Deterministic Tournament Selection in Local Search for Maximum Edge Weight Clique on Large Sparse Graphs

Zongjie Ma[1⋆], Yi Fan[2,1], Kaile Su[1], Chengqian Li[3], and Abdul Sattar[1]

[1] Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
[2] Department of Computer Science, Jinan University, Guangzhou, China
[3] Department of Computer Science, Sun Yat-sen University, China
zongjie.ma@griffithuni.edu.au

Abstract. The maximum edge weight clique (MEWC) problem is important in both theories and applications. During last decades, there has been much interest in finding optimal or near-optimal solutions to this problem. Many existing heuristics focuses on academic benchmarks of relatively small size. However, very little attention has been paid to solving the MEWC problem in large sparse graphs. In this work, we exploit the so-called deterministic tournament selection (DTS) strategy to improve the local search MEWC algorithms. Experiments conducted on a broad range of large sparse graphs show that our algorithm outperforms state-of-the-art local search algorithms in this benchmark. Moreover it finds better solutions on a list of them.

1 Introduction

The rapid growth of the Internet, widespread deployment of sensors and other fields produced huge quantity of massive data sets, which has generated a series of computational challenges to existing algorithms. Hence, new algorithms need to be designed to deal with these data sets. Many data sets can be represented as graphs, and the study of large real-world graphs, also known as complex networks [18], has become an active research agenda over recent decades.

Given a simple undirected graph where edges are weighted, the maximum edge weight clique (MEWC) problem is to find a clique whose total edge weight is maximum. This problem exists in many real applications like [6, 14, 15, 1, 2, 7]. However, it is NP-hard and difficult to approximate [11], so improving the algorithms on this problem is of great importance.

Due to the NP-hardness, the research of MEWC problem focuses on developing heuristics to find a "good" clique within reasonable time periods. Up to now, there are two types of algorithms for MEWC: complete algorithms *e.g.* [3, 12] and incomplete ones *e.g.* [15, 17]. In this paper we use local search to find a clique whose weight is as great as possible.

⋆ Corresponding author

A correlated problem of the MEWC problem is the maximum vertex weight clique (MVWC) problem which asks for a clique with the greatest total vertex weight. Recently this problem attracts much attention in the constraint optimization community like [21, 10, 20, 4, 8].

Although there has been great progress in MVWC solving, very little attention is being paid to the MEWC problem. The reason may be that MEWC is more complicated and thus difficult to solve, from the viewpoint of algorithm design. For example in MVWC solving, when computing the upper-bound for a vertex v, we can simply sum up the weights of v's neighbors. However in MEWC, we have to sum up the edge weights among v and its neighbors, which is more complicated. Hence, those bounds which are shown to be useful in MVWC solving may lose their power in MEWC solving. So it is not easy to adopt the strategies in [9, 4, 13] to solve the MEWC problem.

On the other hand, local search seems to be a simple but effective approach. According to the literature, local search for MVWC usually moves from one clique to another until the cutoff arrives. During the search procedure, it moves to the neighboring clique with the greatest weight by adding, dropping or swapping vertices, according to some tabu criterion. This approach can easily be adopted to solve the MEWC problem. So it seems that for local search, the MVWC problem and the MEWC problem can be solved in very similar ways. Moreover, some well-known strategies like the multi-neighborhood greedy search, the randomized tabu strategy, the strong configuration checking strategy, the deterministic tournament selection and the data structures, can all be adopted to solve the MEWC problem trivially.

In the literature, LSCC[4] is known to be the most prominent local search MVWC solver. It was shown to be effective on both standard and large benchmarks. Our observations find that it can be adapted to solve the MEWC problem in a straightforward way. Similarly, another local search solver LMY-GRS [8], which is powerful for solving the MVWC problem on large sparse graphs, can also be adapted to solve the MEWC problem easily. Therefore, in this paper we adapt them to deal with edge weights, and evaluate them in MEWC solving.

To the best of our knowledge, current local search methods solve the MVWC problem and the MEWC problem in nearly the same way. That is, there are no local search techniques which are specialized for edge weights or vertex weights. Although current techniques can be widely used due to their generality, they may fail to tailor the local search to specific problem structures.

