Local search with edge weighting and configuration checking heuristics for minimum vertex cover

Author

Cai, Shaowei, Su, Kaile, Sattar, Abdul

Published

2011

Journal Title

Artificial Intelligence

DOI

10.1016/j.artint.2011.03.003

Local Search with Edge Weighting and Configuration Checking Heuristics for Minimum Vertex Cover

Shaowei Cai[a], Kaile Su[b,c], Abdul Sattar[b,d]

[a] *Key laboratory of High Confidence Software Technologies (Peking University),Ministry of Education, Beijing, China*
[b] *Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia*
[c] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing*
[d]*ATOMIC Project, Queensland Research Lab, NICTA*

Abstract

The Minimum Vertex Cover (MVC) problem is a well known combinatorial optimization problem of great importance in theory and applications. In recent years, local search has been shown to be an effective and promising approach to solve hard problems, such as MVC. In this paper, we introduce two new local search algorithms for MVC, called EWLS (Edge Weighting Local Search) and EWCC (Edge Weighting Configuration Checking). The first algorithm EWLS is an iterated local search algorithm that works with a partial vertex cover, and utilizes an edge weighting scheme which updates edge weights when getting stuck in local optima. Nevertheless, EWLS has an instance-dependent parameter. Further, we propose a strategy called Configuration Checking for handling the cycling problem in local search. This is used in designing a more efficient algorithm that has no instance-dependent parameters, which is referred to as EWCC. Unlike previous vertex-based heuristics, the configuration checking strategy considers the induced subgraph configurations when selecting a vertex to add into the current candidate solution.

A detailed experimental study is carried out using the well known DIMACS and BHOSLIB benchmarks. The experimental results conclude that EWLS and EWCC are largely competitive on DIMACS benchmarks, where they outperform other current best heuristic algorithms on most hard instances, and dominate on the hard random BHOSLIB benchmarks. Moreover, EWCC makes a significant improvement over EWLS, while both EWLS and EWCC set a new record on a twenty-year challenge instance. Further, EWCC performs quite well even on structured instances in comparison to the best exact algorithm we know. We also study the run-time behavior of EWLS and EWCC which shows interesting properties of both algorithms.

Keywords: Minimum Vertex Cover, Local Search, Edge Weighting, Configuration Checking

Email addresses: shaowei_cai@126.com (Shaowei Cai), k.su@griffith.edu.au (Kaile Su), a.sattar@griffith.edu.au (Abdul Sattar)

This is an improved and extended version of a paper presented at AAAI-2010, Atlanta, USA.

1. Introduction

1.1. The Problem

Given an undirected graph $G = (V, E)$, a vertex cover is a subset $S \subseteq V$, such that every edge in G has at least one endpoint in S. The Minimum Vertex Cover (MVC) problem is to find the minimum sized vertex cover in a graph. MVC is a prominent NP-hard combinatorial problem with many real-world applications, such as network security, scheduling, VLSI design and industrial machine assignment [50]. It is equivalent to two other well-known NP-hard combinatorial problems: Maximum Independent Set (MIS) problem and Maximum Clique (MC) problem. Algorithms for MVC can be directly used to solve the MC problem, which has a range of applications into areas such as information retrieval, experimental design, signal transmission, computer vision [45], and also in bioinformatics such as aligning DNA and protein sequences [32]. Due to their hardness and importance to many real world applications, even a small progress in solving these three problems can have a significant impact in practice. Therefore, these problems remain an active research agenda within the Artificial Intelligence community, and have been widely investigated for the last several decades [8, 9, 12, 15, 39, 40, 45, 50].

1.2. Motivation of the Local Search Approach

These three problems MVC, MIS, and MC are all NP-hard and the associated decision problems are NP-complete [16]. Furthermore, it is NP-hard to approximate MVC within any factor smaller than 1.3606 [11]; and the state-of-the-art approximation algorithms can only achieve an approximation ratio of $2 - o(1)$ [24, 33]. Besides the inapproximability of MVC, Håstad [25, 26] shows that both MIS and MC are not approximable within $|V|^{1-\epsilon}$ for any $\epsilon > 0$, unless NP=ZPP.[2] Recently, this conclusion has been enhanced that MC is not approximable within $|V|^{1-\epsilon}$ for any $\epsilon > 0$ unless NP=P, derived from a derandomization by Zuckerman [62] from a result of [25]. Moreover, the currently best polynomial-time approximation algorithm for MC proposed by Feige is only guaranteed to find a clique within a factor of $O(n(loglogn)^2/(logn)^3)$ of optimal [14].

Practical algorithms for MVC (MIS, MC) mainly fall into two types: exact ones and heuristic ones. Exact methods include branch-and-bound algorithms, e.g. [9, 13, 39, 40, 44, 49, 56]; and heuristic ones are mainly stochastic local search algorithms, e.g. [12, 18, 50, 51] for MVC, [1–3, 7] for MIS, and [6, 22, 34, 35, 45, 47, 53] for MC. Exact algorithms guarantee the optimality of the solutions they find, but may fail to give a solution within reasonable time for large instances. As the size of the problem increases, the exact methods become futile.

On the other hand, although heuristic algorithms cannot guarantee the optimality of their solutions, they can solve large and hard MVC (MIS, MC) instances in the sense of giving optimal or near-optimal solutions within reasonable time. Note that there are

[2] ZPP is the class of problems that can be solved in expected polynomial time by a probabilistic algorithm with zero error probability.

many large and hard problems in real world, for which one must resort to heuristic approaches to obtain good solutions within reasonable time.

Heuristic algorithms have been successfully used to solve combinatorial problems efficiently. For example, heuristic methods for SAT can significantly outperform DPLL-based methods on random problems [36]. A modern heuristic SAT solver can solve hard instances with over a million variables and several million constraints within reasonable time [21]. Heuristic local search algorithms are often quite effective at finding near-optimal solutions for MAXSAT [38]. For graph combinatorial optimization problems, such as MVC, MIS and MC, a number of heuristic algorithms have also been proposed.

However, the results are far from satisfactory, especially on the large hard instances. For example, there has been a hard challenging instance $frb100$-40 of MVC, which is a graph with 4000 vertices. Instance $frb100$-40 has a 3900-sized optimal vertex cover, whereas the size of the best solution found before the presented work [8] is only 3903[3]. Since large and hard SAT instances can be solved efficiently, one may consider transforming MVC (MIS, MC) problems into SAT problems and solving them by SAT solvers. However, the size of those SAT instances transformed from MVC (MIS, MC) problems may become much larger, and they may lose some structural information. Indeed, general solvers like SAT solvers do not perform better than specific solvers on these problems. Therefore, a major challenge is to make a significant progress on algorithms for large hard instances of MVC (MIS, MC) problems. According to the above considerations, we focus on the local search algorithms for hard instances of MVC in this work.

1.3. Previous Heuristics

There are a number of heuristic approaches to MVC. An evolutionary approach to MVC and related survey on this kind of algorithms are presented in [12]. Ant colony approaches have been proposed in [51] and [18]. The recent Cover Edge Randomly (COVER) algorithm [50] is an iterative best improvement algorithm using edge weights to guide the local search. Shown by results on DIMACS and BHOSLIB benchmarks in [50], COVER is the best one among heuristic methods for MVC.

As for MIS, existing heuristic algorithms include: Optimised Crossover Heuristic (OCH) [1]; QSH, which is based on optimization of a quadratic over a sphere [7]; and the evolutionary algorithm Widest Acyclic Orientation (WAO) [3]. Recently, an $iterated\ local\ search$ algorithm based on improving swaps is proposed in [2], which shows significantly better performance than previous MIS heuristics. However, results on the DIMACS benchmarks as presented in [2] and [45] indicate that the MC algorithm called DLS-MC dominates in performance except for two MANN instances.

Compared with MVC and MIS, more work has been done on MC problem. Reported in [45], there are five heuristic algorithms achieving state-of-the-art performance: Reactive Local Search (RLS) [4], QUALEX-MS [6], Deep Adaptive Greedy Search (DAGS) [22], *k-opt* algorithm [34] (which has evolved into $iterated$ *k-opt*

[3]http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

algorithm [35]) and Edge-AC+LS [53]. However, as shown in [45], all the above algorithms are dominated by the Dynamic Local Search-Maximum Clique (DLS-MC) algorithm [45] over a large range of benchmark instances. DLS-MC alternates between phases of iterative improvement and of plateau search, using vertex penalties to guide selecting vertices, and has an instance-dependent parameter called penalty delay. Fortunately, its improved version Phased Local Search (PLS) algorithm, has no instance-dependent parameters and is comparable with or more efficient than DLS-MC for all DIMACS instances [46]. To the best of our knowledge, PLS is still one of the best MC algorithms today. Pullan extends PLS into PLS_MVC and PLS_MIS for MVC and MIS problem [47], achieving state-of-the-art performance.

1.4. Main Contributions

In this paper, we propose two new local search algorithms for MVC, namely EWLS (Edge Weighting Local Search), and EWCC (Edge Weighting Configuration Checking), which is an improved version of EWLS. EWLS focuses on finding a partial vertex cover that provides a better upper bound on the size of the minimum vertex cover, and extends it into a vertex cover. It uses an edge weighting scheme, which updates edge weights when stuck in local optima, so that it may discover good candidate solutions hidden behind local optima. As an iterated local search algorithm, in each local search stage EWLS repeatedly swaps a vertex in the partial vertex cover and an end-vertex of an uncovered edge, in order to decrease the total weight of uncovered edges. Also, EWLS employs some other search strategies to improve the quality of local optima. The experimental results show that EWLS achieves excellent performance on a large variety of benchmarks, especially on large and hard instances. On the commonly used DIMACS benchmarks, EWLS is competitive with the state-of-the-art solvers, and outperforms them on most hard instances. On a suite of hard random benchmarks with hidden optimal solutions (BHOSLIB), which is strongly recommended by MC community [23], EWLS delivers better results than the current best heuristic MC algorithm PLS and the current best MVC heuristic algorithm COVER. Nevertheless, EWLS has an instance-dependent parameter, which controls the size of the partial vertex cover.

The second main contribution of this work is an innovative and general heuristic strategy that exploits induced subgraph information to improve the performance of a local search algorithm. The new strategy, called $configuration\ checking$, takes into account the induced subgraph configurations when selecting a vertex to add into the current candidate solution: for a candidate vertex to be added v, if the configuration of the subgraph induced by v and its neighbors has not changed since v's last removing, then v should not be added back to the current candidate solution. This is a very interesting alternative to the traditional tabu method to avoid the cycling problem in local search. We successfully combine the configuration checking heuristic into EWLS, resulting in a non-parameter local search algorithm EWCC. EWCC significantly outperforms EWLS and other heuristics on the hard random BHOSLIB benchmarks consistently. It also shows an overall performance improvement on DIMACS benchmarks over EWLS.

In particular, both EWLS and EWCC contribute significantly to instance frb100-40. This challenging instance is generated based on the exact phase transition of model RB [58, 59, 61], a successful model of random constraint satisfactory problems (CSP).

The story of this challenging instance is as follows[4]: On Feb. 22, 2005, instance $frb100$-40 was generated and made available online as a challenge for MVC (MIS, MC) algorithms, which is a graph with 4000 vertices and has a hidden optimal vertex cover of 3900 vertices. Based on theoretical analysis and experimental results of smaller instances, the designer of this challenge conjectured that it will not be solved on a PC in less than 24 hours within the next two decades. On Jul. 28, 2005, using a local search SAT solver called Wsatcc, Lengning Liu at the University of Kentucky found a vertex cover of 3904 vertices. On Jul. 4, 2007, using a local search solver called COVER, Silvia Richter at Griffth University found a vertex cover of 3903 vertices [50]. In this paper, our two algorithms EWLS and EWCC both find a vertex cover of 3902 vertices, which therefore sets a new record for this twenty-year challenging problem. Noticing that $frb100$-40 was generated based on the exact phase transition of random CSP, we believe our heuristics are also promising for solving the hard CSP instances.

In addition, we present the run-time distributions of EWLS and EWCC, which are well approximated by exponential distributions. As a consequence, both algorithms are very robust $w.r.t.$ the cutoff parameters like the $maxsteps$ parameter and thus, the number of random restarts. Hence, they have close-to-optimal parallelization speedup [28, 29]. We also demonstrate the effectiveness of the underlying mechanisms in the two algorithms.

1.5. Structure of the Paper

The remainder of this paper is organized as follows: we provide some necessary background knowledge in the next section. In Section 3, we present the notions and theory of the edge weighting scheme for solving MVC problem and describe the EWLS algorithm. In Section 4, we give the configuration checking heuristic and introduce the EWCC algorithm which is an improved version of EWLS by utilizing the configuration checking heuristic. Section 5 reports the experimental study of our two algorithms and comparative results to other algorithms, including heuristic algorithms and exact ones. This is followed by a more detailed investigation of the behavior of EWLS and EWCC, and the factors determining their performance in Section 6. Finally, we conclude the paper by summarizing the main contributions and some remarks on future directions in Section 7.

2. Preliminaries

For convenience, we provide a brief introduction of some necessary notions about graphs used in this paper. Please refer to [10] for more details.

An undirected $graph$ $G = (V, E)$ consists of a vertex set V and an edge set $E \subseteq V \times V$, where each edge is an unordered pair of distinct vertices. The notations $V(G)$ and $E(G)$ denote the vertex and edge sets of graph G, respectively. For an edge $e(u, v)$, u and v are the $endpoints$ of edge e, and $endpoint(e) = \{u, v\}$. For a subset $M \subseteq E$, we write $endpoint(M)$ for $\bigcup_{e \in M} endpoint(e)$. $N(v) = \{u \in V | (u, v) \in E\}$ is the

[4] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

set of neighbors of a vertex v, and we denote $N[v] = N(v) \cup \{v\}$. The $degree$ of v is $d(v) = |N(v)|$. The $maximum\ degree$ over all vertices of a graph G is denoted by $\Delta(G)$. For a subset of vertices $X \subseteq V$, we use $G[X]$ to denote the $induced\ subgraph$ of G whose vertex set is X and whose edge set is the subset of $E(G)$ consisting of those edges with both endpoints in X. The $complementary\ graph$ of a graph $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(u, v)|u, v \in V, u \neq v$ and $(u, v) \notin E\}$. For a set S, we use $random(S)$ to denote a random element of S.

An $independent\ set$ S of a graph $G = (V, E)$ is a subset of V whose elements are pairwise non-adjacent. A $clique$ K of a graph $G = (V, E)$ is a subset of V whose elements are pairwise adjacent. The $maximum\ independent\ set$ ($maximum\ clique$) problem is to find the maximum sized independent set (clique) in a graph.

