the Koblenz Network Collection). Our experimental results indicate that ReConSLS outperforms other competitors in terms of solution quality and running time on the majority of testing graphs.

1.2 Paper structure

We organize the remainder of the paper as follows. Section 2 gives preliminary definitions and notations. In Sect. 3, we present the new algorithm named ReConSLS along with the details of graph reduction, clique construction and local search. Extensive experimental results are shown in Sect. 4 to demonstrate the effectiveness of ReConSLS. Finally, we conclude the paper and point out future work in Sect. 5.

2 Preliminaries

Given an undirected graph $G = (V, E)$, where $V = \{v_1, v_2, \cdots, v_n\}$ is the set of vertices and $E = \{e_1, e_2, \cdots, e_m\}$ is the set of edges, a clique C is a subset of V, such that all vertices in C are connected. MCP is to find a clique with the maximum number of vertices in the given graph $G = (V, E)$. The MVWCP, in which the graph has a weight function W_v that assigns a positive weight to each vertex, and the weight of a clique C is defined as $W_v(C) = \Sigma_{i \in C} W_i$, is to find a clique with the maximum $W_v(C)$ in the given graph $G = (V, E, W_v)$. The MEWCP, in which the graph has a weight function W_e that assigns a positive weight to each edge, for an edge e_{ij}, the weight of e

is denoted by $W_{e_{ij}}$, the weight of a clique C is then defined as $W_e(C) = \Sigma_{e_{ij}, i, j \in C} W_{e_{ij}}$, is to find a clique with the maximum $W_e(C)$ in the given graph $G = (V, E, W_e)$. We use $W^*(G)$ to denote the maximum weight of clique in G.

2.1 Basic operations and scoring functions

Given an undirected graph $G = (V, E)$, an edge $e \in E$ is expressed as a pair of two vertices in V, i.e., e_{uv}, where $u \in V$ and $v \in V$ are two endpoints of e. For a vertex $v \in V$, the set of v's neighboring vertices is denoted by $N(v) = \{u \in V \mid e_{uv} \in E\}$; the degree of v is denoted by $d(v) = |N(v)|$. We also denote $N[v] = N(v) \cup \{v\}$.

To find a clique with good solution-quality, the local search based method usually moves from one clique to another until the time limit is reached, it then returns the clique with the best solution-quality found so far. Our algorithm framework contains three types of search steps: *add*, *swap*, and *drop*. Assume that C is a clique. In order to preserve the property of a clique, we define three sets as follows:

- $S_{add} = \{u | u \notin C, u \in N(v) \text{ for all } v \in C\}$, where C is still a clique after adding u into C if $u \in S_{add}$.
- $S_{swap} = \{e_{uv} | u \notin C, v \in C, e_{uv} \notin E, u \in N(w) \text{ for all } w \in C \backslash \{v\}\}$, where C is still a clique after both adding u into C and dropping v from C if $(u, v) \in S_{swap}$.
- $S_{drop} = \{v | v \in C\}$.

Let C be a clique, $W(C)$ denotes the weight of clique C. $Ascore(v, C)$ represents the increment of $W(C)$ after adding v into C, i.e., $Ascore(v, C) = W(C \cup \{v\}) - W(C)$. $Sscore(u, v, C)$ corresponds the increment of $W(C)$ after both adding u into C and dropping v from C, i.e., $Sscore(u, v, C) = W(C\backslash\{v\} \cup \{u\}) - W(C)$. $Dscore(v, C)$ is the increment of $W(C)$ after dropping v from C, i.e., $Dscore(v, C) = W(C\backslash\{v\}) - W(C)$.

Besides, given an undirected graph $G = (V, E)$ as well as a clique C in G, a vertex $v \in V$ has two possible states: inside C or outside C. We define the number of steps that has occurred since v last changed its states, as the *age* of v, denoted as $age(v)$. The cardinality of C is $|C|$.

2.2 Strong configuration checking

For local search algorithms, it is well acknowledged that a severe issue is the so-called cycling problem, i.e., frequently revisiting candidate solutions which have been explored lately. Cai et al. (2011) proposed an efficient strategy called configuration checking (CC) to handle the cycling problem. The CC strategy is usually implemented with a Boolean array named $confChange$, in which $confChange(v) = 1$ means that v's circumstance information (also known as configuration) has changed since the most recent change of v's state. Till now, the CC strategy has been successfully applied to a number of well-known NP-hard problems, including Boolean satisfiability (Cai and Su 2012; Luo et al. 2012, 2015a; Cai and Su 2013; Abramé et al. 2017), maximum satisfiability (Luo et al. 2015b, 2017), minimum vertex cover (Cai et al. 2011, 2013). Based on the CC strategy, literature Wang et al. (2016) proposed a strategy called strong

Algorithm 1: The *ReConSLS* Algorithm

```
   Input: graph $G = (V, E, W_e)$
1  initialize $C^* := \emptyset$, $ContinuousNoImprovedNum := 0$, $C_0 := \emptyset$;
2  while termination criteria are not met do
3      $C := CliqueConstruction(G)$;
4      $ContinuousNoImprovedNum++$;
5      if $W(C) > W(C^*)$ then
6          $C^* := C$; $ContinuousNoImprovedNum := 0$;
7      $C_s := StochasticLocalSearch(G, C)$;
8      if $W(C_s) > W(C^*)$ then
9          $C^* := C_s$; $ContinuousNoImprovedNum := 0$;
10     if ContinuousNoImprovedNum>threshold and $W(C_0) < W(C^*)$ then
11         $G := GraphReduction(G, C^*)$;
12         if the vertex set of $G = C^*$ then
13             return $C^*$;
14         $C_0 := C^*$;
15 return $C^*$;
```

configuration checking (SCC) in the context of MCP solving. The SCC strategy works according to the following rules:

(1) initially, for each vertex v, $confChange(v)$ is set to 1;
(2) when v is added into the current clique, $confChange(u)$ is set to 1 for all $u \in N(v)$;
(3) when v is removed from the current clique, $confChange(v)$ is set to 0;
(4) when v is removed from the current clique and u is added into the clique, $confChange(v)$ is set to 0.

