Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Two Efficient Local Search Algorithms for Maximum Weight Clique Problem

Yiyuan Wang[1,3], **Shaowei Cai**[2], **and Minghao Yin**[1,3*]
[1]School of Computer Science and Information Technology, Northeast Normal University, Changchun, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[3]Symbol Computation and Knowledge Engineer of Ministry of Education, Jilin University, Changchun, China
yiyuanwangjlu@126.com; caisw@ios.ac.cn; ymh@nenu.edu.cn

Abstract

The Maximum Weight Clique problem (MWCP) is an important generalization of the Maximum Clique problem with wide applications. This paper introduces two heuristics and develops two local search algorithms for MWCP. Firstly, we propose a heuristic called strong configuration checking (SCC), which is a new variant of a recent powerful strategy called configuration checking (CC) for reducing cycling in local search. Based on the SCC strategy, we develop a local search algorithm named LSCC. Moreover, to improve the performance on massive graphs, we apply a low-complexity heuristic called Best from Multiple Selection (BMS) to select the swapping vertex pair quickly and effectively. The BMS heuristic is used to improve LSCC, resulting in the LSCC+BMS algorithm. Experiments show that the proposed algorithms outperform the state-of-the-art local search algorithm MN/TS and its improved version MN/TS+BMS on the standard benchmarks namely DIMACS and BHOSLIB, as well as a wide range of real world massive graphs.

Introduction

Given an undirected graph G=(V,E), a clique C of G is a subset of V such that each pair of vertices in C is mutually adjacent. The maximum clique problem (MCP) consists in finding a clique with maximum number of vertices. An important generalization of MCP is the maximum weight clique problem (MWCP), in which each vertex is associated with a non-negative integer, and the goal is to find a clique with the largest total weight. Obviously, MWCP reduces to MCP if each vertex has the same weight. MWCP has been widely used in many fields, from theoretical computer science to valuable applications (Ballard and Brown 1982; Balasundaram and Butenko 2006; Gomez Ravetti and Moscato 2008).

As is known, the decision version of MCP is one of Karp's prominent 21 NP-complete combinatorial problems (Karp 1972). Both MCP and MWCP has been proved to be NP-hard and state-of-the-art approximation algorithms can only achieve an approximate ratio of O(n(log log n)^2/(log n)^3) (Feige 2004). Thus it is common to see that a huge amount of effort has been devoted to finding a "good" clique within reasonable time. Up to now, there are mainly two types of algorithms for MCP and MWCP, i.e. exact algorithms and heuristic algorithms.

A number of exact algorithms have been proposed to solve MCP and MWCP. A classic branch and bound algorithm is MCQ (Tomita and Seki 2003), which uses a heuristic vertex order for independent set partition. The MCQ algorithm is further improved by computing the degree of vertices dynamically, resulting in the MaxCliqueDyn algorithm (Konc and Janezic 2007). Recently, another paradigm encodes MCP into MaxSAT and then applies MaxSAT reasoning to improve the upper bound (Li and Quan 2010; Li, Fang, and Xu 2013). For MWCP, an early branch and bound algorithm was proposed in (Babel 1994). An improved branch and bound algorithm based on error-correcting codes was offered in (Östergård 2001), and the algorithm in (Yamaguchi and Masuda 2008) computes the upper bound based on the longest path in a directed acyclic graph constructed from the original graph. Very recently, Fang et. al. proposes a MaxSAT-based algorithm for MWCP, and applies Top-k failed literal detection for improving the upper bound (Fang et al. 2014).

Although exact algorithms can guarantee the optimality of their solutions, they may fail to solve hard instances of large scale. For solving large sized instances, a popular approach is local search, which can find an approximate solution within reasonable time. There are numerous local search algorithms for MCP (Singh and Gupta 2006; Pullan and Hoos 2006; Pullan 2006; Guturu and Dantu 2008; Wu and Hao 2013; Benlic and Hao 2013). Among these algorithms, DLS (Pullan and Hoos 2006) is a milestone algorithm which employs vertex penalties which are dynamically adjusted during the search. DLS is further improved into a two phase algorithm called Phased Local Search (PLS) (Pullan 2006). In (Wu and Hao 2013), the proposed tabu search algorithm is presented based on k-fixed penalty strategy. (Benlic and Hao 2013) introduces a break out local search for solving MCP. Also, MCP is closely related to minimum vertex cover (MinVC) and maximum independent set (MaxIS) problems, and algorithms for these two problems can be directly used to solve MCP.

Compared to MCP, there are relatively fewer heuristics on MWCP. The reason may be that MWCP is more complicated and thus difficult to solve, from the viewpoint of al-

*Corresponding author

gorithm design. In (Bomze, Pelillo, and Stix 2000), a parallel, distributed heuristic for approximating the MWCP based on dynamics principles is developed and studied in various branches of mathematical biology. Busygin (Busygin 2006) presents a new fast heuristic method using a nonlinear programming formulation for the MWCP. Pullan extends the Phased Local Search (PLS) algorithm to MWCP (Pullan 2008). According to the literatures, the current best local search algorithm for MWCP is called MN/NT (Wu, Hao, and Glover 2012), which is a multi-neighborhood local search algorithm whose key features include a combined neighborhood and a dedicated tabu mechanism.

