Two New Local Search Strategies for Minimum Vertex Cover

Shaowei Cai[1] **Kaile Su**[2,3]* **Abdul Sattar**[3,4]
[1]Key laboratory of High Confidence Software Technologies, Peking University, Beijing, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[3]Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia
[4]ATOMIC Project, Queensland Research Lab, NICTA
shaowei_cai@126.com; {k.su,a.sattar}@griffith.edu.au

Abstract

In this paper, we propose two new strategies to design efficient local search algorithms for the minimum vertex cover (MVC) problem. There are two main drawbacks in state-of-the-art MVC local search algorithms: First, they select a pair of vertices to be exchanged simultaneously, which is time consuming; Second, although they use edge weighting techniques, they do not have a strategy to decrease the weights. To address these drawbacks, we propose two new strategies: two stage exchange and edge weighting with forgetting. The two stage exchange strategy selects two vertices to be exchanged separately and performs the exchange in two stages. The strategy of edge weighting with forgetting not only increases weights of uncovered edges, but also decreases some weights for each edge periodically. We utilize these two strategies to design a new algorithm dubbed NuMVC. The experimental results show that NuMVC significantly outperforms existing state-of-the-art heuristic algorithms on most of the hard DIMACS instances and all instances in the hard random BHOSLIB benchmark.

Introduction

The Minimum Vertex Cover (MVC) problem consists of, given an undirected graph $G = (V, E)$, finding the minimum sized vertex cover, where a vertex cover is a subset $S \subseteq V$ such that every edge in G has at least one endpoint in S. Two other forms of the MVC problem are the Maximum Clique (MC) problem and Maximum Independent Set (MIS) problem. These three problems are prominent combinatorial optimization problems of great importance in theory and applications (Cai, Su, and Sattar 2011).

MVC, MC and MIS are all NP-hard problems (Garey and Johnson 1979); furthermore, it is NP-hard to approximate MVC within any factor smaller than 1.3606 (Dinur and Safra 2005), although one can achieve an approximation ratio of $2 - o(1)$ (Halperin 2002; Karakostas 2005). Also, negative results hold for approximability within any constant factor in polynomial time for MC and MIS (Håstad 1999; 2001). Practical algorithms for MVC (MC, MIS) mainly fall into two types: exact ones and heuristic ones. Exact algorithms, e.g. (Li and Quan 2010b; 2010a), 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 algorithms become futile. Therefore, large and hard MVC (MC, MIS) instances are typically solved using heuristic approaches, which are mainly stochastic local search algorithms.

A huge amount of effort has been devoted to designing heuristic approaches for MVC, MC and MIS problems, including (Aggarwal, Orlin, and Tai 1997; Battiti and Protasi 2001; Busygin, Butenko, and Pardalos 2002; Shyu, Yin, and Lin 2004; Barbosa and Campos 2004; Pullan 2006; Richter, Helmert, and Gretton 2007; Andrade, Resende, and Werneck 2008; Cai, Su, and Chen 2010; Cai, Su, and Sattar 2011), just to name a few. As one of the three problems (The other two are SAT and Coloring.) in "NP hard problems: The Second DIMACS Implementation Challenge", earlier heuristics for Maximum Clique can be found in (Johnson and Trick 1996).

This work is devoted to a more efficient local search algorithm for MVC. Typically, local search algorithms for MVC solve the problem by solving the k-vertex cover problem iteratively. To solve the k-vertex cover problem, they maintain a current candidate solution of size k, and exchange two vertices iteratively until it becomes a vertex cover, where exchanging two vertices means removing one vertex from the current candidate solution and adding another into it.

In our opinion, there are two drawbacks in state-of-the-art MVC local search algorithms. First, they select a pair of vertices for exchanging simultaneously according to some heuristic (Richter, Helmert, and Gretton 2007; Cai, Su, and Chen 2010; Cai, Su, and Sattar 2011), which is rather time consuming, as will be explained in the section "Two Stage Exchange". The second drawback is about the edge weighting techniques. The basic concept of edge weighting is to increase weight of uncovered clauses to diversity the search. Previous MVC algorithms utilize different edge weighting schemes. For example, COVER (Richter, Helmert, and Gretton 2007) increases weights of uncovered edges each step, while EWLS (Cai, Su, and Chen 2010) and EWCC (Cai, Su, and Sattar 2011) increase weights of uncovered edges only when meeting local optima. However, all these algorithms do not have a mechanism to decrease the weights. We believe this is not fruitful because the weighting decisions made too long ago may mislead the search.

