EWLS: A New Local Search for Minimum Vertex Cover*

Shaowei Cai[1,2] **Kaile Su**[1,3†] **Qingliang Chen**[4]
[1]Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, China
[2] School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China
[3]Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
[4]Department of Computer Science, Jinan University, Guangzhou 510632, China
shaowei_cai@126.com; sukl@pku.edu.cn; tpchen@jnu.edu.cn

Abstract

A number of algorithms have been proposed for the Minimum Vertex Cover problem. However, they are far from satisfactory, especially on hard instances. In this paper, we introduce Edge Weighting Local Search (EWLS), a new local search algorithm for the Minimum Vertex Cover problem. EWLS is based on the idea of extending a partial vertex cover into a vertex cover. A key point of EWLS is to find a vertex set that provides a tight upper bound on the size of the minimum vertex cover. To this purpose, EWLS employs an iterated local search procedure, using an edge weighting scheme which updates edge weights when stuck in local optima. Moreover, some sophisticated search strategies have been taken to improve the quality of local optima. Experimental results on the broadly used DIMACS benchmark show that EWLS is competitive with the current best heuristic algorithms, and outperforms them on hard instances. Furthermore, on a suite of difficult benchmarks, EWLS delivers the best results and sets a new record on the largest instance.

Introduction

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 combinatorial optimization problem with many applications (Richter, Helmert, and Gretton 2007), and it is equivalent to two other well-known optimization problems: Maximum Independent Set (MIS) problem and Maximum Clique (MC) problem. For their importance in theory and applications, these problems have been widely investigated in AI community (Pullan and Hoos 2006; Richter, Helmert, and Gretton 2007; Fellows et al. 2009).

MVC is NP-hard and the associated decision problem is NP-complete (Garey and Johnson 1979); furthermore, it is NP-hard to approximate MVC within any factor smaller than 1.3606 (Dinur and Safra 2005), and state-of-the-art approximation algorithms can only achieve an approximation ratio of $2 - o(1)$ (Halperin 2002; Karakostas 2005). Therefore, for large and hard instances 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, on random problems, heuristic methods for SAT can significantly outperform DPLL-based methods. A modern SAT solver can solve hard instances with over a million variables and several million constraints within reasonable time (Gomes et al. 2008). For graph 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 hard instances. For example, there is a hard instance with 4000 vertices, which has a 3900-sized optimal vertex cover while the size of the best solution found up to now is 3903[1].

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. This calls for more essential progress on algorithms for hard instances of MVC (MIS, MC) problems.

In this paper, we propose a new local search algorithm for MVC, dubbed EWLS (Edge weighting Local Search). Most state-of-the-art solvers, including the ones we introduce in the following section, search for a vertex cover (or clique, respectively) during the search procedure. EWLS uses a rather different framework, which focuses on finding a vertex set that provides a better upper bound on the size of the minimum vertex cover, and extends it into a vertex cover. EWLS makes use of edge weights, which are dynamically modified during the search to level the cost landscape; thus it may discover good candidate solutions hidden behind local optima.

Constraint weighting is an effective technique for escaping local optima, which has been widely used in SAT and CSP, such as clause weighting in SAT (Morris 1993;

*This work is supported by 973 Program (2010CB328103, 2009CB320701 and 2005CB321902), National Natural Science Foundation of China (60725207 and 60973033), Key Research Project of Ministry of Education (210257), and ARC Future Fellowship FT0991785.

†Corresponding author

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

Hutter, Tompkins, and Hoos 2002; Thornton 2005), and was introduced to MVC by the COVER algorithm (Richter, Helmert, and Gretton 2007). Although the edge weighting scheme in EWLS falls into constraint weighting technique, 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, as the weighting scheme proposed in SAT community does (Morris 1993; Hutter, Tompkins, and Hoos 2002). Also, EWLS employs some search strategies to improve the quality of local optima.

We show that EWLS achieves excellent performance on a large variety of benchmarks, especially on large and hard instances. On the DIMACS benchmark, EWLS is competitive with state-of-the-art solvers, and outperforms them on hard instances. Furthermore, on the BHOSLIB benchmark, which is strongly recommended by researchers in MC community (Grosso, Locatelli, and Pullan 2008), EWLS delivers the best results and sets a new record on the largest instance.

The remainder of this paper is organized as follows. In the next section, we introduce some necessary background knowledge and related work. Then we describe the EWLS algorithm. Experimental results demonstrating the performance of EWLS are presented next. Finally, we give some concluding remarks.