In this paper, we develop two local search algorithms for MWCP. Firstly, we propose a new heuristic, which is a variant of the configuration checking (CC) strategy. CC is a recently proposed mechanism to avoid the cycling problem during local search, and has been successfully applied in a number of NP-hard problem, such as MinVC (Cai, Su, and Sattar 2011), SAT (Cai and Su 2012; 2013; André, Djamal, and Donia 2014), and MaxSAT (Luo et al. 2015). We follow this line of research by attempting to apply CC strategy to solve MWCP. However, a direct application of CC strategy does not lead to a successful algorithm, because the forbidding strength of CC is usually too weak in the context of MWCP. We propose a new strategy called Strong CC strategy (SCC for short), which is stricter than CC and reduces more unnecessary search areas. Based on SCC, we develop a local search algorithm called LSCC (Local search with SCC). Experiments comparing LSCC with a state-of-the-art local search algorithm MN/TS show its superiority on standard benchmarks DIMACS (Johnson and Trick 1996) and BHOSLIB (Xu et al. 2005).

Moreover, to improve the performance on massive graphs, we apply a low-complexity heuristic called Best from Multiple Selection (BMS) to select the swapping vertex pair quickly and effectively. A very recent work (Cai 2015) proposed a simple and fast local search algorithm called FastVC for solving MinVC in massive graphs, which is based on two low-complexity heuristics. Inspired by the success of BMS in FastVC (Cai 2015), we also use the BMS heuristic, which approximates the best-greedy swap heuristic (Wu, Hao, and Glover 2012) and has a lower complexity. We also enhance the BMS heuristic with the SCC strategy. Using the BMS heuristic, we improve LSCC and the resulting algorithm is called LSCC+BMS, and also improve MN/TS and obtain MN/TS+BMS. Experiments show that LSCC+BMS outperforms MN/NT and its improved version MN/TS+BMS on a broad range of massive real graphs (Rossi and Ahmed 2015). We also conduct experiments to analyze the effectiveness of the two proposed heuristics.

In the next section, we introduce some necessary background knowledge. After that, we propose the SCC strategy for MWCP and present the LSCC algorithm, along with related experiments. Then, we improve LSCC on massive graphs with the BMS heuristic and obtain the LSCC+BMS algorithm, along with experiments on massive graphs. Finally we make conclusions and outline the future work.

Preliminary

Given an undirected graph G=(V,E) where V=$\{v_1, v_2, \ldots, v_n\}$ is the set of vertices and E=$\{e_1, e_2, \ldots, e_m\}$ is the set of edges. In graph G, each edge is a 2-element subset of V. For an edge e=$\{v,u\}$, we say that vertices u and v are the endpoints of edge, and u is adjacent to v. A clique C of G is a subset of V where each pair of vertices is adjacent. The MCP problem is to find a clique with the most vertices. When each vertex v_i is associated with a positive integer weight, MCP is extended to MWCP which asks for a clique of the maximum total weight. Given a weighting function w: $V \rightarrow Z^+$, the weight of a clique C is $w(C)$=$\sum_{v \in C} w(v)$. The neighborhood of a vertex v is $N(v)$=$\{u \in V | (v,u) \in E\}$. For a vertex v, its age is defined as the number of steps since the last time it changed its state (being selected or not).

Typically, local search algorithms for MWCP (as in MCP) maintain a current clique C, and modify it iteratively by three operators: Add, Drop and Swap. The operator Add refers to adding a vertex into the clique C, providing that the vertex is adjacent to all vertices in C. The operator Drop refers to removing a vertex from C. The operator Swap exchanges one vertex $u \in C$ with another vertex $v \notin C$ which is adjacent to every vertex in C but u. Usually, the operator Drop is considered only when Add and "good" Swap operations are impossible.

Review of Configuration Checking

Revisiting the same part of a search space is referred as the cycling problem, which is a severe issue in local search. Recently, Cai et. al proposed a strategy called configuration checking (CC) (Cai, Su, and Sattar 2011), which exploits the problem structure to reduce cycling in local search. The CC strategy has been successfully used in local search algorithms for combinatorial optimization problems such as MinVC (Cai, Su, and Sattar 2011) and Set Covering (Wang et al. 2015), as well as constraint satisfaction problems such as Satisfiability (Cai and Su 2013; André, Djamal, and Donia 2014) and Maximum Satisfiability (Luo et al. 2015).

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 configuration 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 configuration of a vertex refers to the state of its neighbouring vertices. The CC strategy is usually implemented with a Boolean array named $confChange$, where $confChange(v)$=1 means v is allowed to be added to the candidate solution and $confChange(v)$=0 means v is forbidden to be added to the candidate solution.

A straightforward CC strategy for MWCP can be easily devised. In the beginning, $confChange(v)$ is initialized as 1 for each vertex v, as each vertex is allowed to be selected initially. During the search, when a vertex v is added to the current clique, $confChange(v')$ is set to 1 for each vertex $v' \in N(v)$. When a vertex v is removed out of the current clique, $confChange(v)$ is set to 0 and $confChange(v')$ is

set to 1 for each vertex $v' \in N(v)$. For a swap step, where a vertex v is removed from the current clique and vertex u is added into the clique, then $confChange(v)$ is set to 0 and $confChange(v')$ is set to 1 for $v' \in N(v) \cup N(u)$.

Strong Configuration Checking

In this section, we discuss the drawbacks of the CC strategy when it is applied to MWCP, and propose a new variant of CC for MWCP (also MCP), which is referred to as Strong Configuration Checking (SCC).

We observe that, in local search algorithms with the three operators Add, Drop and Swap, the CC strategy would mislead the search by allowing too many vertices to be added. According to CC, $confChange$ values of related vertices are updated along with each operation. However, some intuitive analyses suggest that it is not always advisable to set the neighbouring vertices' $confChange$ values to 1 upon each operation.

For the Add operation, the clique is being extended by a vertex, and thus it is quite reasonable to allow the selected vertex's neighbors to be added by setting their $confChange$ values to 1. Indeed, those vertices are very encouraged to be added into the clique.