*Corresponding author

To address these two drawbacks in MVC local search algorithms, this paper proposes two new strategies, namely two stage exchange and edge weighting with forgetting. According to the two stage exchange strategy, the two vertices to be exchanged are selected separately and exchanged in two stage: it first selects a vertex and removes it from the current candidate solution, and then selects a vertex and adds it. The second strategy we propose is edge weighting with forgetting, which updates edge weights each step, and reduces edge weights by multiplying a constant factor smaller than one to forget the earlier weighting decisions when the averaged weight achieves a threshold. This is the first time a forgetting mechanism is introduced into edge weighting local search algorithms for MVC. Note that smoothed clause weighting techniques in local search for the Boolean satisfiability problem (SAT) are also "forgetting" mechanisms to some extent. However, according to our experiments, direct applications of smoothed weighting techniques cannot improve but even weaken the performance of our MVC algorithm. For more discussions about the differences between the forgetting mechanism in this work and smoothed weighting techniques, please refer to the discussion section.

We combine these two strategies in a simple local search algorithm called NuMVC. To evaluate the performance of NuMVC, we compare it with PLS (Pullan 2006), COVER (Richter, Helmert, and Gretton 2007) and EWCC (Cai, Su, and Sattar 2011), which are the current best heuristic algorithms for MVC (MC, MIS). Our experiments show that NuMVC is largely competitive on the DIMACS benchmark, where it outperforms the other solvers on most of the hard instances; moreover, NuMVC dominates on the whole BHOSLIB benchmark, dramatically improving the existing results. Hence this work represents a significant progress in heuristic algorithms for MVC (MC, MIS).

The remainder of this paper is organized as follows. In the next section, we introduce some definitions and notations used in this paper. Then we present the two strategies: two stage exchange and edge weighting with forgetting. After that, we describe the NuMVC algorithm, and present experimental results demonstrating the performance of NuMVC. Finally, we give some concluding remarks.

Definitions and Notations

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 a 2-element subset of V. For an edge $e = \{u, v\}$, u and v are the *endpoints* of edge e. Two vertices are neighbors if and only if they both belong to some edge. $N(v) = \{u \in V | \{u, v\} \in E\}$ is the set of neighbors of a vertex v.

Given an undirected graph $G = (V, E)$, a *candidate solution* for MVC is a subset of vertices. An edge $e \in E$ is *covered* by a candidate solution X if at least one endpoint of e belongs to X. During the search procedure, NuMVC always maintains a current candidate solution. For convenience, in the rest of this paper, we use C to denote the current candidate solution. The *state* of a vertex v is denoted by $s_v \in \{1, 0\}$, where $s_v = 1$ means $v \in C$, and $s_v = 0$ means $v \notin C$. 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. The *age* of a vertex is the number of steps since its state was last changed.

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 non-negative integer number $w(e)$ as its weight. We use $\overline{w}$ to denote the mean value of all edge weights.

Let w be a weighting function for G. For a candidate solution X, we set the cost of X as

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

which indicates the total weight of edges uncovered by X. We take $cost(G, X)$ as the *evaluation function*, and NuMVC prefers candidate solutions with lower costs.

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, measuring the benefit of changing the state of vertex v. Obviously, $dscore(v) \leq 0$ if $v \in C$, and $dscore(v) \geq 0$ if $v \notin C$.

c' 可能是加入一个node也可能是删除一个node

Two Stage Exchange

In this section, we introduce the two stage exchange framework, which the NuMVC algorithm is based on to exchange a pair of vertices.

As with most state-of-the-art local search algorithms for MVC, NuMVC is an iterated k-vertex cover algorithm. When finding a k-vertex cover, NuMVC removes one vertex from the current candidate solution C and goes on to search for a $(k-1)$-vertex cover. In this sense, the core of NuMVC is a k-vertex cover algorithm — given a positive integer number k, searching for a k-sized vertex cover. To find a k-vertex cover, NuMVC begins with a candidate solution C of size k, and then exchanges two vertices iteratively until C becomes a vertex cover.

Most local search algorithms for MVC select a pair of vertices to be exchanged simultaneously according to some heuristic. For example, COVER selects a pair of vertices that maximize $gain(u, v)$ (Richter, Helmert, and Gretton 2007); EWLS (Cai, Su, and Chen 2010) and EWCC (Cai, Su, and Sattar 2011) select a random pair of vertices with $score(u, v) > 0$. A drawback of selecting two vertices to be exchanged simultaneously is that, the evaluation of a pair of vertices not only depends on the evaluations (such as *dscore*) of the two vertices, but also involves the relationship between the two vertices, like "do they belong to a same edge". Therefore, it is rather time consuming to evaluate each candidate pair of vertices.