1.1 Our Contributions

In this paper we proposed a strategy which is specialized for edge weights. It selects some edges with great weight, and uses their endpoints as the starting point of local search. The intuition is that the search space is huge and we can

[4] In [20], there are LSCC and LSCC+BMS. LSCC is better on DIMACS and BHOSLIB. LSCC+BMS is better on large crafted graphs. For simplicity, we write both versions as LSCC if it is understood from the context or there are no confusions.

only visit a very small part of the space within reasonable time periods, so we have to choose some promising parts. More specifically we choose an edge with great weight by the deterministic tournament selections (DTS) [16]. Given a set S and a positive integer k, the DTS heuristic works as follows: *randomly select k elements from S with replacements and then return the best one.* Based on DTS for selecting edges, we develop a new local search solver called LS-DTS[5], which is dedicated to solve the MEWC problem on large sparse graphs.

We conduct experiments on a broad range of large sparse graphs. The experimental results show that our solver LS-DTS significantly outperforms LSCC, LSCC+BMS and LMY-GRS.

2 Preliminaries

Formally the MEWC problem is defined over a graph $G = (V, E, w)$, where $V = \{v_1, \ldots, v_n\}$ is the vertex set, each edge $e \in E$ is a 2-element subset of V and $w : E \mapsto R_{\geq 0}$ is the weighting function on E. A *clique* C is a subset of V s.t. each pair of vertices in C is mutually adjacent. The MEWC problem is to find a clique which maximizes

$$\sum_{v_i, v_j \in C \text{ and } i \neq j} w(\{v_i, v_j\}). \tag{1}$$

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 $N(v) = \{u | u \text{ and } v \text{ are neighbors.}\}$ to denote the set of v's neighbors. We use $d_{max}(G)$ to denote the maximum degree of graph G, suppressing G if understood from the context.

2.1 Multi-neighborhood Search

Usually for finding a good 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: `add`, `swap` and `drop`, which guide the local search to move in the clique space. In [8], two sets S_{add} and S_{swap} were defined as below which ensures that the clique property is preserved.

$$S_{add} = \begin{cases} \{v | v \notin C, v \in N(u) \text{ for all } u \in C\} & \text{if } |C| > 0; \\ \emptyset, \text{ otherwise,} \end{cases} \tag{2}$$

$$S_{swap} = \begin{cases} \{(u, v) | u \in C, v \notin C, \{u, v\} \notin E, v \in N(w) \text{ for all } w \in C \backslash \{u\}\} & \text{if } |C| > 1; \\ \emptyset, \text{ otherwise,} \end{cases} \tag{3}$$

[5] https://github.com/math6068/LS-DTS

These definitions have a nice property as below, which make them desirable in solving large sparse graphs.

Proposition 1 *$|S_{add}| \leq d_{\max}$, and $|S_{swap}| \leq 2d_{\max}$.*

In large sparse graphs d_{max} is always small, so this proposition shows that S_{add} and S_{swap} are always small. Therefore the complexity of best-picking over these two sets are guaranteed to be low.

2.2 Scoring Function

Let S be a set of vertices[6], then we define $w(S)$ as

$$\sum_{v_i,v_j \in S \text{ and } \{v_i,v_j\} \in E} w(\{v_i, v_j\}). \tag{4}$$

We use $score(v, S)$ to denote the increase of $w(S)$ when v is added into or dropped from S as below

$$score(v, S) = \begin{cases} w(S \cup \{v\}) - w(S) \text{ if } v \notin S; \\ w(S \backslash \{v\}) - w(S) \quad \text{if } v \in S. \end{cases} \tag{5}$$

Then we use $score(u, v, S)$ to denote the increase of $w(S)$ when u and v are swapped, that is,

$$score(u, v, S) = w(S \backslash \{u\} \cup \{v\}) - w(S), \tag{6}$$

where $u \in S$, $v \notin S$ and $\{u, v\} \notin E$.

Notice that the score values may be negative. In our solver we will use the score value to measure the benefits of a local move. For efficiency, we maintain the score values of adding and dropping with the proposition below.

Proposition 2

1. *$score(u, \emptyset) = 0$ for all $u \in V$;*
2. *$score(v, S \backslash \{w\}) = score(v, S) + w(\{v, w\})$ for all $v \in (N(w) \cap S)$;*
3. *$score(v, S \cup \{w\}) = score(v, S) - w(\{v, w\})$ for all $v \in (N(w) \cap S)$;*
4. *$score(v, S \backslash \{w\}) = score(v, S) - w(\{v, w\})$ for all $v \in (N(w) \backslash S)$;*
5. *$score(v, S \cup \{w\}) = score(v, S) + w(\{v, w\})$ for all $v \in (N(w) \backslash S)$.*

Then we compute the score of swapping with the proposition below.