The Drop operation indicates the algorithm meets a local optimum and is rolling back by removing a vertex from the clique. In this case, we believe the neighbouring vertices of the removed vertex should not be encouraged to be added to the clique.

The Swap operation usually serves as a form of diversification, by leading the search switch to another clique near the current one. Since we are not of certain that the Swap step is leading the search towards a better clique, in our algorithm, we adopt a conservative strategy — not encouraging more neighbouring vertices of the swapped vertices, but only those with $confChange$ value already being 1.

Based on the above considerations, we modify CC into a more restrictive version, which is called Strong Configuration Checking (SCC). This heuristic is specified by the following four rules.

SCC-InitialRule. In the beginning of the search procedure, $confChange(v)$ is set to 1 for each vertex v.

SCC-AddRule. When v is added into the current clique, $confChange(v')$ is set to 1 for each $v' \in N(v)$.

SCC-DropRule. When a vertex v is removed from the current clique, $confChange(v)$ is switched to 0.

SCC-SwapRule. When u is removed from the current clique and v is added into this clique, $confChange(u)$ is switched to 0.

In a nut shell, SCC only allows a vertex v to be added to the current clique when some of v's neighbors have been added since v's last removal, while CC allows the adding of v when some of v's neighbors have been either added or removed. The CC strategy usually works well with weighting techniques, so the missing of weighting techniques in our algorithm may be a reason for the failure of the original CC strategy. We also note that there is a concept called promising variable in SAT (Li and Huang 2005), which allows a variable to be flipped if its score becomes positive because of the flips of its neighboring variables. This concept is in some sense similar to CC strategies including SCC.

Algorithm 1: LSCC (G, *cutoff*)

Input: graph $G = (V, E, w)$, the *cutoff* time
Output: A maximum weight clique C of G

1 $C^* := \emptyset$
2 **while** *elapsed time* $<$ *cutoff* **do**
3 initialize $confChange$;
4 $C := InitGreedyConstruction()$;
5 $C_{localbest} := C$;
6 **for** $step = 0; step < L; step$++ **do**
7 $v :=$ a vertex in $AddSet$ with the biggest Δ_{add} and $confChange(v) = 1$, breaking ties in favor of the oldest one;
8 $(u, u') :=$ a vertex pair in $SwapSet$ such that $confChange(u') = 1$ with the biggest Δ_{swap}, breaking ties in favor of the oldest one;
9 **if** $AddSet \neq \emptyset$ **then**
10 $C := (\Delta_{add} > \Delta_{swap})? (C \cup \{v\}) : (C \setminus \{u\} \cup \{u'\})$;
11 **else**
12 $x :=$ a vertex in C with the biggest Δ_{drop}, breaking ties in favor of the oldest one;
13 $C := (\Delta_{drop} > \Delta_{swap})? (C \setminus \{x\}) : (C \setminus \{u\} \cup \{u'\})$;
14 update $confChange$ according to *SCC* rules;
15 **if** $w(C) > w(C_{localbest})$ **then** $C_{localbest} := C$;
16 **if** $w(C_{localbest}) > w(C^*)$ **then** $C^* := C_{localbest}$;
17 **return** C^*;

The LSCC Algorithm

Based on the SCC heuristic, we develop a local search algorithm named LSCC (Local search with SCC). LSCC works with the three operators Add, Swap and Drop. We maintain a set for the Add and Swap operators respectively. With the current clique denoted by C, the two sets are defined as follows. The set for the Drop operator is simply C.

$AddSet = \{v | v \notin C, v \in N(u) \text{ for } \forall u \in C\}$

$SwapSet = \{(u, v)) | u \in C, v \notin C, v \in N(y) \text{ for } \forall y \in C \setminus \{u\}\}$

We use Δ_{add}, Δ_{swap} and Δ_{drop} to denote the change on the value of $w(C)$ for operation Add, Drop, and Swap respectively. Obviously, we can calculate them according to the following equations.

- for a vertex $v \in AddSet$, $\Delta_{add}(v) = w(v)$;
- for a vertex $u \in C$, $\Delta_{drop}(u) = -w(u)$;
- for a vertex pair $(u, v) \in$*SwapSet*, $\Delta_{swap}(u, v) = w(v) - w(u)$.

In our algorithm, the vertices of the operations are explicit from the context and thus omitted.

The pseudo code of LSCC is outlined in Algorithm 1, as described below. In the beginning, LSCC initializes the best found maximum clique C^* as an empty set. There is an outer loop (lines 2-16) and an inner loop (lines 6-15). In each inner loop ($step < L$), LSCC searches for a local optimal clique

denoted as $C_{localbest}$. After each inner loop, if $w(C_{localbest})$ is larger than $w(C^*)$, C^* is updated by $C_{localbest}$ (line 16). Finally, LSCC returns C^* when the algorithm reaches a time limit.

Before each inner loop, LSCC constructs an initial candidate solution C greedily by iteratively selecting a vertex that is adjacent to all vertices in C until no such vertex exists, with ties broken randomly (line 4). The greedy initialization process is very simple and remains effective for massive graphs. Also, with the random tie-breaking mechanism, the procedure is able to find diversified initial solutions in different rounds. Then, $C_{localbest}$ is initialized as C (line 5).