In contrast to earlier MVC local search algorithms, NuMVC selects the two vertices for exchanging separately in two stages. In each iteration, NuMVC first selects a vertex $u \in C$ with the highest *dscore* and removes it. After that, NuMVC selects a uniformly random uncovered edge e, and chooses one endpoint v of e with the higher *dscore* under some restrictions and adds it into C.

Selecting the two vertices for exchanging separately may in some cases miss some greedier vertex pairs which consist of two neighboring vertices. Nevertheless, as is usual in local search algorithms, there is a tradeoff between the accuracy of heuristics and the complexity per step. Let R and A denote the set of candidate vertices for removing and adding separately. The time complexity per step for selecting the exchanging vertex pair simultaneously is $|R| \cdot |A|$; while the complexity per step for selecting the two vertices separately, as in NuMVC, is only $|R| + |A|$.

To sum up, the two stage exchange framework slightly brings down the greediness of the algorithm, and significantly cuts down the time complexity per step. It is worthy to note that, as heuristics in a local search algorithm are often based on intuition and experience rather than on theoretically or empirically derived principles and insights, we cannot say for certain that being less greedy is not a good thing (Hoos and Stützle 2005). On the other hand, a lower time complexity is always what we expect.

Edge Weighting with Forgetting

This section presents a new edge weighting technique called edge weighting with forgetting, which plays an important role in NuMVC.

The proposed strategy of edge weighting with forgetting works as follows: each edge is associated to a positive integer number as its weight, and each edge weight is initialized as one. Then in each iteration, edge weights of the uncovered edges are increased by one. Moreover, when the average weight achieves a threshold, all edge weights are reduced to forget the earlier weighting decisions using the formula $w(e) := \rho \cdot w(e)$, where $\rho < 1$.

Edge weighting techniques, as a form of diversification, have been used in MVC local search algorithms. For example, COVER (Richter, Helmert, and Gretton 2007) updates edge weights each step, while EWLS (Cai, Su, and Chen 2010) and EWCC (Cai, Su, and Sattar 2011) update edge weights only when meeting local optima. However, these edge weighting techniques do not have a mechanism to decrease the weights. The essential difference between edge weighting with forgetting and previous edge weighting techniques is that it introduces a forgetting mechanism to reduce edge weights periodically.

The intuition behind the forgetting mechanism is that the weighting decisions made too long ago are no longer helpful and may mislead the search. Therefore, these weighting decisions should be considered less important than the recent ones. For example, consider two edges e_1 and e_2 with $w(e_1) = 1000$ and $w(e_2) = 100$ at some step. We use $\Delta w(e)$ to denote the increase of $w(e)$. According to the *evaluation function*, in the next period of time, the algorithm is likely to cover e_1 more frequently than e_2, and we may assume that during this period $\Delta w(e_1) = 50$ and $\Delta w(e_2) = 500$, which makes $w(e_1) = 1000 + 50 = 1050$ and $w(e_2) = 100 + 500 = 600$. Without a forgetting mechanism, the algorithm would still prefer e_1 to e_2 to be covered in the future search. This is not reasonable, as during this period the number of steps in which e_2 is covered is much less than the number of steps in which e_1 is covered. Thus, e_2 should take priority to be covered for the sake of diversification. Now let us consider the case with the forgetting mechanism (assuming $\rho = 0.3$ which is the setting in our experiments). Suppose $w(e_1) = 1000$ and $w(e_2) = 100$ when the algorithm performs the forgetting. The forgetting mechanism reduces the edge weights as $w(e_1) = 1000 \times 0.3 = 300$ and $w(e_2) = 100 \times 0.3 = 30$. After a period of time, with $\Delta w(e_1) = 50$ and $\Delta w(e_2) = 500$, we have $w(e_1) = 300 + 50 = 350$ and $w(e_2) = 30 + 500 = 530$. In this case, the algorithm prefers to cover e_2 rather than cover e_1 in the future search, as we expect.

The NuMVC Algorithm

In this section, we present the NuMVC algorithm, which is based on the strategies of two stage exchange and edge weighting with forgetting.