Proposition 3 *$score(u, v, S) = score(u, S) + score(v, S)$.*

A vertex has two possible states: inside and outside the candidate solution. We use $age(v)$ to denote the number of steps since last time v changed its state.

[6] S does not need to be a clique.

2.3 The Strong Configuration Checking Strategy

Recently, [5] proposed a strategy called configuration checking (CC), which exploits the problem structure to reduce cycling in local search. Roughly speaking, for combinatorial problems whose tasks are to find an optimal set of elements, the idea of CC can be described as follows. For an element (such as a vertex), if its local environment remains the same as the last time it was removed out of the candidate set, then it is forbidden to be added back into the candidate set. Typically, the local environment of a vertex refers to the state of its neighboring vertices.

The CC strategy is usually implemented with an array named $confChange$, where $confChange(v) = 1$ means v's local environment has changed since last time it was removed, and $confChange(v) = 0$ otherwise.

Later [20] 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 back into C only if one of its neighbors is added into C. More specifically the SCC strategy is specified as the following rules.

1. Initially $confChange(v)$ is set to 1 for each vertex v;
2. When v is added, $confChange(n)$ is set to 1 for all $n \in N(v)$;
3. When v is dropped, $confChange(v)$ is set to 0;
4. When $(u, v) \in S_{swap}$ are swapped, $confChange(u)$ is set to 0.

3 DTS for Selecting Edges

Usually the edge weights can vary considerably among each other, so if an edge has a great weight, it is likely to be contained in a good clique. By selecting such an edge, we give higher priority to visit a clique containing it. This provides a promising starting point for the later local search procedure.

In this work we proposed a strategy which is specialized for edge weights. It is based on the well-known deterministic tournament selection which is widely used in genetic algorithms [16]. Given a set S and a positive integer k, the DTS heuristic works as follows: *randomly select k elements from S with replacements and then return the best one.*

We formalize the DTS for selecting edges in Algorithm 1 as below.

The DTS heuristic has some advantages: (1) it is greedy because it approximates the best-picking heuristic well; (2) it provides some diversification in that it chooses an element among some very good ones; (3) the greediness and the randomness can easily be controlled by a parameter.

Actually the parameter k controls the greediness, *i.e.*, a greater k means more greediness while a smaller k means less greediness.

Algorithm 1: DTS

input : an edge set E, a positive integer k
output: a vertex v

```
1 S ← ∅ ;
2 e* ← a random edge from E;
3 for iteration ← 2 to k do
4 |   e ← a random edge from E;
5 |   if w(e) > w(e*) then e* ← e;
6 v ← a random vertex in e*;
7 return v;
```

4 The LS-DTS Algorithm

The top level algorithm of LS-DTS is shown in Algorithm 2, where the `localMove()` procedure is shown in Algorithm 3. For simplicity, in Algorithm 3 we write $score(v)$ in short for $score(v, C)$, and $score(u, v)$ in short for $score(u, v, C)$.

Algorithm 2: LS-DTS

input : A graph $G = (V, E, w_{\mathcal{E}})$ and the $cutoff$
output: The best clique that was found

```
1 C ← ∅; C* ← ∅; step ← 1; confChange(v) ← 1 for all v ∈ V;
2 while elapsed time < cutoff do localMove() ;
3 return C*;
```

In each local move, LS-DTS selects a neighboring clique with the greatest weight according to the SCC criterion. Every L steps, the search is restarted.

The details of `localMove()` are shown in Algorithm 3. In Algorithm 3, LS-DTS adopts the multi-neighborhood greedy search from MN/TS [21] which is shown between Line 3 and Line 11. In this greedy search procedure, as LSCC, LS-DTS exploits the Strong Configuration Checking (SCC) strategy [19] in place of the randomized tabu strategy in MN/TS. The SCC strategy, which is a dynamic tabu management strategy, exploits local environment information to determine in which condition a forbidden operation will become allowed.

When C becomes an empty clique in Line 1, LS-DTS starts the local search from a vertex return by `DTS()`. Furthermore, Line 14 ensures that the search will restart every L steps. Anyway, LS-DTS sets L to be 4,000 just as what MN/TS, LSCC and LMY-GRS do.