In each inner loop, LSCC chooses one operator to modify the current clique C. It first selects a vertex $v \in AddSet$ with the biggest Δ_{add} and $confChange(v)$=1 (line 7), and selects a swapping pair $(u, u') \in SwapSet$ such that $confChange(u')$=1 with the biggest Δ_{swap} (line 8).[1] Both ties are broken by preferring the oldest one. If an Add operation is possible, LSCC compares Δ_{add} and Δ_{swap}, and chooses the operation with the bigger benefit to perform (lines 9-10). On the contrary, if $AddSet$ is empty, which means no Add operation is possible, then LSCC performs either a Swap or Drop operation. It picks a vertex $x \in C$ with the biggest Δ_{drop} (i.e. the smallest weight) (line 12), and then compares Δ_{swap} and Δ_{drop} and chooses the operation with the bigger benefit to perform (line 13).

After each operation, the values of $confChange$ are updated according to the corresponding SCC rules (line 14), and if $w(C)$ is larger than $w(C_{localbest})$, $C_{localbest}$ is updated by C (line 15).

Evaluation of LSCC on Standard Benchmarks

We carry out extensive experiments to evaluate the performance of the LSCC algorithm for MWCP on two standard benchmarks, including DIMACS and BHOSLIB. DIMACS benchmarks are from the Second DIMACS Implementation Challenge (Johnson and Trick 1996) including problems from real applications and randomly generated graphs. BHOSLIB instances are generated randomly based on the model RB at the phase transition area (Xu et al. 2005). These instances are originally unweighted, and to obtain the corresponding MWCP instances, we use the same method as in (Pullan 2008; Wu, Hao, and Glover 2012). For the ith vertex v_i, $w(v_i)$=(i mod 200)+1.

For comparison, we choose MN/TS (Wu, Hao, and Glover 2012) to represent a state-of-the-art algorithm for solving MWCP. MN/TS is open-source and implemented in C++. Our algorithm LSCC is also implemented in C++. Both of two algorithms are compiled by g++ 4.6.2 with the $-$O2 option. For the search depth L, MN/TS and LSCC set L=4000 for all instances. MN/TS employs a tabu heuristic and the tabu tenure TL is set to 7 as in (Wu, Hao, and Glover 2012).

In order to demonstrate the effectiveness of the SCC heuristic, we also compare LSCC with its variant LCC (Local search with CC) which utilizes the original CC strategy (as introduced in Section 2.1) instead of SCC.

[1]When the $AddSet$ (or $SwapSet$) set is empty, the returned vertex (or vertex pair) is denoted by -1 (or (-1,-1)), and the corresponding Δ value is set to $-\infty$.

Instance	MN/TS	LCC	LSCC	$\delta_{max}(\delta_{avg})$
	$w_{max}(w_{avg})$	$w_{max}(w_{avg})$	$w_{max}(w_{avg})$	
C2000.9	10999 (10948.5)	10267(9948)	10999 (10922.6)	0(-25.9)
p_hat1500-3	10321(10314.4)	10321(10130.1)	10321(10321)	0(**6.6**)
MANN_a27	12281(12270.6)	12275(12268.8)	12283(12283)	**2**(**12.4**)
MANN_a45	34192(34167)	34183(34175.9)	34254(34242.1)	**62**(**75.1**)
MANN_a81	111128(111074.6)	111135(111084.8)	111135(111118.1)	**7**(**10.2**)
frb56-25-1	5916(5815.6)	5669(5588.1)	5916(5825.7)	0(**10.1**)
frb56-25-2	5872(5790.8)	5589(5550.7)	5886(5813.7)	**14**(**22.9**)
frb56-25-3	5859(5780.4)	5689(5545.7)	5859(5777.6)	0(-2.8)
frb56-25-4	5892(5818.9)	5712(5311.7)	5892(5821.1)	0(**2.2**)
frb56-25-5	5839(5750.9)	5597(5536.9)	5839(5754.2)	0(**3.3**)
frb59-26-1	6591(6516)	6318(6108.9)	6591(6538.3)	0(**22.3**)
frb59-26-2	6645(6542.8)	6320(6190.1)	6645(6546.9)	0(**4.1**)
frb59-26-3	6608(6579.5)	6178(6105.5)	6608(6505.7)	0(-73.8)
frb59-26-4	6592(6463.7)	6246(6076.2)	6592(6488.6)	0(**24.9**)
frb59-26-5	6584(6491)	6269(6100.2)	6584(6512.6)	0(**21.6**)

Table 1: Experiment results of MN/TS, LCC, and LSCC on DIMACS and BHOSLIB benchmarks.
DIMACS instances for which MN/TS and LSCC find the same quality cliques very quickly are not reported. A positive δ_{max} or δ_{avg} indicates LSCC finds better quality clique than MN/TS.

All the experiments were run on Ubuntu Linux, with 3.1 GHZ CPU and 8GB memory. For each instance, each algorithm is performed 100 independent runs with different random seeds, where each run is terminated upon reaching a given time limit (1000 seconds). For each instance, w_{max} is the weight of the maximum clique found, and w_{avg} is the average weight over the 100 runs. We also report the difference δ_{max} and δ_{avg} between the maximum and average weight values of clique found by LSCC and MN/TS.

Experiment results on the DIMACS are shown in Table 1. Most DIMACS instances are so easy that MN/TS and LSCC find the same quality cliques very quickly, and thus are not report. The results show that LSCC find better quality cliques than MN/TS and LCC on DIMACS instances. Particularly, LSCC obtains new best solutions for MANN_a27, MANN_a45, and MANN_a81. LSCC is consistently superior on the MANN domain. For p_hat1500-3, LSCC is the only algorithm that finds a 10321-sized clique consistently in 100% runs. Finally, we note that LSCC succeeds in finding the best known solution for all DIMACS instances, indicating its robustness.

