A New Solver for the Minimum Weighted Vertex Cover Problem

Hong Xu, T. K. Satish Kumar, and Sven Koenig

University of Southern California, Los Angeles CA 90089, USA
hongx@usc.edu, tkskwork@gmail.com, skoenig@usc.edu

Abstract. Given a vertex-weighted graph $G = \langle V, E \rangle$, the minimum weighted vertex cover (MWVC) problem is to choose a subset of vertices with minimum total weight such that every edge in the graph has at least one of its endpoints chosen. While there are good solvers for the unweighted version of this NP-hard problem, the weighted version—i.e., the MWVC problem—remains understudied despite its common occurrence in many areas of AI—like combinatorial auctions, weighted constraint satisfaction, and probabilistic reasoning. In this paper, we present a new solver for the MWVC problem based on a novel reformulation to a series of SAT instances using a primal-dual approximation algorithm as a starting point. We show that our SAT-based MWVC solver (SBMS) significantly outperforms other methods.

1 Introduction

Given a directed or undirected graph $G = \langle V, E \rangle$, a *vertex cover* of G is defined as a collection of vertices $S \subseteq V$ such that every edge in E has at least one of its endpoint vertices in S. A *minimum vertex cover* (MVC) of G is a vertex cover of minimum cardinality. When G is vertex-weighted—i.e., each vertex $v_i \in V$ has a non-negative weight w_i associated with it—the *minimum weighted vertex cover* (MWVC) for it is defined as a vertex cover of minimum total weight.

Two important combinatorial problems equivalent to the MVC problem are the *maximum independent set* (MIS) problem and the *maximum clique* (MC) problem [8]. The MVC problem and its equivalent MIS and MC problems have numerous real-world applications such as in AI scheduling, logistics and operations management, and VLSI design. More recent applications have also been discovered in information retrieval, signal processing, and sequence alignment in computational genomics [14].

Since the MVC problem is a special case of the MWVC problem, the latter not only captures all of the real-world combinatorial problems that the MVC problem can model but also captures a wide range of other combinatorial problems central to AI. For example, consider a simple combinatorial auction problem. We are given a set of items with bids placed on subsets of the items. Each bid has a valuation. The goal is to pick a set of winning bids that maximizes the total valuation—i.e., revenue of the auctioneer—and allocates each item to

at most one winning bid. This can be modeled as a *maximum weighted independent set* (MWIS) problem—equivalent to the MWVC problem—as follows. We create a vertex for each bid such that the weight of the vertex is equal to the valuation of that bid. Two vertices are connected by an edge if and only if their corresponding bids have a non-empty intersection. It is easy to see that the winning bids correspond to the vertices in the MWIS for the graph.

In [19,20], the MWVC problem has also been identified as being fundamental to solving *weighted constraint satisfaction problems* (WCSPs). Any combinatorial problem posed as a WCSP is equivalent to the MWVC problem for its associated *constraint composite graph* [19, 20]. An efficient solver for the MWVC problem, therefore, has important implications on how well we can solve the plethora of real-world problems that can be modeled as WCSPs. Examples include—but are not limited to—representing and reasoning about user preferences [3], oversubscription planning with goal preferences [10], and various resource allocation problems. Quite importantly, WCSPs also arise as *energy minimization problems* (EMPs) in probabilistic settings. In computer vision applications, for example, tasks such as image restoration, total variation minimization, and panoramic image stitching can be formulated as EMPs derived in the context of *markov random fields* [17, 18].

The MVC problem has received a lot of recent attention in response to the DIMACS Implementation Challenge [14]. There are both exact and heuristic algorithms for solving the MVC problem. Exact algorithms mainly use branch-and-bound techniques [21, 28]. While they guarantee optimality, they may not scale efficiently to be able to solve large problem instances. Heuristic and local search methods, on the other hand, can provide near-optimal solutions to larger and harder problem instances [6, 26]. As a matter of fact, the NuMVC solver [5] integrates many interesting local search techniques for the MVC problem and performs very well in practice.

While there are reasonably good solvers for the MVC problem, the MWVC problem remains understudied. Clearly, the MWVC problem is a generalization of the MVC problem and is harder to solve efficiently. Exact algorithms based on the branch-and-bound technique are not expected to do well for large instances of the MWVC problem simply because they do not scale well even for large instances of the MVC problem. Moreover, the local search techniques used in the best solvers for the MVC problem are also not expected to generalize well to the MWVC problem. This is because the MVC problem is fixed-parameter tractable while the MWVC problem is not [7]. The local search solvers for the MVC problem [5, 26] heavily rely on this property as they solve the fixed-parameter vertex cover problem in their inner loops.

In this paper, we present a new solver for the MWVC problem based on a novel reformulation to a series of SAT instances using a primal-dual approximation algorithm as a starting point. Our SAT-based MWVC solver (SBMS) implements an anytime algorithm that trades off running time with the quality of the produced solution. Moreover, SBMS also reports on how good the produced solution is guaranteed to be with respect to the optimal solution. In many cases,

SBMS converges to the optimal solution in a few iterations and reports it within the allocated amount of time. Empirical results show that SBMS significantly outperforms other methods.