Preliminaries and Related Work

A *graph* $G = (V, E)$ consists of a *vertex set* V and an *edge set* $E \subseteq V \times V$, where each edge is a pair of distinct vertices. For an edge $e(u, v)$, u and v are the *endpoints* of edge e, and $endpoint(e) = \{u, v\}$.

MIS problem and MC problem can be reduced to MVC problem (Richter, Helmert, and Gretton 2007). All these problems are NP-hard and the associated decision problems are NP-complete (Garey and Johnson 1979). Moreover, besides the inapproximability of MVC, both MIS and MC are not approximable within $|V|^{1-\epsilon}$ for any $\epsilon > 0$, unless NP=ZPP (Håstad 1999; 2001). Therefore, for large and hard instances, we have to resort to heuristic approaches.

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 (Evans 1998). Ant colony approaches have been proposed in (Shyu, Yin, and Lin 2004) and (Gilmour and Dras 2006). The recent Cover Edge Randomly (COVER) algorithm (Richter, Helmert, and Gretton 2007) is an iterative best improvement algorithm using edge weights to guide the local search. Shown by results on DIMACS benchmark and BHOSLIB benchmark, COVER is the best one among heuristic methods to MVC.

For MIS, existing heuristic algorithms include: Optimised Crossover Heuristic (OCH) (Aggarwal, Orlin, and Tai 1997); QSH, which is based on optimization of a quadratic over a sphere (Busygin, Butenko, and Pardalos 2002); and the evolutionary algorithm Widest Acyclic Orientation (WAO) (Barbosa and Campos 2004). Recently, an *iterated local search* algorithm based on improving swaps is proposed in (Andrade, Resende, and Werneck 2008). Seen from results on the DIMACS benchmark, it is dominated by the best MC algorithm except two $MANN$ instances.

Compared with MVC and MIS, more work has been done on MC problem. Reported in (Pullan and Hoos 2006), there are five heuristic algorithms achieving state-of-the-art performance: Reactive Local Search (RLS) (Battiti and Protasi 2001), QUALEX-MS (Busygin 2002), Deep Adaptive Greedy Search (DAGS) (Grosso, Locatelli, and Croce 2004), *k-opt* algorithm (Katayama, Hamamoto, and Narihisa 2004) which has evolved into *iterated k-opt* algorithm (Katayama, Sadamatsu, and Narihisa 2007) and Edge-AC+LS (Solnon and Fenet 2006). However, Dynamic Local Search-Maximum Clique (DLS-MC) algorithm proposed in (Pullan and Hoos 2006) delivers the best results on DIMACS benchmark. DLS-MC alternates between phases of iterative improvement and 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 (Pullan 2006). Even at the time of writing, PLS is still one of the best algorithms for the DIMACS benchmark as we know. Pullan extends PLS to MVC and MIS problem (Pullan 2009), achieving state-of-the-art performance.

Edge Weighting Local Search for MVC

In this section, we present the EWLS algorithm, which is an iterated local search algorithm based on the idea of extending a vertex set into a vertex cover and uses edge weighting technique.

Basic Notation and Definitions

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. 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 exchanging two vertices: a vertex $u \in C$ is removed from C, and a vertex $v \notin C$ is put into C. Each edge $e \in E$ is associated with a positive integer number $w(e)$ as its weight. 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 weights of edges not covered by C. The *evaluation function* is $g : C \mapsto cost(G, C)$, which means EWLS prefers candidate solutions with lower cost. 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 $v \notin C$,

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

which indicates the *score* of exchanging u and v.

Lemma 1 *Given $G = (V, E)$ and C the current candidate solution, 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.*

The lemma can be proved according to the definition of $dscore$ and $score$ by some elementary set-theoretic arguments. EWLS calculates the $score$ of an exchanging step according to Lemma 1.

Partial vertex cover is an important notion in our work, so we give its definition 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.

In the above definition, a $(k, 0)$-partial vertex cover is a k-vertex cover. 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 *For an undirected graph $G = (V, E)$, a (k, t)-partial vertex cover of G provides an upper bound that is equal to $k + t$ on the size of the minimum vertex cover of G.*

The EWLS Algorithm

An essential idea of EWLS is to find a partial vertex cover that can be extended into an optimal vertex cover. We have a general framework as follows. Whenever finding a partial vertex cover providing a better upper bound, EWLS extends it into a vertex cover, and then removes some vertices from C, and goes on to find a new upper bound. In this way, the MVC problem is transformed to 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 in an $iterated\ local\ search$ scheme, which applies a so called $local\ search\ stage$ to an initial candidate solution until it meets a local optimum; then it perturbs the final candidate solution and executes the next stage.