A vertex v is allowed to be added into the clique if $confChange(v) = 1$.

3 The ReConSLS algorithm

In this section, we propose a new algorithm for MEWCP, which works in three phases, i.e., clique construction, local search and graph reduction. We first describe the algorithm, and then present details about the three phases of the algorithm.

3.1 The framework of ReConSLS

The pseudo-code of ReConSLS is outlined in Algorithm 1. The framework of the algorithm is described as follows. After initialization, ReConSLS executes a WHILE loop until the time limit is reached or the simplified graph of G is a clique. In each WHILE loop: firstly, a clique is constructed by iteratively adding vertices into a set that is initialized to be empty (line 3), the variable ContinuousNoImprovedNum is a consecutive loop counter which increases by 1 after each construction (line 4) and sets to 0 if a better solution is found (line 6, 9). Then the local search phase is executed (line 7). Finally, if ContinuousNoImprovedNum is greater than a positive integer threshold

(In this paper, the threshold is set to 10) and the weight of the current best clique is larger than that of the clique that has been used in the most recent graph reduction phase, the graph reduction phase is executed (line 11).

3.2 Graph reduction

A graph can be reduced to a simplified graph with the identical maximum weight clique by applying proper graph reduction rules. Intuitively, it is easier for a local search algorithm for finding a larger clique weight in the simplified graph due to a smaller search space. As computing the upper bound for a vertex in MEWCP is more complicated than that in MVWCP, it is not easy to directly adapt the bound-computing strategies in Cai and Lin (2016), Fang et al. (2014) and Jiang et al. (2017) for solving MEWCP. In this subsection, we propose an upper bound function for edge-weighted graphs and describe the graph reduction algorithm.

As in literature Cai and Lin (2016), we give the definition of upper bound of a vertex.

Definition 1 Given an edge-weighted graph $G = (V, E, W_e)$, for a vertex $v \in V$, an upper bound of v is a positive integer, denoted by $UB_{func}(v)$, such that for $\forall C$ in G that $v \in C$, $UB_{func}(v) \geq W(C)$.

Based on the Definition 1, we consider the following reduction rule:

Reduction Rule: Given an edge-weighted graph $G = (V, E, W_e)$ and a clique C_0 in G, for $\forall v \in V$, if $UB_{func}(v) < W(C_0)$, delete v and its incident edges from G.

We denote this rule by $Rule(UB_{func}, C_0)$, where UB_{func} is the upper bound function and C_0 is the input clique. We can apply this rule in the algorithm of solving MEWCP for graph reduction, which depends on the following proposition being true.

Proposition 1 *Given an edge-weighted graph $G = (V, E, W_e)$, G' is a simplified graph by applying $Rule(UB_{func}, C_0)$ on G. Then $W(G) = max\{W(C_0), W(G')\}$. The proof of Proposition 1 has been given in the literature Cai and Lin (2016).*

The upper bound function is fundamental in the reduction rule. We provide some notations before proposing the upper bound function. The sum of the weight of the edges adjacent to v is denoted by $WN(v) = \Sigma_{u \in N(v)} W_{e_{uv}}$. For a vertex v, the maximum weight of the edges adjacent to v is denoted by $Emax(v) = max\{W_{e_{uv}} | u \in N(v)\}$, the minimum weight of the edges adjacent to v is denoted by $Emin(v) = min\{W_{e_{uv}} | u \in N(v)\}$. For a clique C, we say that an edge e_{uv} is in C iff $i \in C$ and $j \in C$.

In order to describe the upper bound function, we propose a concept called the contribution from vertex u to vertex v. The contribution from u to v is defined as the sum of the weights of edges that connected with vertex u and may be in cliques containing vertex v, and is denoted by $CB(u, v)$. From this definition, we learn that u makes contribution to v iff u and v are neighbors. It is clear that the value of cardinality of cliques containing v will not be greater than $d(v) + 1$. The number of edges that u can contribute to v will not be greater than $d(v)$. In order to define the upper bound of vertex v, we first define the upper limit of $CB(u, v)$, denoted by $UL_{CB}(u, v)$. Suppose

Algorithm 2: The *GraphReduction* Algorithm

```
   Input: the edge weighted graph $G = (V, E, W_e)$, a clique $C_0$
1  initialize $isPending$ [] to 0, $pendingRm := \emptyset$;
2  foreach $v \in V(G)$ do
3      if $UB_{func}(v) < W(C_0)$ then
4          $pendingRm := pendingRm \cup \{v\}$; $isPending[v] := 1$;
5  while $\emptyset \neq pendingRm$ do
6      $u$ := pop a vertex from $pendingRm$;
7      remove $u$ and its incident edges from $G$;
8      foreach $v \in N(u)$ do
9          if $0 = isPending[v]$ and $UB(v) < W(C_0)$ then
10             $pendingRm := pendingRm \cup \{v\}$; $isPending[v] := 1$;
11 return;
```

that vertices u and v are neighbors, then the maximum number of edges adjacent to u within any clique C containing u and v is not greater than $min\{d(u), d(v)\}$. Assume that $d(v) < d(u)$, C is any clique containing u and v, let $EMAX = d(v) * Emax(u)$, $EMIN = WN(u) - (d(u) - d(v)) * Emin(u)$, the sum of weight of any $d(v)$ edges within C that are adjacent to u is not greater than $min\{EMAX, EMIN\}$. Thus, we give the definition of $UL_{CB}(u, v)$ as follows.

$$UL_{CB}(u, v) = \begin{cases} WN(u), & d(v) \geq d(u); \\ min\{EMAX, EMIN\}, & d(v) < d(u). \end{cases}$$

Based on the concept of contribution, we propose an efficient upper bound function. The definition is described as follows.

$$UB(v) = \begin{cases} 0, & d(v) = 0; \\ WN(v), & d(v) = 1; \\ \frac{WN(v) + \Sigma_{u \in N(v)} UL_{CB}(u,v)}{2}, & d(v) > 1. \end{cases}$$

Our graph reduction algorithm is based on our upper bound function and the reduction rule. The pseudo-code is outlined in Algorithm 2.