For better understanding the NuMVC algorithm, we first describe a strategy called configuration checking (CC), which is used in NuMVC. The CC strategy was proposed in (Cai, Su, and Sattar 2011) for handling the cycling problem in local search, i.e., revisiting a candidate solution that has been visited recently (Michiels, Aarts, and Korst 2007), and has been used in local search algorithms for MVC (Cai, Su, and Sattar 2011) as well as SAT (Cai and Su 2011). The CC strategy in NuMVC works as follows: For a vertex $v \notin C$, if all its neighboring vertices never change their *states* since v's last removal from C, then v should not be added back to C. An implementation of the CC strategy is to maintain a boolean array $confChange$. During the search procedure, vertices with $confChange[v] = 0$ are forbidden to be added into C. The $confChange$ array is initialized as an all-1 array. After that, when a vertex v is removed from C, $confChange[v]$ is reset to 0; when a vertex v changes its $state$, for each $u \in N(v)$, $confChange[u]$ is set to 1.

We outline the NuMVC algorithm in Algorithm 1, as described below. In the beginning, all edge weights are initialized as 1, and $dscores$ of vertices are computed accordingly; $confChange[v]$ is initialized as 1 for each vertex v; then the current candidate solution C is constructed by adding the vertex with the highest $dscore$ iteratively until it becomes a vertex cover, and the best solution C^* is initialized as C.

After the initialization, the loop (lines 7-18) is executed until a given cutoff time is reached. During the search procedure, once there is no uncovered edge, which means C is a vertex cover, NuMVC updates the best solution C^* as C (line 9), and then removes the vertex with the highest $dscore$ from C (line 10), going on to search for a vertex cover with a smaller size. We note that, in C, the vertex with the highest $dscore$ has the minimum absolute value of $dscore$ since all these $dscores$ are negative.

In each iteration of the loop, NuMVC swaps two vertices according to the strategy of two stage exchange (line 12-16). Specifically, it first selects a vertex $u \in C$ with the highest $dscore$ to remove, breaking ties in favor of the oldest vertex. NuMVC keeps track of the vertex last inserted into C, and prevents it from being removed immediately. After removing u, NuMVC selects a random uncovered edge e, and selects one of e's endpoints to add into C as follows: If only one endpoint of e satisfies $confChange[v] = 1$, then that vertex is selected; if both endpoints of e satisfy $confChange[v] = 1$, then NuMVC selects the one with the higher $dscore$, breaking ties in favor of the older vertex. The exchange is finished by adding the selected vertex into C. Along with exchanging the two vertices u and v, the $confChange$ array is updated accordingly.

Algorithm 1: NuMVC

```
1  NuMVC (G,cutoff)
   Input: graph $G = (V, E)$, the cutoff time
   Output: vertex cover of $G$
2  begin
3      initialize edge weights and $dscores$ of vertices;
4      initialize the $confChange$ array as an all-1 array;
5      construct $C$ greedily until it is a vertex cover;
6      $C^* := C$;
7      while elapsed time < cutoff do
8          if there is no uncovered edge then
9              $C^* := C$;
10             remove a vertex with the highest $dscore$ from $C$;
11             continue;
12         choose a vertex $u \in C$ with the highest $dscore$,
           breaking ties in favor of the oldest one;
13         $C := C \backslash \{u\}$, $confChange(u) := 0$ and
           $confChange(z) := 1$ for each $z \in N(u)$;
14         choose an uncovered edge $e$ randomly;
15         choose a vertex $v \in e$ such that
           $confChange[v] = 1$ with higher $dscore$, breaking
           ties in favor of the older one;
16         $C := C \cup \{v\}$, $confChange(z) := 1$ for each
           $z \in N(v)$;
17         $w(e) := w(e) + 1$ for each uncovered edge $e$;
18         if $\overline{w} \geq \gamma$ then $w(e) := \lfloor \rho \cdot w(e) \rfloor$ for each edge $e$;
19     return $C^*$;
```

At the end of each iteration, NuMVC updates the edge weights (line 17-18). Weights of all uncovered edges are increased by one. Moreover, NuMVC uses the forgetting mechanism. Specifically, if the average weight of all edges achieves a threshold, then all edge weights are multiplied by a constant factor ρ ($\rho < 1$) and rounded down to the integer as edge weights are defined as integers in NuMVC. The forgetting mechanism forgets the earlier weighting decision to some extent, as these past effects are generally no longer helpful and may mislead the search.

We conclude this section by the following observation, which guarantees the executability of line 15.