Lastly we remind readers that we adopted the data structures in LMY-GRS [8] to implement our solver.

5 Experimental Evaluation

In this section, we carry out extensive experiments to evaluate LS-DTS on a wide range of large sparse graphs.

Algorithm 3: LocalMove

```
1  if C = ∅ then
2      v ← DTS(); add v into C;
3  v ← a vertex in S_add s.t. confChange(v) = 1 with the biggest score(v), breaking ties in
     favor of the oldest one; otherwise v ← NULL;
4  (u, u′) ← a pair in the S_swap s.t. confChange(u′) = 1 with the biggest score(u, u′),
     breaking ties in favor of the oldest u′; otherwise (u, u′) ← (NULL, NULL);
5  if v ≠ NULL then
6      if (u, u′) = (NULL, NULL) or score(v) > score(u, u′) then C ← C ∪ {v} ;
7      else C ← C\{u} ∪ {u′} ;
8  else
9      v ← a vertex in C with the biggest score, breaking ties in favor of the oldest one;
10     if (u, u′) = (NULL, NULL) or score(v) > score(u, u′) then C ← C\{v} ;
11     else C ← C\{u} ∪ {u′} ;
12 step ← step + 1;
13 if w(C) > w(C*) then C* ← C;
14 if step mod L = 0 then
15     drop all vertices in C;
16     step ← step + 1;
17 update confChange array according to SCC rules;
18 if w(C) > w(C*) then C* ← C;
```

5.1 The Competitors

So far as we know the most prominent local search solver for the MEWC problem is PLS [17] which extends the Phased Local Search algorithm to MEWC. It solves the MVWC and the MEWC problem in very similar ways.

Considering that there has been great progress in MVWC solving, *e.g.*, the multi-neighborhood greedy search and the strong configuration checking strategy, the approach in [17] falls behind. So in this paper, we adapted two recent local search solvers LSCC [7] and LMY-GRS[8] to solve the MEWC problem, since they represent state-of-the-art.

5.2 Experiment setup

All the solvers in this work were implemented in C++, and compiled by g++ 4.6.3 with the '-O3' option. The experiments were conducted on a cluster equipped with Intel Xeon E5-2670 v3 2.3GHz with 32GB RAM, running CentOS6.

For all the solvers the search depth L was set to 4,000. In LSCC+BMS, the BMS parameter k was set to 100 as is in [20]. For LS-DTS, the DTS parameter k was fixed to 50 for all the experiments[9].

Each solver is executed on each instance with a time limit of 1,000 seconds, with seeds from 1 to 10. For each algorithm on each instance, we report the maximum edge weight "w_{max}") and averaged edge weight ("w_{avg}") of the cliques found by the algorithm.

[7] https://github.com/math6068/MEWC-LSCC-BMP

[8] https://github.com/math6068/LMY-GRS

[9] Different values of k are only used to test parameter sensitivity, which will be discussed in section 5.5.

5.3 Details of Benchmarks

We downloaded all 139 instances[10], which were originally online[11], and then transformed to DIMACS graph format.

In many of these large graphs there are millions of vertices and dozens of millions of edges. To obtain the corresponding MEWC instances, we use the same method as in [17]. For the edge $\{i, j\}$, $w(\{u, v\}) = ((i + j) \mod 200) + 1$.

The graphs used in our experiments can be divided into 11 classes: biological networks, collaboration networks, facebook networks, interaction networks, infrastructure networks, amazon recommend networks, retweet networks, scientific computation networks, social networks, technological networks, web link networks. There is also a group of temporal reachability networks, where the graphs are small and the algorithms quickly found the same quality solution on all the graphs. Hence, the result in this group is not reported in our experiment.

5.4 Main Results

The main experimental results are shown in Table 1. From this table we find that,

1. LS-DTS significantly outperforms all other solvers in terms of the solution quality.
2. Compared to LMY-GRS, LS-DTS finds better and worse solutions in 11 and 6 graphs respectively .

Since LS-DTS is based on LMY-GRS, we further compare LS-DTS and LMY-GRS in the following.

Time and Step Improvements For the 85 instances where LS-DTS and LMY-GRS return both the same w_{max} and w_{avg} values, we compare the averaged time, as well as and the number of steps to locate the respective solutions. From Table 2 we observe that:

1. The time columns show that LS-DTS is faster than LMY-GRS on most of these instances.
2. The step columns illustrate that our *heuristic* is more clever than LMY-GRS on most graphs, in that it needs significantly less steps to locate the solutions.