Vertex cover is closely related to independent set and clique. A vertex set S is an independent set of G iff $V \backslash S$ is a vertex cover of G; a vertex set K is a clique of G iff $V \backslash K$ is a vertex cover of the complementary graph $\overline{G}$. Hence, the MIS problem and MC problem can be reduced to MVC problem. To find the maximum independent set of a graph G, one can find the minimum vertex cover C for G and return $V \backslash C$. Similarly, to find the maximum clique of a graph G, one can find the minimum vertex cover C for the complementary graph $\overline{G}$, and return $V \backslash C$.

3. Edge Weighting and The EWLS Algorithm

In this section we present EWLS, an iterated local search algorithm for the Minimum Vertex Cover problem, based on an edge weighting scheme. EWLS works with a partial vertex cover of a graph G. A partial vertex cover here means a set of vertices to cover all or partial edges in a given graph. A partial vertex cover may leave some edges uncovered; EWLS repeatedly swaps a vertex in the current partial vertex cover and an end-vertex of an uncovered edge, in order to decrease the total weight of uncovered edges. When finding a better partial vertex cover, EWLS extends it into a vertex cover as the current best solution.

3.1. Edge Weighting Scheme

The edge weighting scheme plays an important role in EWLS. Each edge is associated to a positive integer number as its weight, and edge weights of the uncovered edges are increased when stuck in local optima. Note that the edge weighting scheme in EWLS is a more general technique similar to constraint weighting, which is an effective diversification technique for local search and has been widely used in SAT and CSP, for example, clause weighting in SAT [30, 31, 43, 54, 55]. Our results therefore provide further evidence for the effectiveness and general applicability of this algorithmic technique.

Edge weighting scheme in EWLS increases the cost of local optima it meets, to level the cost landscape. As a result, the algorithm may find good candidate solutions hidden behind local optima. To do so, EWLS initializes all edge weights to 1, and maintains a set L of uncovered edges in the following search process. When it gets stuck in a local optimum, for each edge $e \in L$, the edge weight is increased by 1, and then EWLS takes a random step to continue to search from another starting point.

Although the COVER algorithm also makes use of edge weights, EWLS is quite different from COVER. COVER is an $iterative\ best\ improvement$ algorithm and updates edge weights in each step, while EWLS is an $iterated\ local\ search$ algorithm and updates edge weights only when stuck in local optima.

3.2. Formal Notions and Theoretical Basis

Given an undirected graph $G = (V, E)$, a $candidate\ solution$ is a subset of vertices $C \subseteq V$. An edge $e \in E$ is $covered$ by a candidate solution C if at least one endpoint of e belongs to C.

An edge weighted undirected graph is an undirected graph $G = (V, E)$ combined with a weighting function w so that each edge $e \in E$ is associated with a positive integer number $w(e)$ as its weight. We use a triple (V, E, w) to denote an edge weighted undirected graph, where (V, E) is an undirected graph.

Let w be a weighting function for G. Then, We set

$$cost(G, C) = \sum_{e \in E \text{ and } e \text{ is not covered by } C} w(e)$$

which indicates the cost of C, that is, the total weight of edges not covered by C.

For a vertex $v \in V$,

$$dscore(v) = cost(G, C) - cost(G, C')$$

where $C' = C \backslash \{v\}$ if $v \in C$, and $C' = C \cup \{v\}$ otherwise. Obviously, $dscore(v) \leq 0$ if $v \in C$, and $dscore(v) \geq 0$ if $v \notin C$. For two vertices $u, v \in V$, where $u \in C$ and $v \notin C$,

$$score(u, v) = cost(G, C) - cost(G, [C \backslash \{u\}] \cup \{v\})$$

measuring the benefit of swapping u and v.

During the search procedure, EWLS always maintains a current candidate solution C and a set L of edges not covered by C. The step to a neighboring candidate solution consists of swapping two vertices: a vertex $u \in C$ is removed from C, and a vertex $v \notin C$ is put into C. The $evaluation\ function$ is $g : C \mapsto cost(G, C)$, which means EWLS prefers candidate solutions with lower cost.

Lemma 1. *Given a graph $G = (V, E)$, and C the current candidate solution, w the edge weighting function, for a pair of vertices $u, v \in V$, where $u \in C$ and $v \notin C$, $score(u, v) = dscore(u) + dscore(v) + w(e(u, v))$ if $e(u, v) \in E$; and $score(u, v) = dscore(u) + dscore(v)$ otherwise.*

EWLS calculates the $score$ of a swapping vertex pair according to Lemma 1. We give the proof of Lemma 1 in appendix A.

Partial vertex cover is an important notion in EWLS, and it is defined as follows:

Definition 2. For an undirected graph $G = (V, E)$, a k-sized vertex set $P \subseteq V$ is a (k, t)-$partial\ vertex\ cover$ $(0 \leq t \leq |E|)$ if $|E| - t$ edges of G are covered by P.

Clearly, a $(k, 0)$-partial vertex cover is a k-vertex cover, as it covers all edges by the definition. Generally, a (k, t)-partial vertex cover can be extended into a vertex cover whose size is at most $k + t$, since we need at most t vertices to cover t edges. A (k, t)-partial vertex cover can be denoted by a k-sized vertex set $P \subseteq V$ and a t-sized set $L \subseteq E$ of edges not covered by P. According to the definition of partial vertex cover, we have the following lemma.

Lemma 3. *Given a graph $G = (V, E)$, a (k, t)-partial vertex cover of G provides an upper bound that equals $k + t$ on the size of the minimum vertex cover of G.*

Example 4. In Figure 1, the solid vertices are selected for covering and the weight of each edge is labeled. The current candidate solution $C = \{v_3, v_5, v_6\}$ is a (3,2)-partial vertex cover, as $|C| = 3$ and there are 2 uncovered edges ($e(v_1, v_2)$ and $e(v_1, v_4)$); the cost of C $cost(C, G) = w(e(v_1, v_2)) + w(e(v_1, v_4)) = 1 + 3 = 4$; $dscore(v_1) = w(e(v_1, v_2)) + w(e(v_1, v_4)) = 1 + 3 = 4$ and $dscore(v_3) = -w(e(v_2, v_3)) - w(e(v_3, v_4)) = -6 - 3 = -9$.

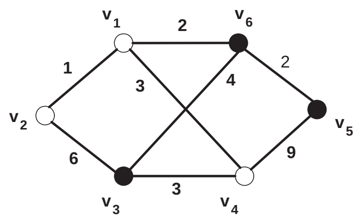

Figure 1: A simple graph labeled with edge weights

3.3. Description of EWLS Algorithm

An essential idea underlying EWLS is to find a partial vertex cover that can be extended into an optimal vertex cover. To do so, we adopt a general framework as follows. Whenever finding a partial vertex cover that provides a better upper bound, EWLS extends it into a vertex cover and stores it as the best vertex cover it has found. Then EWLS removes some vertices from the current partial vertex cover C, and goes on to find a new upper bound. In this way, the MVC problem is transformed into a series of new problems: given a graph $G = (V, E)$ and an integer number k, to find a (k, t)-partial vertex cover that minimizes t, i.e., the number of uncovered edges. EWLS solves these problems using an $iterated\ local\ search$ scheme, which applies a $local\ search\ stage$ to an initial candidate solution until it meets a local optimum; then it perturbs the final candidate solution and executes the next stage.

Based on the above considerations, we outline the EWLS algorithm (Algorithm 1), as described below:

In the beginning, EWLS creates two set variables L and UL. L is the set of uncovered edges; and $UL \subseteq L$ is the set of those unchecked by $ChooseSwapPair$ in the current local search stage. Both of them are set to E. Moreover, edge weights are initialized as 1, and $dscores$ of vertices are computed accordingly. Also, to construct the current candidate solution C, a loop is executed until C becomes a vertex cover. In each iteration, the vertex with the highest $dscore$ is added to C (breaking ties randomly). Finally, the upper bound ub is initialized as $|C|$, and the best solution C^* is initialized as C. Whenever finding a new upper bound, EWLS selects vertices with the highest $dscore$ in C (breaking ties randomly) and removes them until $|C| = ub - delta$. We note that, in C, the vertex with the highest $dscore$ has the minimum absolute value of $dscore$ since all these $dscores$ are negative.

Algorithm 1: EWLS

1 EWLS($G, delta, maxSteps$)
Input: graph $G = (V, E)$,delta(adjust size of C according to $|C| = ub - delta$), $maxSteps$
Output: vertex cover of G
2 **begin**
3     $step := 0$; $L := E$; $UL := E$;
4   initialize all edge weights as 1 and compute $dscores$ of vertices;
5   construct C greedily until it's a vertex cover;
6     $ub := |C|$; $C^* := C$;
7   remove vertices with the highest $dscore$ from C until $|C| = ub - delta$;
8   **while** $step < maxSteps$ **do**
9   **if** $((u, v) := ChooseSwapPair(C, L, UL)) \neq (0, 0)$ **then**
10             $C := [C \backslash \{u\}] \cup \{v\}$;
11   **else**
12             $w(e) := w(e) + 1$ for each $e \in L$;
13             $C := [C \backslash \{u\}] \cup \{v\}$ where $u := random(C)$ and $v := random(endpoint(L))$;
14         $tabuAdd := u$;
15         $tabuRemove := v$;
16   **if** $|C| + |L| < ub$ **then**
17             $ub := |C| + |L|$;
18   **if** $L = \emptyset$ **then**
19                 $C^* := C$;
20   **else**
21   construct C^+ greedily that covers L;
22                 $C^* := C \cup C^+$;
23   remove vertices with the highest $dscore$ from C until $|C| = ub - delta$;
24         $step := step + 1$;
25   **return** C^*;
26 **end**

After the initialization, the loop (lines 8-24) is executed until a limited number of steps denoted by $maxSteps$ is reached. In each iteration, if $ChooseSwapPair$ successfully finds a pair of vertices to swap, then an improving step is executed by swapping the two vertices (lines 10). If $ChooseSwapPair$ fails, which means EWLS gets stuck in a local optimum, then the edge weights are updated by incrementing the weights of all edges in L, by one (line 12). After updating the edge weights, C is perturbed by a random step, which swaps a random vertex $u \in C$ and a random vertex $v \in endpoint(L)$ (line 13). EWLS employs a two-step random process (first picking an uncovered edge randomly, and then picking an end-vertex of that edge randomly) to pick a random vertex $v \in endpoint(L)$. EWLS keeps track of the vertices last inserted into C and last removed from C, and prevents them from being rolled back immediately (lines 14-15).

At the end of each step, if a new upper bound is found, EWLS will do some updatings (lines 16-23). The upper bound ub is updated (line 17), and the best solution C^* is updated in one of two ways (lines 18-22). If $L = \emptyset$, which means C is a vertex cover, C^* is set to be C; otherwise, EWLS extends C into a vertex cover by constructing a vertex set C^+ that covers the uncovered edges, and C^* is updated as $C \cup C^+$. EWLS uses a greedy strategy to construct C^+, which chooses a vertex that covers most uncovered edges each time. Finally, EWLS selects vertices with the highest $dscore$ in C and removes them to maintain $|C| = ub - delta$, and continues to search for a better upper bound (line 23).

3.4. Further Comments on EWLS

In this section, we give some more comments on the EWLS algorithm. We present the $ChooseSwapPair$ function, which gives details of how EWLS selects the exchanging vertex pair. We also present the data structure of the edge sets L and UL, and show the details of maintaining these two sets. An intuitive explanation is also given on the specific strategy for removing vertices when adjusting the size of current partial vertex cover. Finally, we end this section by drawing a summary of EWLS.

The $ChooseSwapPair$ Function

The most important part of EWLS algorithm is the function $ChooseSwapPair$. Its pseudo code is given in Algorithm 2.

Algorithm 2: function ChooseSwapPair

ChooseSwapPair(C,L,UL)
Input: current candidate solution C, uncovered edge set L, edge set UL of uncovered edges unchecked in the current local search stage
Output: a pair of vertices
begin
    if $S := \{(u,v)|u \in C, v \in endpoint(e_{oldest}), u \neq tabuRemove, v \neq tabuAdd$ *and* $score(u,v) > 0\} \neq \emptyset$ **then**
        return $random(S)$;
    else
        foreach $e \in UL$, *from old to young* **do**
            if $S := \{(u,v)|u \in C, v \in endpoint(e),$ $u \neq tabuRemove, v \neq tabuAdd$ *and* $score(u,v) > 0\} \neq \emptyset$ **then**
                return $random(S)$;
    return (0,0);
end

In EWLS, the age of an uncovered edge is the current step number minus the step number at which the edge became uncovered most recently. For example, let the uncovered edge set $L = \{e_1, e_3\}$ at the 100^{th} step; the last time e_1 became uncovered is at the 30^{th} step, and the last time e_3 became uncovered is at the 51^{th} step. Then we say e_1's age is 70, and e_3's age is 49; thus e_1 is older than e_3.

The $ChooseSwapPair$ function chooses a pair of vertices under the constraints $u \neq tabuRemove$, $v \neq tabuAdd$ and $score(u, v) > 0$, where $u \in C$ and $v \in endpoint(L)$. When choosing the vertex v to put into C, the function prefers endpoints of older uncovered edges. In detail, it first checks the oldest edge e_{oldest} in L (refer to $e_{i|L|}$ in Figure 2). If there exist vertex pairs (u, v) that satisfy the constraints (line 3), the function returns one of them randomly. Otherwise, it checks the edges in UL, according to the order from old to young. If for some edge $e \in UL$, there exist vertex pairs (u, v) that satisfy the constraints (line 7), then the function returns one of them randomly. Finally, if the function fails to find such a vertex pair, it then returns (0,0).

Maintaining L and UL

To maintain the set L of uncovered edges, we employ a doubly linked list. During the search procedure, when an edge becomes uncovered, it is inserted into the head of the list, and hence the age of edges in L increases in the order from head to tail. As mentioned before, $UL \subseteq L$ is the set of those unchecked by $ChooseSwapPair$ in the current local search stage. At the beginning of each local search stage we reset UL as L, while as the search goes on, we use a pointer to distinguish UL from the checked edges (Figure 2). Note that whenever an edge becomes uncovered and is inserted into the list, it is considered unchecked. Therefore, in our actual implementation, UL contains the edges that remain unchecked after their latest insertions into the list.

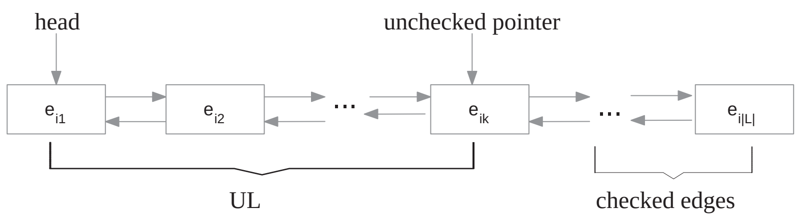

Figure 2: List of L, consisting of UL and the checked edges

Removing Vertices for Adjusting the Size of Partial Vertex Cover

Whenever a new upper bound on the size of the minimum vertex cover is found, EWLS decreases the size of the current partial vertex cover C by removing vertices with the highest $dscore$ from C until $|C| = ub - delta$.