Proposition 1 *For an uncovered edge e, there is at least one vertex $v \in e$ such that $confChange[v] = 1$.*

Proof: Let $e = \{v_1, v_2\}$, the proof includes two cases.
(a) *There is at least one of v_1 and v_2 which never changes its state after initialization.* Without loss of generality, we assume v_1 is such a vertex. In the initialization, we have that $confChange[v_1] = 1$. After that, only removing v_1 from C (which corresponds to v's state s_v changing to 0 from 1) can make $confChange[v_1]$ to be 0, but v_1 never changes its state after initialization, so we have $confChange[v_1] = 1$.
(b) *Both v_1 and v_2 change their states after initialization.* As e is uncovered, we have $v_1 \notin C$ and $v_2 \notin C$. Without loss of generality, we assume the last removing of v_1 happens before the last removing of v_2. When the last time v_1 is removed, $v_2 \in C$ holds. Afterwards, v_2 is removed, which means v_2 changes its state, so $confChange[v_1]$ is set to 1 as $v_1 \in N(v_2)$. ∎

Empirical Results

We evaluate NuMVC against the best known local search algorithms for MVC (MC, MIS) on standard benchmarks in the literature, i.e., the DIMACS and BHOSLIB benchmarks. We first present a brief introduction to the DIMACS and BHOSLIB benchmarks, and describe some preliminaries about the experiments. Then, for each benchmark set, we compare NuMVC with the best known local search algorithms. We also perform additional experiments to study the effectiveness of the two stage exchange strategy.

Seen from recent literature, there are five state-of-the-art heuristic algorithms for MVC (MC, MIS): three MVC algorithms COVER (Richter, Helmert, and Gretton 2007), EWLS (Cai, Su, and Chen 2010) and EWCC (Cai, Su, and Sattar 2011), and two MC algorithms DLS-MC (Pullan and Hoos 2006) and PLS (Pullan 2006). EWCC and PLS are the improved versions of EWLS and DLS-MC respectively, and show better performance over their original versions on DIMACS and BHOSLIB benchmarks. Therefore, we compare NuMVC only with PLS, COVER and EWCC.

The Benchmarks

The DIMACS benchmark is taken from the Second DIMACS Challenge Test Problems (1992-1993)[1]. These instances were generated from real world problems such as coding theory, fault diagnosis, Kellers conjecture and the Steiner Triple Problem, etc, and random graphs in various models, such as the *brock* and p_hat families. These 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 BHOSLIB[2] (Benchmarks with Hidden Optimum Solutions) instances were generated randomly in the phase transition area according to the model RB (Xu and Li 2000; Xu et al. 2007). The SAT version of the BHOSLIB benchmark is extensively used in the SAT competition[3]. Nevertheless, SAT solvers are much weaker than MVC solvers on these problems (Cai, Su, and Sattar 2011), which remains justifiable when referring to the results of the SAT Competition 2011 on this benchmark. The BHOSLIB benchmark is famous for its hardness and so influential as strongly recommended by the MVC (MC, MIS) community (Grosso, Locatelli, and Pullan 2008; Cai, Su, and Sattar 2011). It has been widely used in the recent literature as a reference point for new heuristics to MVC, MC and MIS. Besides these 40 instances, there is a large instance frb100-40 with 4,000 vertices and 572,774 edges, which is designed for challenge.

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

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

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

Graph		PLS		COVER		EWCC		NuMVC	
Instance	k^*	suc	time	suc	time	suc	time	suc	time
brock400_2	371	**100**	**0.15**	3	1947	20	1778	96	572
brock400_4	367	100	**0.03**	82	960	100	25.38	100	4.89
brock800_2	776	**100**	**3.89**	0	n/a	0	n/a	0	n/a
brock800_4	774	**100**	**1.31**	0	n/a	0	n/a	0	n/a
C2000.9	1920	0	n/a	0	n/a	0	n/a	**1**	**1994**
C4000.5	3982	100	**67**	100	658	100	739	100	152
gen400_p0.9_55	345	100	15.17	100	0.35	100	0.05	100	**0.02**
keller6	3302	92	559	100	68	100	3.76	**100**	**2.37**
MANN_a45	690	1	1990	94	714	88	763	**100**	**86**
MANN_a81	2221	0	n/a	1	1995	1	1986	**27**	**1657**
p_hat1500-1	1488	100	**2.36**	100	18.10	100	9.79	100	3.75

Table 1: Comparative results on the DIMACS benchmark

Experiment Preliminaries

NuMVC is implemented in C++. The codes of COVER are downloaded on line[4] and those of PLS and EWCC are provided by their authors respectively. All the four solvers are compiled by g++ with the '-O2' option. All experiments are run on a 3 GHz Intel Core 2 Duo CPU E8400 and 4GB RAM under Linux. To execute the DIMACS machine benchmarks[5], this machine requires 0.19 CPU seconds for r300.5, 1.12 CPU seconds for r400.5 and 4.24 CPU seconds for r500.5.