The results of BHOSLIB instances are also shown in Table 1. For focusing on hard instances, we only present two groups of the largest-sized instances, which are much more difficult than other small instances. The results illustrate that LSCC outperforms MN/TS for these instances. Moreover, LSCC improves the maximum clique of one instance frb56-25-2. For instances where both algorithms find the same quality maximum weight clique, the averaged weight of cliques found by LSCC is larger than that of MN/TS, except for frb56-25-3 and frb59-26-3. Finally, the comparison between LSCC and LCC also confirms the effectiveness of the SCC heuristic.

Algorithm 2: the BMS heuristic

```
pick a random vertex pair (v,v′)∈ SwapSet with confChange(v′)=1;
Δ*_swap := Δ_swap(v, v′);
for i = 0;i < k;i++ do
    pick a random vertex pair (u,u′)∈ SwapSet with confChange(u′)=1;
    if (Δ_swap(u, u′) > Δ*_swap)||(Δ_swap(u, u′) = Δ*_swap&age(u′) < age(v′)) then
        (v,v′):=(u,u′);
        Δ*_swap := Δ_swap(u, u′);
return (v,v′);
```

Improving LSCC for Massive Graphs

Although LSCC performs quite well on standard benchmarks, it is not so effective on massive graphs. In this section, we employ a heuristic called Best from Multiple Selection (BMS) to improve LSCC, resulting in an improved algorithm called LSCC+BMS. We show the efficiency of LSCC+BMS and the effectiveness of the underlying heuristics by experiments on a broad range of massive graphs.

The BMS Heuristic and LSCC+BMS Algorithm

In LSCC, we use the best-picking heuristic to choose a swapping vertex pair of the best benefit (w.r.t. Δ_{swap}) from the $SwapSet$ to swap. With a suitable criterion, this kind heuristic could guide the search towards the most promising area, and is thus commonly adopted in local search algorithms (Wu, Hao, and Glover 2012; Cai et al. 2013). Such best-picking heuristics are suitable for most cases, but work not well in massive graphs where the $SwapSet$ is usually very large and finding a best pair not only wastes a lot of time but also cannot guarantee that this move is the best one for the quality of solution.

Based on the considerations above, we apply a fast and effective heuristic named Best from Multiple Selection (BMS) for choosing a vertex pair from $SwapSet$, which costs little time while at the same time can pick a pair of good quality. The BMS heuristic strikes a good balance between the quality of the vertex pair and the time complexity. A formal description of the BMS heuristic is shown in Algorithm 2.

Basically, the BMS heuristic chooses k swapping pairs (v,v') randomly, and then returns the best swapping pair w.r.t. the value of Δ_{swap}, where k is a parameter. A trick for accelerating BMS is to pick the best pair when $|SwapSet| < k$. Moreover, we use the SCC strategy to help BMS exclude some unreasonable vertex pairs.

There are two differences between the BMS heuristic in our algorithm and the original on in (Cai 2015). First, the BMS heuristic in FastVC is used to select a vertex to drop, while BMS in our algorithm is used to choose a swapping vertex pair. Secondly and more importantly, we combine the configuration checking technique in the BMS heuristic to prune some "not promising" candidates, while BMS in FastVC does not have any mechanism to exclude not promising candidates.

We use the BMS heuristic to improve the LSCC algorithm, simply by replacing the best-picking heuristic (i.e., line 8 in Algorithm 1) for choosing the swapping vertex pair with the BMS heuristic. The resulting algorithm is thus called LSCC+BMS.

Experiments on Massive Graphs

We evaluate LSCC+BMS on real-world massive graphs from Network Data Repository online (Rossi and Ahmed 2015), which have recently been used in testing the performance of local search methods and parallel algorithms (Rosin 2014; Rossi et al. 2014; Cai 2015). For the sake of space, we do not report the results on graphs with less than 1000 vertices, for which both algorithms find the same quality solutions quickly.

Note that MN/TS fails to find a clique for many of the massive graphs, mainly due to its memory-expensive data structure and high-complexity heuristics. For more interesting comparison, we improve MN/TS by better data structure as well as the BMS heuristic, so that it can also handle massive graphs well. The resulting algorithm is termed as MN/TS+BMS. For the BMS heuristic in both LSCC+BMS and MN/TS+BMS, we set the k parameter to 100, according to some preliminary experiments.

The experiment settings are the same as in the preceding section. In this experiment, δ_{max} and δ_{avg} denote the difference between the maximum and average weight values of clique found by LSCC+BMS and MN/TS+BMS. Also, there are a considerable portion of instances for which LSCC+BMS and MN/TS+BMS find the same quality clique in all runs, that is, $\delta_{max}(\delta_{avg}) = 0(0)$. For these instances, we report another statistics δ_{time}, which represents that the difference of run time between LSCC+BMS and MNTS. For instances where MN/TS fails to find a clique within the time limit, the column for MN/TS is marked as "n/a".

The results on massive graphs are summarized in Table 2, where a positive δ_{max} or δ_{avg} indicates LSCC+BMS finds better quality clique than MN/TS+BMS. MN/TS is essentially worse than the other two algorithms, and we focus on the comparison between MN/TS+BMS and LSCC+BMS. Overall, LSCC+BMS finds better solutions than MN/TS+BMS on these massive graphs. Specially, we observe that LSCC+BMS finds cliques that MN/TS+BMS cannot reach for 17 graphs, and for another 20 graphs where they both can find the same quality cliques, LSCC+BMS does so with a better average solution quality. For the remaining 49 instances, the two algorithms find solutions of the same quality consistently. For 40 out of these 49 instances, LSCC+BMS is faster than MN/TS+BMS. The averaged run time of LSCC+BMS over these 49 instances is only half that of MN/TS+BMS.