One may notice that a greedy way for adjusting the size of C is to obtain C from C^* by removing vertices with the highest $dscore$ from C^* until $|C| = ub - delta$. However, when finding a new upper bound ($|C|+|L| < ub$), we have $|L| < delta$ since $|C| + delta = ub$ always holds during the search. We shall illustrate in Section 6.3 that the optimal $delta$ is a small number (usually not bigger than 4); thus, when finding a new upper bound, L contains very few edges. So there are unlikely common vertices among these uncovered edges, which means C^+ contains $|L|$ vertices, each for one uncovered edge. Hence, by removing a vertex $v \in C^+$ from C^*, we just make one edge become uncovered, as a result the vertices in C^+ are likely to have the higher $dscores$ (the smaller absolute value of $dscore$) than those in C. Therefore, for convenience, we directly remove those vertices with the highest $dscore$ from C until $|C| = ub - delta$, which can be seen as a two-phased procedure: (1) removing all vertices in C^+ from C^*; (2) removing those vertices with the highest $dscore$ from the remaining vertices in C^*, until there are $(ub - delta)$ vertices left, which constitute the new current partial vertex cover.

To sum up, EWLS strikes a balance between guided search and the diversity. In each step, candidate vertex pairs for swapping are chosen according to an improving heuristic; however, one of them is selected randomly to swap. Moreover, the old-to-young search strategy makes EWLS prefer to cover old uncovered edges, which keeps L lively so that the search region in each local search stage is wide enough for the algorithm to reach a local optimum of high quality. Also, EWLS takes a random step when stuck in local optima to provide additional diversification. Finally, the edge weighting scheme makes EWLS unlikely to converge in a small region by filling up the local optima.

4. Configuration Checking and The EWCC Algorithm

In this section, we propose a novel strategy called $configuration\ checking$ for handling the cycling problem in local search. This significant strategy is used to improve the EWLS algorithm, resulting in a new non-parameter local search algorithm EWCC.

4.1. Configuration Checking

Most successful heuristics for MVC (MIS, MC) adopt hill-climbing heuristics [2, 4, 8, 22, 23, 35, 45, 46, 50], i.e., the search is led towards candidate solutions consisting of the best (or better) vertices under their evaluations. These hill-climbing heuristics may easily encounter the cycling phenomenon, i.e., returning to a candidate solution that has been visited recently. This is because the vertices of high evaluation are likely to be added back to the current candidate solution no long after its last removing. This cycling phenomenon wastes a local search much time and also prevents it from getting out of local minima.

The cycling problem is an inherent problem of local search as the method does not allow our algorithms to memorize, in a compact way, all previously visited parts of the search space. Moreover, it is impractical to incorporate local search with an additional mechanism to remember all previously visited candidate solutions, which requires exponential space and huge time consumption for checking.

To overcome the cycling problem, some naive methods such as random walk and non-improving moves are incorporated into local search algorithms. Besides, a simple trick called *tabu* was proposed by Glover [19, 20] and widely used in local search algorithms [4, 41, 50, 52]. To prevent the local search to immediately return to a previously visited candidate solution and to avoid cycling, the tabu mechanism forbids reversing the recent changes, where the "strength" of forbidding is controlled by a parameter called *tabu tenure*. Recently, a deterministic exploitation strategy which relies on the so-called *promising decreasing variables* was proposed by Li and Huang in [37]. This strategy in some way can reduce the cycling problem of local search algorithms for SAT. When selecting variables to flip, this deterministic exploitation prefers *promising decreasing variables*. The exploitation strategy based on the notion of promising decreasing variable has been used in most wining local search algorithms in the recent SAT competitions. This indicates that successfully handling the cycling problem can significantly influence the performance of local search algorithms.

Although a number of methods have been proposed for handling the cycling problem of local search, for all heuristic algorithms we know, the vertices to add into the current candidate solution are selected solely based on vertex information such as degrees [34, 35], penalties [45, 46], and scores [8, 50], etc. Here we propose a method which as we know is the first time that takes into account the vertices' circumstance. As we mentioned before, it is impractical to remember all previously visited candidate solutions. To make it practicable, a compromising method is to remember each vertex's circumstance information and prevent a vertex to get into the circumstance it just leaves. The intuition behind this idea is that by reducing cycles on local structures of the candidate solution, we reduce cycles on the whole candidate solution.

The above considerations results in a novel heuristic strategy called $configuration\ checking$ that considers the induced subgraph configuration when selecting the vertex to be added. The configuration checking strategy is based on the concept $configuration$, which denotes a vertex's circumstance. Note that there are different versions of definition of $configuraion$ for various application scenarios of the configuration checking heuristic, which mainly includes the unweighted version, the edge weighted version, and the vertex weighted version. We give the definitions of configuration for the unweighted version and the edge weighted version as follows.

Definition 5. Given an undirected graph $G = (V, E)$ and C the current candidate solution, the $state$ of a vertex v $s_v \in \{1, 0\}$, where $s_v = 1$ means $v \in C$, and $s_v = 0$ means $v \notin C$.

Definition 6. (Configuration of unweighted version) Given an undirected graph $G = (V, E)$ and C the current candidate solution, the $configuration$ of a subgraph H of G is a vector S_H consisting of state of all vertices in H.

Definition 7. (Configuration of edge weighted version) Given an edge weighted undirected graph $G = (V, E, w)$ and C the current candidate solution, the $configuration$ of a subgraph H of G is a two-tuples $< S_H, W_H >$, where S_H is a vector consisting of state of all vertices in H, and W_H is a vector consisting of weight of all edges in H.

Similarly, configuration of vertex weighted version can be defined.

As is usually the case, suppose the local search procedure maintains a current candidate solution C, the configuration checking heuristic can be described as following: when selecting a vertex to add into C, for a vertex $v \notin C$, if the configuration of the induced subgraph $G[N[v]]$ (recalling the definitions in Section 2) has not changed since

v's last removing from C, which means the circumstance of v has not changed, then v should not be added back to C. This strategy is reasonable in terms of avoiding cycles; otherwise, the algorithm is led to a scenario it has recently faced to, which is likely to cause a cycle.

Different from previous heuristics which usually refer to the vertex information but neglect its circumstance in making decisions, the configuration checking heuristic takes vertices' circumstance into account. It appears reasonable and helpful to incorporate such a circumstance-concerning strategy to the traditional vertex-based heuristics, as the best decision on a vertex should come from not only its evaluation, but also its circumstance, such as the community it belongs to and the relationship with the community. As mentioned before, the configuration checking strategy is proposed to handle the cycling problem in local search — in this sense it provides an interesting alternative to the standard use of tabu mechanism to avoid the cycling problem in combinatorial search problems.

4.2. The EWCC Algorithm

The configuration checking strategy is tested by implementing it within the EWLS algorithm. This results into a new local search algorithm EWCC that requires no instance-dependent parameters. We adopt the edge weighted version of $configuration$, as EWLS utilizes an edge weighting scheme which updates edge weights of uncovered edges when stuck in local optima. Experimental results show that EWCC makes a significant improvement over EWLS on a wide range of instances.

Maintaining Configuration Changing Information

In order to implement the configuration checking strategy in EWLS, we employ an array $confChange$, whose element is an indicator — $confChange[v] = 1$ means the configuration of $G[N[v]]$ has changed since v's last leaving C; and $confChange[v] = 0$ on the contrary. We maintain the $confChange$ array as follows:

- Rule 1: In the beginning, for each vertex v, $confChange[v]$ is initialized as 1.
- Rule 2: When removing v from C, $confChange[v]$ is reset to 0.
- Rule 3: When u changes its $state$, for each $v \in N(u)\backslash C$, $confChange[v]$ is set to 1.
- Rule 4: When updating the weight of edge $e(u, v)$, both $confChange[u]$ and $confChange[v]$ are set to 1.

Note that the configuration checking strategy is used to decide whether or not a vertex $v \notin C$ can be added into C, judged from the value of $confChange[v]$. So we do not update $confChange[v]$ for a vertex $v \in C$ in the above maintaining scheme, since it is not necessary.

Description of EWCC Algorithm

With the configuration checking strategy, we develop an iterated k-vertex cover algorithm EWCC (abbreviatory for Edge Weighting Configuration Checking), which is obtained from EWLS by applying subtle but significant modifications. An algorithmic description of EWCC is presented in Algorithm 3. EWCC follows the same overall procedure as EWLS; however, it has two distinguishing features as described below:

- EWCC does not utilize the tabu mechanism on selecting the vertex for adding in a swapping step as EWLS; instead, it uses the configuration checking strategy to avoid the cycling problem (see Algorithm 4). We will see from the experimental results that the configuration checking heuristic is more efficient than the tabu mechanism.
- EWCC does not have the $delta$ parameter, in other words, EWCC does not utilize the partial vertex cover concept. Indeed, this parameter is set to 1 permanently in EWCC, which means EWCC is an iterated k-vertex cover algorithm — when finding a k-vertex cover, it removes one vertex from C and goes on to search for a (k-1)-vertex cover, and hence it updates the current best solution in a straightforward way (line 16-18).

We conducted a few experiments to study the performance of these two heuristics, and found that the partial vertex cover concept did not significantly improve the performance of EWCC. It would be interesting to find out why the configuration checking and the partial vertex cover heuristics do not cooperate well. We leave this direction of investigation for future work.

5. Empirical Results

In this section, we first present a brief introduction to the benchmarks we adopted, and describe some preliminaries about our experiments. Then, we divide the experiments into four parts. The purpose of the first part is to demonstrate the performance of EWLS and EWCC in detail, and also to assess the effectiveness of the configuration checking strategy. The second part is to compare EWLS and EWCC with other state-of-the-art heuristic algorithms, and the third and fourth part is to compare EWCC with the state-of-the-art exact algorithms and SAT solvers respectively.

5.1. The Benchmarks

Having a good set of benchmarks is fundamental to demonstrate the effectiveness of new solvers. We use two widely studied well known benchmark sets in MVC (MIS, MC) research, the DIMACS benchmarks and the BHOSLIB benchmarks. Most of the DIMACS instances are structured ones, while the BHOSLIB instances are random ones of high difficulty.

Algorithm 3: EWCC

EWCC($G, maxSteps$)
Input: graph $G = (V, E)$, $maxSteps$
Output: vertex cover of G
begin
    $step := 0$; $L := E$; $UL := E$;
    initialize all edge weights as 1 and compute $dscores$ of vertices;
    initialize $confChange[v]$ as 1 for each vertex v;
    construct C greedily until it's a vertex cover;
    $C^* := C$;
    remove a random vertex with highest $dscore$ from C;
    while $step < maxSteps$ **do**
        if $((u, v) := ChooseSwapPair2(C, L, UL)) \neq (0, 0)$ **then**
            $C := [C \backslash \{u\}] \cup \{v\}$, update the $confChange$ array according to Rule 2 and Rule 3;
        else
            $w(e) := w(e) + 1$ for each $e \in L$, update the $confChange$ array according to Rule 4;
            $C := [C \backslash \{u\}] \cup \{v\}$ where $u := random(C)$ and $v := random(endpoint(L))$, update the $confChange$ array according to Rule 2 and Rule 3;
        $tabuRemove := v$;
        if $L = \emptyset$ **then**
            $C^* := C$;
            remove a random vertex with highest $dscore$ from C;
        $step := step + 1$;
    return C^*;
end

Algorithm 4: function ChooseSwapPair2

ChooseSwapPair2(C,L,UL)
Input: current candidate solution C, uncovered edge set L, edge set UL of uncovered edges unchecked in the current local search stage
Output: a pair of vertices
begin
    if $S := \{(u, v)|u \in C, v \in endpoint(e_{oldest}), u \neq tabuRemove, confChange[v] = 1$ *and* $score(u, v) > 0\} \neq \emptyset$ **then**
        return $random(S)$;
    else
        foreach $e \in UL$, *from old to young* **do**
            if $S := \{(u, v)|u \in C, v \in endpoint(e), u \neq tabuRemove, confChange[v] = 1$ *and* $score(u, v) > 0\} \neq \emptyset$ **then**
                return $random(S)$;
    return (0,0);
end

DIMACS Benchmarks

The DIMACS benchmarks are taken from the Second DIMACS Implementation Challenge (1992-1993)[5]. Thirty seven graphs were selected by the organizers for a summary to indicate the effectiveness of algorithms, comprising the Second DIMACS Challenge Test Problems. These DIMACS MAX-CLIQUE instances were generated from real world problems such as coding theory, fault diagnosis problems and Kellers conjecture, etc, in addition to randomly generated graphs and graphs where the maximum clique has been hidden by incorporating low-degree vertices. These problem instances range in size from less than 50 vertices and 1,000 edges to greater than 3,300 vertices and 5,000,000 edges. The DIMACS benchmarks have been widely used for MVC, MIS and MC algorithms [2, 4, 8, 22, 23, 35, 45, 46, 50].

BHOSLIB Benchmarks

BHOSLIB (Benchmarks with Hidden Optimum Solutions) is a suite of hard random benchmarks, which arises from the SAT'04 Competition[6]. These 40 BHOSLIB instances were translated from SAT instances which were generated randomly in the phase transition area according to the model RB [59]. Generally, those phase-transition instances generated by model RB have been proven to be hard both theoretically [60] and practically [58, 61]. The BHOSLIB benchmarks are famous for their hardness and so influential as strongly recommended by the MC community [23]. They have been widely used in the recent literature as a reference point for new heuristics to MVC, MIS and MC [7]. Besides these 40 instances, there is a large instance with 4,000 vertices and 572,774 edges, which is designed for challenge.

5.2. Experiment Preliminaries

Before we discuss the experimental performance results, let us introduce some preliminary information about our experiments.

- **Implementation:** Both EWLS and EWCC are implemented in C++; the alternative heuristic solvers are also implemented in C++ by their authors. We compile all these 4 solvers by the g++ compiler with the '-O2' option.

- **Computing Platform:** All experiments were run on a 3 GHz Intel Core 2 Duo CPU E8400 and 4GB RAM under Linux. To execute the DIMACS machine benchmarks[8], this machine required 0.193 CPU seconds for r300.5, 1.118 CPU seconds for r400.5 and 4.242 CPU seconds for r500.5.

- **Termination Criterion:** Most empirical results on the performance of algorithms found in the literature are in the form of statistics on the solution quality and the average CPU time over successful trials obtained after a fixed run-time. This is a direct way to compare the efficiency of different algorithms, and is

[5]ftp://dimacs.rutgers.edu/pub/challenges

[6]http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

[7]http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/list-graph-papers.htm

[8]ftp://dimacs.rutgers.edu/pub/dsj/clique/

adopted in algorithm competitions such as the SAT competitions and MAX-SAT evaluations. We also adopt this termination criterion. In our experiments, each run terminates upon reaching a given cutoff time.