Further observations show that on each graph in Table 2, LS-DTS found the same quality solution in all runs, so as LMY-GRS. This shows that the two solvers are insensitive to seeds over these graphs.

The step columns also show the superiority of our strategy, because the number of steps needed to locate a solution only relies on the strategy. It is irrelevant to the running environment, the data structures as well as the programming techniques.

[10] http://lcs.ios.ac.cn/∼caisw/Resource/realworld%20graphs.tar.gz
[11] http://www.graphrepository.com/networks.php

Table 1. Experimental results on large sparse graphs where the four algorithms find different w_{max} or w_{avg} values. We bold the better values in shaded cells.

Graph	LSCC $w_{max}(w_{avg})$	LSCC+BMS $w_{max}(w_{avg})$	LMY-GRS $w_{max}(w_{avg})$	LS-DTS $w_{max}(w_{avg})$
ca-citeseer	451135(254584.1)	451135(322432.0)	451135(**451135.0**)	451135(**451135.0**)
ca-coauthors-dblp	5661008(5359200.8)	5661008(5109263.7)	5661008(**5661008.0**)	5661008(**5661008.0**)
ca-dblp-2010	276575(221523.5)	276575(206365.0)	276575(**276575.0**)	276575(**276575.0**)
ca-hollywood-2009	245095624(229025126.6)	245095624(**245095624.0**)	245095624(166832908.6)	245095624(226282461.2)
ca-MathSciNet	32364(20300.9)	25393(18195.9)	32364(**32364.0**)	32364(**32364.0**)
inf-roadNet-CA	597(597.0)	597(597.0)	1050(**1050.0**)	1050(**1050.0**)
inf-roadNet-PA	993(993.0)	993(993.0)	1164(**1164.0**)	1164(**1164.0**)
inf-road-usa	567(567.0)	567(558.0)	1092(**1092.0**)	1092(**1092.0**)
rec-amazon	1686(1531.6)	1686(1566.4)	1866(**1866.0**)	1866(**1866.0**)
rt-retweet-crawl	4020(4020.0)	8262(6989.4)	8262(**7413.6**)	8262(6989.4)
sc-ldoor	38990(37260.0)	39570(37846.0)	40610(**40610.0**)	40610(**40610.0**)
sc-msdoor	39550(38236.0)	39550(38314.0)	40250(**40250.0**)	40250(**40250.0**)
sc-nasasrb	51040(50657.6)	51040(50657.6)	51040(**51040.0**)	51040(**51040.0**)
sc-pkustk11	77580(69262.0)	77580(73692.0)	77580(**77580.0**)	77580(**77580.0**)
sc-pkustk13	99915(95501.5)	94080(93468.0)	99915(94393.5)	99915(**96825.0**)
sc-pwtk	51888(50281.2)	51888(50522.8)	51888(**51888.0**)	51888(**51888.0**)
sc-shipsec1	45126(33624.7)	45126(36190.1)	45126(**45126.0**)	45126(**45126.0**)
sc-shipsec5	48576(47223.6)	48576(47335.4)	48576(**48576.0**)	48576(**48576.0**)
soc-digg	123757(94136.7)	123757(102589.9)	123757(111055.5)	123757(**115291.4**)
soc-flixster	47685(**47685.0**)	47685(**47685.0**)	47685(47677.0)	47685(47640.0)
soc-FourSquare	45982(**45982.0**)	45982(**45982.0**)	45982(43121.2)	45982(43539.8)
soc-gowalla	30226(**24149.2**)	22630(22630.0)	22630(22590.0)	30226(23389.6)
soc-lastfm	11266(10412.7)	10047(10047.0)	11266(**10887.3**)	11266(10656.5)
soc-livejournal	81460(28406.8)	81460(32161.4)	2289993(1222371.1)	2289993(**1958073.9**)
soc-orkut	96682(45829.9)	74911(46446.7)	72188(50334.5)	96682(**55740.6**)
soc-pokec	25603(17587.3)	19959(17794.4)	**38202**(25734.1)	30121(24361.6)
socfb-A-anon	25615(20483.1)	32532(20994.7)	32532(**28129.3**)	32532(**28129.3**)
socfb-B-anon	22441(18330.7)	22441(18198.5)	28384(23161.7)	28384(**23308.6**)
socfb-MIT	54696(**54696.0**)	54696(**54696.0**)	54696(54677.6)	54696(**54696.0**)
socfb-Penn94	93079(**93079.0**)	93079(**93079.0**)	93079(92045.1)	93079(**93079.0**)
socfb-Stanford3	131675(**131675.0**)	131675(**131675.0**)	131675(**131675.0**)	131675(131665.0)
socfb-UF	149419(149419.0)	149419(149419.0)	149419(148600.5)	149419(149419.0)
tech-as-skitter	179915(158512.6)	179915(162907.7)	179915(**166687.1**)	179915(**166687.1**)
tech-p2p-gnutella	903(886.5)	903(888.9)	903(**903.0**)	903(**903.0**)
web-it-2004	8035767(4233906.9)	9282115(5332859.0)	9308691(**9308691.0**)	9308691(**9308691.0**)
web-sk-2005	401124(339784.8)	401124(390852.0)	401124(**401124.0**)	401124(**401124.0**)
web-uk-2005	12572200(12559868.4)	12572200(12146305.8)	12572200(**12572200.0**)	12572200(**12572200.0**)
web-wikipedia2009	46785(29626.8)	46785(32679.2)	46832(**46832.0**)	46832(46827.3)