2 Background

The MVC problem is a well known NP-hard problem [8]. There exists a simple factor-2 approximation algorithm for it that runs in polynomial time [29].[1] There are also polynomial-time algorithms that yield slightly better approximation factors but are more involved [15]. However, the MVC problem is also known to be APX-complete. It cannot be approximated arbitrarily well unless P = NP [29]. Furthermore, PCP theorems yield inapproximability results for designing polynomial-time algorithms with approximation factors better than 1.36 [9].[2]

The MWVC problem is harder than the MVC problem since it is a generalization of the latter. The negative results associated with the MVC problem therefore carry over to the MWVC problem. Fortunately, the MWVC problem is still amenable to a fairly simple polynomial-time factor-2 approximation algorithm based on the idea of *linear programming duality* [29]. However, unlike the MVC problem, the MWVC problem is not fixed-parameter tractable [7]. The MVC problem is in fact studied as a central problem in parameterized complexity theory and can be formulated as a half-integral linear programming problem whose dual yields a maximum matching in the corresponding graph [29].

The good solvers for the MVC problem are based on local search [5, 6, 26]. They implicitly exploit the fixed-parameter tractability of the MVC problem in their inner loops. In order to solve the k-vertex cover problem—i.e., find a vertex cover of size k—in their inner loops, they maintain a current set of vertices of size k and iteratively exchange two vertices—one inside and one outside of this set—until it becomes a valid vertex cover. The state-of-the-art solver for the MVC problem, NuMVC [5], also exploits the fixed-parameter tractability of the MVC problem but with added optimizations.

The NuMVC solver mainly introduces two new techniques not present in its predecessors [5]. The first optimization decomposes the exchange process into two stages—one stage for removing a vertex from the set and the other for adding a vertex to the set. This decomposition leads to linear-time subroutines for each stage instead of the original quadratic-time subroutine that deliberates all pairs of vertices for a possible exchange. The second optimization involves weighting the edges across different iterations while simultaneously employing a mechanism to forget weighting decisions made too far in the past [5].

[1] While the MVC is approximable within a constant factor, this has no implications on the MIS problem. In fact, the MIS problem is one of the hardest combinatorial problems and has no polynomial-time constant-factor approximation algorithm unless P = NP [29].

[2] This inapproximability result is tighter under the *unique games conjecture* [16].

3 Reformulations of the MWVC Problem

Given that there are no standard solvers for the understudied MWVC problem, we develop a solver based on reformulating it to a series of SAT instances. We study the usefulness of this reformulation in comparison to modeling the MWVC problem as an Integer Linear Program (ILP), a Pseudo-Boolean Optimization (PBO) problem, a MAX-SAT problem, or an Answer Set Program (ASP).

3.1 Reformulation as an ILP or a PBO Problem

For a given vertex-weighted undirected (or directed) graph $G = \langle V, E \rangle$, the MWVC problem can be formulated as an ILP as follows. We simply associate a 0/1 variable X_i with each vertex $v_i \in V$. X_i indicates the presence of v_i in the MWVC. Here, w_i is the non-negative weight associated with vertex v_i.

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{|V|} w_i X_i \\ \forall\ v_i \in V: \quad & X_i \in \{0,1\} \\ \forall\ (v_i, v_j) \in E: \quad & X_i + X_j \geq 1 \end{aligned} \tag{1}$$

To reformulate the MWVC problem as a PBO problem, we simply change the "type" of each variable X_i in the ILP formulation from a 0/1 integer to a Boolean variable.

3.2 Reformulation as a MAX-SAT Problem

The MWVC problem can also be formulated as a weighted MAX-SAT problem—simply referred to as the "MAX-SAT problem" here. In a MAX-SAT problem, we are given a set of clauses on Boolean variables. Each clause has a reward associated with satisfying it. The goal is to find a complete assignment of Boolean values to all variables so as to maximize the sum of the rewards associated with the satisfied clauses. The MAX-SAT problem is a well known NP-hard problem [8].

The reformulation of the MWVC problem to the MAX-SAT problem is easy to understand by first modeling the complement of the MWVC problem—i.e., the MWIS problem—as a MAX-SAT problem. Once again, we associate a Boolean variable X_i with each vertex $v_i \in V$ of weight w_i. For each edge $(v_i, v_j) \in E$, we create the clause $(\overline{X_i} \vee \overline{X_j})$ with a very high reward so that there is no incentive to violate it.[3] These clauses represent an independent set in the graph. For each vertex $v_i \in V$, we also add the singleton clause X_i with an associated reward of w_i. It is easy to see that solving the MAX-SAT problem over all these clauses with their associated rewards solves the MWIS problem on the given graph.

[3] It suffices for this reward to be greater than the sum of the weights of all vertices in the graph.

3.3 Reformulation as an ASP

To formulate the MWVC problem as an ASP, we use a constant to represent each vertex. We define a predicate "edge" to represent the edges in the graph. We also define a predicate "picked" to represent whether a vertex is in the MWVC. We define a function "cost" to denote the cost of picking a vertex. Equation 2 captures the nature of undirected edges and vertex cover constraints. The goal is to minimize the sum of the costs of all picked vertices.

$$\begin{aligned} edge(X,Y) &\leftarrow edge(Y,X) \\ picked(X) \vee picked(Y) &\leftarrow edge(X,Y) \end{aligned} \tag{2}$$

3.4 Reformulation as a Series of SAT Instances

An instance of the MWVC problem can be reformulated as a series of SAT instances with each SAT instance answering the question: "Is there a vertex cover of weight less than a given test weight w_{test}?" Solving these SAT instances iteratively converges to a solution of the MWVC problem since we can conduct binary search for the cost of the optimal solution within the interval $[0, \sum_{i=1}^{|V|} w_i]$.[4]

Formulating Each SAT Instance: Consider associating a Boolean variable X_i with each vertex $v_i \in V$ of weight w_i. X_i indicates the presence of v_i in the MWVC. Each SAT instance is intended to search for a vertex cover of weight less than a test weight w_{test}. The clauses in the SAT instance should therefore encode two properties: (a) the validity of the vertex cover; and (b) the weight of the vertex cover being less than w_{test}.