- **Result Reporting Methodology:** For each instance, each algorithm performs 100 independent runs within a cutoff time with different random seeds. We report the following information: the optimal (or minimum known) vertex cover size (k^*); the cutoff time ("CT"); the number of successful runs in which a solution of size k^* is found ("suc"); and the run-time in CPU seconds to find the best solution averaged over all successful runs ("time"). The results in bold indicate the best performance for an instance.

  In the detailed performance report of our two algorithms (Section 5.3), we also report the number of steps to find the best solution averaged over all successful runs, and the optimized $delta$ (d) for EWLS.

5.3. Performance of EWLS and EWCC

Performance on DIMACS Benchmarks

Table 1 illustrates the performance of EWLS and EWCC on the DIMACS benchmarks. Most instances are very easy and they can be solved by both algorithms within 1 second. In 34 of the 37 instances, EWLS and EWCC find an optimal (or best known) solution. Note that 2 of the 3 failed instances are $brock$ graphs. Furthermore, of the 34 successful instances, EWLS and EWCC do so consistently, i.e. in all 100 runs for 32 and 31 instances respectively. Overall, both algorithms achieve excellent results but for the special $brock$ artificial graphs. The instances where the optimal size is not reached consistently by both algorithms are summarized in Table 2.

As is clear from Table 1, EWCC outperforms EWLS on most instances, including some hard instances such as the $brock$ instances, $C2000.9$, and $keller6$. Nevertheless, EWCC shows a significant degraded performance on two $MANN$ instances.

Since the only algorithmic difference between EWCC and EWLS is that EWCC employs the configuration checking strategy instead of the tabu mechanism on choosing a vertex to add into the current candidate solution, we attribute the performance improvement to the configuration checking strategy reducing cycling steps. An empirical evidence for this conjecture is that EWCC shows an improvement in step performance over EWLS on most DIMACS instances.

Performance on BHOSLIB Benchmarks

Table 3 presents performance of EWLS and EWCC on the hard random BHOSLIB benchmarks. Both EWLS and EWCC successfully solve all BHOSLIB instances in terms of reaching an optimal solution, and the worst solution they find never exceeds $k^* + 1$. Moreover, referring to other works on this suite of benchmarks[9], including those on the equivalent benchmarks for other problems such as SAT, MAXSAT, CSP, etc, EWCC and EWLS solve the most instances with 100% success rate (29 of 40 instances) while requiring less time.

[9]http://www.nlsde.buaa.edu.cn/~kexu

Graph			EWLS				EWCC		
Instance	k^*	CT(s)	d	suc	time	steps	suc	time	steps
brock200_2	188	1000	1	100	0.264	119476	100	**0.244**	**117800**
brock200_4	183	1000	1	100	3.184	1696748	100	**2.080**	**1233612**
brock400_2	371	1000	1	4	338.641	122253257	**12**	374.520	185029672
brock400_4	367	1000	1	100	74.146	27107873	100	**25.376**	**10305332**
brock800_2	776	1000	1	0	n/a	n/a	0	n/a	n/a
brock800_4	774	1000	1	0	n/a	n/a	0	n/a	n/a
C125.9	91	1000	1	100	< 0.001	**70**	100	< 0.001	73
C250.9	206	1000	1	100	< 0.001	**1541**	100	< 0.001	1743
C500.9	443	1000	1	100	0.236	115481	100	**0.199**	**99203**
C1000.9	932	1000	1	100	3.326	896259	100	**2.657**	**758546**
C2000.5	1984	1000	1	100	2.471	**73177**	100	**2.429**	76000
C2000.9	1921	2400	1	21	557.745	66666972	**34**	858.733	107898557
C4000.5	3982	2400	1	100	**686.472**	**7257183**	100	738.909	8802400
DSJC500.5	487	1000	1	100	0.018	3179	100	**0.014**	**2344**
DSJC1000.5	985	1000	1	100	0.798	69431	100	**0.592**	**58466**
gen200_p0.9_44	156	1000	1	100	< 0.001	2434	100	< 0.001	**1296**
gen200_p0.9_55	145	1000	1	100	< 0.001	299	100	< 0.001	**242**
gen400_p0.9_55	345	1000	1	100	0.074	41906	100	**0.049**	**29450**
gen400_p0.9_65	335	1000	1	100	< 0.001	1724	100	< 0.001	**1586**
gen400_p0.9_75	325	1000	1	100	< 0.001	1074	100	< 0.001	**920**
hamming8-4	240	1000	1	100	< 0.001	1	100	< 0.001	1
hamming10-4	984	1000	1	100	0.030	5010	100	**0.019**	**2913**
keller4	160	1000	1	100	< 0.001	74	100	< 0.001	**70**
keller5	749	1000	1	100	0.025	5314	100	**0.019**	**4373**
keller6	3302	1000	1	100	4.934	226592	100	**3.757**	**185507**
MANN_a27	252	1000	1	100	0.002	**7195**	100	0.002	8105
MANN_a45	690	2400	1	**100**	**194.969**	**124004605**	94	698.505	404529167
MANN_a81	2221	2400	3	**22**	**621.683**	207424642	1	634.460	145174870
p_hat300-1	292	1000	1	100	0.002	215	100	0.002	**148**
p_hat300-2	275	1000	1	100	0.001	80	100	0.001	**75**
p_hat300-3	264	1000	1	100	< 0.001	794	100	< 0.001	**679**
p_hat700-1	689	1000	1	100	0.035	2363	100	**0.028**	**1794**
p_hat700-2	656	1000	1	100	0.009	222	100	0.009	**212**
p_hat700-3	638	1000	1	100	0.007	970	100	**0.006**	**884**
p_hat1500-1	1488	1000	1	100	13.587	419374	100	**9.784**	**334296**
p_hat1500-2	1435	1000	1	100	0.091	906	100	**0.081**	**708**
p_hat1500-3	1406	1000	1	100	0.053	3309	100	**0.042**	**2779**

Table 1: Results on DIMACS benchmarks. To obtain a meaningful comparison, k^* for C2000.9 is set to 1921 which is known to be 1920, obtained within 10^9 steps, 9 hours [23]. In order to maximize the probability of reaching the optimal solution in every run, the cutoff time for $C2000.9$, $C4000.5$ and two large $MANN$ instances is set to 2400 seconds.

As can be seen from Table 3, EWCC shows a significant improvement over EWLS. We emphasize that EWCC has no instance-dependent parameter while the performance of EWLS is given by optimizing the $delta$ parameter. For instances solved by both algorithms with 100% success rate, EWCC always finds an optimal solution more quickly. For other instances, EWCC succeeds in more runs than EWLS, except frb59-26-2.

Again, as in the DIMACS benchmarks, EWCC shows an improvement in step performance on BHOSLIB instances, which is more significant — from frb35-17-2 on, for instances solved by both algorithms with 100% success rate, EWCC always spends less steps, except frb56-25-3. Based on these results, we conjecture that the configuration checking strategy is effective in reducing cycling steps in local search.

On the other hand, noticing that EWCC performs more steps but a little faster on frb56-25-3, one might speculate that the good performance of EWCC is a consequence of more efficient implementation. However, EWCC is actually implemented on the codes of EWLS, just by adapting a few codes for the configuration checking strategy.

Graph Instance	EWLS size	EWLS suc	EWLS time	EWLS steps	EWCC size	EWCC suc	EWCC time	EWCC steps
brock400_2	371(374.84,375)	4	338.641	122253257	371(374.52,375)	12	374.520	185029672
brock800_2	779	100	0.366	59157	779	100	0.489	75309
brock800_4	779	100	0.617	98963	779	100	0.619	103494
C2000.9	1921(1921.84,1923)	21	557.745	66666972	1921(1921.66,1922)	34	858.733	107898557
MANN_a45	690	100	194.969	124004605	690(690.06,691)	94	698.505	404529167
MANN_a81	2221(2221.78,2222)	22	621.683	207424642	2221(2222.89,2223)	1	634.460	145174870

Table 2: Results of DIMACS instances where the optimal size was not reached consistently within the cutoff time by both EWLS and EWCC. size = min(average, max) vertex cover size; suc = number of runs which find the best solution; run-time and steps are averaged over all runs that find the best solution.

Graph			EWLS				EWCC		
Instance	k^*	CT(s)	d	suc	time	steps	suc	time	steps
frb30-15-1	420	1000	2	100	0.069	**29095**	100	**0.062**	29854
frb30-15-2	420	1000	1	100	0.096	43222	100	**0.090**	**42847**
frb30-15-3	420	1000	2	100	0.465	199161	100	**0.329**	**160758**
frb30-15-4	420	1000	2	100	0.071	**30075**	100	**0.069**	33223
frb30-15-5	420	1000	2	100	0.246	106133	100	**0.169**	**82382**
frb35-17-1	560	1000	2	100	1.178	**419575**	100	**1.061**	436347
frb35-17-2	560	1000	2	100	1.316	460821	100	**0.899**	**363737**
frb35-17-3	560	1000	2	100	0.305	112705	100	**0.209**	**88393**
frb35-17-4	560	1000	2	100	1.326	476318	100	**1.057**	**441708**
frb35-17-5	560	1000	2	100	0.711	251727	100	**0.580**	**236252**
frb40-19-1	720	1000	2	100	0.722	226711	100	**0.552**	**199420**
frb40-19-2	720	1000	2	100	17.680	5216744	100	**11.295**	**3935909**
frb40-19-3	720	1000	3	100	3.626	1014101	100	**2.965**	**1012203**
frb40-19-4	720	1000	1	100	18.751	5735325	100	**13.787**	**4592540**
frb40-19-5	720	1000	2	100	87.495	25283646	100	**41.714**	**14231964**
frb45-21-1	900	1000	4	100	14.594	3224649	100	**9.066**	**2583572**
frb45-21-2	900	1000	2	100	26.637	6507751	100	**14.927**	**4245886**
frb45-21-3	900	1000	2	100	96.047	23568777	100	**56.404**	**16118564**
frb45-21-4	900	1000	2	100	18.628	46633662	100	**14.671**	**4290939**
frb45-21-5	900	1000	3	100	84.679	19147891	100	**41.861**	**11865589**
frb50-23-1	1100	1800	2	100	262.313	56455242	100	**123.803**	**29773443**
frb50-23-2	1100	1800	2	59	509.296	106684368	**80**	631.688	154734494
frb50-23-3	1100	1800	1	42	692.210	155456585	**54**	797.524	189561481
frb50-23-4	1100	1800	1	100	31.573	7299068	100	**23.847**	**5792613**
frb50-23-5	1100	1800	2	100	117.730	25697612	100	**84.832**	**20774461**
frb53-24-1	1219	2000	4	25	745.425	129528395	**30**	985.294	217902696
frb53-24-2	1219	2000	3	58	603.768	109004768	**81**	772.619	170480847
frb53-24-3	1219	2000	2	100	165.720	32678443	100	**116.726**	**26143480**
frb53-24-4	1219	2000	2	50	772.885	147937643	**81**	**642.068**	**140122496**
frb53-24-5	1219	2000	2	100	250.262	48559793	100	**125.366**	**27379752**
frb56-25-1	1344	2200	3	28	742.227	124088139	**62**	826.916	165580423
frb56-25-2	1344	2200	2	32	730.987	134692382	**54**	868.426	178346136
frb56-25-3	1344	2200	4	100	307.964	**51838431**	100	**285.046**	57511723
frb56-25-4	1344	2200	4	100	234.376	39241213	100	**183.438**	**37005137**
frb56-25-5	1344	2200	3	100	94.242	16390263	100	**79.865**	**15955311**
frb59-26-1	1475	2400	2	21	1015.145	171236143	**22**	**1012.230**	186973450
frb59-26-2	1475	2400	3	**18**	**1116.251**	**179661855**	8	1139.360	209757895
frb59-26-3	1475	2400	3	42	664.507	103219920	**65**	913.653	168513191
frb59-26-4	1475	2400	4	18	841.930	125795679	**26**	1080.650	197407398
frb59-26-5	1475	2400	1	100	334.044	60010102	100	**173.675**	**32789882**

Table 3: Results on BHOSLIB benchmarks

The real reason of the small disagreement between time performance and step performance is that EWLS sets a large $delta$ to perform well for this instance, which results in a large L, and hence large complexity per step, as we will discuss the impact of the $delta$ parameter on the performance of EWLS in Section 6.3.

New Record for The 20-year Challenge Instance

For the challenge instance $frb100$-40, which has a minimum vertex cover of size 3900, the designer of the BHOSLIB benchmarks conjectured in 2005 that this instance would not be solved on a PC in less than 24 hours within the next two decades[10]. The last best known 3903-solution was established by the solver COVER on Jul. 4, 2007. More specifically, the author ran the solver 100 times with different seeds. Of those 100 runs, 25 found a solution of size 3903, and the other runs all found a solution of size 3904. Of the 25 successful runs that found the better solution, the quickest one did so in 71.09 seconds while the median run-time was 1193.92 seconds[7]. It should be noted that COVER runs by given k^* in advance, which makes the problem easier but is not natural for MVC.

We run EWLS (with $delta = 6$) and EWCC 100 independent trials given cutoff time 4800 seconds on this instance, respectively. For EWLS, 4 runs find a 3902-sized vertex cover, or equivalently, a 98-sized independent set (reported in Appendix B) with the average time of 2823.05 seconds and the quickest one does so in 1239.87 seconds, and 59 runs find a 3903-sized solution with average time of 2470.39; the remaining runs all find a 3904-sized solution. For EWCC, 1 run finds a 3902-sized vertex cover in 2856.43 seconds, and 82 runs find a 3903-sized solution with average time of 2145.09 seconds; the remaining runs all find a 3904-sized solution.

