A Warning Propagation-Based Linear-Time-and-Space Algorithm for the Minimum Vertex Cover Problem on Giant Graphs

Hong Xu(✉), Kexuan Sun, Sven Koenig, and T. K. Satish Kumar

University of Southern California, Los Angeles, CA 90089, United States of America
{hongx, kexuansu, skoenig}@usc.edu, tkskwork@gmail.com

Abstract. A vertex cover (VC) of a graph G is a subset of vertices in G such that at least one endpoint vertex of each edge in G is in this subset. The minimum VC (MVC) problem is to identify a VC of minimum size (cardinality) and is known to be NP-hard. Although many local search algorithms have been developed to solve the MVC problem close-to-optimally, their applicability on giant graphs (with no less than 100,000 vertices) is limited. For such graphs, there are two reasons why it would be beneficial to have linear-time-and-space algorithms that produce small VCs. Such algorithms can: (a) serve as preprocessing steps to produce good starting states for local search algorithms and (b) also be useful for many applications that require finding small VCs quickly. In this paper, we develop a new linear-time-and-space algorithm, called MVC-WP, for solving the MVC problem on giant graphs based on the idea of warning propagation, which has so far only been used as a theoretical tool for studying properties of MVCs on infinite random graphs. We empirically show that it outperforms other known linear-time-and-space algorithms in terms of sizes of produced VCs.

1 Introduction

Thanks to the advancement of technologies such as the Internet and database management systems, datasets have been growing tremendously over the past decade. Many of the resulting datasets can be modeled as graphs, such as social networks, brain networks, and street networks. Therefore, it is essential to develop algorithms to solve classical combinatorial problems on giant graphs (with no less than 100,000 vertices).

A *vertex cover* (VC) on an undirected graph $G = \langle V, E \rangle$ is defined as a set of vertices $S \subseteq V$ such that every edge in E has at least one of its endpoint vertices in S. A *minimum VC* (MVC) is a VC on G of minimum size (cardinality), i.e., there exists no VC whose size is smaller than that of an MVC. The MVC problem is to find an MVC on a given graph G. Its decision version is known to be NP-complete [18]. An *independent set* (IS) on G is a set of vertices $T \subseteq V$ such that no two vertices in T are adjacent to each other. The complement of a (maximum)

IS is a (minimum) VC and vice versa, i.e., for any (maximum) IS T, $V \setminus T$ is always a (minimum) VC.

The MVC problem has been widely used to study various real-world and theoretical problems. For example, in practice, it has been used in computer network security [11], in crew scheduling [24], and in the construction of phylogenetic trees [1]. In theoretical research, it has been used to prove the NP-completeness of various other well-known problems, such as the set cover problem and the dominating set problem [19]. It is also a fundamental problem studied in the theory of fixed-parameter tractability [13].

Various researchers have developed exact solvers [10, 21, 27, 29] for the MVC problem and its equivalents. However, none of these solvers work well for large problem instances of the MVC problem due to its NP-hardness. Furthermore, solving the MVC problem within any approximation factor smaller than 1.3606 is also NP-hard [8].

To overcome the poor efficiency of exact algorithms and the high approximation factor of polynomial-time approximation algorithms, researchers have focused on developing non-exact local search algorithms [2, 5, 6, 22] for solving the MVC problem and its equivalents. These algorithms often require a preprocessing step to construct a VC (usually the smaller the better) before starting the local search. While polynomial-time procedures work well for the preprocessing step on regular-sized graphs, they are prohibitively expensive on giant graphs. On giant graphs, this preprocessing step needs to terminate fast and should use only a moderate amount of memory. Therefore, it is important to develop a linear-time-and-space algorithm to find a small VC.

In addition, many real-world applications on giant graphs require the identification of small VCs but not necessarily MVCs. One example of such applications is the influence-maximization problem in social networks [14]. Here, too, linear-time-and-space algorithms for finding small VCs are important.