3.3 Clique construction

In this subsection, we present a clique construction algorithm. This algorithm contains two components: first, selecting a starting vertex, and add it into an empty set C; second, iteratively adding a vertex to expand the clique C.

We use $CandSet$ to denote the set containing candidate vertices. After initialization, each candidate vertex in $CandSet$ could be utilized as a starting vertex to construct a clique. During the adding vertex selection phase, given a current clique C, the property of $CanSet$ is identical with S_{add} which was proposed in Sect. 2.1, i.e., C is still a clique after adding u into C if $u \in CandSet$.

To select a starting vertex, we utilize our upper bound function $UB_{func}(v)$ as the selection criterion and utilize the BMS strategy to select a vertex in $CandSet$. The BMS strategy has been proposed for efficiently selecting a good-quality element from a large-sized candidate element set which uses a parameter k_s to control the sampling size. For instance, k_s sets to 1, referring to randomly select a vertex from $CandSet$; k_s sets to 100, indicating randomly selecting 100 vertices from $CandSet$ and then selecting v with the largest value of $UB_{func}(v)$ from the vertices previously selected. In our algorithm, k_s is set to 1 and 100 alternatively, which means randomly and greedily selecting alternatively.

To select adding vertices, inspired by the benefit estimation function of vertex addition in literature Cai and Lin (2016), we use $CandNeighborWeight(v)$ to denote the possible benefit after adding v into C, i.e, $CandNeighborWeight(v) = \Sigma_{u\in(N(v)\cap S_{add})} W_{e_{uv}}$. Based on $CandNeighborWeight(v)$, we propose a new function $B(v)$ to estimate the benefit of adding vertex v, i.e., $B(v) = Ascore(v, C) + CandNeighborWeight(v)/2$. We select the adding vertex based on $B(v)$, and adopt the dynamic BMS heuristic proposed in literature Cai and Lin (2016). For the parameter k of the dynamic BMS, it starts with a small value (k_0). The dynamic BMS utilized in our algorithm is similar with that in FastWClq (Cai and Lin 2016), but with two differences. (1) The dynamic BMS utilized in our algorithm adjusts k more frequently than that utilized in FastWClq. After each starting vertex selection phase, the value of k is increased via $k := 2k$; (2) whenever k exceeds the maximum value k_m, it resets to $k = k_0$ (In FastWclq, resets to $k := ++k_0$).

3.4 Stochastic local search

Based on the SCC strategy, in this phase, we utilize the BMS heuristic to strike a good balance between the quality of the selected neighboring clique and the time complexity. We also adopt a random walk strategy to choose the dropping vertex from the current clique.

The pseudo-code of the local search procedure is outlined in Algorithm 3. In each search step, according to the SCC strategy, Algorithm 3 randomly selects k vertices from S_{add} and then selects v with the maximum $Ascore$ from the vertices previously selected (line 3). It also randomly selects k vertex pairs from S_{swap} and then selects (u, u') with the maximum $Sscore$ from the vertex pairs previously selected (line 4). If $null = v$, then Algorithm 3 selects v' with the maximum $Dscore$ from S_{drop} (line 11), if $(null, null) = (u, u')$ or $Dscore(v') > Sscore(u, u')$, then Algorithm 3 determines to select a vertex to drop from the current clique; with probability p, it randomly selects a vertex from S_{drop} to be dropped (line 13-14), otherwise it drops vertex v' from the current clique (line 15).

Algorithm 3: The *StochasticLocalSearch* Algorithm

```
   Input: graph $G = (V, E, W_e)$, the current clique $C$
1  initialize $confChange$ [] to 0, $C^* := C$;
2  for $i$=0; $i$<$L$; $i$++ do
3      $v$ := vertex in $S_{add}$ with the biggest $Ascore(v)$ with BMS (with $k_{add}$) and $confChange(v) = 1$,
       breaking ties in favor of the oldest one; otherwise $v := null$;
4      $(u, u')$ := a vertex pair in $S_{swap}$ with the biggest $Sscore(u, u')$ with BMS (with $k_{swap}$) and
       $confChange(u) = 1$, breaking ties in favor of the oldest one; otherwise $(u, u') := (null, null)$;
5      if $null \neq v$ then
6          if $(u, u') = (null, null)$ or $Ascore(v) > Sscore(u, u')$ then
7              $C := C \cup \{v\}$;
8          else
9              $C := C \backslash \{u'\} \cup \{u\}$;
10     else
11         $v'$ := vertex in $S_{drop}$ with the biggest $Dscore(v')$, breaking ties in favor of the oldest one;
12         if $(u, u') = (null, null)$ or $Dscore(v') > Sscore(u, u')$ then
13             if with fixed probability $p$ then
14                 $v'$ := choose vertex in $S_{drop}$ randomly;
15             $C := C \backslash \{v'\}$;
16         else
17             $C := C \backslash \{u'\} \cup \{u\}$;
18     if $W(C) > W(C^*)$ then
19         $C^* := C$;
20 return $C^*$;
```

4 Experimental evaluations

In this section, we conduct experiments to compare ReConSLS against existing state-of-the-art competitors on a broad range of real-world large graphs, in order to evaluate the efficiency of ReConSLS for solving MEWCP.

4.1 The benchmarks

To comprehensively evaluate the effectiveness of our ReConSLS algorithm, we conducted experiments on two benchmarks:

(i) We downloaded all 139 graphs online,[1] which were originally taken from the Network Data Repository.[2] These graphs have been recently used for evaluating the performance of algorithms for solving MVWCP (Cai and Lin 2016; Wang et al. 2016; Fan et al. 2017a; Jiang et al. 2017), MEWCP (Fan et al. 2017b), Coloring (Rossi and Ahmed 2014) and MVC (Cai 2015).