5.4. Comparison with Other Heuristics

We compare our algorithms with the following two heuristic algorithms:

- **PLS** [46]: Heuristic Max-Clique algorithm. It is the improved version of the stochastic local search algorithm DLS-MC [45]. Shown in [45], DLS-MC dominates other state-of-the-art MaxClique solvers over a large range of the DIMACS instances. While in [46], PLS is compared directly with DLS-MC by measuring performance using the same techniques as utilized in [45], and the results indicate that PLS is comparable or more efficient than DLS-MC for all DIMACS instances. Therefore, PLS can be regarded as one of the best MC heuristic algorithms (to the best of our knowledge).
- **COVER** [50]: Heuristic Minimum Vertex Cover algorithm. It is proposed in [50], which was one of the three candidates of the Springer Best Paper Award at the 30th Annual German Conference on Artificial Intelligence[11]. Shown by experimental results [50], COVER is competitive with the state-of-the-art stochastic local search algorithm DLS-MC on DIMACS benchmarks and is the best algorithm on the BHOSLIB benchmarks as we know. Specially, it establishes new

[10] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm
[11] http://www.ki2007.uos.de/

Graph			EWLS		EWCC		PLS		COVER	
Instance	k^*	CT(s)	suc	time	suc	time	suc	time	suc	time
brock400_2	371	1000	4	338.641	12	347.520	**100**	**0.145**	3	228.520
brock400_4	367	1000	100	74.146	100	25.376	**100**	**0.029**	47	702.573
brock800_2	776	1000	0	n/a	0	n/a	**100**	**3.886**	0	n/a
brock800_4	774	1000	0	n/a	0	n/a	**100**	**1.314**	0	n/a
C2000.9	1921	2400	21	557.745	**34**	**858.733**	34	1014.611	7	1337.397
C4000.5	3982	2400	100	686.472	100	738.909	**100**	**66.702**	100	658.33
gen400_p0.9_55	345	1000	100	0.074	**100**	**0.049**	100	15.171	100	0.353
keller6	3302	1000	100	4.934	**100**	**3.757**	90	342.972	100	68.214
MANN_a45	690	2400	**100**	**194.969**	94	698.505	1	985.910	97	643.638
MANN_a81	2221	2400	**22**	**621.683**	1	634.460	0	n/a	2	1875.320
p_hat1500-1	1488	1000	100	13.587	100	9.784	**100**	**2.355**	100	18.095

Table 4: Comparative results on DIMACS benchmarks. To obtain a meaningful comparison, k^* for $C2000.9$ is set to 1921 which is known to be 1920. In order to maximize the probability of reaching the optimal solution in every run, the cutoff time for $C2000.9$, $C4000.5$ and two large $MANN$ instances is set to 2400 seconds.

records on three very large problem instances — $MANN\_a45$, $MANN\_a81$, and the challenge instance $frb100$-40.

To sum up, PLS and COVER are the best known heuristic algorithm to date in literature for the MC and the MVC problem respectively. Also the best MIS heuristic algorithm we find in literature [2] is also dominated by DLS-MC (also by PLS) except two $MANN$ instances. Therefore, we consider these two algorithms (PLS and COVER) as the most suitable candidates to be used in our comparative study.

Comparative Results on DIMACS Benchmarks

The comparative results on the Second DIMACS Challenge Test Problems are shown in Table 4. Most DIMACS instances are too easy for a modern solver. The instances not appearing in the table are solved by all solvers with 100% success rate within 4 seconds. Overall, the results from the comparative performance on these DIMACS instances can be summarized as follows:

- EWLS is dominant for the $MANN\_a45$ and $MANN\_a81$ instances.
- EWCC is dominant for the $C2000.9$, $keller6$ and $gen400\_p0.9\_55$ instances.
- PLS is dominant for four $brock$ instances, the $C4000.5$ and $p\_hat1500$-1 instances.

The results show EWLS and EWCC are competitive with PLS, given the fact that they perform better on some hard instances but fail on a class of artificial graphs ($brock$). Particularly, the performance of EWLS on two large $MANN$ instances and the performance of EWCC on $keller6$ are (as far as we know) the best reported, and indicates a significant improvement on the best known algorithms (PLS and COVER) in literature.

Nevertheless, EWLS, EWCC and COVER all fail on some $brock$ graphs which are generated by explicitly incorporating low-degree vertices into the optimal vertex cover to defeat greedy heuristics. Indeed, most algorithms preferring higher-degree vertices such as GRASP, RLS, and k-opt also failed in these graphs. Remark that, PLS performs

Graph			EWLS		EWCC		PLS		COVER	
Instance	k^*	CT(s)	suc	time	suc	time	suc	time	suc	time
frb40-19-1	720	1000	100	0.722	**100**	**0.552**	100	10.421	100	1.580
frb40-19-2	720	1000	100	17.680	**100**	**11.295**	100	85.254	100	17.180
frb40-19-3	720	1000	100	3.626	**100**	**2.965**	100	9.058	100	5.059
frb40-19-4	720	1000	100	18.751	100	13.787	100	77.393	**100**	**11.794**
frb40-19-5	720	1000	100	87.495	**100**	**41.714**	86	373.058	100	124.153
frb45-21-1	900	1000	100	14.594	**100**	**9.066**	100	52.175	100	14.427
frb45-21-2	900	1000	100	26.637	**100**	**14.927**	100	170.308	100	37.692
frb45-21-3	900	1000	100	96.047	**100**	**56.404**	8	442.780	100	109.831
frb45-21-4	900	1000	100	18.628	**100**	**14.671**	100	111.375	100	20.888
frb45-21-5	900	1000	100	84.679	**100**	**41.861**	97	248.291	100	105.138
frb50-23-1	1100	1800	100	262.313	**100**	**123.803**	30	859.997	100	267.752
frb50-23-2	1100	1800	59	509.296	**80**	631.688	3	547.666	48	594.427
frb50-23-3	1100	1800	42	692.210	**54**	797.524	2	1429.810	37	585.273
frb50-23-4	1100	1800	100	31.573	**100**	**23.847**	100	92.946	100	32.734
frb50-23-5	1100	1800	100	117.730	**100**	**84.832**	77	680.537	100	167.929
frb53-24-1	1219	2000	25	745.425	**30**	985.294	1	231.270	17	796.249
frb53-24-2	1219	2000	58	603.768	**81**	772.619	6	1316.510	50	558.181
frb53-24-3	1219	2000	100	165.720	**100**	**116.726**	20	856.849	99	256.704
frb53-24-4	1219	2000	50	772.885	**81**	**642.068**	21	962.502	48	810.777
frb53-24-5	1219	2000	100	250.262	**100**	**125.366**	10	1549.25	95	341.714
frb56-25-1	1344	2200	28	742.227	**62**	826.916	1	1259.190	26	839.331
frb56-25-2	1344	2200	32	730.987	**54**	868.426	0	n/a	17	815.253
frb56-25-3	1344	2200	100	307.964	**100**	**285.046**	0	n/a	97	492.755
frb56-25-4	1344	2200	100	234.376	**100**	**183.438**	12	1225.533	93	362.35
frb56-25-5	1344	2200	100	94.242	**100**	**79.865**	30	959.249	100	168.166
frb59-26-1	1475	2400	21	1015.145	**22**	**1012.230**	0	n/a	17	883.755
frb59-26-2	1475	2400	**18**	**1116.251**	8	1139.360	0	n/a	9	678.554
frb59-26-3	1475	2400	42	664.507	**65**	913.653	6	1537.731	26	1149.02
frb59-26-4	1475	2400	18	841.930	**26**	1080.650	1	2224.220	4	1596.32
frb59-26-5	1475	2400	100	334.044	**100**	**173.675**	37	1202.803	98	430.602

Table 5: Comparative results on BHOSLIB benchmarks

well on $brock$ family because it comprises three sub-algorithms, one of which favors the lower degree vertices.

Comparative Results on BHOSLIB Benchmarks

Comparative results on BHOSLIB benchmarks are shown in Table 5. For concentrating on the considerable gaps in comparisons, we do not report the two groups of small instances (frb30, frb35), as they can be solved in a few seconds, and the corresponding results are consistent with Table 5. Table 5 illustrates that EWCC and EWLS are the best two algorithms for the BHOSLIB benchmarks. EWCC is the best solver for all instances in terms of both quality and run-time, with the exceptions of frb40-19-4 whose dominant algorithm is COVER, and frb59-26-2 whose dominant algorithm is EWLS. The results undoubtedly demonstrate that EWCC delivers the best performance on the BHOSLIB benchmarks, which remains justifiable when referring to other works on this suite of benchmarks[12], including those on its equivalent benchmarks of other problems, such as SAT, MAXSAT, CSP, etc.

[12] http://www.nlsde.buaa.edu.cn/˜kexu

5.5. Comparison with Exact Algorithms

Generally, exact algorithms and local search algorithms are somewhat complementary in their applications. As is usually the case, exact algorithms perform better on structured instances while local search algorithms perform better on random ones. For example, this phenomenon is typical in SAT solving, which is clear from the results of SAT competitions[13]. A natural question is then whether this phenomenon also happens in MVC (MIS, MC) solving.

For these three problems MVC, MIS and MC, most of exact algorithms are designed for the MaxClique problem [9, 13, 39, 40, 44, 49, 56]. The recent branch-and-bound MC algorithm called MaxCLQ which utilizes MaxSAT technologies to improve upper bounds [39] shows a considerable progress. The evaluation in [39] on some random graphs and the DIAMCS benchmarks indicates MaxCLQ significantly outperforms previous exact MC algorithms. Further, the MaxCLQ algorithm is improved using two strategies called Extended Failed Literal detection and Soft Clause Relaxation, resulting in a better algorithm denoted by MaxCLQdyn+EFL+SCR [40]. Due to the great success of MaxCLQdyn+EFL+SCR, we compare our algorithm only with MaxCLQdyn+EFL+SCR.

We compare EWCC with MaxCLQdyn+EFL+SCR on the DIMACS benchmarks. The results of MaxCLQdyn+EFL+SCR is taken from [40]. MaxCLQdyn+EFL+SCR is not evaluated on the BHOSLIB benchmarks which are harder and require more effective technologies for exact algorithms [40].

The run-times of MaxCLQdyn+EFL+SCR are obtained on a 3.33 GHz Intel Core 2 Duo CPU with linux and 4 Gb memory, which required 0.172 seconds for r300.5, 1.016 seconds for r400.5 and 3.872 seconds for r500.5 to execute the DIMACS machine benchmarks [40]. The corresponding run-times for our machine is 0.193, 1.118, 4.242. So, we multiply the reported run-times of MaxCLQdyn+EFL+SCR by 1.098 (=(4.242/3.872+1.118/1.016)/2=1.098, the average of the two largest ratios). This normalization is based on the way established in the Second DIMACS Implementation Challenge for Cliques, Coloring, and Satisfiability, and is widely used for comparing different MaxClique algorithms [39, 40, 45, 46].

Graph Instance	k^*	EWCC suc	EWCC time	MaxCLQdyn+EFL +SCR time	Graph Instance	k^*	EWCC suc	EWCC time	MaxCLQdyn+EFL +SCR time
brock400_1	373	0	n/a	**286.578**	p_hat300-3	264	100	**0.001**	1.307
brock400_2	371	12	374.520	**125.062**	p_hat500-3	450	100	**0.007**	50.409
brock400_3	369	100	**45.054**	251.442	p_hat700-2	656	100	**0.009**	3.272
brock400_4	367	100	**25.376**	119.243	p_hat700-3	638	100	**0.006**	1141.920
brock800_1	777	0	n/a	**5861.124**	p_hat1000-2	954	100	**0.022**	108.944
brock800_2	776	0	n/a	**5138.098**	p_hat1000-3	932	100	**0.029**	113860.404
brock800_3	775	0	n/a	**3298.392**	p_hat1500-1	1488	100	9.784	**3.096**
brock800_4	774	0	n/a	**2391.444**	p_hat1500-2	1435	100	**0.081**	866.514
keller5	749	100	**0.019**	6884.461	sanr200_0.9	158	100	**0.001**	5.20
MANN_a45	690	94	698.505	**21.169**	sanr400_0.7	379	100	**0.012**	97.722

Table 6: EWCC performance as compared to the best exact MaxClique algorithm MaxCLQdyn+EFL+SCR for the DIMACS benchmarks.

Table 6 presents the performance of EWCC and MaxCLQdyn+EFL+SCR on the

[13] http://www.satcompetition.org/

DIMACS benchmarks. Those instances that can be solved in 1 second by both algorithms are not reported.

The results suggest that EWCC is much better on random instances such as the p_hat and $sanr$ instances. We believe that EWCC outperforms MaxCLQdyn+EFL+SCR on other hard random benchmarks like BHOSLIB ones, as MaxCLQdyn+EFL+SCR are not evaluated due to their high hardness [40], while EWCC performs well.

For hard structured instances, we note that MaxCLQdyn+EFL+SCR is mainly evaluated on the $brock$ instances where EWCC performs worst, but not evaluated on the five open DIMACS instances ($MANN\_a81$, $hamming10$-4, $johnson32$-2-4, $keller6$, and $p\_hat1500$-3) [40]. Although MaxCLQdyn+EFL+SCR overall performs better, EWCC also outperform MaxCLQdyn+EFL+SCR significantly on some structured instances, such as the two $brock$ instances and the $keller5$ instance. Moreover, local search algorithms based on the vertex penalty mechanism such as DLS-MC [45] and PLS [46] significantly outperform MaxCLQdyn+EFL+SCR on the $brock$ instances.

Based on the comparative results, we conclude that in MVC (MIS, MC) solving, local search algorithms are significant better at random instances. More interestingly, local search algorithms are competitive with exact algorithms on structured instances.

5.6. Comparison with SAT solvers

We also compare the performance of EWCC with the results from the SAT Competition 2004, as some corresponding SAT instances of the BHOSLIB benchmarks were also used as handmade benchmarks for the SAT Competition 2004 (55 SAT solvers)[14].

Instance	SAT 2004 results	EWCC suc	EWCC time	Instance	SAT 2004 results	EWCC suc	EWCC time
frb40-19-1	Solved by 28 solvers	100	0.552	frb53-24-1	Unsolved	6	152.252
frb40-19-2	Solved by 27 solvers	100	11.295	frb53-24-2	Unsolved	21	131.210
frb45-21-1	Solved by 8 solvers	100	9.066	frb56-25-1	Unsolved	19	135.085
frb45-21-2	Solved by 5 solvers	100	14.927	frb56-25-2	Unsolved	18	113.936
frb50-23-1	Solved by 1 solver	91	92.471	frb59-26-1	Unsolved	4	112.193
frb50-23-2	Solved by 1 solver	28	137.158	frb59-26-2	Unsolved	1	56.150

Table 7: EWCC performance as compared to SAT Competition 2004 for the BHOSLIB benchmarks. The cutoff time for EWCC is set to be 300 seconds in this table. For each instance, the number of SAT solvers (from a total of 55 solvers) in the first stage of the SAT 2004 Competition that were able to solve the corresponding SAT problem, the number of successful trials (from a total of 100) in which EWCC located the optimal minimum vertex cover, and the average time over these successful trials are shown

The SAT Competition 2004 was based on a two-stage ranking. In the first stage, all 55 solvers were run on each instance one time within 600 seconds. The competition ran on two clusters of Linux boxes. One was composed of Athlon 1800+ with 1 GB memory, and the other was composed of Intel Xeon 2.4 GHz with 1 GB memory [5]. In order to obtain a meaningful comparison, the cutoff time for EWCC is set to be 300 seconds (half of the cutoff time in the first stage of the competition) for each instance. The reader may be concerned by the fact that such a scaling may lead to some loss of

[14] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm

accuracy, nevertheless the considerable differences are indeed worthwhile to note. As can be seen from Table 7, EWCC achieves a significant improvement on the results of SAT Competition 2004.