The Effectiveness of SCC and BMS

To study the effectiveness of SCC and BMS heuristics, we compare LSCC+BMS with LSCC and LCC. Note that LSCC works with SCC and without BMS, and LCC works with the original CC strategy. Table 3 shows LSCC finds better solutions than LCC, which illustrates the effectiveness of SCC on massive graphs. Thanks to the BMS strategy,

Instance	MN/TS w_{max} (w_{avg})	MN/TS+BMS w_{max} (w_{avg})	LSCC+BMS w_{max} (w_{avg})	δ_{max} (δ_{avg})	δ_{time}
bio-dmela	805(805)	805(805)	805(805)	0(0)	0
bio-yeast	629(629)	629(629)	629(629)	0(0)	**0.42**
ca-AstroPh	5338(5338)	5338(5338)	5338(5338)	0(0)	**46.14**
ca-citeseer	n/a	8838(8838)	8838(8838)	0(0)	**133.22**
ca-coauthors-dblp	n/a	37884(28196)	37884(34622)	0(**6426**)	
ca-CondMat	n/a	2887(2887)	2887(2887)	0(0)	**24.1**
ca-CSphd	489(489)	489(489)	489(489)	0(0)	**0.1**
ca-dblp-2010	n/a	7575(7087.9)	7575(7479.8)	0(**391.9**)	
ca-dblp-2012	n/a	14108(10197)	14108(14108)	0(**3911**)	
ca-Erdos992	958(958)	958(958)	958(958)	0(0)	**0.19**
ca-GrQc	4279(4279)	4279(4279)	4279(4279)	0(0)	**0.27**
ca-HepPh	24533(24533)	24533(24533)	24533(24533)	0(0)	**0.59**
ca-hollywood-09	n/a	222720(121846)	222720(211311)	0(**89465**)	
ca-MathSciNet	n/a	2792(2374.3)	2792(2543)	0(**168.7**)	
ia-email-EU	n/a	1350(1350)	1350(1350)	0(0)	-0.36
ia-email-univ	1473(1473)	1473(1473)	1473(1473)	0(0)	**0.05**
ia-enron-large	n/a	2490(2490)	2490(2490)	0(0)	-2.54
ia-fb-messages	791(791)	791(791)	791(791)	0(0)	**0.01**
ia-reality	374(374)	374(374)	374(374)	0(0)	**0.56**
ia-wiki-Talk	n/a	1884(1884)	1884(1884)	0(0)	-1.19
inf-power	888(888)	888(888)	888(888)	0(0)	**0.51**
inf-roadNet-CA	n/a	597(594.5)	752(613.4)	**155**(**18.9**)	
inf-roadNet-PA	n/a	597(596.5)	599(599)	**2**(**2.5**)	
rec-amazon	n/a	942(942)	942(942)	0(0)	**7.63**
sc-ldoor	n/a	4018(3836.1)	4081(3936.4)	**63**(**100.3**)	
sc-msdoor	n/a	4088(3959.4)	4088(4043.9)	0(**84.5**)	
sc-nasasrb	n/a	4548(4441)	4548(4548)	0(**107**)	
sc-pkustk11	n/a	5091(4769.8)	5298(5298)	**207**(**528.2**)	
sc-pkustk13	n/a	5853(5565.4)	5928(5874.6)	**75**(**309.2**)	
sc-pwtk	n/a	4548(4372)	4620(4603.2)	**72**(**231.2**)	
sc-shipsec1	n/a	3255(3100.4)	3540(3381.5)	**285**(**281.1**)	
sc-shipsec5	n/a	4500(4338.8)	4524(4445.4)	**24**(**106.6**)	
soc-BlogCatalog	n/a	4803(4803)	4803(4803)	0(0)	**35.90**
soc-brightkite	n/a	3672(3653.8)	3672(3655.7)	0(**1.9**)	
soc-buzznet	n/a	2981(2981)	2981(2981)	0(0)	**22.86**
soc-delicious	n/a	1547(1523.3)	1547(1543.5)	0(**20.2**)	
soc-digg	n/a	4675(4675)	5303(4800.6)	**628**(**125.6**)	
soc-douban	n/a	1682(1682)	1682(1682)	0(0)	**19.35**
soc-epinions	n/a	1657(1657)	1657(1657)	0(0)	**27.48**
soc-flickr	n/a	7050(6998.1)	7083(7083)	**33**(**84.9**)	
soc-flixster	n/a	3805(3036.4)	3805(3500.9)	0(**464.5**)	
soc-FourSquare	n/a	3064(3043.6)	3064(3053.6)	0(**10**)	
soc-gowalla	n/a	2335(2209)	2335(2291.8)	0(**82.8**)	
soc-lastfm	n/a	1773(1773)	1773(1773)	0(0)	**65.84**
soc-livejournal	n/a	2521(2050.4)	3120(2327.7)	**599**(**277.3**)	
soc-LiveMocha	n/a	1784(1784)	1784(1784)	0(0)	-4.97
soc-pokec	n/a	2341(1984.3)	3191(2075.3)	**850**(**91**)	
soc-slashdot	n/a	2811(2811)	2811(2811)	0(0)	-21.24
soc-twitter-follows	n/a	808(785.1)	808(808)	0(**22.9**)	
soc-youtube	n/a	1961(1961)	1961(1961)	0(0)	-18.63
soc-youtube-snap	n/a	1787(1787)	1787(1787)	0(0)	-51.79
socfb-A-anon	n/a	2576(2096.5)	2777(2196.4)	**201**(**99.9**)	
socfb-B-anon	n/a	2513(1986.9)	2537(2071.3)	**24**(**84.4**)	
socfb-Berkeley13	n/a	4906(4906)	4906(4906)	0(0)	**7.24**
socfb-CMU	4141(4141)	4141(4141)	4141(4141)	0(0)	**1.11**
socfb-Duke14	3694(3694)	3694(3694)	3694(3694)	0(0)	**12.25**
socfb-Indiana	n/a	5412(5412)	5412(5412)	0(0)	**29.67**
socfb-MIT	3658(3658)	3658(3658)	3658(3658)	0(0)	**0.74**
socfb-OR	n/a	3523(3523)	3523(3523)	0(0)	**120.2**
socfb-Penn94	n/a	4738(4709.3)	4738(4738)	0(**28.7**)	
socfb-Stanford3	5769(5769)	5769(5769)	5769(5769)	0(0)	**10.52**
socfb-Texas84	n/a	5546(5524.5)	5546(5546)	0(**21.5**)	
socfb-UCLA	n/a	5595(5595)	5595(5595)	0(0)	**26.42**
socfb-UConn	5733(5733)	5733(5733)	5733(5733)	0(0)	**2.24**
socfb-UCSB37	5669(5669)	5669(5669)	5669(5669)	0(0)	**46.66**
socfb-UF	n/a	6043(6021)	6043(6043)	0(**22**)	
socfb-UIllinois	n/a	5730(5721.6)	5730(5730)	0(**8.4**)	
socfb-Wisconsin87	n/a	4239(4239)	4239(4239)	0(0)	**27.29**
tech-as-caida2007	n/a	1869(1869)	1869(1869)	0(0)	-0.41
tech-as-skitter	n/a	5527(4387)	5703(5271.8)	**176**(**884.8**)	
tech-internet-as	n/a	1692(1692)	1692(1692)	0(0)	-0.38
tech-p2p-gnutella	n/a	703(675.8)	703(703)	0(**27.2**)	
tech-RL-caida	n/a	1861(1861)	1861(1861)	0(0)	**26.25**
tech-routers-rf	1460	1460(1460)	1460(1460)	0(0)	**0.12**
tech-WHOIS	6154(6154)	6154(6154)	6154(6154)	0(0)	**19.60**
web-arabic-2005	n/a	10558(10529)	10558(10558)	0(**29**)	
web-BerkStan	3249(3249)	3249(3249)	3249(3249)	0(0)	**2.2**
web-edu	2077(2077)	2077(2077)	2077(2077)	0(0)	**5.06**
web-google	1749(1749)	1749(1749)	1749(1749)	0(0)	**0.1**
web-indochina-2004	6997(6997)	6997(6997)	6997(6997)	0(0)	**8.58**
web-it-2004	n/a	43842(36402)	45477(45313)	**1635**(**8911**)	
web-sk-2005	n/a	11925(10775)	11925(11925)	0(**1150**)	
web-spam	2503(2503)	2503(2503)	2503(2503)	0(0)	**3.4**
web-uk-2005	n/a	54850(54850)	54850(54850)	0(0)	**467.52**
web-webbase-2001	3574(3574)	3574(3574)	3574(3574)	0(0)	**110.69**
web-wikipedia2009	n/a	1997(1582.3)	3455(2451)	**1458**(**868.7**)	