For NuMVC, we set $\gamma = |V|/2$ and $\rho = 0.3$ for all runs, except for the challenging instance frb100-40, where $\gamma = 5000$ and $\rho = 0.3$. Other state-of-the-art MVC (MC, MIS) algorithms also have parameters, such as DLS-MC (Pullan and Hoos 2006) and EWLS (Cai, Su, and Chen 2010). Moreover, the parameters in DLS-MC and EWLS vary on different instances.

For each instance, each algorithm is performed 100 runs with different random seeds, where each run is terminated upon reaching a given cutoff time which is set to 2000 seconds. We report the following information for each instance: the optimal (or minimum known) vertex cover size (k^*); the number of successful runs in which a solution of size k^* is found ("suc"); and the averaged run time in CPU seconds over all runs ("time"), where the run time of a failed run is considered to be the cutoff time. If there are no successful runs, the "time" column is marked with "n/a". The results in bold indicate the best performance for an instance.

The averaged run time over only successful runs cannot indicate comparative performance of algorithms correctly unless the evaluated algorithms have close success rates, and can be calculated by $\frac{\text{“}time\text{”}*100-cutoff*(100-\text{“}suc\text{”})}{\text{“}suc\text{”}}$, so we do not report these statistics.

DIMACS Benchmark Results

The results on the DIMACS benchmark are shown in Table 1. Most DIMACS instances are so easy that they can be solved by all solvers with 100% success rate within 2 seconds, and thus are not reported in the table.

Table 1 shows that NuMVC outperforms COVER and EWCC on all instances and competes well with PLS.

Graph		PLS		COVER		EWCC		NuMVC	
Instance	k^*	suc	time	suc	time	suc	time	suc	time
frb40-19-1	720	100	10.42	100	1.58	100	0.55	100	**0.24**
frb40-19-2	720	100	85.25	100	17.18	100	11.30	100	**4.08**
frb40-19-3	720	100	9.06	100	5.06	100	2.97	100	**1.07**
frb40-19-4	720	100	77.39	100	11.79	100	13.79	100	**2.76**
frb40-19-5	720	95	496	100	124	100	41.71	100	**10.14**
frb45-21-1	900	100	52.31	100	14.34	100	9.07	100	**2.71**
frb45-21-2	900	100	170	100	38	100	15	100	**5**
frb45-21-3	900	21	1737	100	110	100	56	100	**14**
frb45-21-4	900	100	111	100	21	100	15	100	**4**
frb45-21-5	900	100	261	100	105	100	42	100	**11**
frb50-23-1	1100	30	1658	100	268	100	124	100	**38**
frb50-23-2	1100	3	1956	48	1325	82	905	**100**	**177**
frb50-23-3	1100	2	1989	39	1486	56	1348	**95**	**606**
frb50-23-4	1100	100	93	100	33	100	24	100	**8**
frb50-23-5	1100	79	967	100	168	100	85	100	**19**
frb53-24-1	1219	1	1982	17	1796	30	1696	**86**	**895**
frb53-24-2	1219	6	1959	50	1279	81	1006	**100**	**205**
frb53-24-3	1219	20	1771	99	273	100	117	100	**51**
frb53-24-4	1219	21	1782	48	1428	81	900	**100**	**266**
frb53-24-5	1219	10	1955	95	423	100	125	100	**40**
frb56-25-1	1344	1	1993	24	1698	56	1268	**100**	**470**
frb56-25-2	1344	0	n/a	17	1598	52	1387	**97**	**659**
frb56-25-3	1344	0	n/a	97	537	100	285	100	**121**
frb56-25-4	1344	11	1915	93	476	100	183	100	**50**
frb56-25-5	1344	27	1719	100	168	100	80	100	**27**
frb59-26-1	1475	0	n/a	16	1607	21	1778	**88**	**843**
frb59-26-2	1475	0	n/a	9	1881	7	1930	**37**	**1677**
frb59-26-3	1475	3	1978	21	1768	64	1294	**96**	**636**
frb59-26-4	1475	0	n/a	3	1980	20	1745	**79**	**1004**
frb59-26-5	1475	30	1708	98	431	100	174	100	**62**

Table 2: Comparative results on the BHOSLIB benchmark

NuMVC significantly outperforms PLS on five instances, including the two putatively hardest instances C2000.9 and MANN_a81 (Richter, Helmert, and Gretton 2007; Grosso, Locatelli, and Pullan 2008; Cai, Su, and Sattar 2011), as well as keller6, MANN_a45 and gen400_p0.9_55. For C2000.9, only NuMVC finds a 1920-sized solution, and NuMVC finds a 1921-sized solution in 70 runs, while this number is 31, 6 and 32 for PLS, COVER, and EWCC respectively.