6. Discussion

In this section, we perform additional empirical analysis to gain a deeper understanding of the run-time behavior of our two algorithms and the efficacy of their underlying mechanisms. Specifically, we explore the variability in run-time between multiple independent runs on the same problem instance; the effectiveness of the edge weighting scheme; the impact of the $delta$ parameter on the performance of EWLS; the efficacy of the old-to-young strategy; and more insights into the configuration checking strategy.

6.1. Variability in Run-Time

The variability of run-time between multiple independent runs on a given instance is an important aspect of the behavior of stochastic local search algorithms. Hoos and Stützle proposed a methodology for studying this aspect based on run-time distributions (RTDs) and run-length distributions (RLDs) [28, 29], which has been widely used in empirical analysis of stochastic local search algorithms in AI community [27, 45, 52, 57]. We follow this methodology for our analysis.

The investigation is performed using two DIMACS instances and two BHOSLIB instances. For DIMACS benchmarks, we select $C1000.9$ and $p\_hat1500$-1, both of which are of reasonable size and difficulty. $C1000.9$ is a density graph and $p\_hat1500$-1 has a wide vertex degree range. For BHOSLIB benchmarks, $frb53$-24-5 and $frb59$-26-5 are selected. They are typical instances for studying the variability of run-time of EWLS and EWCC since they are neither too easy that can be solved very quickly nor too hard to reach a good success rate.

The empirical RTD and RLD graphs of EWLS and EWCC are shown in Figure 3 and Figure 4, for the DIMACS instances and BHOSLIB instances respectively (each instance is based on 100 independent runs that all reached the respective optimal solution). According to the graphs, both EWLS and EWCC show a large variability in run-time (also in run-length). Further investigation shows that the RLDs and RTDs are quite well approximated by exponential distributions, labeled $ed[m](x) = 1 - 2^{-x/m}$, where m is the median of the distribution. To test the goodness of the approximations, we use $Kolmogorov\text{-}Smirnov$ tests, which fail to reject the null hypothesis that the sampled run-times stem from the exponential distributions shown in the figures at a standard significance level $\alpha = 0.05$ with $p\text{-}values$ between 0.19 and 0.96.

This observation of exponential RLDs and RTDs is consistent with similar results for other high performance SLS algorithms, e.g., for MaxClique [45], for SAT [27, 28], for MAXSAT [52], and for scheduling problems [57]. By the arguments [27–29] made for stochastic local search algorithms that characterized by an exponential RTD (and RLD), we conclude that, for EWLS and EWCC, the probability of finding an optimal solution within a fixed amount of time (or steps) does not depend on the number of search steps done in the past. Consequently, they are robust $w.r.t.$ the cutoff parameters like the $maxsteps$ parameter and thus, the number of random restarts. Hence,

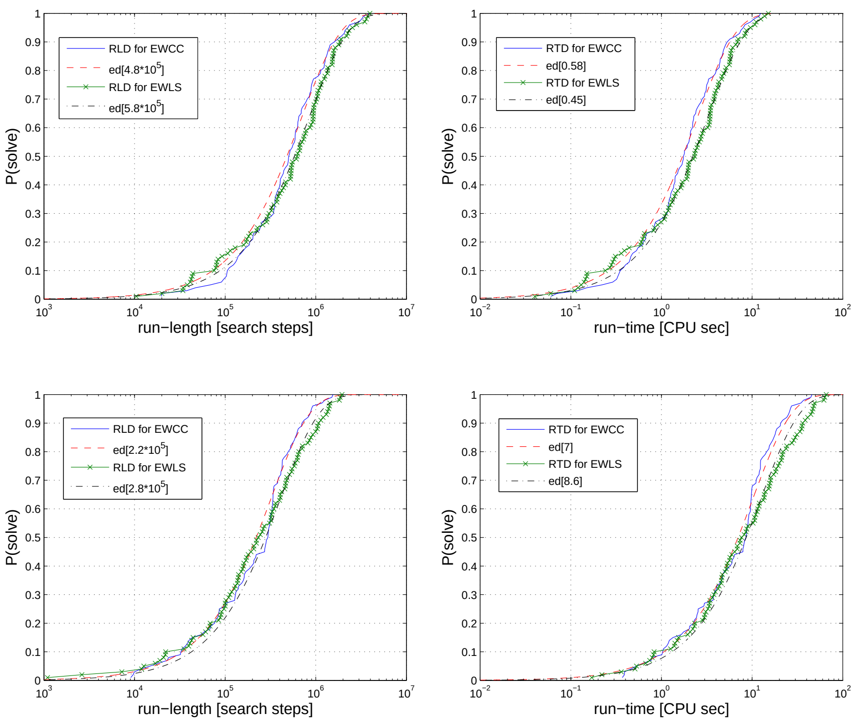

Figure 3: Run-length distributions (left) and run-time distributions (right) for EWLS and EWCC applied to $C1000.9$ (top) and $p\_hat1500$-1 (bottom); these empirical RLDs and RTDs are well approximated by exponential distributions, labeled $ed[m](x) = 1 - 2^{-x/m}$ in the plots.

performing multiple independent runs of them in parallel will result in close-to-optimal parallelization speedup. Similar observations were made for most of the other difficult DIMACS instances and BHOSLIB instances.

These RTD and RLD graphs show that EWCC outperforms EWLS, which is more significant on the two BHOSLIB instances. This is also demonstrated by Table 1 and Table 3 in Section 5.

6.2. The Effectiveness of Edge Weighting Scheme

We have demonstrated the effectiveness of the configuration checking strategy by comparing the performance of EWLS and EWCC, which shows that EWCC outperforms EWLS significantly on most DIMACS benchmarks and BHOSLIB benchmarks. Here we study the effectiveness of the edge weighting scheme in the same way, by comparing the performance of EWLS and its alternative version which we refer as EWLS_0 without edge weighting scheme. EWLS_0 works all the same with EWLS, except for not updating the edge weights (that is, deleting line 12 in Algorithm 1), in other words, the edge weight of each edge is always 1 during the search procedure.

We perform EWLS_0 on several hard instances with the optimal $delta$ parameter settings. The results show that the performance of EWLS_0 degrades significantly on

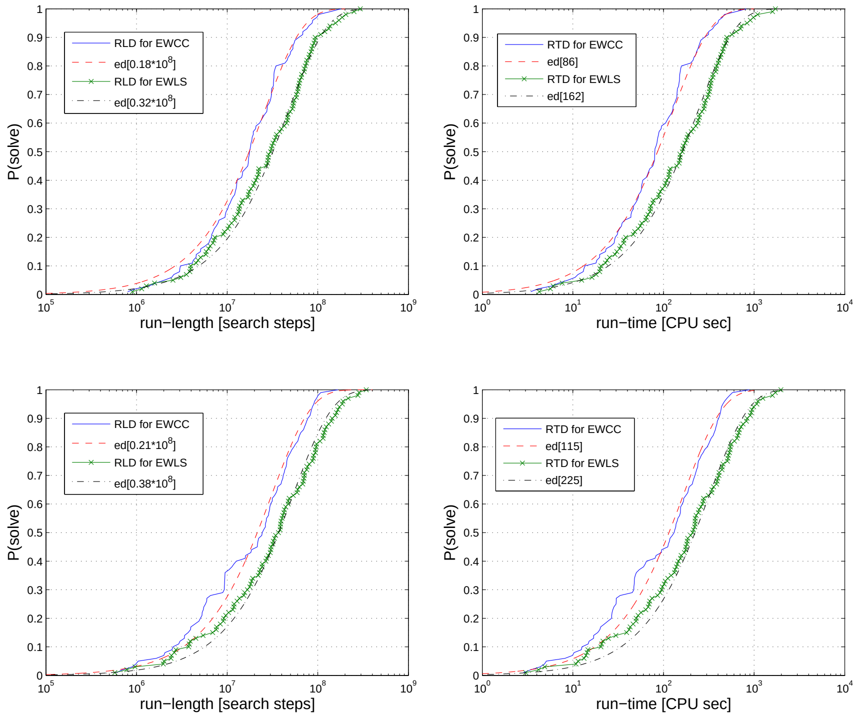

Figure 4: Run-length distributions (left) and run-time distributions (right) for EWLS and EWCC applied to frb53-24-5 (top) and frb59-26-5 (bottom); these empirical RLDs and RTDs are well approximated by exponential distributions, labeled $ed[m](x) = 1 - 2^{-x/m}$ in the plots.

these hard instances. We also perform EWLS$_0$ on frb100-40 and 12% runs find a 3903-sized solution while none finds a 3902-sized solution. Based on these results, we conclude that the edge weighting scheme plays a key role in EWLS.

Graph			EWLS			EWLS$_0$		
Instance	$k*$	CT(s)	suc	time	steps	suc	time	steps
C2000.9	1921	2400	**21**	557.745	66666972	1	53.610	8022464
keller6	3302	1000	**100**	4.934	226592	70	456.159	16690123
frb53-24-1	1219	2000	**25**	745.425	129528395	1	560.550	136983373
frb56-25-1	1344	2200	**28**	742.227	124088139	5	675.348	148886230
frb59-26-1	1475	2400	**21**	1015.145	171236143	1	1137.620	231604550

Table 8: Comparative results between EWLS and EWLS$_0$

6.3. The $delta$ Parameter

The $delta$ parameter is a non-adaptive parameter that should be provided in order to execute EWLS. This parameter influences the performance of EWLS by controlling the size of the current candidate solution, i.e., the current partial vertex cover. Specifically, EWLS adjusts the size of the current candidate solution C during the search process to maintain the equation $|C| = ub - delta$.

An investigation about the $delta$ parameter is performed using 4 DIMACS benchmarks and 4 BHOSLIB benchmarks. For DIMACS benchmarks, we select $brock400\_4$, $C2000.5$, $MANN\_a81$ and $p\_hat1500\text{-}1$, which are from different classes and of appropriate hardness, and cover all optimal values of $delta$ that appear in DIMACS benchmarks. For BHOSLIB benchmarks, frb50-23-3, frb53-24-2, frb56-25-3 and frb59-26-1 are selected, which are of different sizes and different hardness, and cover all optimal values of $delta$ that appear in BHOSLIB benchmarks. We run EWLS with different values of $delta$ within the same cutoff time for each of these 8 instances.

Instance	d	suc	time	steps	time/step	Instance	d	suc	time	steps	time/step
brock400_4	**1**	100	74.146	27107873	2.73	frb50-23-3	**1**	42	692.210	155456585	4.45
	2	92	53.106	14911288	3.56		2	31	792.994	162250210	4.89
	3	0	n/a	n/a	n/a		3	0	n/a	n/a	n/a
	4	0	n/a	n/a	n/a		4	0	n/a	n/a	n/a
	5	0	n/a	n/a	n/a		5	0	n/a	n/a	n/a
C2000.5	**1**	100	2.471	73177	33.76	frb53-24-2	1	40	579.803	126594742	4.58
	2	100	4.109	113269	36.11		2	51	540.416	105610398	5.11
	3	100	17.181	452588	37.96		**3**	58	603.768	109004768	5.54
	4	100	157.638	4109690	38.36		4	48	663.545	110960787	5.98
	5	29	6.871	173373	39.63		5	44	816.342	125398250	6.51
MANN_a81	1	11	934.792	326850188	2.86	frb56-25-3	1	90	692.210	110050260	4.45
	2	18	510.480	181665726	2.81		2	92	443.208	84831320	5.22
	3	22	621.683	207424642	3.00		3	100	359.264	64732310	5.55
	4	2	685.240	234671242	2.92		**4**	100	307.964	51838431	5.94
	5	8	440.686	142616979	3.09		5	99	291.235	45923315	6.32
p_hat1500-1	**1**	100	13.587	419374	32.39	frb59-26-1	1	18	1114.757	190556780	5.85
	2	56	9.396	277012	33.92		**2**	21	1116.251	179661855	6.21
	3	22	7.315	209301	34.95		3	19	1315.865	201295040	6.54
	4	51	11.502	329956	34.86		4	0	n/a	n/a	n/a
	5	0	n/a	n/a	n/a		5	0	n/a	n/a	n/a

Table 9: EWLS performance with different values of $delta$, where time per step (time/step) is measured in 10^{-6} second. The cutoff time for each instance is the same as that in Table 1 and Table 3.

The performance of EWLS with different values of $delta$ on the representative instances is reported in Table 9. The results illustrate that the $delta$ parameter has a big impact on the performance of EWLS. The optimal setting for the $delta$ parameter varies between different instances; consequently, considerable manual tuning is typically required to obtain good performance. Besides, unsurprisingly, we found from the experiments that for the same instance, the optimal value of $delta$ varies between different runs (each run is denoted by its random seed). Therefore, adjusting the value of $delta$ automatically during the search procedure not only makes EWLS more applicable by eliminating this parameter, but also helps it perform better at each run. From Table 9, we have the following observations, which may be helpful for doing this.

- As the value of $delta$ increases, complexity per step increases; in some cases, increasing the value of $delta$ appropriately can accelerate the speed of converging to an optimal solution and improve the success probability.
  Intuitively it can be explained as follows. According to $|C| = ub - delta$, a larger $delta$ results in a smaller C, which leaves more uncovered edges. Note that $ChooseSwapPair$ searches a vertex pair for swapping by scanning the uncovered edge set L, so a larger L means increased complexity per step, and also provides a wider search region at each local search stage which may improve the

quality of local optima.

- EWLS becomes inefficient when the value of $delta$ is "too large" (depending on the instance, but usually those $deltas$ greater than 5 are "too large").
  The intuitive explanation for this observation is that C^+ (see Algorithm 1) is constructed simply, but not optimally. Even if C and C^+ are optimal, $C \cup C^+$ is unlikely to be an optimal solution if both C and C^+ are of considerable size, since they are not independent to each other.

The observations from Table 9 provide some insights about how the $delta$ parameter impacts the performance and behavior of EWLS, which is helpful for adjusting this parameter automatically. We believe that, with the adapted $delta$ (or the adapted size of the current partial vertex cover), the concept of partial vertex cover would be more applicable to improve MVC local search algorithms, including EWCC. We leave this direction for future work.

6.4. The Old-to-Young Strategy

When scanning the uncovered edge set L to search for the vertex to be added in a swapping step, EWLS adopts a specific old-to-young strategy. Generally, the old-to-young strategy in EWLS is motivated by the idea of preferring older solution components for the current candidate solution, where the age of a solution component is defined to be the current step number minus the step number at which it is selected most recently. This idea, as a diversification technique, has been widely used in local search algorithms. Many successful local search algorithms, such as the SAT local search algorithms HSAT [17], Novelty [42], and Novelty++ [37], and also the MVC local search algorithm COVER [50], break ties in favor of the least recently changed solution components.