The validity of the vertex cover is enforced simply by having a clause $(X_i \vee X_j)$ for each $(v_i, v_j) \in E$. The weight of the vertex cover being less than w_{test} is enforced by converting the arithmetic operations involved into Boolean operations just like in a digital circuit.

Figure 1 illustrates how to make use of a digital circuit to enforce that the weight of a vertex cover is less than a test weight. In other words, it enforces the condition $\sum_{i=1}^{|V|} w_i X_i - w_{test} < 0$. For simplicity of exposition, assume that all weights are non-negative integers. Each given weight w_i is first converted to its 2's complement representation. For example, w_1 is converted to '0101' in the figure. Replacing the '1's in this binary representation by X_i represents the term $w_i X_i$. To represent $-w_{test}$ on the left side of the condition, we simply use its 2's complement. For example, $-w_t = -3$ in the figure is represented as '1101'. A hierarchy of adder circuits adds all these terms—two numbers at a time as shown in Figure 1—and produces a final output that represents the quantity $\sum_{i=1}^{|V|} w_i X_i - w_{test}$. Since we require it to be negative, we simply enforce that the final sign-bit s is '1'.

[4] We can compute much more informed lower and upper bounds as explained later.

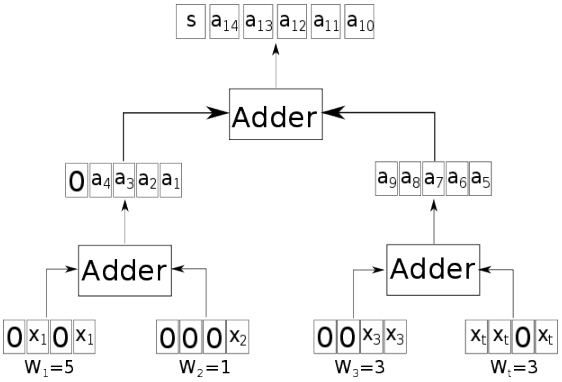

Fig. 1. Shows how to use a digital circuit to enforce the weight of a vertex cover to be less than a given test weight. Assume that there are 3 vertices, v_1, v_2, and v_3, with associated Boolean variables X_1, X_2, and X_3, respectively. The corresponding weights are $w_1 = 5$, $w_2 = 1$, and $w_3 = 3$. The test weight is $w_t = 3$. w_1 is converted to its binary representation '0101' and the '1's are replaced by X_1 to represent the term w_1X_1. w_2 and w_3 are converted in a similar way. For w_t, however, the binary representation '0011' is converted to its negative '1101' in 2's complement representation to represent $-w_t$ (X_t is set to '1'). The final output of the adder circuits represents the quantity $w_1X_1 + w_2X_2 + w_3X_3 - w_t$. The internal variables of the hierarchy of adder circuits are added to the SAT encoding. The constraints dictated by the gates of the digital circuit are added as clauses to the SAT encoding. The final sign-bit s is set to '1' in the SAT encoding to enforce that the result is negative as required.

Once we have a digital circuit, we can convert it into a CNF Boolean formula—i.e., a SAT instance—with Tseitin transformation. The internal variables of the hierarchy of adder circuits are added to the SAT encoding. The constraints dictated by the gates of the digital circuit are also added as clauses to the SAT encoding.[5] Each integer is represented using a non-redundant number of bits. When we add two integers with the longer of the two having k bits, the result is allocated $k + 1$ bits. All operations are done consistently with the 2's complement representation of integers. This reformulation is similar to [30] in the context of translating CSPs into SAT, to [11] in the context of translating pseudo-Boolean constraints into SAT, to [24] in the context of solving disjunctive temporal reasoning problems efficiently, and to [4, 27] in the context of solving planning problems.

Several issues need to be addressed in this reformulation of the MWVC problem. Some of them are: (a) the number of auxiliary variables in the SAT instances; (b) the number of clauses in the SAT instances; and (c) the precision of the numbers used to specify the weights. However, these issues have already been addressed in [4, 11, 24, 27] for SAT encodings of other combinatorial problems.

[5] We skip a detailed discussion of this transformation since it is similar to the works of various authors mentioned later.

The arithmetic operations that we encode using the digital circuit are very simple: addition ('+'), negation ('-'), and comparison ('<'). This makes the circuit representation compact with only logarithmic depth. If each weight has an L-bit representation, then there are about $|V|$ numbers with L bits each in the bottom level, $|V|/2$ numbers with $L+1$ bits each in the next level, and so on. This leads to $O(L|V|)$ variables in the SAT encoding. The number of internal gates is thus of the same order. This makes the SAT encoding small enough to be solvable by powerful SAT solvers.[6] When the weights are not integral, scaling techniques similar to those in [24] can be used.