Edge weighting scheme in EWLS increases the cost of local optima it meets, making the cost landscape flatter; the algorithm may thus find good candidate solutions hidden behind local optima. To this end, EWLS maintains a set L of uncovered edges. When it gets stuck in local optima, for each edge $e \in L$, $w(e)$ is increased by 1, and then EWLS perturbs C to continue to search from another starting point.

Based on the above considerations, we outline the EWLS algorithm as Algorithm 1. Further comments on the algorithm are given 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 the function $ChooseExchangePair$ in the current local search stage. Both of them are set to E. Moreover, edges weights are initialised 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 (ties are broken randomly). Finally, the upper bound ub is initialised as $|C|$, and the best solution C^* is initialised as C. Whenever finding a new upper bound, EWLS selects vertices with the highest $dscore$ in C 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 $dscore$ are negative.

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 $ChooseExchangePair$ successfully finds a pair of vertices to exchange, then an improving step is executed by exchanging the two vertices (lines 10-12). EWLS keeps track of the vertices last inserted into C and last removed from C, and prevents them from being rolled back immediately. If $ChooseExchangePair$ 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 14). After updating the edge weights, C is perturbed by a random step, which exchanges a random vertex $u \in C$ and a random vertex $v \in \bigcup_{e \in L} endpoint(e)$ (line 15).

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 edge weights and dscore of vertices;
5      construct C greedily until it's a vertex cover;
6      ub := |C|; C* := C;
7      remove some vertices from C;
8      while step < maxSteps do
9          if ((u, v) := ChooseExchangePair(C, L, UL)) ≠
           (0, 0) then
10             C := [C\{u}] ∪ {v};
11             tabuAdd := u;
12             tabuRemove := v;
13         else
14             update edge weights;
15             take a random step;
16         if |C| + |L| < ub then
17             ub := |C| + |L|;
18             if L = ∅ then
19                 C* := C;
20             else
21                 construct C+ greedily that covers L;
22                 C* := C ∪ C+;
23             remove some vertices from C;
24         step := step + 1;
25     return C*;
26 end
```

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 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 removes vertices from C to continue to search for a better upper bound (line 23).

Obviously, larger $delta$ results in smaller C, which leaves more uncovered edges. We will show that $ChooseExchangePair$ searches a vertex pair for exchanging by scanning L; as a result, larger L means the algorithm searches a wider region at each local search stage and may find better local optima. However, the algorithm becomes inefficient when $delta$ is too large (depending on the instance), because C^+ 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 a considerable size.

The most important part of EWLS algorithm is the function $ChooseExchangePair$. Its pseudo code is given in Algorithm 2. Before we come to describe it, let's introduce a notion first. In EWLS, the age of an uncovered edge is the current step number minus the step number that it 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.

Algorithm 2: function ChooseExchangePair