When choosing the vertex for inserting into C in the $ChooseSwapPair$ function, EWLS first checks the oldest edge in L and then checks UL, rather than check other checked edges, although they are older than edges in UL. This specific strategy is based on our conjecture that it is likely that most of the checked edges still do not satisfy the choosing constraints in the following steps until they are covered and become uncovered again as an edge in UL. Thus, it would be a waste of time to recheck all edges in $L \backslash UL$ (the checked edges). To make a balance between preferring older edges (which would improve the step performance) and low time consumption per step, we just check the oldest edge in L and then go to check UL.

To verify this conjecture, we conduct some experiments for comparing the performance of EWLS with its two alternative versions, EWLS$_1$ and EWLS$_2$. EWLS$_1$ checks all edges in L from old to young; EWLS$_2$ checks all edges in UL from old to young, but does not recheck any checked edges.

The investigation is performed using 5 DIMACS benchmarks and 5 BHOSLIB benchmarks. For the DIMACS benchmarks, we select $brock200\_4$, $C1000.9$, $keller6$, $MANN\_a45$ and $p\_hat1500$-1, which are from different classes and of appropriate hardness. For BHOSLIB benchmarks, $frb45$-21-1, $frb50$-23-3, $frb53$-24-3, $frb56$-25-4 and $frb59$-26-4 are selected, which are of different sizes and varying hardness. From Table 10, we have the following observations, which support our conjectures to some extent.

Instance	k^*	CT(s)	Algorithm	suc	time	steps	time/step
brock200_4	183	1000	EWLS	100	3.184	1696748	1.88
			EWLS$_1$	100	2.657	1362619	*1.95*
			EWLS$_2$	100	3.667	*2037874*	1.80
C1000.9	932	1000	**EWLS**	100	3.326	896259	3.71
			EWLS$_1$	100	4.536	1054497	*4.30*
			EWLS$_2$	100	3.819	*1091106*	3.50
keller6	3302	1000	EWLS	100	4.943	*226592*	21.81
			EWLS$_1$	100	5.832	216712	*26.91*
			EWLS$_2$	100	4.164	198643	20.96
MANN_a45	690	2400	**EWLS**	100	194.969	124004605	1.56
			EWLS$_1$	100	241.581	148208716	*1.63*
			EWLS$_2$	100	232.121	*150503543*	1.54
p_hat1500-1	1488	1000	**EWLS**	100	13.587	419374	32.39
			EWLS$_1$	100	15.211	394754	*38.53*
			EWLS$_2$	100	18.215	*552802*	32.95
frb45-21-1	900	1000	EWLS	100	14.594	3224649	4.53
			EWLS$_1$	100	13.386	2413533	*5.54*
			EWLS$_2$	100	18.068	*4417659*	4.09
frb50-23-3	1100	1800	**EWLS**	42	692.210	155456585	4.45
			EWLS$_1$	30	748.632	150327876	*4.98*
			EWLS$_2$	35	761.672	*179217109*	4.25
frb53-24-3	1219	2000	**EWLS**	100	165.720	32678443	5.07
			EWLS$_1$	100	218.007	36825526	*5.92*
			EWLS$_2$	100	178.661	*38175477*	4.68
frb56-25-4	1344	2200	**EWLS**	100	234.376	39241213	5.97
			EWLS$_1$	100	321.202	39851453	*8.06*
			EWLS$_2$	100	378.561	*68579975*	5.52
frb59-26-4	1475	2400	**EWLS**	18	841.930	125795679	6.69
			EWLS$_1$	7	1281.137	*142665611*	*8.98*
			EWLS$_2$	13	696.889	114057238	6.11

Table 10: Comparative results of EWLS with different search strategies for scanning L, where time per step (time/step) is measured in 10^{-6} second. For each instance, the values of the $delta$ parameter in three algorithms are all set to be the optimal value in EWLS. The algorithm in **bold** is the best algorithm for an instance, while the results in *italic* indicate the worst performance for an instance.

- Overall, EWLS outperforms both EWLS$_1$ and EWLS$_2$ on these representative instances.
- The step performance of EWLS and EWLS$_1$ are comparable, which are significantly better than EWLS$_2$; EWLS$_2$ performs the worst in terms of step on all these representative instances except $keller6$ and $frb59$-26-4.
- The run-time per step of EWLS and EWLS$_2$ are comparable, which are significantly better than EWLS$_1$; EWLS$_1$ has the highest time consumption per step on all these representative instances.
- For some instances, such as $p\_hat1500$-1, $frb45$-21-1 and $frb56$-25-4, rechecking the oldest edge in L (as EWLS does) gains a significant improvement on step performance (compared to EWLS$_2$), while the improvement by rechecking more checked edges (as EWLS$_1$ does) is less significant: the law of diminishing returns appears remarkably in this scenario.

6.5. More Insights to The Configuration Checking Strategy

We now discuss some insights about the configuration checking strategy through an experimental study. The key feature of the CC strategy is indeed its power to handle the cycling problem. Our experimental results demonstrate this. For DIMACS benchmarks, we select all instances that can not be solved within 1 second by both EWLS and EWCC, except the two $brock800$ instances where both EWLS and EWCC totally fail. For BHOSLIB benchmarks, we select two instances of different hardness for each of the five large groups, as well as the challenge instance $frb100$-40.

Let $S = \{v | v \in V \backslash C$ and $confChange[v] = 0\}$, consisting of vertices for which the configuration has not been changed since they were removed from the current candidate solution. We run EWCC on each instance 10 times with different random seeds, within the cutoff time in Table 1 and Table 3; for each run, the size of S ($|S|$) and the ratio $\frac{|S|}{|V \backslash C|}$ (where C is the current candidate solution) averaged over all steps are computed. We report for each instance the mean value and the standard deviation of $|S|$ and $\frac{|S|}{|V \backslash C|}$ over the 10 runs, in the form of $\frac{mean}{standard\ deviation}$.

Instance	$\lvert S\rvert$	$\frac{\lvert S\rvert}{\lvert V \backslash C\rvert}$	Instance	$\lvert S\rvert$	$\frac{\lvert S\rvert}{\lvert V \backslash C\rvert}$
brock200_4	$\frac{0.8456}{0.00708}$	$\frac{0.04975}{0.00044}$	frb45-21-1	$\frac{2.3354}{0.03511}$	$\frac{0.0523}{0.00108}$
brock400_2	$\frac{1.4587}{0.00216}$	$\frac{0.0561}{0.00008}$	frb45-21-3	$\frac{2.2947}{0.10585}$	$\frac{0.0521}{0.00057}$
brock400_4	$\frac{1.4418}{0.02680}$	$\frac{0.0555}{0.00106}$	frb50-23-1	$\frac{2.5227}{0.19606}$	$\frac{0.0517}{0.00130}$
C1000.9	$\frac{3.4168}{0.04079}$	$\frac{0.0506}{0.00053}$	frb50-23-3	$\frac{2.85892}{0.06701}$	$\frac{0.0572}{0.00132}$
C2000.5	$\frac{0.7568}{0.00503}$	$\frac{0.0474}{0.00035}$	frb53-24-1	$\frac{2.8439}{0.03577}$	$\frac{0.0536}{0.00067}$
C2000.9	$\frac{4.4805}{0.21037}$	$\frac{0.0569}{0.00257}$	frb53-24-3	$\frac{2.7152}{0.04402}$	$\frac{0.0516}{0.00087}$
C4000.5	$\frac{0.8247}{0.00413}$	$\frac{0.0489}{0.00028}$	frb56-25-1	$\frac{2.9740}{0.04743}$	$\frac{0.0531}{0.00082}$
keller6	$\frac{2.6849}{0.19274}$	$\frac{0.0473}{0.00369}$	frb56-25-3	$\frac{2.7903}{0.06534}$	$\frac{0.0500}{0.00108}$
MANN_a45	$\frac{2.8835}{0.03406}$	$\frac{0.0084}{0.00009}$	frb59-26-1	$\frac{3.0517}{0.07798}$	$\frac{0.0517}{0.00124}$
MANN_a81	$\frac{4.2277}{0.04637}$	$\frac{0.0039}{0.00004}$	frb59-26-3	$\frac{3.0913}{0.05977}$	$\frac{0.0525}{0.00089}$
p_hat1500-1	$\frac{0.5308}{0.04173}$	$\frac{0.0473}{0.00357}$	frb100-40	$\frac{4.8507}{0.08021}$	$\frac{0.0484}{0.00098}$

Table 11: Statistics on the configuration checking strategy

As shown in Table 11, the ratio $\frac{|S|}{|V \backslash C|}$ varies from 4.73% to 5.72% between different instances (except two MANN instances), which means there is a small but non-negligible portion of such vertex v that $confChange[v] = 0$ in the set $V \backslash C$ each step. Although the amount of such vertices is relatively small, by prohibiting inserting these vertices into C, we improve EWLS significantly in terms of both run-time performance and step performance. For $MANN\_a45$ and $MANN\_a81$, the value of ratio $\frac{|S|}{|V \backslash C|}$ is 0.84% and 0.39% respectively, which means the CC strategy helps little for these two $MANN$ instances; this is coincident with the failures of EWCC on these two $MANN$ instances.

However, we note that the implementation of the CC strategy in this paper (the

4 rules in Section 4) is an approximate one and does not reflect its spirit accurately, recalling that the spirit of the CC strategy is that a vertex $v \notin C$ for which the configuration is not changed after its last removing from C is forbidden to be added back into C. To see this, consider an edge (u, v) where $u \notin C$ and $v \in C$, v is removed from the current candidate solution C ($confChange[v]$ is set to 0 according to Rule 2); after that, u is inserted ($confChange[v]$ is set to 1 according to Rule 3) and then removed from C ($confChange[v]$ is set to 1 according to Rule 3). Suppose other neighbors of v do not change their states and the edge weights are not updated during this period of time. In this case, the configuration for v is considered changed ($confChange[v] = 1$) by the implementation in this paper, but it is not really changed since the state of all v's neighbors and the weight of all edges incident to v are the same as the last time v is removed from C. We believe there would be many such v causing the cycling problem.

A naive accurate implementation of the CC strategy is to store the configuration for a vertex v (store state of its neighbors, also edge weights in $G[N[v]]$ in edge weighted version of configuration and vertex weights in vertex weighted version) when it is removed from C, and check the configuration when needed, say, when considering inserting it into C.

Nevertheless, as is usual in local search algorithms, there is a tradeoff between the accuracy of heuristics and the complexity per step. It is rather time-consuming to execute the CC strategy in this naive accurate way in EWCC. One may replace the implementation of the CC strategy in EWCC by the naive implementation. The resulting algorithm needs $O(\Delta(G))$ (recalling the definitions in Section 2) for both storing and checking the configuration for a vertex. Therefore, the time complexity per step for the CC strategy is $O(\Delta(G)|L|) + O(\Delta(G)) = O(\Delta(G)|L|)$ (check the configuration for the vertices of uncovered edges and store the configuration for the removed vertex). While in EWCC, the time complexity per step for the CC strategy is only $O(|L|)+1+O(\Delta(G)) = O(max(\Delta(G), |L|))$ (check the $confChange$ indicator for the vertices of uncovered edges, reset the $confChange$ indicator for the removed vertex and update the $confChange$ indicator for the neighbors of the two exchanged vertices).

To make a balance between the accuracy of the CC strategy and the complexity per step, we adopt the approximate implementation of the CC strategy in EWCC. Is there a way to accurately implement the configuration checking strategy efficiently? If there is one, then does it lead to a further improvement of EWCC? These are interesting issues to be addressed in future. We also note that for some other algorithms, there would be a better implementation of the CC strategy than the approximate one in EWCC.

7. Conclusions and Future Work

In this paper, we studied stochastic local search approaches to a challenging problem in graph theory that has a wide range of applications in AI and computer science. We have proposed two novel search strategies EW (Edge Weighting) and CC (Configuration Checking) to address the well known NP-hard MVC problem. We employed these two strategies in the development of two new local search algorithms which we call as EWLS and EWCC. The EWCC is an extension of EWLS. This was achieved by integrating the configuration checking strategy within EWLS. The EWCC not only significantly outperforms EWLS, but also it does not require instance-dependent parameters as EWLS does. Thus, it makes a worthwhile advancement to the state-of-the-art on local search algorithms for MVC.

The EW strategy we proposed is in some way inspired by the clause weighting technique in SAT solving [30, 31, 43, 54, 55]. It is a novel idea for solving the MVC problem. The EW strategy associates a weight to each edge and increases the cost of local optima it meets. This means the algorithm may find good candidate solutions hidden behind local optima. It is important to note that our EW strategy is rather different from the strategy adopted in the COVER algorithm which also makes use of edge weights. While COVER is an iterative best improvement algorithm that updates edge weights in each step, the EW strategy in EWLS updates edge weights only when the search being stuck in local optima.

The CC strategy we proposed is to handle the cycling problem in local search. It takes the induced subgraph configuration into account when selecting vertices to add into the current candidate solution, in order to prevent a vertex from getting into the circumstance it just leaves; and hence reduces cycling steps. This novel strategy is essentially different from previous heuristics on MVC, MIS and MC problems, which usually refer to the vertex information but neglect its circumstance in making decisions. It is exciting that the CC strategy is surprisingly effective while being conceptually simple; indeed, EWCC dramatically improves EWLS just by adopting the CC strategy instead of using the standard tabu mechanism as in EWLS. We believe that it also provides an interesting alternative to the standard use of the tabu mechanism to avoid the cycling phenomenon in other combinatorial search problems.

We have carried out experiments on two important benchmark sets, DIMACS and BHOSLIB. Compared with the current best heuristic algorithms, EWLS and EWCC are largely competitive on DIMACS benchmarks and outperform them on most hard instances, and dominate on the hard random BHOSLIB benchmarks. Moreover, EWCC makes a significant improvement over EWLS, particularly on the BHOSLIB benchmarks, where EWCC delivers the best results which significantly improve existing ones. Further, they perform quite well even on structured instances in comparison to the best exact MC algorithm. Notably, both EWLS and EWCC set a new record to the twenty year challenging instance frb100-40. In this sense, this work takes a promising step towards solving hard instances of MVC, MIS and MC problems.

Furthermore, we have shown that EWLS and EWCC are characterized by exponential RTD and RLD, which means they are robust $w.r.t.$ the cutoff parameters and the number of random restarts, and hence have close-to-optimal parallelization speedup [28, 29]. We also perform some further investigations to illustrate the effectiveness of the underlying mechanisms in these two algorithms.

A significant future direction is to apply the configuration checking heuristic to various combinatorial search problems. The CC strategy is very simple and the notions (such as neighbors and state of solution components) it uses are common in combinatorial problems. From its ability to incorporate and guide another procedure, in amended form as a subroutine, the CC strategy may be viewed as a meta-strategy for combinatorial problem solving. Given the success of the CC strategy and its generality, we believe it is also effective for solving other hard combinatorial search problems. Also, based on the discussions in Section 6.5, it would be a valuable work to implement the CC strategy accurately with low time price to handle the cycling problem better.