Optimizations: Once we have the ability to answer the question of whether there is a vertex cover with a weight less than a given test weight w_{test}, we can employ binary search in the interval $[0, \sum_{i=1}^{|V|} w_i]$ to converge to the MWVC. However, this naive strategy is not very effective without the following optimizations that significantly reduce the number of iterations—i.e., the number of SAT instances to be solved.

The first optimization, *quasi binary search*, is based on the following observation. Suppose, in some iteration, the binary search is in the interval $[L, U]$, the test weight is $w_{test} = (L+U)/2$, and the SAT solver determines that there exists a vertex cover with a weight less than the given test weight w_{test}. Then, the SAT solver is also able to produce a candidate solution with weight $w' < w_{test}$. In the next iteration, therefore, the interval for the binary search can be reduced to $[L, w']$ instead of $[L, w_{test}]$. This can reduce the number of iterations significantly whenever we find a "good" solution, i.e., a small w'.

The second optimization is to make use of an approximation algorithm to produce tighter lower and upper bounds for use in the very first iteration instead of the conservative interval $[0, \sum_{i=1}^{|V|} w_i]$. Clarkson's primal-dual factor-2 approximation algorithm can be used to do so [29]. This algorithm is motivated by a linear programming perspective on the MWVC problem. Using a simple greedy strategy, it constructs integral primal and integral dual solutions simultaneously with the cost of the primal solution being at most twice the cost of the dual solution. The cost of the optimal solution should be in between; and, therefore, the greedily constructed primal solution serves as a factor-2 approximation. If the cost of such an approximate solution is S, then we can set $[S/2, S]$ as the binary search interval in the very first iteration. It is unlikely that we can do better since finding a $2-\epsilon$ approximation for the MVC or MWVC problem is UG-hard [16].[7]

The third optimization is to run an MVC solver by ignoring all the weights before the first iteration. The cost of the MVC solution produced can then be evaluated to serve as an upper bound for the first iteration of the binary search. However, this method is not guaranteed to be effective since it is completely oblivious to the weights. Nonetheless, it could often produce something useful.

[6] In fact, this approach is employed by CircuitTSAT, a state-of-the-art solver for disjunctive temporal reasoning problems [24].

[7] UG-hard means "Unique Games-hard", i.e., hard under the unique games conjecture.

4 Empirical Evaluation

Graph			SBMS				Gurobi		cliquer	
Instance	Vertices	MVC	Running Time	Iteration	Bounds	Initial Bounds	Running Time	Bounds	Running Time	Bounds
frb30-15-1	450	420	49.83	8	-	[218, 437]	22.80	-	15.29	-
frb30-15-2	450	420	40.84	8	-	[219, 438]	11.76	-	30.26	-
frb30-15-3	450	420	36.22	8	-	[218, 437]	34.05	-	120.33	-
frb30-15-4	450	420	40.38	8	-	[219, 439]	29.10	-	0.99	-
frb30-15-5	450	420	34.84	8	-	[219, 438]	10.38	-	0.15	-
frb35-17-1	595	560	65.73	8	-	[292, 584]	84.87	-	14.20	-
frb35-17-2	595	560	84.39	8	-	[292, 584]	>7200	[560, 561]	53.66	-
frb35-17-3	595	560	66.97	8	-	[291, 582]	>7200	[560, 561]	>7200	[-, 582]
frb35-17-4	595	560	55.37	8	-	[292, 584]	>7200	[560, 561]	5189.27	-
frb35-17-5	595	560	54.70	8	-	[290, 581]	>7200	[560, 561]	98.84	-
frb40-19-1	760	720	90.76	8	-	[371, 743]	>7200	[720, 722]	>7200	[-, 736]
frb40-19-2	760	720	131.52	9	-	[372, 745]	>7200	[720, 722]	>7200	[-, 733]
frb40-19-3	760	720	127.73	9	-	[372, 744]	>7200	[720, 721]	273.22	-
frb40-19-4	760	720	243.98	9	-	[372, 744]	>7200	[720, 722]	1555.14	-
frb40-19-5	760	720	198.27	9	-	[372, 745]	>7200	[720, 722]	42.77	-
frb45-21-1	945	900	2955.26	9	-	[465, 930]	>7200	[900, 904]	>7200	[-, 917]
frb45-21-2	945	900	235.59	9	-	[465, 930]	>7200	[900, 903]	>7200	[-, 917]
frb45-21-3	945	900	2036.46	9	-	[465, 930]	>7200	[900, 902]	>7200	[-, 913]
frb45-21-4	945	900	884.90	9	-	[465, 931]	>7200	[900, 902]	>7200	[-, 914]
frb45-21-5	945	900	1958.17	9	-	[465, 931]	>7200	[900, 903]	>7200	[-, 922]
frb50-23-1	1150	1100	3208.50	10	-	[556, 1133]	>7200	[1100, 1104]	>7200	[-, 1102]
frb50-23-2	1150	1100	>7200	9	[1100, 1101]	[567, 1135]	>7200	[1100, 1103]	>7200	[-, 1113]
frb50-23-3	1150	1100	111.09	10	-	[567, 1135]	>7200	[1100, 1105]	>7200	[-, 1112]
frb50-23-4	1150	1100	113.10	10	-	[567, 1135]	>7200	[1100, 1104]	1868.10	-
frb50-23-5	1150	1100	113.68	10	-	[568, 1137]	>7200	[1100, 1104]	>7200	[-, 1129]
frb53-24-1	1272	1219	>7200	8	[1219, 1221]	[625, 1250]	>7200	[1219, 1225]	>7200	[-, 1232]
frb53-24-2	1272	1219	114.87	10	-	[625, 1251]	>7200	[1219, 1224]	>7200	[-, 1239]
frb53-24-3	1272	1219	>7200	9	[1219, 1220]	[628, 1256]	>7200	[1219, 1224]	>7200	[-, 1237]
frb53-24-4	1272	1219	>7200	9	[1219, 1220]	[628, 1257]	>7200	[1219, 1224]	>7200	[-, 1228]
frb53-24-5	1272	1219	120.37	10	-	[627, 1255]	>7200	[1219, 1226]	>7200	[-, 1247]
frb56-25-1	1400	1344	>7200	9	[1344, 1345]	[692, 1384]	>7200	[1344, 1350]	>7200	[-, 1365]
frb56-25-2	1400	1344	>7200	9	[1344, 1345]	[691, 1383]	>7200	[1344, 1352]	>7200	[-, 1371]
frb56-25-3	1400	1344	6717.57	10	-	[692, 1384]	>7200	[1344, 1348]	>7200	[-, 1377]
frb56-25-4	1400	1344	>7200	9	[1344, 1345]	[692, 1385]	>7200	[1344, 1350]	>7200	[-, 1348]
frb56-25-5	1400	1344	120.31	10	-	[690, 1381]	>7200	[1344, 1350]	>7200	[-, 1379]
frb59-26-1	1534	1475	>7200	9	[1475, 1476]	[757, 1514]	>7200	[1475, 1482]	>7200	[-, 1493]
frb59-26-2	1534	1475	>7200	9	[1475, 1476]	[757, 1515]	>7200	[1475, 1481]	>7200	[-, 1513]
frb59-26-3	1534	1475	>7200	9	[1475, 1476]	[757, 1514]	>7200	[1475, 1482]	>7200	[-, 1509]
frb59-26-4	1534	1475	>7200	8	[1475, 1477]	[756, 1513]	>7200	[1475, 1481]	>7200	[-, 1516]
frb59-26-5	1534	1475	131.04	10	-	[759, 1519]	>7200	[1475, 1481]	>7200	[-, 1496]