Table 2: Experiment results on the massive graphs.

Instance	LCC $w_{max}(w_{avg})$	LSCC $w_{max}(w_{avg})$	LSCC+BMS $w_{max}(w_{avg})$
ca-coauthors-dblp	31925(25484.4)	**37884**(34425.8)	**37884**(**34622.6**)
ca-dblp-2010	**7575**(6966.8)	**7575**(7439.8)	**7575**(**7479.8**)
ca-hollywood-2009	**222720**(90209.1)	**222720**(199902.8)	**222720**(**211311.4**)
ca-MathSciNet	2611(1991)	2611(2393.1)	**2792**(**2543**)
inf-roadNet-CA	594(574.9)	597(597)	**752**(**613.4**)
inf-roadNet-PA	597(579.3)	597(597)	**599**(**599**)
sc-ldoor	4060(3733.7)	4074(3922.5)	**4081**(**3936.4**)
sc-msdoor	3941(3749.9)	4074(4036.7)	**4088**(**4043.9**)
sc-pkustk11	**5298**(4741.9)	**5298**(5090.5)	**5298**(**5298**)
sc-pkustk13	5853(5662.8)	**5928**(5864.1)	**5928**(**5874.6**)
sc-shipsec1	**3540**(3116.8)	**3540**(3373.2)	**3540**(**3381.5**)
sc-shipsec5	4440(4041.8)	4500(4444.8)	**4524**(**4445.4**)
socfb-B-anon	1907(1521.7)	2470(1993.2)	**2537**(**2071.3**)
soc-delicious	1466(1446.5)	**1547**(1542.8)	**1547**(**1543.5**)
soc-digg	4429(4240.3)	4675(4675)	**5303**(**4800.6**)
soc-flickr	6717(6138.1)	**7083**(7058.1)	**7083**(**7083**)
soc-flixster	3311(2184.3)	**3805**(3162.3)	**3805**(**3500.9**)
soc-FourSquare	3038(2982.5)	**3064**(3024.7)	**3064**(**3053.6**)
soc-lastfm	1695(1599.9)	**1773**(1769.4)	**1773**(**1773**)
soc-pokec	1960(1619.3)	**3191**(2020.2)	**3191**(**2075.3**)
soc-youtube-snap	**1787**(1571.9)	**1787**(1744.2)	**1787**(**1787**)
tech-as-skitter	5506(4302.4)	**5703**(5258.2)	**5703**(**5271.8**)
web-arabic-2005	10445(10445)	**10558**(10546.7)	**10558**(**10558**)
web-wikipedia2009	1879(1087.3)	1997(1378.9)	**3455**(**2451**)

Table 3: Comparing LCC, LSCC and LSCC+BMS on typical massive graphs

LSCC+BMS acquires better cliques than LSCC in terms of both w_{max} and w_{avg}.