```
1  ChooseExchangePair(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
2  begin
3      if S := {(u,v)|u ∈ C, v ∈ {v1*, v2*} and
       score(u,v) > 0} ≠ ∅ then
4          return random(S) ;
5      else
6          foreach e(v1,v2) ∈ UL, from old to young do
7              if S := {(u,v)|u ∈ C, v ∈ {v1,v2} and
               score(u,v) > 0} ≠ ∅ then
8                  return random(S);
9      return (0,0);
10 end
```

The function $ChooseExchangePair$ chooses a pair of vertices $u \in C$ and $v \in \bigcup_{e \in L} endpoint(e)$ that $score(u,v) > 0$. 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 $e^*(v_1^*, v_2^*)$ in L. If there exists at least one vertex pair $u \in C$ and $v \in \{v_1^*, v_2^*\}$ such that $score(u,v) > 0$, the function returns one of them randomly. Otherwise, it goes to check the edges in UL, according to the order from old to young. If for some edge $e(v_1, v_2)$ in UL, the set $S = \{(u,v)|u \in C, v \in \{v_1, v_2\}$ and $score(u,v) > 0\}$ is not empty, then the function returns one vertex pair $(u,v) \in S$ randomly. Finally, if the function fails to find such a vertex pair, it returns (0,0).

To sum up, EWLS strikes a balance between guided search and the diversity. In each step, candidate vertex pairs for exchanging are chosen according to improving heuristics; however, one of them is selected randomly to exchange. 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.

Empirical Performance Results

We evaluate the performance of EWLS on the DIMACS Clique benchmark set and the BHOSLIB benchmark set. For each benchmark set, we compare with the solvers with the best results as far as we can find in literatures.

Our algorithm is implemented in C++, compiled by the g++ compiler with the '-O2' option. All experiments were run on a machine with a 3 GHz CPU and 4GB RAM under Linux. To execute the DIMACS machine benchmarks[2], this machine required 0.19 CPU seconds for r300.5, 1.12 CPU seconds for r400.5 and 4.24 CPU seconds for r500.5. In the following, all CPU times refer to our machine's. CPU times in other ones are scaled to ours according to the run-time on machine benchmarks.

We perform 100 independent trials with different random seeds on each instance. The parameter $maxSteps$ is set to 10^8 for all DIMACS instances and 4×10^8 for all BHOSLIB instances because of their higher difficulty. Each run was stopped in advance before the step limit when an optimal/best known solution was detected. All algorithms for comparison also use the same termination criterion as EWLS. For each instance, we report the following information: the minimum known vertex cover size (k^*); the number of runs (out of 100) in which a solution of size k^* is found ($rate$); and the run-time in CPU seconds averaged over all successful runs. We also report the optimised $delta$ (d) for EWLS, and the dominant solver is highlighted in bold text.

DIMACS Benchmark Results

The DIMACS benchmark is taken from the Second DIMACS Implementation Challenge (1992-1993)[3]. Thirty seven graphs were selected by the organizers for a summary to indicate the effectiveness of algorithms, comprising the Second DIMACS Challenge Test Problems. We compare EWLS with the solvers PLS and COVER, both of which also run 100 times with the same step limit as EWLS. We download COVER[4] and run its iterative version COVER-I[5] on our

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

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

[4] http://www.informatik.uni-freiburg.de/~srichter/

[5] COVER is a k-vertex cover solver and needs to know k^*, while COVER-I runs without knowing k^*

Graph		EWLS			PLS		COVER-I	
Instance	k^*	d	rate	CPU(s)	rate	CPU(s)	rate	CPU(s)
brock400_2	371	1	2	40.902	**100**	0.111	0	n/a
brock400_4	367	1	96	56.816	**100**	0.031	2	242.025
brock800_2	776	1	0	n/a	**100**	7.157	0	n/a
brock800_4	774	1	0	n/a	**100**	1.918	0	n/a
C2000.9	1921	1	**18**	327.875	0	n/a	0	n/a
C4000.5	3982	1	100	686.472	**100**	43.886	100	658.33
keller6	3302	1	**100**	4.934	36	161.568	100	68.214
MANN_a45	690	1	**56**	68.069	0	n/a	40	171.823
MANN_a81	2221	3	**7**	115.751	0	n/a	0	n/a
p_hat1500-1	1488	1	100	13.587	**100**	0.961	100	18.095

Table 1: Results on DIMACS benchmark

machine with the same setting, while PLS is not available to us and the results are taken from (Pullan 2006).

The results on the Second DIMACS Challenge Test Problems are shown in Table 1. Most DIMACS instances are too easy for a modern solver. The instances not appearing in the table are solved by the three solvers with 100% success rate in less than 2 seconds. To obtain a meaningful comparison, k^* for $C2000.9$ is set to 1921. Indeed, the best known 1920-vertex solution was found within 10^9 steps (Grosso, Locatelli, and Pullan 2008), and we are not aware of any algorithm finding an optimal solution in 10^8 steps.