Nevertheless, PLS significantly outperforms NuMVC on the brock graphs and C4000.5. The brock graphs are designed to defeat greedy heuristics by explicitly incorporating low-degree vertices into the optimal vertex cover. Indeed, most algorithms preferring higher-degree vertices such as GRASP, RLS, k-opt, COVER and EWCC also failed in these graphs. PLS performs well on the brock family because it comprises three sub-algorithms, one of which favors the lower degree vertices.

Recently, a branch-and-bound MC algorithm named MaxCLQ (Li and Quan 2010b) and its improved version MaxCLQdyn+EFL+SCR (Li and Quan 2010a) show a considerable progress on solving DIMACS instances by exact algorithms. It would be interesting to compare NuMVC with MaxCLQdyn+EFL+SCR. We leave this for future work.

BHOSLIB Benchmark Results

Table 2 shows comparative results on the BHOSLIB benchmark. 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 within 2 seconds by

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

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

all solvers, and the results are consistent with Table 2.

The results demonstrate that NuMVC significantly outperforms all the other algorithms on all BHOSLIB instances, in terms of both success rate and averaged run time. We take a further look at the performance comparison between NuMVC and EWCC, as EWCC performs obviously better than PLS and COVER on this benchmark. NuMVC solves 33 instances out of 40 with 100% success rate, 4 more instances than EWCC does; For instances solved by both algorithms with 100% success rate, the overall averaged run time is 25 seconds for NuMVC and 74 seconds for EWCC; For other instances, the averaged success rate is 90% for NuMVC, compared to 50% for EWCC.

The excellent performance of NuMVC is further underlined by the large gaps between NuMVC and the other solvers on the hard instances. For those hard instances where all solvers fail to find an optimal solution with 100% success rate, NuMVC achieves an averaged success rate of 82.57%, dramatically better than that of PLS, COVER and EWCC, which are 0.85%, 17.43% and 35.71% respectively. Obviously, the experimental results show that NuMVC delivers the best performance for this hard random benchmark, vastly improving the existing performance results.

Also, we would like to remark that the performance of NuMVC on the BHOSLIB benchmark is better than a four core version of CLS (Pullan, Mascia, and Brunato 2011), even if we do not divide the runtime of NuMVC by 4 (the number of cores utilized by CLS). If we consider the machine speed ratio and divide the runtime of NuMVC by 4, then NuMVC would be dramatically better than CLS on the BHOSLIB benchmark.

For the challenging instance frb100-40, which has a hidden minimum vertex cover of size 3900, the designer of the BHOSLIB benchmark conjectured that this instance will not be solved on a PC in less than a day within the next two decades[6]. The latest record for this challenging instance is a 3902-sized vertex cover found by EWLS, and also EWCC.

Solver	3902		$\leq$ 3903	
	suc	time	suc	time
COVER	0	n/a	33	2768
EWCC	1	2856	79	2025
NuMVC	4	2955	93	1473

Table 3: Results on frb100-40 Challenging Instance

We run COVER, EWCC and NuMVC 100 trials within 4000 seconds on frb100-40. Given the failure of PLS on large BHOSLIB instances, we do not run PLS on this instance. Table 3 shows that NuMVC significantly outperforms EWCC and COVER on frb100-40. For NuMVC, 4 runs find a 3902-sized solution with the average time of 2955 seconds, and 93 runs find a 3903-sized solution (including 3902-sized) with the average time of 1473 seconds.

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

Graph Instance	PLS #steps/sec	COVER #steps/sec	EWCC #steps/sec	NuMVC #steps/sec
C4000.5	85,318	8,699	11,927	513,307
MANN_a45	1546,625	279,514	578,656	991,476
p_hat_1500-1	170,511	19,473	34,111	297,220
frb53-24-5	841,346	128,971	219,038	570,425
frb56-25-5	801,282	116,618	199,441	522,561
frb59-26-5	706,436	108,534	189,536	511,014

Table 4: Complexity per step on selected instances

The Effectiveness of Two Stage Exchange

We study the effectiveness of the two stage exchange heuristic by comparing the number of search steps per second on representative instances.

Seen from Table 4, the number of search steps per second of NuMVC is much more than those of the other two MVC local search solver COVER and EWCC, where the increase ranges from several to several dozen times. This indicates that the two stage exchange strategy can significantly accelerate MVC local search algorithms. Although PLS performs more steps per second than NuMVC, PLS is an MC local search algorithm whose search scheme is essentially different from those of MVC local search algorithms.