Robustness From Tables 1 to 2, among all the 11 classes of graphs,

1. LS-DTS is superior in 7 classes.
2. LMY-GRS is better in 4 classes.

So LS-DTS is more robust than LMY-GRS.

5.5 Parameter Sensitivity

We tested LS-DTS with two different parameter setting for k, *i.e.*, $k = 20$ and $k = 80$. The results are shown in Table 3. We use #win(w_{max}) to denote number of graphs where the new setting finds better w_{max} than the defaulting setting (k=50). Similarly we use #lose(w_{max}), #win(w_{avg}), #lose(w_{avg}) and #draw.

From Table 3 we find that the two variants perform very close to the default solver. That is, our solver is insensitive to the parameter k.

Table 2. Comparative performances on the instances where LMY-GRS and LS-DTS return the same w_{max} and w_{avg} values. We bold the better values in shaded cells.

Graph	time		#step	
	LMY-GRS	LS-DTS	LMY-GRS	LS-DTS
bio-celegans	0.336	**0.167**	17620.4	**8820.5**
bio-diseasome	<0.001	<0.001	97.1	**94.1**
bio-dmela	**0.039**	0.043	**1974.5**	2140.1
bio-yeast	<0.001	<0.001	626.5	**617**
ca-AstroPh	28.262	**20.192**	404137	**347673**
ca-citeseer	46.753	**4.566**	3255350	**321791**
ca-coauthors-dblp	115.205	**75.297**	663396	**316425**
ca-CondMat	**9.737**	10.588	**593689**	652068
ca-CSphd	**0.001**	0.002	**1737.4**	1760.7
ca-dblp-2010	16.256	**5.151**	810286	**265328**
ca-dblp-2012	5.899	**2.848**	258712	**118697**
ca-Erdos992	**0.001**	0.003	108.6	**83.6**
ca-GrQc	0.01	**0.009**	**1364.8**	2545.2
ca-HepPh	0.258	**0.139**	5611.6	**1554**
ca-MathSciNet	**201.822**	201.919	11162200	**7602890**
ca-netscience	0.036	**0.023**	30856.3	**18025.8**
ia-email-EU	**0.115**	0.119	957.6	**587.7**
ia-email-univ	0.005	**0.002**	500.5	**484.6**
ia-enron-large	**24.663**	30.129	**47745.9**	53325.3
ia-enron-only	**0.028**	0.030	**7216.7**	8418.7
ia-fb-messages	0.008	**0.006**	**106.3**	116
ia-infect-dublin	0.080	**0.024**	13278.9	**3689.5**
ia-infect-hyper	**0.118**	0.208	**10035.8**	19232
ia-reality	0.005	0.005	274.6	**223.2**
ia-wiki-Talk	**0.729**	1.442	**1084.9**	1406
inf-power	0.010	**0.004**	2267.5	**2035.8**
inf-roadNet-CA	**3.939**	6.724	**1912020**	2810280
inf-roadNet-PA	1.244	**1.234**	536135	**525639**
inf-road-usa	92.853	**39.837**	26994100	**11596300**
rec-amazon	**0.075**	0.111	**38215.2**	60678.7
rt-retweet	<0.001	<0.001	**83.6**	121.5
rt-twitter-copen	<0.001	<0.001	74.4	**72.8**
sc-msdoor	261.813	**252.347**	31414800	**28946000**
sc-nasasrb	**7.633**	8.643	**808128**	972502
sc-pkustk11	140.514	**83.618**	10601300	**5037680**
sc-pwtk	55.054	**36.550**	5729330	**4299690**
sc-shipsec1	20.841	**17.613**	2784070	**1876090**
sc-shipsec5	78.511	**18.981**	11293200	**2702440**
soc-BlogCatalog	**120.191**	205.804	**7770.2**	14141.4