The results show that EWLS is competitive with PLS and performs better on hard instances. EWLS finds a solution of size k^* for 35 out of 37 instances, while this number is 34 for PLS and 32 for COVER-I. Furthermore, EWLS significantly outperforms PLS and COVER-I on three hard instances ($C2000.9$, $MANN\_a45$, and $MANN\_a81$), where PLS does not find a solution of size k^*. On the putatively hardest instance $MANN\_a81$, EWLS finds an optimal solution in 7 runs, while PLS only finds solutions of size k^*+2 and COVER-I finds a solution of size k^*+1 in 5 runs.

Nevertheless, both EWLS and COVER-I fail in some graphs of the $brock$ family which is generated by explicitly incorporating low-degree vertices into the optimal cover to defeat greedy heuristics. Indeed, most algorithms that prefers the higher degree vertices such as GRASP, RLS, and k-opt also failed in these graphs. Remark that, PLS performs well on $brock$ family because it comprises three sub-algorithms, one of which favors the lower degree vertices.

BHOSLIB Benchmark Results

We also consider a new benchmark that is much more difficult, the BHOSLIB (Benchmark with Hidden Optimum Solutions) instances arising from the SAT'04 Competition[6]. These 40 BHOSLIB instances were translated from SAT instances generated randomly in the phase transition area according to the model RB, and have been proven to be hard both theoretically and practically (Xu et al. 2007). The BHOSLIB benchmark has been widely used in the recent literature as a reference point for new heuristics to MVC, MC and MIS[7]. Besides these 40 instances, there is a large instance with 4,000 vertices and 572,774 edges, which is designed for challenge. We compare EWLS with the solvers

[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

Graph		EWLS			PLS-MVC		COVER-I	
Instance	k^*	d	rate	CPU(s)	rate	CPU(s)	rate	CPU(s)
frb50-23-1	1100	2	**100**	262.313	87	479.845	100	267.752
frb50-23-2	1100	2	**61**	535.889	59	622.067	53	730.832
frb50-23-3	1100	1	42	692.21	22	786.944	47	967.853
frb50-23-4	1100	1	**100**	31.573	100	37.682	100	32.734
frb50-23-5	1100	2	**100**	117.730	100	132.791	100	167.929
frb53-24-1	1219	4	**29**	885.421	9	891.715	19	994.628
frb53-24-2	1219	3	**62**	659.381	41	933.330	62	946.14
frb53-24-3	1219	2	**100**	165.720	86	541.244	100	281.189
frb53-24-4	1219	2	51	783.279	**59**	593.589	59	1099.910
frb53-24-5	1219	2	**100**	250.262	94	540.461	99	416.047
frb56-25-1	1344	3	29	773.586	14	809.778	**34**	1255.851
frb56-25-2	1344	2	**32**	730.987	10	806.611	22	1230.623
frb56-25-3	1344	4	**100**	307.964	18	1028.537	99	536.067
frb56-25-4	1344	4	**100**	234.376	89	695.234	98	466.007
frb56-25-5	1344	3	**100**	94.242	96	428.927	100	168.166
frb59-26-1	1475	2	**21**	1015.145	2	748.640	21	1421.241
frb59-26-2	1475	3	**18**	1116.251	2	272.768	11	1149.700
frb59-26-3	1475	3	**44**	726.424	24	934.994	38	1765.950
frb59-26-4	1475	4	**19**	900.275	14	1003.104	6	2069.310
frb59-26-5	1475	1	**100**	334.044	97	472.659	100	478.587

Table 2: Results on BHOSLIB benchmark

PLS-MVC and COVER. We run COVER-I on our machine with the same setting, while the solver PLS-MVC is not available to us and the results are taken from (Pullan 2009).

Performance results on BHOSLIB benchmark are shown in Table 2. Here we focus on the hard instances, so the instances solved by the three solvers with 100% success rate in less than 200 seconds are not reported here. Note that EWLS solves all BHOSLIB instances in terms of reaching an optimal solution. Furthermore, referring to literatures on this benchmark, EWLS solves the most instances with 100% success rate (29 of 40 instances) while requiring less time for most of these instances.

The results illustrate that EWLS is the dominant solver for all instances in terms of both quality and run-time, with the exceptions of frb50-23-3, frb53-24-4 and frb56-25-1. For frb50-23-3, however, there is not an obvious dominant solver.

The excellent performance of EWLS is further underlined by the large gaps between EWLS and the other solvers on the frb59 family, the largest and hardest family. The results undoubtedly demonstrate that EWLS delivers the best performance on the BHOSLIB benchmark, which remains justifiable when referring to other literatures on this benchmark.

For the challenge instance $frb100\text{-}40$, which has a minimum vertex cover of size 3900, the designer of the BHOSLIB benchmark conjectures that this instance will not be solved on a PC in less than a day within the next two decades[6]. The best solution found up to now is a vertex cover of size 3903 in 71.09 seconds by the solver COVER on a 2.13GHz/2GB computer. In detail, 25 out of 100 runs found a 3903-sized solution with the median run-time of 1193.92 seconds[6]. It should be noted that COVER runs by given k^* in advance, which is not natural for MVC and makes the problem easier. We run EWLS (with $delta = 6$ and $maxSteps = 3 \times 10^8$) 100 independent trials on this instance. 4 runs find a 3902-sized vertex cover, or equivalently, a 98-sized independent set (reported in Appendix) with the average time of 2823.05 seconds and the fastest one does so in 1239.87 seconds. Of the other 96 runs, 59 find a 3903-sized solution and 37 find a 3904-sized solution.