Acknowledgement

This work is supported by 973 Program (2010CB328103 and 2009CB320701), National Natural Science Foundation of China (60725207, 60821003, 60973033 and 61073033), ARC Future Fellowship FT0991785, and Open Project SYSKF1003 of State Key Laboratory of Computer Science. We would like to thank the anonymous referees for their helpful comments and suggestions.

Appendix A: Proof to Lemma 1

Proof: In this proof, $E(G)$ is the universal set. $u \in C$ and $v \notin C$. Let

$E_0 = \{e|e$ is not covered by $C\}$,

$E_1 = \{e|e$ is not covered by $C \backslash \{u\}\}$,

$E_2 = \{e|e$ is covered by u and not covered by $C \backslash \{u\}\}$.

Note that $C = \{u\} \cup [C \backslash \{u\}]$, we can rewrite E_0 as

$E_0 = \{e|e$ is not covered by u and not covered by $C \backslash \{u\}\}$.

Then we have $E_1 = E_0 \cup E_2$ and $E_0 \cap E_2 = \emptyset$. Thus

$$\begin{aligned} dscore(u) &= cost(G,C) - cost(G, C \backslash \{u\}) \\ &= \sum_{e \in E_0} w(e) - \sum_{e \in E_1} w(e) \\ &= \sum_{e \in E_0} w(e) - (\sum_{e \in E_0} w(e) + \sum_{e \in E_2} w(e)) \\ &= - \sum_{e \in E_2} w(e) \end{aligned}$$

Let $E_3 = \{e|e$ is not covered by $C \cup \{v\}\}$,

$E_4 = \{e|e$ is covered by v and not covered by $C\}$.

We can rewrite E_3 as

$E_3 = \{e|e$ is not covered by v and not covered by $C\}$,

Obviously, $E_0 = E_3 \cup E_4$ and $E_3 \cap E_4 = \emptyset$. Thus

$$\begin{aligned} dscore(v) &= cost(G,C) - cost(G, C \cup \{v\}) \\ &= \sum_{e \in E_0} w(e) - \sum_{e \in E_3} w(e) \\ &= \sum_{e \in E_0} w(e) - (\sum_{e \in E_0} w(e) - \sum_{e \in E_4} w(e)) \\ &= \sum_{e \in E_4} w(e) \end{aligned}$$

Let $E_5 = \{e|e$ is not covered by $C \backslash \{u\} \cup \{v\}\}$,

$E_6 = \{e|e$ is covered by v and not covered by $C \backslash \{u\}\}$.

Similarly, we rewrite E_5 as

$E_5 = \{e|e$ is not covered by v and not covered by $C \backslash \{u\}\}$.

Then we have $E_1 = E_5 \cup E_6$ and $E_5 \cap E_6 = \emptyset$.

$$
\begin{aligned}
score(u,v) &= cost(G,C) - cost(G,[C\backslash\{u\}]\cup\{v\}) \\
&= \sum_{e\in E_0} w(e) - \sum_{e\in E_5} w(e) \\
&= \sum_{e\in E_0} w(e) - (\sum_{e\in E_1} w(e) - \sum_{e\in E_6} w(e)) \\
&= \sum_{e\in E_0} w(e) - (\sum_{e\in E_0} w(e) + \sum_{e\in E_2} w(e) - \sum_{e\in E_6} w(e)) \\
&= \sum_{e\in E_6} w(e) - \sum_{e\in E_2} w(e)
\end{aligned}
$$

Let $E_7 = \{e|e$ is covered by u and v, and not covered by $C\backslash\{u\}\}$,
Rewrite $E_4 = \{e|e$ is not covered by u, covered by v, and not covered by $C\backslash\{u\}\}$,
then we have E_6=$E_4 \cup E_7$ and $E_4 \cap E_7 = \emptyset$. So

$$score(u,v) = \sum_{e\in E_4} w(e) + \sum_{e\in E_7} w(e) - \sum_{e\in E_2} w(e)$$

If $e(u,v) \in E$, then $E_7 = \{e(u,v)\}$,

$$
\begin{aligned}
score(u,v) &= \sum_{e\in E_4} w(e) + w(e(u,v)) - \sum_{e\in E_2} w(e) \\
&= dscore(u) + dscore(v) + w(e(u,v))
\end{aligned}
$$

Otherwise, $E_7 = \emptyset$,

$$score(u,v) = \sum_{e\in E_4} w(e) - \sum_{e\in E_2} w(e) = dscore(u) + dscore(v) \quad \blacksquare$$

Appendix B: Larger independent set for frb100-40

Here we report a 98-vertex independent set for frb100-40.
5, 54, 113, 145, 177, 212, 253, 293, 331, 366, 439, 470, 512, 528, 562, 618, 656, 694, 744, 787, 832, 868, 891, 941, 964, 1008, 1076, 1094, 1149, 1181, 1238, 1241, 1282, 1348, 1390, 1416, 1474, 1490, 1547, 1578, 1623, 1664, 1681, 1722, 1786, 1806, 1844, 1890, 1955, 1999, 2040, 2046, 2116, 2130, 2188, 2216, 2244, 2326, 2362, 2421, 2480, 2516, 2558, 2589, 2608, 2679, 2698, 2743, 2788, 2820, 2878, 2885, 2935, 2993, 3026, 3078, 3084, 3152, 3213, 3241, 3299, 3357, 3373, 3429, 3444, 3515, 3538, 3600, 3638, 3648, 3705, 3760, 3762, 3834, 3875, 3895, 3928, 3994.

References

[1] C. Aggarwal, J. Orlin, R. Tai, Optimized crossover for the independent set problem, Operations Research 45 (1997) 226–234.

[2] D. V. Andrade, M. G. C. Resende, R. F. F. Werneck, Fast local search for the maximum independent set problem, in: Proc. of WEA-08, 2008, pp. 220–234.

[3] V. C. Barbosa, L. C. D. Campos, A novel evolutionary formulation of the maximum independent set problem, J. Comb. Optim. 8 (4) (2004) 419–437.

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

[5] D.L. Berre, L.Simon, Fifty-five solvers in vancouver: the SAT 2004 competition, in: Proc. of SAT-04, 2004, pp. 321–344.

[6] S. Busygin, A new trust region technique for the maximum clique problem, internal report, http://www.busygin.dp.ua (2002).

[7] S. Busygin, S. Butenko, P. M. Pardalos, A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere, J. Comb. Optim. 6 (3) (2002) 287–297.

[8] S. Cai, K. Su, Q. Chen, Ewls: A new local search for minimum vertex cover, in: Proc. of AAAI-10, 2010, pp. 45–50.

[9] R. Carraghan, P. Pardalos, An exact algorithm for the maximum clique problem, Operations Research Letters 9 (6) (1990) 375–382.

[10] R. Diestel, Graph Theory Third Edition, Springer-Verlag, NewYork, 2005.

[11] I. Dinur, S. Safra, On the hardness of approximating minimum vertex cover, Annals of Mathematics 162 (2) (2005) 439–486.

[12] I. Evans, An evolutionary heuristic for the minimum vertex cover problem, in: Proc. of EP-98, 1998, pp. 377–386.

[13] T. Fahle, Simple and fast: Improving a branch-and-bound algorithm for maximum clique, in: Proc. of ESA-02, 2002, pp. 485–498.

[14] U. Feige, Approximating maximum clique by removing subgraphs, SIAM J. Discrete Math. 18 (2) (2004) 219–225.

[15] M. R. Fellows, F. A. Rosamond, F. V. Fomin, D. Lokshtanov, S. Saurabh, Y. Villanger, Local search: Is brute-force avoidable?, in: Proc. of IJCAI-09, 2009, pp. 486–491.

[16] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, San Francisco, CA, USA, 1979.

[17] I.P. Gent, T. Walsh, Towards an understanding of hill-climbing procedures for SAT, in: Proc. of AAAI-93, 1993, pp. 28–33.

[18] S. Gilmour, M. Dras, Kernelization as heuristic structure for the vertex cover problem, in: ANTS Workshop, 2006, pp. 452–459.

[19] F. Glover, Tabu search – part i, ORSA Journal on Computing 1 (3) (1989) 190–206.

[20] F. Glover, Tabu search – part ii, ORSA Journal on Computing 2 (1) (1990) 4–32.

[21] C. P. Gomes, H. Kautz, A. Sabharwal, B. Selman, Satisfiability solvers, in: Handbook of Knowledge Representation, Elsevier, 2008, pp. 89–134.

[22] A. Grosso, M. Locatelli, F. D. Croce, Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem, J. Heuristics 10 (2) (2004) 135–152.

[23] A. Grosso, M. Locatelli, W. J. Pullan, Simple ingredients leading to very efficient heuristics for the maximum clique problem, J. Heuristics 14 (6) (2008) 587–612.

[24] E. Halperin, Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs, SIAM Journal on Computing 31 (5) (2002) 1508–1623.

[25] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, Acta Math 182 (1999) 105–142.

[26] J. Håstad, Some optimal inapproximability results, J. ACM 48 (4) (2001) 798–859.

[27] H. H. Hoos, T. Stützle, Towards a characterisation of the behaviour of stochastic local search algorithms for sat, Artif. Intell. 112 (1-2) (1999) 213–232.

[28] H. H. Hoos, T. Stützle, Local search algorithms for sat: An empirical evaluation, J. Autom. Reasoning 24 (4) (2000) 421–481.

[29] H. H. Hoos, T. Stützle, Stochastic Local Search: Foundations & Applications, Elsevier / Morgan Kaufmann, 2004.

[30] F. Hutter, D. A. D. Tompkins, H. H. Hoos, Scaling and probabilistic smoothing: Efficient dynamic local search for sat, in: Proc. of CP-02, 2002, pp. 233–248.

[31] A. Ishtaiwi, J. Thornton, Anbulagan, A. Sattar, D. N. Pham, Adaptive clause weight redistribution, in: Proc. of CP-06, 2006, pp. 229–243.

[32] Y. Ji, X. Xu, G. D. Stormo, A graph theoretical approach for predicting common rna secondary structure motifs including pseudoknots in unaligned sequences, Bioinformatics 20 (10) (2004) 1603–1611.

[33] G. Karakostas, A better approximation ratio for the vertex cover problem, in: Proc. of ICALP-05, 2005, pp. 1043–1050.

[34] K. Katayama, A. Hamamoto, H. Narihisa, Solving the maximum clique problem by k-opt local search, in: Proc. of SAC-04, 2004, pp. 1021–1025.

[35] K. Katayama, M. Sadamatsu, H. Narihisa, Iterated k-opt local search for the maximum clique problem, in: Proc. of EvoCOP-07, 2007, pp. 84–95.

[36] H. A. Kautz, A. Sabharwal, B. Selman, Incomplete algorithms, in: Handbook of Satisfiability, IOS Press, 2009, pp. 185–203.

[37] C. Li, W. Huang, Diversification and determinism in local search for satisfiability, in: Proc. of SAT-05, 2005, pp. 158–172.

[38] C. Li, F. Manyà, MaxSAT, Hard and Soft Constraints, in: Handbook of Satisfiability, IOS Press, 2009, pp. 613–631.

[39] C. Li, Z. Quan, An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem, in: Proc. of AAAI-10, 2010, pp. 128–133.

[40] C. Li, Z. Quan, Combining graph structure exploitation and propositional reasoning for the maximum clique problem, in: Proc. of ICTAI-10, 2010, pp. 344-351.

[41] B. Mazure, L. Sais, É. Grégoire, Tabu search for sat, in: Proc. of AAAI-97, 1997, pp. 281–285.

[42] D. McAllester, B. Selman, H. Kautz, Evidence for invariants in local search, in: Proc. of AAAI-97, 1997, pp. 321–326.

[43] P. Morris, The breakout method for escaping from local minima, in: Proc. of AAAI-93, 1993, pp. 40–45.

[44] P. R. J. Östergård, A fast algorithm for the maximum clique problem, Discrete Applied Mathematics 120 (1-3) (2002) 197–207.

[45] W. Pullan, H. H. Hoos, Dynamic local search for the maximum clique problem, J. Artif. Intell. Res. (JAIR) 25 (2006) 159–185.

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

[47] W. Pullan, Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers, Discrete Optimization 6 (2009) 214–219.

[48] W. Pullan, F. Mascia, M. Brunato, Cooperating local search for the maximum clique problem, J. Heuristics,doi:10.1007/s10732-010-913195.

[49] J. C. Régin, Using constraint programming to solve the maximum clique problem, in: Proc. of CP-03, 2003, pp. 634–648.

[50] S. Richter, M. Helmert, C. Gretton, A stochastic local search approach to vertex cover, in: Proc. of KI-07, 2007, pp. 412–426.

[51] S. J. Shyu, P. Yin, B. M. T. Lin, An ant colony optimization algorithm for the minimum weight vertex cover problem, Annals OR 131 (1-4) (2004) 283–304.

[52] K. Smyth, H. H. Hoos, T. Stützle, Iterated robust tabu search for max-sat, in: Canadian Conference on AI, 2003, pp. 129–144.

[53] C. Solnon, S. Fenet, A study of aco capabilities for solving the maximum clique problem, J. Heuristics 12 (3) (2006) 155–180.

[54] J. Thornton, D. N. Pham, S. Bain, V. F. Jr., Additive versus multiplicative clause weighting for sat, in: Proc. of AAAI-04, 2004, pp. 191–196.

[55] J. Thornton, Clause weighting local search for sat, J. Autom. Reasoning 35 (1-3) (2005) 97–142.

[56] E. Tomita, T. Kameda, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments, J. Global Optimization 44 (2) (2009) 311.

[57] J. P. Watson, L. D. Whitley, A. E. Howe, Linking search space structure, runtime dynamics, and problem difficulty: A step toward demystifying tabu search, J. Artif. Intell. Res. (JAIR) 24 (2005) 221–261.

[58] K. Xu, W. Li, Exact phase transitions in random constraint satisfaction problems, J. Artif. Intell. Res. (JAIR) 12 (2000) 93–103.

[59] K. Xu, F. Boussemart, F. Hemery, C. Lecoutre, A simple model to generate hard satisfiable instances, in: Proc. of IJCAI-05, 2005, pp. 337–342.

[60] K. Xu, W. Li, Many hard examples in exact phase transitions, Theoretical Computer Science 355 (2006) 291-302.

[61] K. Xu, F. Boussemart, F. Hemery, C. Lecoutre, Random constraint satisfaction: Easy generation of hard (satisfiable) instances, Artif. Intell. 171 (8-9) (2007) 514–534.

[62] D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, in: Proc. of STOC-06, 2006, pp. 681–690.