Table 1. Shows the performances of SBMS, Gurobi and cliquer on unweighted BHOSLIB benchmark problem instances. The column "Iteration" indicates the number of iterations needed to produce the optimal solution or reach the running time limit of 2 hours. The column "Initial Bounds" indicates the bounds generated by Clarkson's algorithm. The column "Running Time" indicates the running time in seconds. When the running time exceeds the running time limit, the upper bound in column "Bounds" indicates the cost of the current candidate solution and the lower bound indicates that there cannot be a solution of lower cost. If either the lower bound or upper bound is not specified, it is marked with a '-'. When a problem instance is solved within the running time limit, the cost of the produced solution matches the entry in column "MVC" and the column "Bounds" is marked with a '-' in such a case.

We now compare the ILP, MAX-SAT, PBO, ASP and SAT-based approaches on a variety of MWVC problem instances. We also make important observations about the behaviors of these solvers. For the ILP-based solver, we use Gurobi [13], a state-of-the-art solver for mathematical programming, and lp_solve [1], a popular open source mixed integer linear programming solver. For the MAX-SAT-based solver, we use EvaSolver [23], a state-of-the-art MAX-SAT solver. For the PBO-based solver, we use WBO [22]. For the ASP-based solver, we use

Graph			Running Time of SBMS (mins)							
Instance	Vertices	MWVC	Q+C+N	C+N	Q+C	Q+N	Q	C	N	None
frb30-15-1	450	825	38.33	38.32	37.68	60.00	35.10	37.49	29.99	35.23
frb30-15-2	450	825	59.97	59.98	58.98	75.12	74.87	59.00	75.00	74.80
frb30-15-3	450	790	0.84	0.84	36.43	0.87	36.84	36.32	0.86	36.73
frb30-15-4	450	825	16.92	16.84	14.47	18.79	18.33	14.39	18.80	18.71
frb30-15-5	450	827	28.28	28.34	47.80	27.73	43.13	47.77	27.75	44.35

Table 2. Shows the performance of SBMS on a subset of the weighted BHOSLIB benchmark problem instances. "Q" refers to enabling quasi binary search. "C" refers to enabling Clarkson's algorithm. "N" refers to enabling the use of an initial upper bound derived from running NuMVC for 30 seconds. "None" refers to disabling all optimizations. All running times include the time to perform optimizations as well as the time to perform the actual search. The running times in the "Q+C+N" column and the "C+N" column are almost identical because the evolution of the bounds for these benchmark instances is not affected much by quasi binary search.

clingo from Potassco—the Potsdam Answer Set Solving Collection [12]. Because the MWVC problem is equivalent to the MWIS problem, we can also use a clique-based solver that searches for the maximum weighted clique in the edge-complement graph. We therefore additionally use one such state-of-the-art solver in our experiments. In particular, we use cliquer [25] for this purpose. For the SAT-based solver, we use SBMS, which makes use of Lingeling [2], a state-of-the-art complete SAT solver. For SBMS, we also use Clarkson's primal-dual factor-2 approximation algorithm for the MWVC problem [29] to generate the initial lower and upper bounds for the quasi binary search. For the BHOSLIB and DIMACS benchmark problems described below, SBMS also runs NuMVC [5] for 30 seconds in order to yield a possibly tighter upper bound. Except for Gurobi and EvaSolver for which we used prebuilt binaries, all solvers were implemented in C++, were compiled by gcc 4.9.2 with the -O3 option, and were run on a GNU/Linux workstation with Intel Xeon Processor E3-1240 v3 (8MB Cache, 3.4GHz) and 16GB RAM.