(ii) In addition, we downloaded 18 unweighted undirected graphs from KONECT (The Koblenz Network Collection) (Kunegis 2013).[3] (**We chose those graphs with**

[1] http://www.ios.ac.cn/~caisw/Resource/realworld%20graphs.tar.gz.

[2] http://networkrepository.com/networks.php.

[3] http://konect.uni-koblenz.de/.

more than 300K edges and not included in the Network Data Repository. The graphs we used are available online[4])

Many of these real-world graphs have millions of vertices and dozens of millions of edges. Since the original graphs are unweighted, we transformed them into an edge weighted version using the same weighting method as in Pullan (2008) – for an edge e_{ij}, the weight of edge e_{ij} is set to $W_{e_{ij}} = ((i + j)\ mod\ 200) + 1$.

4.2 Experimental setup

We include four state-of-the-art SLS solvers as the competitors. CERS (Fan et al. 2017b) and LSMR (Li et al. 2018) are two efficient solvers for MEWCP in large graphs. Recently, breakthroughs have been made in MVWCP solving, resulting in several state-of-the-art solvers, such as LSCC, LSCC + BMS (Wang et al. 2016), FastWClq (Cai and Lin 2016) and WLMC (Jiang et al. 2017). Considering that it is not easy to adapt FastWClq and WLMC to solve MEWCP, we utilized LSCC and LSCC + BMS as two competitors by adapting such two solvers to solve MEWCP. We implemented ReConSLS, LSCC and LSCC + BMS in C++. The source code of ReConSLS is available online (see footnote 4). CERS was also implemented in C++ by its author and could be downloaded online[5]. LSMR was kindly provided by its author. All solvers were compiled by g++ (version 4.8.5) with the option '-O3'. All the experiments in this paper were carried out on a workstation under the operating system CentOS (version: 7.6.1810), with Intel(R) Xeon(R) CPU E5-2620 2.10GHz CPU, 20MB L3 cache and 128GB RAM.

In our experiments, all solvers are randomized ones. As a result, to make our evaluations statistically significant, each algorithm was performed 10 runs on each graph with seeds from 1 to 10. The cutoff time for each solver run was set to 1000 CPU seconds. For each run, we recorded the final solution and the time for seeking out the final solution. For each solver on each graph, we report the best clique weight, denoted by 'Wmax', the averaged clique weight over all runs, denoted by 'Wavg' (we do not report 'Wavg' if 'Wavg' = 'Wmax'), and the averaged time of finding the final solution in each run, denoted by 'time' (the unit is CPU second). We use 'N/A' to denote that a solver failed to run on that graph. In Tables 2 and 4 , in each category, we report the number of graphs where the solver finds the best clique weight among all competing solvers, denoted by '#max', the number of graphs where the solver finds the best averaged clique weight weight among all competing solvers, denoted by '#avg'. The number of graphs in each category is indicated by '#graph'. The number of vertices and edges in each graph is indicated by '|V|' and '|E|', respectively. The results in **boldface** show the best performance for the related graph or category in the same table. In this experiments, unspecified time units are **CPU seconds**.

In ReConSLS, parameters k_0 and k_m for dynamic BMS heuristic were set to 8 and 128, respectively. In the local search phase, the search depth L was set to 100, k_{add} and k_{swap} were set to 128, p for randomly choose the vertex to drop was set to 50%.

[4] https://github.com/acl-star/MEWCP-ReConSLS.

[5] https://github.com/Fan-Yi/Local-Search-for-Maximum-Edge-Weight-Clique.

In LSCC and LSCC+BMS, the search depth L was set to 4000, the BMS parameter k for LSCC+BMS was set to 100 as suggested in literature Wang et al. (2016).

4.3 Experimental results

4.3.1 Experimental results on graphs from network repository

Table 1 presents the comparative results of ReConSLS, CERS, LSMR, LSCC and LSCC+BMS on graphs from Network Repository. For the sake of space, we report 35 graphs and do not report the graphs that all the solvers find the same best clique weight within 25 s. The complete results are available online (see footnote 4). From Table 1, we can see that ReConSLS stands out as the best solver on this benchmark. On all 35 graphs reported in Table 1, ReConSLS finds the best clique weight on all of them. ReConSLS also finds the best averaged clique weight on all the 35 graphs, while the figure for CERS, LSMR, LSCC and LSCC+BMS is 33, 20, 27 and 32, respectively. In terms of running time, ReConSLS finds the best averaged clique weight with the shortest averaged time on 31 of the 35 graphs. Furthermore, on 16 of the 35 graphs, ReConSLS's speed of finding the best averaged clique weight is more than 3 times as fast as the second fastest competitor's. The comparative results on all the 139 graphs from Network Repository was presented in Table 2. The 139 graphs are divided into 12 categories according to the filename of these graphs. On all 12 categories, ReConSLS gives the best clique weight and best averaged clique weight with the shortest averaged time on all of them.

4.3.2 Experimental results on graphs from KONECT

Table 3 presents the comparative results of ReConSLS, CERS, LSMR, LSCC and LSCC+BMS on 18 graphs from KONECT. From Table 3, ReConSLS provides a performance advantage in terms of both clique weight and runtime. On all 18 instances, all the solvers find the best clique weight for all of them except LSMR. ReConSLS also finds the best clique weight on all 10 runs for all 18 graphs, while this figure for CERS, LSMR, LSCC and LSCC+BMS is only 13, 13, 14 and 13, respectively. In terms of running time, ReConSLS finds the best averaged clique weight with the shortest averaged time on 14 of the 18 instances. Furthermore, on 8 of the 18 graphs, ReConSLS's speed of finding the best averaged clique weight is more than 3 times as fast as the second fastest competitor's.