Conclusions and Future Work

We present a local search algorithm for the minimum vertex cover problem, EWLS, which is based on a new idea that finds a better partial vertex cover and extends it into a vertex cover. EWLS uses edge weighting scheme to level the cost landscape so that it would not converge in a small region. Also, it utilizes an old-to-young search strategy to make a wide search region for each local search stage to improve the quality of local optima. Evaluated on the DIMACS benchmark and the hard BHOSLIB benchmark, EWLS is consistently superior on large, hard instances, compared with the current best algorithms on these benchmarks. Furthermore, EWLS delivers the best results on the BHOSLIB benchmark, significantly improving the existing ones and sets a new record for the twenty years long challenge instance frb100-40. In this sense, this work takes a promising step towards solving hard instances of MVC, MC and MIS problems.

As with the DLS-MC algorithm (Pullan and Hoos 2006), EWLS has an instance dependent parameter too. Fortunately, its optimal value is always one of $\{1, 2, 3, 4\}$ and thus is easy to tune. Indeed, it has the same value ($delta = 1$) for all DIMACS instances except one instance. An obvious direction of future work is to eliminate the parameter of EWLS. It is also interesting to apply the ideas in EWLS to other combinatorial optimization problems.

Acknowledgment

We would like to thank the anonymous referees for their comments which lead to significant improvements in this paper. Thanks are also due to Tian Liu for bringing our attention to this direction. Particularly, the first author are extremely grateful to Yanfei Lv for all his help.

Appendix: 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

Aggarwal, C.; Orlin, J.; and Tai, R. 1997. Optimized crossover for the independent set problem. *Operations Research* 45:226–234.

Andrade, D. V.; Resende, M. G. C.; and Werneck, R. F. F. 2008. Fast local search for the maximum independent set problem. In *Workshop on Experimental Algorithms*, 220–234.

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

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

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

Busygin, S. 2002. A new trust region technique for the maximum clique problem. Internal report, http://www.busygin.dp.ua.

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

Evans, I. 1998. An evolutionary heuristic for the minimum vertex cover problem. In *Proc. EP-98*, 377–386.

Fellows, M. R.; Rosamond, F. A.; Fomin, F. V.; Lokshtanov, D.; Saurabh, S.; and Villanger, Y. 2009. Local search: Is brute-force avoidable? In *Proc. of IJCAI-09*, 486–491.

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

Gilmour, S., and Dras, M. 2006. Kernelization as heuristic structure for the vertex cover problem. In *ANTS Workshop*, 452–459.

Gomes, C. P.; Kautz, H.; Sabharwal, A.; and Selman, B. 2008. Satisfiability solvers. In *Handbook of Knowledge Representation*. Elsevier.

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

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

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

Håstad, J. 1999. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math* 182:105–142.

Håstad, J. 2001. Some optimal inapproximability results. *J. ACM* 48(4):798–859.

Hutter, F.; Tompkins, D. A. D.; and Hoos, H. H. 2002. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *Proc. of CP-02*, 233–248.

Karakostas, G. 2005. A better approximation ratio for the vertex cover problem. In *Proc. of ICALP-05*, 1043–1050.

Katayama, K.; Hamamoto, A.; and Narihisa, H. 2004. Solving the maximum clique problem by k-opt local search. In *Proc. SAC-2004*, 1021–1025.

Katayama, K.; Sadamatsu, M.; and Narihisa, H. 2007. Iterated k-opt local search for the maximum clique problem. In *EvoCOP*, 84–95.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proc. of AAAI-93*, 40–45.

Pullan, W., and Hoos, H. H. 2006. Dynamic local search for the maximum clique problem. *J. Artif. Intell. Res.* 25:159–185.

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

Pullan, W. 2009. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization* 6:214–219.

Richter, S.; Helmert, M.; and Gretton, C. 2007. A stochastic local search approach to vertex cover. In *Proc. of KI-07*, 412–426.

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

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

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

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