Discussion

The forgetting mechanism in NuMVC is inspired by smoothing techniques in clause weighting local search algorithms for SAT. However, it is different from those smoothing techniques in SAT local search algorithms.

According to the way that clause weights are smoothed, there are three main smoothing techniques in SAT local search to our knowledge: the first is to pull all clause weights to their mean value using $w_i := \rho \cdot w_i + (1-\rho) \cdot \overline{w}$, as in ESG (Schuurmans, Southey, and Holte 2001), SDF (Schuurmans and Southey 2001) and SPAS (Hutter, Tompkins, and Hoos 2002); the second is to subtract one from all clause weights which are bigger than 1, as in DLM (Wu and Wah 2000) and PAWS (Thornton et al. 2004); and the last is employed in DDWF (Ishtaiwi et al. 2005), which transfers weights from neighbouring satisfied clauses to unsatisfied ones.

The forgetting mechanism in NuMVC is different from these smoothing techniques in SAT solving. Although the second smoothing technique can be also seen as a forgetting mechanism to some extent, the forgetting mechanism in NuMVC is done by multiplying all edge weights by a constant factor ρ that is smaller than one, rather than by subtracting weights. Also, NuMVC forgets weights on the condition that the averaged weight achieves a threshold. The forgetting mechanism in NuMVC is novel and has not been used in smoothing techniques in literature we have found.

We conducted a few experiments to study the performance of the alternative versions of NuMVC by replacing the forgetting mechanism with the smoothing techniques similar to those in SAT local search mentioned above, and found that those smoothing techniques only seemed to weaken the performance. It would be interesting to find out the reasons of the success of the forgetting mechanism and the failure of the smoothing techniques in MVC edge weighting local search algorithms like NuMVC. We leave this direction of investigation for future work.

Conclusions and Future Work

We presented two new local search strategies for the minimum vertex cover (MVC) problem, namely two stage exchange and edge weighting with forgetting. Based on these two strategies, we developed a local search algorithm for MVC, called NuMVC. The NuMVC algorithm is evaluated against the best known local search algorithms for MVC (MC, MIS) on standard benchmarks, i.e., the DIMACS and BHOSLIB benchmarks. The results are convincing and concluding in this highly competitive field.

We believe that the algorithm can be further improved by more elaborate techniques, especially by better heuristics for the two stage exchange framework and more effective edge weighting schemes. Also, it is interesting to investigate how the forgetting mechanism in NuMVC can be applied to local search algorithms for other combinatorial search problems.

Acknowledgement

This work is partially supported by 973 Program (2010CB328103), ARC grants FT0991785 and DP120102489, and Open Project SYSKF1003 of State Key Laboratory of Computer Science. We would like to thank the anonymous referees for their helpful comments.

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.

Cai, S., and Su, K. 2011. Local search with configuration checking for sat. In *Proc. of ICTAI-11*, 59–66.

Cai, S.; Su, K.; and Chen, Q. 2010. Ewls: A new local search for minimum vertex cover. In *Proc. of AAAI-10*, 45–50.

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

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

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

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.

Hoos, H., and Stützle, T. 2005. *Stochastic Local Search: Foundations and Applications*. San Francisco, CA, USA: Morgan Kaufmann.

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.

Ishtaiwi, A.; Thornton, J.; Sattar, A.; and Pham, D. N. 2005. Neighbourhood clause weight redistribution in local search for sat. In *Proc. of CP-05*, 772–776.

Johnson, D. S., and Trick, M., eds. 1996. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA.

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

Li, C. M., and Quan, Z. 2010a. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *Proc. of ICTAI (1)*, 344–351.

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

Michiels, W.; Aarts, E. H. L.; and Korst, J. H. M. 2007. *Theoretical aspects of local search*. Springer.

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

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

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

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

Schuurmans, D., and Southey, F. 2001. Local search characteristics of incomplete sat procedures. *Artif. Intell.* 132(2):121–150.

Schuurmans, D.; Southey, F.; and Holte, R. C. 2001. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. of IJCAI-01*, 334–341.

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.

Thornton, J.; Pham, D. N.; Bain, S.; and Jr., V. F. 2004. Additive versus multiplicative clause weighting for sat. In *Proc. of AAAI-04*, 191–196.

Wu, Z., and Wah, B. W. 2000. An efficient global-search strategy in discrete lagrangian methods for solving hard satisfiability problems. In *Proc. of AAAI/IAAI-00*, 310–315.

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

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.