Since the MWVC problem has not received much attention, there do not exist any benchmark instances for it. However, benchmark instances for the MVC problem do exist, such as the BHOSLIB and DIMACS suites used in [5]. We created MWVC versions of these instances by arbitrarily assigning a weight of $i \bmod 3 + 1$ to a vertex with index i to achieve repeatability of the experiments. As argued before, the number of variables in the SAT encoding increases linearly with the size of the bit representations of the weights and thus only logarithmically with their values (scaled to be non-negative integers).

Clearly, any good MWVC solver should also perform well on regular MVC problem instances. Our first experiment, therefore, used the unweighted version of the BHOSLIB instances. In essence, we solved hard benchmark instances of the MVC problem using a complete solver. Table 1 shows our performance results. We solved more than 50% of these benchmark instances quite comfortably. Even in the cases that were not solved within the running time limit, SBMS returned solutions with the guarantee that they were no more than a cost of 1

Graph			Running Times (secs)						
Instance	Vertices	Edges	Gurobi	lp_solve	EvaSolver	WBO	clingo	cliquer	SBMS
brock200_2	200	10024	32.41	168.44	83.86	>3600	23.28	<0.01	212.34
brock200_4	200	6811	52.64	1078.76	491.10	>3600	558.61	0.08	1438.92
brock400_2	400	20014	>3600	>3600	>3600	>3600	>3600	226.74	>3600
brock400_4	400	20035	>3600	>3600	>3600	>3600	>3600	208.74	>3600
brock800_2	800	111434	>3600	>3600	>3600	>3600	>3600	3220.27	>3600
brock800_4	800	111957	>3600	>3600	>3600	>3600	>3600	2826.08	>3600
C1000.9	1000	49421	>3600	>3600	>3600	>3600	>3600	>3600	>3600
C125.9	125	787	0.72	28.64	1649.65	>3600	>3600	3.37	>3600
C2000.5	2000	999164	>3600	>3600	>3600	>3600	>3600	>3600	>3600
C2000.9	2000	199468	>3600	>3600	>3600	>3600	>3600	>3600	>3600
C250.9	250	3141	2058.10	>3600	>3600	>3600	>3600	>3600	>3600
C4000.5	4000	3997732	>3600	>3600	>3600	>3600	>3600	>3600	>3600
C500.9	500	12418	>3600	>3600	>3600	>3600	>3600	>3600	>3600
DSJC1000.5	1000	249674	>3600	>3600	>3600	>3600	>3600	43.42	>3600
DSJC500.5	500	62126	>3600	>3600	>3600	>3600	>3600	0.46	>3600
gen200_p0.9_44	200	1990	3.38	>3600	>3600	>3600	>3600	1722.84	>3600
gen200_p0.9_55	200	1990	0.10	2921.30	872.60	>3600	>3600	43.05	>3600
gen400_p0.9_55	400	7980	>3600	>3600	>3600	>3600	>3600	>3600	>3600
gen400_p0.9_65	400	7980	>3600	>3600	>3600	>3600	>3600	>3600	>3600
gen400_p0.9_75	400	7980	381.13	>3600	>3600	>3600	>3600	>3600	>3600
hamming10-4	1024	89600	>3600	>3600	>3600	>3600	>3600	>3600	>3600
hamming8-4	256	11776	1.34	>3600	800.69	>3600	>3600	0.97	71.42
keller4	171	5100	0.63	475.42	55.65	>3600	132.62	0.03	69.69
keller5	776	74710	1510.35	>3600	>3600	>3600	>3600	>3600	>3600
keller6	3361	1026582	>3600	>3600	>3600	>3600	>3600	>3600	>3600
MANN_a27	378	702	<0.01	0.11	0.12	0.16	>3600	>3600	43.66
MANN_a45	1035	1980	0.01	1.30	0.82	1.26	>3600	>3600	242.38
MANN_a81	3321	6480	0.03	19.54	10.12	18.85	>3600	>3600	1783.03
p_hat1500-1	1500	839327	>3600	>3600	>3600	>3600	>3600	0.98	>3600
p_hat1500-2	1500	555290	>3600	>3600	>3600	>3600	>3600	>3600	>3600
p_hat1500-3	1500	277006	>3600	>3600	>3600	>3600	>3600	>3600	>3600
p_hat300-1	300	33917	56.33	309.62	17.88	>3600	3.01	<0.01	131.43
p_hat300-2	300	22922	94.83	>3600	>3600	>3600	1512.84	0.13	>3600
p_hat300-3	300	11460	1764.59	>3600	>3600	>3600	>3600	47.11	>3600
p_hat700-1	700	183651	>3600	>3600	2887.61	>3600	1548.55	0.04	3532.76
p_hat700-2	700	122922	>3600	>3600	>3600	>3600	>3600	1247.51	>3600
p_hat700-3	700	61640	>3600	>3600	>3600	>3600	>3600	>3600	>3600

Table 3. Shows the performances of Gurobi, lp_solve, EvaSolver, WBO, clingo, cliquer and SBMS on weighted DIMACS benchmark problem instances. When the running time exceeds the running time limit of 1 hour, the corresponding entry is marked with ">3600".

away from the optimal ones.[8] The running times of lp_solve, EvaSolver, WBO and clingo are not listed in Table 1 since none of them could solve any of the benchmark instances within 2 hours. It is also easy to see that SBMS significantly outperforms even Gurobi and cliquer both in terms of the number of problem instances solved as well as the quality of the bounds produced.