4.3.3 The effectiveness of the graph reduction algorithm and the ConSLS algorithm

To evaluate the effectiveness of the graph reduction algorithm proposed in Sect. 3.2, as well as the techniques utilized in the clique construction phase and the stochastic local search phase, we disable the graph reduction algorithm in ReConSLS, resulting in another solver named ConSLS.

We compare ReConSLS with ConSLS, CERS, LSMR, LSCC and LSCC+BMS on all the 157 graphs. The related results are listed in Table 4. According to Table 4, we

Table 1 Experimental results on graphs from network repository (For the sake of space, we do not report on graphs that all the solvers find the same best clique weight within 25 s, we do not report ‘Wavg’ which is equal to ‘Wmax’)

Graphs |V|, |E|	ReConSLS Wmax/Wavg (time)	CERS Wmax/Wavg (time)	LSMR Wmax/Wavg (time)	LSCC Wmax/Wavg (time)	LSCC+BMS Wmax/Wavg (time)
col/ca-coauthors-dblp	5661008	5661008	5661008	5661008	5661008
540K, 15M	(**5.791**)	(30.954)	(202.037)	(12.041)	(10.468)
col/ca-hollywood-2009	245095624	245095624	245095624	245095624	245095624
1M, 56M	(**27.621**)	(293.191)	(41.9)	(59.879)	(45.771)
col/ca-MathSciNet	32364	32364	32364	32364	32364
333K, 821K	(**0.5**)	(3.309)	(36.371)	(32.127)	(28.829)
fb/socfb-A-anon	**32532**	**32532**	**32532/**	**32532**	**32532**
3M, 24M	(**13.209**)	(47.128)	29501.9(361.87)	(37.738)	(45.194)
fb/socfb-B-anon	**28384**	**28384**	**28384/**	**28384**	**28384**
3M, 21M	(**24.585**)	(86.065)	23797.3(463.451)	(129.526)	(181.901)
fb/socfb-Duke14	**55940**	**55940/**	**55940**	**55940**	**55940**
10K, 506K	(**0.198**)	55929.2(1.639)	(1.806)	(0.358)	(0.67)
fb/socfb-uci-uni	**2210**	**2210**	N/A/	**2210**	**2210**
59M, 92M	(119.391)	(204.223)	N/A(N/A)	(89.336)	(**71.63**)
inf/inf-roadNet-CA	1050	1050	975/	1050	1050
2M, 3M	(**4.195**)	(6.534)	817.9(441.359)	(228.841)	(221.563)
inf/inf-roadNet-PA	1164	1164	1164	1164	1164
1M, 2M	(**1.704**)	(3.025)	(261.547)	(32.653)	(33.221)
inf/inf-road-usa	**1092**	**1092**	N/A/	1017/	1017/
24M, 29M	(**70.128**)	(87.841)	N/A(N/A)	864.2(521.612)	864.2(524.229)
rec/rec-amazon	1866	1866	1866	1866	1866
92K, 126K	(**0.009**)	(0.429)	(29.845)	(0.517)	(0.53)

Table 1 continued

Graphs |V|, |E|	ReConSLS Wmax/Wavg (time)	CERS Wmax/Wavg (time)	LSMR Wmax/Wavg (time)	LSCC Wmax/Wavg (time)	LSCC+BMS Wmax/Wavg (time)
ret/rt-retweet-crawl	**8262**	**8262**	**8262/**	**8262/**	**8262**
1M, 2M	(**8.042**)	(20.597)	5707.2(151.553)	4868.4(115.35)	(101.195)
sci/sc-ldoor	**40610**	**40610**	**40610/**	**40610**	**40610**
952K, 21M	(**7.948**)	(33.821)	39830(450.446)	(57.817)	(56.408)
sci/sc-msdoor	**40250**	**40250**	**40250/**	**40250**	**40250**
416K, 9M	(**3.152**)	(17.262)	40098(449.816)	(67.457)	(64.99)
sci/sc-pkustk11	77580	77580	77580	77580	77580
88K, 3M	(**0.952**)	(6.693)	(38.991)	(25.114)	(12.744)
sci/sc-pkustk13	99915	99915	99915	99915	99915
95K, 3M	(**2.884**)	(48.517)	(72.302)	(77.931)	(109.506)
sci/sc-pwtk	**51888**	**51888**	**51888/**	**51888**	**51888**
218K, 6M	(**2.022**)	(8.322)	51832.8(345.381)	(6.436)	(7.247)
sci/sc-shipsec1	45126	45126	45126	45126	45126
140K, 2M	(**0.639**)	(3.042)	(75.638)	(8.03)	(7.729)
sci/sc-shipsec5	48576	48576	48576	48576	48576
179K, 2M	(**0.86**)	(4.633)	(33.622)	(12.053)	(12.11)
soc/soc-digg	**123757**	**123757**	**123757/**	**123757**	**123757**
771K, 6M	(**21.744**)	(267.384)	123738.4(275.439)	(42.349)	(60.326)
soc/soc-flickr	166552	166552	166552	166552	166552
514K, 3M	(**2.002**)	(11.661)	(96.149)	(3.237)	(4.471)
soc/soc-flixster	47685	47685	47685	47685	47685