Conclusion

We developed two local search algorithms for the Maximum Weight Clique problem (MWCP). We first propose a variant of the configuration checking (CC) strategy, called Strong Configuration Checking (SCC), which is used in developing a local search algorithm named LSCC. Experiments on standard benchmarks show its superiority over the current best local search algorithm for MWCP namely the MN/TS algorithm.

We further improve LSCC for massive graphs by applying a cost effective heuristic for choosing the swapping vertex pair, namely Best from Multiple Selection (BMS), and obtain the LSCC+BMS algorithm. We also use BMS to improve the MN/TS algorithm. Experimental results on massive graphs show that the BMS heuristic significantly improves the performance of the algorithms on massive graphs, and that LSCC+BMS significantly performs better than MN/TS+BMS. We also carry out extensive experiments to analyze the effectiveness of the SCC and BMS heuristics.

In the future, we plan to further study variants of CC in the context of MWCP and MCP, and to exploit other properties of vertices such as subscore (Cai and Su 2013), to improve the algorithms. For massive graphs, it is interesting to design low-complexity heuristics to improve the Add and Drop operations in local search algorithms for MWCP.

Acknowledgments

This work was supported in part by China National 973 program 2014CB340301, NSFC under Grant Nos. (61370156, 61502464, 61503074) and Program for New Century Excellent Talents in University (NCET-13-0724). We would like to thank the anonymous referees for their helpful comments.

References

André, A.; Djamal, H.; and Donia, T. 2014. Improving configuration checking for satisfiable random k-SAT instances. In *Proceedings of International Symposium on Artificial Intelligence and Mathematics, ISAIM 2014*.

Babel, L. 1994. A fast algorithm for the maximum weight clique problem. *Computing* 52(1):31–38.

Balasundaram, B., and Butenko, S. 2006. Graph domination, coloring and cliques in telecommunications. In *Handbook of Optimization in Telecommunications*. 865–890.

Ballard, D., and Brown, C. 1982. Computer vision. *New Jersey: Prentice Hall*.

Benlic, U., and Hao, J.-K. 2013. Breakout local search for maximum clique problems. *Computers & Operations Research* 40(1):192–206.

Bomze, I. M.; Pelillo, M.; and Stix, V. 2000. Approximating the maximum weight clique using replicator dynamics. *Neural Networks, IEEE Transactions on* 11(6):1228–1241.

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

Cai, S., and Su, K. 2012. Configuration checking with aspiration in local search for sat. In *Proceedings of AAAI 2012*, 334–340.

Cai, S., and Su, K. 2013. Local search for boolean satisfiability with configuration checking and subscore. *Artificial Intelligence* 204:75–98.

Cai, S.; Su, K.; Luo, C.; and Sattar, A. 2013. Numvc: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research* 687–716.

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

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

Fang, Z.; Li, C.-M.; Qiao, K.; Feng, X.; and Xu, K. 2014. Solving maximum weight clique using maximum satisfiability reasoning. In *Proceedings of ECAI 2014*, volume 263, 303.

Feige, U. 2004. Approximating maximum clique by removing subgraphs. *SIAM Journal on Discrete Mathematics* 18(2):219–225.

Gomez Ravetti, M., and Moscato, P. 2008. Identification of a 5-protein biomarker molecular signature for predicting alzheimers disease. *PloS one* 3(9):e3111.

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

Johnson, D. S., and Trick, M. A. 1996. *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*, volume 26. American Mathematical Soc.

Karp, R. 1972. Reducibility among combinatorial problems. *Complexity of Computer Computations* 85–103.

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

Li, C. M., and Huang, W. 2005. Diversification and de- terminism in local search for satisfiability. In *Proceedings of Theory and Applications of Satisfiability Testing, SAT 2005*, 158–172.

Li, C. M., and Quan, Z. 2010. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In *AAAI*, volume 10, 128–133.

Li, C.-M.; Fang, Z.; and Xu, K. 2013. Combining maxsat reasoning and incremental upper bound for the maximum clique problem. In *Proceedings of ICTAI 2013*, 939–946.

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

Östergård, P. R. 2001. A new algorithm for the maximum-weight clique problem. *Nordic Journal of Computing* 8(4):424–436.

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

Pullan, W. 2006. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization* 12(3):303–323.

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

Rosin, C. D. 2014. Unweighted stochastic local search can be effective for random csp benchmarks. *arXiv preprint arXiv:1411.7480*.

Rossi, R. A., and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Rossi, R. A.; Gleich, D. F.; Gebremedhin, A. H.; and Patwary, M. M. A. 2014. Fast maximum clique algorithms for large graphs. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, 365–366.

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

Tomita, E., and Seki, T. 2003. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete mathematics and theoretical computer science*. 278–289.

Wang, Y.; Ouyang, D.; Zhang, L.; and Yin, M. 2015. A novel local search for unicost set covering problem using hyperedge configuration checking and weight diversity. *SCIENCE CHINA Information Sciences*.

Wu, Q., and Hao, J.-K. 2013. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization* 26(1):86–108.

Wu, Q.; Hao, J.-K.; and Glover, F. 2012. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research* 196(1):611–634.

Xu, K.; Boussemart, F.; Hemery, F.; Lecoutre, C.; et al. 2005. A simple model to generate hard satisfiable instances. In *Proceedings of IJCAI*, 337–342.

Yamaguchi, K., and Masuda, S. 2008. A new exact algorithm for the maximum weight clique problem. *ITC-CSCC: 2008* 317–320.