As expected, the running times of NuMVC are smaller on these benchmark instances compared to the running times of SBMS [5]. However, NuMVC also fails to find an optimal solution on a few of these benchmark instances. In addition, NuMVC solves only MVC problem instances and, furthermore, is an incomplete solver that cannot prove the optimality of the produced solution nor provide optimality bounds. Some other state-of-the-art complete solvers, like MaxCLQdyn+EFL+SCR [21], were not included in the evaluation of NuMVC in [5] since their performance was poor.[9] SBMS, therefore, is a state-of-the-art complete solver for MVC instances.

[8] frb53-24-1 and frb59-26-4 are the only two exceptions with a gap of 2.

[9] See the second paragraph on page 18 of [5] that states "... MaxCLQdyn+EFL+SCR is not evaluated on BHOSLIB benchmark which is much harder and requires more effective technologies for exact algorithms ...".

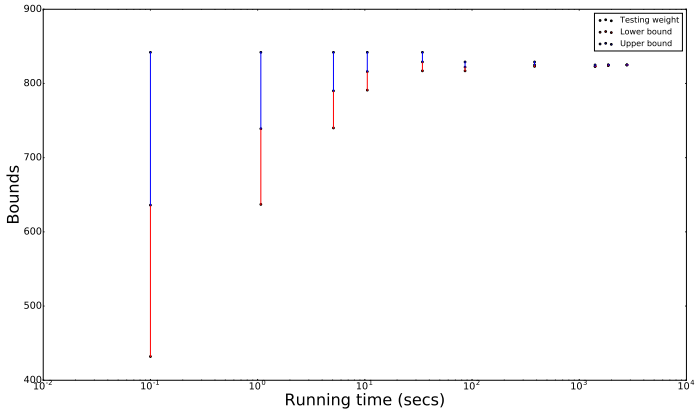

Fig. 2. Shows the evolution of the lower and upper bounds with the running time of SBMS on the weighted BHOSLIB instance frb30-15-1. The mid-point of the bounds is used as the testing weight for the SAT instance posed at that time.

Our second experiment used the weighted BHOSLIB instances. None of lp_solve, EvaSolver, WBO, clingo or cliquer could solve any of these instances in less than 2 hours. Table 2 shows the performance of SBMS—and its variants with various optimization features enabled or disabled—on the first five weighted benchmark instances that it could solve.[10] For the instances that it could not solve, SBMS still produced useful bounds. For generality, our third experiment used a different set of benchmark instances—the weighted DIMACS instances. Table 3 shows the performances of Gurobi, lp_solve, EvaSolver, WBO, clingo, cliquer and SBMS on these instances. Once again, SBMS produces useful bounds when it cannot solve a problem instance.

To understand the anytime property of SBMS, we also ran experiments to observe patterns in its behavior. Figures 2 and 3 show the typical behavior of SBMS on a fixed benchmark instance. Figure 2 illustrates that the intervals between the optimality bounds typically decrease very quickly, and the solver thus finds a good solution fast. SBMS spends most of the time in trying to improve a good solution to the optimal solution. This "diminishing returns" property which is so pronounced that it is apparent despite a log scale used in the figure is very desirable for an anytime algorithm. Figure 3 reinforces this observation by showing that the SAT instances in the early iterations are much easier to solve than in later iterations (which have smaller intervals between optimality bounds). Thus, by the time the SAT instances get hard to solve, the solver has already found a good solution and is only trying to improve it further.

[10] Gurobi was competitive with SBMS on these five instances.

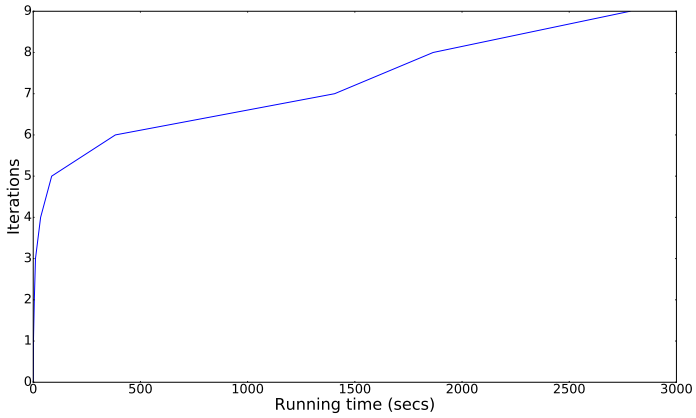

Fig. 3. Shows the iteration number as a function of the running time of SBMS on the weighted BHOSLIB instance frb30-15-1.