Table 1 continued

Graphs |V|, |E|	ReConSLS Wmax/Wavg (time)	CERS Wmax/Wavg (time)	LSMR Wmax/Wavg (time)	LSCC Wmax/Wavg (time)	LSCC + BMS Wmax/Wavg (time)
3M, 8M	(5.498)	(16.95)	(188.161)	(6.674)	(**5.006**)
soc/soc-FourSquare	45982	45982	45982	45982	45982
639K, 3M	(**5.915**)	(99.325)	(21.657)	(168.832)	(14.523)
soc/soc-gowalla	**30226**	**30226**	**30226**	**30226/**	**30226**
197K, 950K	(**2.851**)	(37.038)	(26.358)	29466.4(390.045)	(347.044)
soc/soc-lastfm	11266	11266	11266	11266	11266
1M, 5M	(**8.891**)	(12.439)	(118.135)	(34.966)	(24.438)
soc/soc-livejournal	**2289993**	**2289993**	2157827/	**2289993/**	**2289993/**
4M, 28M	(**19.139**)	(106.825)	1247883.3(485.849)	2134981.8(378.917)	2212487.4(365.109)
soc/soc-orkut	**105549/**	**105549/**	N/A/	**105549/**	**105549/**
3M, 106M	**105475**(475.155)	99305.1(435.907)	N/A(N/A)	97568.7(233.93)	96304.1(272.138)
soc/soc-pokec	**38202**	**38202**	**38202/**	**38202/**	**38202**
2M, 22M	(**44.299**)	(123.91)	29166.9(411.257)	36942.1(477.04)	(491.217)
soc/soc-twitter-follows	1625	1625	1625	1625	1625
405K, 713K	(**1.003**)	(3.162)	(32.846)	(2.944)	(1.934)
tec/tech-as-skitter	179915	179915	179915	179915	179915
2M, 11M	(**10.25**)	(64.881)	(113.958)	(84.341)	(77.636)
tem/scc_reality	**76649665**	**76649665**	**76649665**	**76649665/**	**76649665**
7K, 5M	(44.377)	(56.345)	(**19.302**)	76372857(393.69)	(393.252)
tem/scc_retweet-crawl	**19648**	**19648**	**19648/**	**19648**	**19648**
1M, 24K	(**0.059**)	(0.462)	19126.4(340.887)	(0.27)	(0.261)

Table 1 continued

Graphs |V|, |E|	ReConSLS Wmax/Wavg (time)	CERS Wmax/Wavg (time)	LSMR Wmax/Wavg (time)	LSCC Wmax/Wavg (time)	LSCC+BMS Wmax/Wavg (time)
tem/scc_twitter-copen	16912230	16912230	16912230	16912230	16912230
9K, 474K	(**0.47**)	(7.961)	(1.656)	(40.283)	(27.56)
web/web-it-2004	9308691	9308691	9308691	9308691	9308691
509K, 7M	(**2.939**)	(10.115)	(98.14)	(6.531)	(5.542)
web/web-wikipedia2009	**46832**	**46832**	**46832/**	**46832/**	**46832**
2M, 5M	(**4.183**)	(18.645)	46789.7(184.192)	46822.6(360.346)	(218.143)

Table 2 Experimental results on 139 graphs from network repository

Graph category (#graph)	ReConSLS #max/#avg (time)	CERS #max/#avg (time)	LSMR #max/#avg (time)	LSCC #max/#avg (time)	LSCC + BMS #max/#avg (time)
bio/bio(4)	**4/4**(**0.01**)	**4/4**(0.29)	**4/4**(**0.01**)	**4/4**(**0.01**)	**4/4**(**0.01**)
col/ca(13)	**13/13**(**2.70**)	**13/13**(25.84)	**13/13**(25.27)	**13/13**(8.98)	**13/13**(7.39)
fb/socfb(18)	**18/18**(**9.21**)	**18**/17(22.70)	17/15(103.67)	**18/18**(17.60)	**18/18**(21.02)
inf/inf(4)	**4/4**(**19.01**)	**4/4**(24.42)	2/2(425.74)	3/3(195.78)	3/3(194.75)
int/ia(9)	**9/9**(**0.02**)	**9/9**(0.45)	**9/9**(0.14)	**9/9**(0.12)	**9/9**(0.14)
rec/rec(1)	**1/1**(**0.01**)	**1/1**(0.43)	**1/1**(29.85)	**1/1**(0.52)	**1/1**(0.53)
ret/rt(3)	**3/3**(**2.68**)	**3/3**(7.05)	**3**/2(50.52)	**3**/2(38.45)	**3/3**(33.73)
sci/sc(8)	**8/8**(**2.36**)	**8/8**(15.52)	**8**/5(184.72)	**8/8**(32.12)	**8/8**(34.10)
soc/soc(23)	**23/23**(**25.84**)	**23**/22(51.41)	21/19(117.54)	**23**/19(76.28)	**23**/21(69.66)
tec/tech(7)	**7/7**(**1.52**)	**7/7**(9.75)	**7/7**(17.59)	**7/7**(13.40)	**7/7**(12.63)
tem/scc(37)	**37/37**(**1.25**)	**37/37**(2.06)	**37**/36(10.66)	**37**/36(12.18)	**37/37**(11.68)
web/web(12)	**12/12**(**0.96**)	**12/12**(4.06)	**12**/11(25.91)	**12**/11(31.64)	**12/12**(19.84)
summary(139)	**139/139**(**6.95**)	**139**/137(17.04)	134/124(65.40)	138/131(30.71)	138/136(28.71)

Table 3 Comparative results of ReConSLS, CERS, LSMR, LSCC and LSCC + BMS on real world graphs from KONECT