soc-brightkite	394.281	**143.673**	1729650	**600864**
soc-buzznet	**50.438**	171.688	**3340.5**	10203.8
soc-delicious	**25.520**	35.564	**28990.5**	38567
soc-dolphins	<0.001	<0.001	19.1	**16.5**
soc-douban	0.339	**0.198**	5645.7	**2261.5**
soc-epinions	53.221	**27.777**	479236	**259627**
soc-flickr	55.894	**16.425**	22618.9	**4291.7**
soc-karate	<0.001	<0.001	15.6	**8.3**
soc-LiveMocha	0.381	**0.360**	**198.8**	205.5
soc-slashdot	**0.277**	0.294	**529.6**	900.4
soc-twitter-follows	5.489	**5.323**	**17262.5**	18688.1
soc-wiki-Vote	<0.001	<0.001	**105.7**	121
soc-youtube	11.455	**7.652**	5293.5	**3680.2**
soc-youtube-snap	**84.058**	132.746	**8979.9**	12976.7
socfb-A-anon	453.944	**347.777**	461488	**325436**
socfb-Berkeley13	189.775	**135.750**	373326	**276905**
socfb-CMU	21.011	**16.014**	69727.4	**51317.4**
socfb-Duke14	**13.409**	16.866	38746.7	**37527.9**
socfb-Indiana	73.643	**63.928**	235903	**149565**
socfb-OR	48.231	**45.355**	**207749**	223313
socfb-Texas84	**116.618**	176.882	**186177**	244536
socfb-uci-uni	**86.270**	163.999	**66388.8**	136236
socfb-UCLA	114.036	**78.595**	337721	**209763**
socfb-UConn	28.706	**15.912**	127718	**76159.6**
socfb-UCSB37	**57.321**	57.451	266970	**183756**
socfb-UIllinois	153.354	**147.381**	554564	**455851**
socfb-Wisconsin87	96.978	**73.474**	395674	**214891**
tech-as-caida2007	**0.032**	0.040	**38.6**	47.5
tech-as-skitter	265.709	**124.921**	52609.7	**19752.8**
tech-internet-as	0.973	**0.393**	2436.3	**1234.8**
tech-p2p-gnutella	0.122	**0.102**	14829.6	**12201.8**
tech-RL-caida	3.770	**3.461**	**24900.2**	25023.5
tech-routers-rf	0.092	**0.071**	6860.2	**5683.9**
tech-WHOIS	2.169	**0.962**	14634.5	**7375.6**
web-arabic-2005	0.243	**0.179**	**11523.7**	18335.8
web-BerkStan	0.005	0.005	**1171.6**	1687.4
web-edu	0.050	**0.018**	3550.7	**1119.5**
web-google	0.001	0.001	**931.1**	1307.2
web-indochina-2004	0.130	**0.020**	9037	**1054.3**
web-it-2004	12.003	**4.754**	304937	**69621**
web-polblogs	<0.001	<0.001	139.9	**131.5**
web-sk-2005	5.825	**0.580**	1096210	**157384**
web-spam	72.264	**32.147**	776421	**328820**
web-uk-2005	4.455	**2.880**	155745	**72339.4**
web-webbase-2001	7.702	**1.248**	61332.5	**8160**
web-wikipedia2009	**361.855**	362.694	1365900	**1174920**

Table 3. Experimental results on different values of k in DTS, compare to the performances when k is set to 50

k	#win(w_{max})	#lose(w_{max})	#win(w_{avg})	#lose(w_{avg})	#draw
20	0	0	7	8	87
80	0	1	6	7	88

6 Conclusions

In this work, we proposed a new algorithm named LS-DTS for the MEWC problem on large sparse graphs. Also we adapted two recent local search solvers LSCC and LMY-GRS to solve the MEWC problem. Experiments on a broad range of large sparse graphs demonstrate the effectiveness of LS-DTS.