5 Conclusions and Future Work

In this paper, we presented a SAT-based solver for the MWVC problem. We first argued that, because the MWVC problem is not fixed-parameter tractable, none of the state-of-the-art methods for the MVC problem can be easily modified to tackle the MWVC problem. We compared several solvers based on ILP, MAX-SAT, PBO, ASP and SAT reformulations. Our reformulation of the MWVC problem as a series of SAT instances yields an anytime algorithm that exhibits the "diminishing returns" property and quickly converges to a good solution. In most cases, SBMS significantly outperforms the other methods. SBMS uses quasi binary search in the inner loop and a primal-dual approximation for the MWVC to provide a good starting point.

While SBMS appears to provide an alternative to the few competitive solvers that currently exist for the MWVC problem, we presented it here mostly as a strawman solver for the purpose of gaining interest among AI researchers to study this combinatorial problem more closely. Recent results—like in [19,20]—have demonstrated the importance of the MWVC problem for a wide range of other combinatorial problems in AI applications. Future work will not only be directed toward developing a better solver for the MWVC problem but also toward exploring the full implications of having good solvers available.

6 Acknowledgments

The research at USC was supported by NSF under grant numbers 1409987 and 1319966 and a MURI under grant number N00014-09-1-1031. The views and conclusions contained in this document are those of the authors and should not

be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

1. Berkelaar, M., Eikland, K., Notebaert, P.: lp_solve 5.5 open source (mixed integer) linear programming software. In: http://lpsolve.sourceforge.net/5.5/ (2004)
2. Biere, A.: Lingeling, Plingeling and Treengeling Entering the SAT competition 2013. In: Proceedings of the SAT Competition 2013. Department of Computer Science Series of Publications B, vol. B-2013-1, pp. 51–52 (2013)
3. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of Artifical Intelligence Research 21, 135–191 (2004)
4. Büttner, M., Rintanen, J.: Satisfiability planning with constraints on the number of actions. In: the Proceedings of the International Conference on Automated Planning and Scheduling. pp. 292–299 (2005)
5. Cai, S., Su, K., Luo, C., Sattar, A.: NuMVC: An efficient local search algorithm for minimum vertex cover. Journal of Artificial Intelligence Research 46(1), 687–716 (2013)
6. Cai, S., Su, K., Sattar, A.: Local search with edge weighting and configuration checking heuristics for minimum vertex cover. Artificial Intelligence 175(9-10), 1672–1696 (2011)
7. Chen, J., Kanj, I.A., Xia, G.: Improved parameterized upper bounds for vertex cover. In: International Symposium on Mathematical Foundations of Computer Science, pp. 238–249. Springer (2006)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (3rd Edition). MIT Press (2009)
9. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Annals of Mathematics 162(1), 439–485 (2005)
10. Do, M.B., Benton, J., Briel, M.V.D., Kambhampati, S.: Planning with goal utility dependencies. In: Proceedings of the International Joint Conference on Artificial Intelligence. pp. 1872–1878 (2007)
11. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2, 1–26 (2006)
12. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 107–124 (2011)
13. Gurobi Optimization, I.: Gurobi optimizer reference manual (2015), http://www.gurobi.com
14. Johnson, D.J., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society (1996)
15. Karakostas, G.: A better approximation ratio for the vertex cover problem. ACM Transactions on Algorithms 5(4), 1–8 (2009)
16. Khot, S., Regev, O.: Vertex cover might be hard to approximate to within 2-ϵ. Journal of Computer and System Sciences 74(3), 335–349 (2008)
17. Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? IEEE Transactions on Pattern Analysis and Machine Intelligence 26(2), 147–159 (2004)

18. Kolmogorov, V.: Primal-dual algorithm for convex Markov random fields. Tech. Rep. MSR-TR-2005-117, Microsoft Research (2005)
19. Kumar, T.K.S.: A framework for hybrid tractability results in boolean weighted constraint satisfaction problems. In: the Proceedings of the International Conference on Principles and Practice of Constraint Programming. pp. 282–297 (2008)
20. Kumar, T.K.S.: Lifting techniques for weighted constraint satisfaction problems. In: Proceedings of the International Symposium on Artificial Intelligence and Mathematics (2008)
21. Li, C.M., Quan, Z.: Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence. pp. 344–351 (2010)
22. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Proceedings of the International Conference on Theory and Applications of Satisfiability Testing. pp. 495–508 (2009)
23. Narodytska, N., Bacchus, F.: Maximum satisfiability using core-guided MaxSAT resolution. In: Proceedings of the AAAI Conference on Artificial Intelligence. pp. 2717–2723 (2014)
24. Nelson, B., Kumar, T.K.S.: CircuitTSAT: A solver for large instances of the disjunctive temporal problem. In: Proceedings of the International Conference on Automated Planning and Scheduling. pp. 232–239 (2008)
25. Niskanen, S., Östergård, P.R.J.: Cliquer user's guide, version 1.0. Tech. Rep. T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland (2003)
26. Richter, S., Helmert, M., Gretton, C.: A stochastic local search approach to vertex cover. In: Proceedings of the Annual German Conference on Artificial Intelligence (Künstliche Intelligenz). pp. 412–426 (2007)
27. Rosa, E.D., Giunchiglia, E., Maratea, M.: Solving satisfiability problems with preferences. Constraints 15(4), 485–515 (2010)
28. Tomita, E., Kameda, T.: An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. Journal of Global Optimization 37(1), 95–111 (2007)
29. Vazirani, V.V.: Approximation algorithms. Springer (2003)
30. Walsh, T.: SAT v CSP. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming. pp. 441–456 (2000)