Graphs |V|, |E|	ReConSLS Wmax/Wavg (time)	CERS Wmax/Wavg (time)	LSMR Wmax/Wavg (time)	LSCC Wmax/Wavg (time)	LSCC + BMS Wmax/Wavg (time)
Amazon-MDS	3951	3951	3951	3951	3951
335K, 926K	(**0.59**)	(3.075)	(161.879)	(23.123)	(7.929)
Catster-friendships	**329120**	**329120/**	**329120**	**329120/**	**329120**
150K, 5M	(55.545)	329024(411.039)	(**23.762**)	329072(233.708)	(257.492)
DBLP-co-authorship	633443	633443	633443	633443	633443
317K, 1M	(**0.54**)	(2.32)	(14.13)	(1.134)	(1.023)
Dogster-friendships	**92969**	**92969/**	**92969/**	**92969/**	**92969/**
427K, 9M	(**64.338**)	88927.2(299.009)	92626.8(327.275)	92373.6(407.095)	91778.2(313.707)
Douban	4995	4995	4995	4995	4995
155K, 327K	(0.192)	(0.741)	(4.488)	(0.155)	(**0.153**)
Facebook-friendships	47639	47639	47639	47639	47639
64K, 817K	(**0.515**)	(3.611)	(5.62)	(5.393)	(4.641)
Familylinks	**67588403**	**67588403**	**67588403**	**67588403/**	**67588403**
624K, 16M	(**15.266**)	(287.985)	(42.719)	61941584.2(242.374)	(471.863)
Flickr	16500789	16500789	16500789	16500789	16500789
106K, 2M	(**1.045**)	(7.888)	(2.87)	(13.232)	(11.509)
Flickr-links	**476372**	**476372/**	**476372/**	**476372**	**476372**
2M, 16M	(**58.178**)	472212.8(400.255)	470782.4(117.741)	(302.157)	(189.103)
Flixster	51215	51215	51215	51215	51215
3M, 8M	(6.043)	(17.21)	(121.339)	(**4.669**)	(6.1)

Table 3 continued

Graphs |V|, |E|	ReConSLS Wmax/Wavg (time)	CERS Wmax/Wavg (time)	LSMR Wmax/Wavg (time)	LSCC Wmax/Wavg (time)	LSCC + BMS Wmax/Wavg (time)
Gowalla	**29410**	**29410**	**29410**	**29410**	**29410/**
197K, 950K	(**2.684**)	(92.026)	(23.79)	(192.297)	28725.3(164.258)
Hyves	15430	15430	15430	15430	15430
1M, 3M	(1.829)	(8.384)	(72.912)	(**1.749**)	(2.379)
Livejournal-links	**6426160**	**6426160/**	6263339/	**6426160/**	**6426160/**
5M, 49M	(**35.525**)	6423128.2(522.146)	4448625(368.74)	6338088.8(471.455)	6132512.6(469.731)
Orkut	**131452**	**131452/**	N/A/	**131452**	**131452/**
3M, 117M	(**272.405**)	125561.2(425.772)	N/A(N/A)	(369.85)	129953.3(389.909)
Texas	1110	1110	1110	1110	1110
1M, 2M	(**1.216**)	(3.172)	970.8(328.657)	(7.703)	(7.506)
WordNet	60251	60251	60251	60251	60251
146K, 657K	(**0.36**)	(1.867)	(28.351)	(6.611)	(4.872)
YouTube	**20185**	**20185**	**20185**	**20185**	**20185/**
3M, 9M	(**24.627**)	(295.592)	(216.041)	(321.944)	19843.8(427.54)
Youtube-friendship	14648	14648	14648	14648	14648
1M, 3M	(**1.421**)	(11.379)	(11.392)	(3.092)	(2.264)

Table 4 Experimental results of ReConSLS, ConSLS, CERS, LSMR, LSCC and LSCC+BMS on all the graphs

Graph	ReConSLS	ConSLS	CERS	LSMR	LSCC	LSCC+BMS
category	#max/#avg	#max/#avg	#max/#avg	#max/#avg	#max/#avg	#max/#avg
(#graph)	(time)	(time)	(time)	(time)	(time)	(time)
repository(139)	**139/139(6.95)**	**139**/137(10.41)	**139**/137(17.04)	134/124(65.40)	138/131(30.71)	138/136(28.71)
konect(18)	**18/18(30.13)**	**18/18**(33.31)	**18**/13(155.19)	**18**/13(159.54)	**18**/14(144.87)	**18**/13(151.78)
summary(157)	**157/157(9.61)**	**157**/155(13.04)	**157**/150(32.88)	152/137(76.19)	156/145(43.80)	156/149(42.82)

can see that ReConSLS gives the best performance on all the graphs and ConSLS is the second best solver. In terms of solution quality, ReConSLS gives the best clique weight and best averaged clique weight on all of them. In terms of running time, the averaged running time for ReConSLS of finding the largest clique weight is 9.61, and the figure for ConSLS, CERS, LSMR, LSCC and LSCC + BMS is 13.04, 32.88, 76.19, 43.80 and 42.82, respectively. The comparison between ReConSLS and ConSLS indicates the effectiveness of the graph reduction algorithm. The comparison between ConSLS and other competitors shows that the techniques utilized in clique construction phase and stochastic local search phase are efficient.

5 Conclusions and future work

In this paper, we present an effective local search algorithm named ReConSLS, which works in three phases, i.e. clique construction, stochastic local search and graph reduction. We also proposed a new upper bound function for edge-weighted graphs to improve the performance of graph reduction. Experiments on real-world large graphs demonstrate that ReConSLS outperforms other competitors in terms of solution quality and running time on majority of testing graphs. Furthermore, we remove the graph reduction algorithm from ReConSLS, resulting in an alternative algorithm named ConSLS. We conducted experiments to compare ConSLS against ReConSLS, CERS, LSMR, LSCC and LSCC + BMS, and the related results show that ConSLS is the second best solver following ReConSLS, indicating the effectiveness of our upper bound function and the techniques utilized in the clique construction phase and the stochastic local search phase.

For future work, we would like to design more efficient graph reduction algorithm, as well as heuristics applied to MEWCP.

Acknowledgements This work is partially supported by the National Key Research and Development Program of China under Grant 2017YFB0202502. Shaowei Cai is supported by Youth Innovation Promotion Association, Chinese Academy of Sciences (No. 2017150).