As for future works we would like to design more efficient heuristics for the MEWC problem on large sparse graphs and exploit our solver to tackle industrial graphs.

Acknowledgments

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.

This research was supported by use of the NeCTAR Research Cloud and by QCIF(http://www.qcif.edu.au). The NeCTAR Research Cloud is a collaborative Australian research platform supported by the National Collaborative Research Infrastructure Strategy.

References

1. Adamczewski, K., Suh, Y., Lee, K.M.: Discrete tabu search for graph matching. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. pp. 109–117 (2015)
2. Adluru, N., Yang, X., Latecki, L.J.: Sequential monte carlo for maximum weight subgraphs with application to solving image jigsaw puzzles. International Journal of Computer Vision 112(3), 319–341 (2015)
3. Alidaee, B., Glover, F., Kochenberger, G., Wang, H.: Solving the maximum edge weight clique problem via unconstrained quadratic programming. European Journal of Operational Research 181(2), 592 – 597 (2007)
4. Cai, S., Lin, J.: Fast solving maximum weight clique problem in massive graphs. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016. pp. 568–574 (2016)
5. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artif. Intell. 175(9-10), 1672–1696 (2011)
6. Czajkowska, J., Feinen, C., Grzegorzek, M., Raspe, M., Wickenhfer, R.: Skeleton graph matching vs. maximum weight cliques aorta registration techniques. Computerized Medical Imaging and Graphics 46, Part 2, 142 – 152 (2015), information Technologies in Biomedicine

7. Deng, Z., Todorovic, S., Latecki, L.J.: Unsupervised object region proposals for RGB-D indoor scenes. Computer Vision and Image Understanding 154, 127–136 (2017)
8. Fan, Y., Li, C., Ma, Z., Wen, L., Sattar, A., Su, K.: Local search for maximum vertex weight clique on large sparse graphs with efficient data structures. In: AI 2016: Advances in Artificial Intelligence - 29th Australasian Joint Conference, Hobart, TAS, Australia, December 5-8, 2016, Proceedings. pp. 255–267 (2016)
9. Fang, Z., Li, C., Qiao, K., Feng, X., Xu, K.: Solving maximum weight clique using maximum satisfiability reasoning. In: ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014). pp. 303–308 (2014)
10. Fang, Z., Li, C., Xu, K.: An exact algorithm based on maxsat reasoning for the maximum weight clique problem. J. Artif. Intell. Res. (JAIR) 55, 799–833 (2016), http://dx.doi.org/10.1613/jair.4953
11. Feige, U.: Approximating maximum clique by removing subgraphs. SIAM J. Discrete Math. 18(2), 219–225 (2004)
12. Gouveia, L., Martins, P.: Solving the maximum edge-weight clique problem in sparse graphs with compact formulations. EURO Journal on Computational Optimization 3(1), 1–30 (2015)
13. Jiang, H., Li, C., Manyà, F.: An exact algorithm for the maximum weight clique problem in large graphs. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. pp. 830–838 (2017)
14. Ma, T., Latecki, L.J.: Maximum weight cliques with mutex constraints for video object segmentation. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012. pp. 670–677 (2012)
15. Mascia, F., Cilia, E., Brunato, M., Passerini, A.: Predicting structural and functional sites in proteins by searching for maximum-weight cliques. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010 (2010)
16. Miller, B.L., Goldberg, D.E.: Genetic algorithms, tournament selection, and the effects of noise. Complex systems 9(3), 193–212 (1995)
17. Pullan, W.J.: Approximating the maximum vertex/edge weighted clique using local search. J. Heuristics 14(2), 117–134 (2008)
18. Traud, A.L., Mucha, P.J., Porter, M.A.: Social structure of facebook networks. Physica A: Statistical Mechanics and its Applications 391(16), 4165–4180 (2012)
19. Wang, C., Jonckheere, E., Brun, T.: Differential geometric treewidth estimation in adiabatic quantum computation. Quantum Information Processing 15(10), 3951–3966 (2016)
20. Wang, Y., Cai, S., Yin, M.: Two efficient local search algorithms for maximum weight clique problem. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 805–811 (2016)
21. Wu, Q., Hao, J., Glover, F.: Multi-neighborhood tabu search for the maximum weight clique problem. Annals OR 196(1), 611–634 (2012), http://dx.doi.org/10.1007/s10479-012-1124-3