References

Abramé A, Habet D, Toumi D (2017) Improving configuration checking for satisfiable random k-sat instances. Ann Math Artif Intell 79(1–3):5–24

Alidaee B, Glover F, Kochenberger G, Wang H (2007) Solving the maximum edge weight clique problem via unconstrained quadratic programming. Eur J Oper Res 181(2):592–597

Balasundaram B, Butenko S (2006) Graph domination, coloring and cliques in telecommunications. In: Resende MGC, Pardalos PM (eds) Handbook of optimization in telecommunications. Springer, Boston

Ballard DH, Brown CM (1982) Computer vision. Prenice-Hall, Englewood Cliffs

Battiti R, Protasi M (2001) Reactive local search for the maximum clique problem 1. Algorithmica 29(4):610–637

Benlic U, Hao JK (2013) Breakout local search for maximum clique problems. Comput Oper Res 40(1):192–206

Cai S (2015) Balance between complexity and quality: local search for minimum vertex cover in massive graphs. In: Proceedings of IJCAI 2015, pp 747–753

Cai S, Su K (2012) Configuration checking with aspiration in local search for sat. In: AAAI

Cai S, Su K (2013) Local search for boolean satisfiability with configuration checking and subscore. Artif Intell 204:75–98

Cai S, Lin J (2016) Fast solving maximum weight clique problem in massive graphs. In: Proceedings of IJCAI 2016, pp 568–574

Cai S, Su K, Sattar A (2011) Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif Intell 175(9–10):1672–1696

Cai S, Su K, Luo C, Sattar A (2013) NuMVC: an efficient local search algorithm for minimum vertex cover. J Artif Intell Res 46:687–716

Carraghan R, Pardalos PM (1990) An exact algorithm for the maximum clique problem. Oper Res Lett 9(6):375–382

Fan Y, Li N, Li C, Ma Z, Latecki LJ, Su K (2017a) Restart and random walk in local search for maximum vertex weight cliques with evaluations in clustering aggregation. In: Proceedings of international joint conference on artificial intelligence (IJCAI), pp 622–630

Fan Y, Ma Z, Su K, Li C, Rao C, Liu RH, Latecki L (2017b) A local search algorithm for the maximum weight clique problem in large graphs. In: 29rd IEEE international conference on tools with artificial intelligence (ICTAI) 2017. IEEE, pp 1099–1104

Fang Z, Li CM, Qiao K, Feng X, Xu K (2014) Solving maximum weight clique using maximum satisfiability reasoning. In: Proceedings of the twenty-first European conference on artificial intelligence. IOS Press, pp 303–308

Fomeni FD (2017) A new family of facet defining inequalities for the maximum edge-weighted clique problem. Optim Lett 11(1):47–54

Gouveia L, Martins P (2015) Solving the maximum edge-weight clique problem in sparse graphs with compact formulations. EURO J Comput Optim 3(1):1–30

Jiang H, Li CM, Manya F (2017) An exact algorithm for the maximum weight clique problem in large graphs. In: AAAI, pp 830–838

Karp RM (1972) Reducibility among combinatorial problems. J Symb Logic 40(4):618–619

Kunegis J (2013) Konect: the koblenz network collection. In: Proceedings of the 22nd international conference on world wide web. ACM, pp 1343–1350

Li CM, Quan Z (2010) An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In: AAAI, vol 10, pp 128–133

Li CM, Fang Z, Xu K (2013) Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In: 2013 IEEE 25th international conference on tools with artificial intelligence (ICTAI), pp 939–946. IEEE

Li R, Wu X, Liu H, Wu J, Yin M (2018) An efficient local search for the maximum edge weighted clique problem. IEEE Access 6:10743–10753

Luo C, Su K, Cai S (2012) Improving local search for random 3-SAT using quantitative configuration checking. In: Proceedings of ECAI 2012, pp 570–575

Luo C, Cai S, Su K, Wu W (2015a) Clause states based configuration checking in local search for satisfiability. IEEE Trans Cybern 45(5):1028–1041

Luo C, Cai S, Wu W, Jie Z, Su K (2015b) CCLS: an efficient local search algorithm for weighted maximum satisfiability. IEEE Trans Comput 64(7):1830–1843

Luo C, Cai S, Su K, Huang W (2017) CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. Artif Intell 243:26–44

Park K, Lee K, Park S (1996) An extended formulation approach to the edge-weighted maximal clique problem. Eur J Oper Res 95(3):671–682

Pullan W (2006) Phased local search for the maximum clique problem. J Comb Optim 12(3):303–323

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

Pullan W, Hoos HH (2006) Dynamic local search for the maximum clique problem. J Artif Intell Res 25:159–185

Pullan W, Mascia F, Brunato M (2011) Cooperating local search for the maximum clique problem. J Heuristics 17(2):181–199

Rossi RA, Ahmed NK (2014) Coloring large complex networks. Soc Netw Anal Min 4(1):228

San Segundo P, Rodríguez-Losada D, Jiménez A (2011) An exact bit-parallel algorithm for the maximum clique problem. Comput Oper Res 38(2):571–581

Shimizu S, Yamaguchi K, Masuda S (2018) A branch-and-bound based exact algorithm for the maximum edge-weight clique problem. In: International conference on computational science/intelligence & applied informatics. Springer, pp 27–47

Tomita E, Kameda T (2007) An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J Global Optim 37(1):95–111

Tomita E, Seki T (2003) An efficient branch-and-bound algorithm for finding a maximum clique. In: International conference on discrete mathematics and theoretical computer science, pp 278–289

Wang Y, Cai S, Yin M (2016) Two efficient local search algorithms for maximum weight clique problem. In: Proceedings of AAAI 2016, pp 805–811

Wu Q, Hao JK, Glover F (2012) Multi-neighborhood tabu search for the maximum weight clique problem. Ann Oper Res 196(1):611–634

Affiliations

Yi Chu[1,2] · Boxiao Liu[1,2] · Shaowei Cai[3] · Chuan Luo[4,5] · Haihang You[1]

Yi Chu
chuyi@ict.ac.cn

Boxiao Liu
liuboxiao@ict.ac.cn

Shaowei Cai
caisw@ios.ac.cn

Chuan Luo
chuanluosaber@gmail.com

1 State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

2 University of Chinese Academy of Sciences, Beijing 100049, China

3 State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China

4 Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands

5 Microsoft Research, Beijing, China