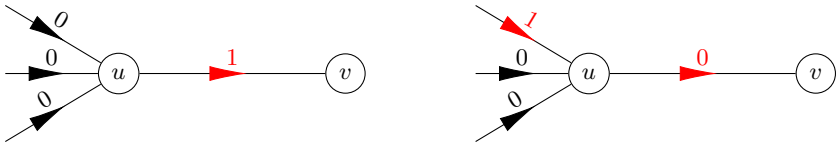
In this paper, we develop a new linear-time-and-space algorithm, called MVC-WP, for solving the MVC problem on giant graphs based on the idea of warning propagation, which has so far only been used as a theoretical tool for studying properties of MVCs on infinite random graphs. We then empirically show that MVC-WP has several advantages over other linear-time-and-space algorithms. We also experiment with variants of MVC-WP to empirically demonstrate the usefulness of various steps in it.

2 Background

In this section, we introduce relevant background on random graph models, warning propagation, and existing linear-time-and-space MVC algorithms known to the authors.

2.1 Random Graph Models

The Erdős-Rényi Model An *Erdős-Rényi model* (ER model) [9] is characterized by two parameters n and p. It generates random graphs with n vertices and



(a) u sends a message of 1 to v since all other incoming messages from its other neighbors are 0.

(b) u sends a message of 0 to v since at least one of its incoming messages from its other neighbors is 1.

Fig. 1: Illustrates the update of a message from $u \in V$ to $v \in V$ in the warning propagation algorithm for the MVC problem on graph $G = \langle V, E \rangle$. Only relevant parts of G are shown, i.e., u, v, and all edges incident to u.

connects every pair of vertices with probability p. We call a graph generated by an ER model an *ER graph*. The degrees of the vertices of an ER graph follow a Poisson distribution. The average degree of vertices is $c = np$.

The Scale-Free Model A *scale-free model* (SF model) [4] is characterized by two parameters n and $\lambda > 2$. It generates random graphs whose vertex degree distribution follows a power law, i.e., $P(d) \sim d^{-\lambda}$. The average degree of vertices is therefore

$$c = \sum_{d=1}^{+\infty} P(d)d = \frac{\zeta(\lambda - 1)}{\zeta(\lambda)}, \tag{1}$$

where $\zeta(x) = \sum_{k=1}^{\infty} \frac{1}{k^x}$ is the Riemann zeta function. For notational convenience, we define $Z(\lambda) = \frac{\zeta(\lambda-1)}{\zeta(\lambda)}$. We call a graph generated by an SF model an *SF graph*.

2.2 Warning Propagation

The *warning propagation algorithm* is a specialized *message passing algorithm* where information is processed locally and exchanged between relevant variables [20]. In the warning propagation algorithm, messages can only take one of two values, namely 0 or 1. To analyze properties of MVCs on infinite random graphs, [26] proposed an algorithm that uses warning propagation for solving the MVC problem to help with their theoretical analysis. In their algorithm, messages are passed between adjacent vertices. A message of 1 from $u \in V$ to $v \in V$ indicates that u is not in the MVC and thus it "warns" v to be included in the MVC. Otherwise, if u is in the MVC, this message would be 0. Based on this intuition, the warning propagation algorithm updates messages according to the following rules: A message from u to v is updated to 1 iff all incoming messages to u from its other neighbors equal 0, i.e., no other adjacent vertices of u require u to be in the VC. Otherwise, this message would be 0. Figure 1 illustrates these rules. Upon convergence, vertices with at least one incoming messages equal to 1 are included in the VC, and other vertices are excluded from the VC. The theoretical analysis in [26] mainly focuses on ER graphs. It shows that, on an

infinitely large ER graph, a message is 1 with probability $W(c)/c$, where $W(\cdot)$ is the Lambert-W function, i.e., the inverse function of $f(x) = xe^x$.

2.3 Known Linear-Time-and-Space MVC Algorithms

MVC-2 This well-known linear-time-and-space factor-2 approximation algorithm for the MVC problem works as follows [25]: In each iteration, MVC-2 first arbitrarily selects an uncovered edge, then marks it as well as the edges incident to its two endpoint vertices as being covered, and finally adds its endpoint vertices to the VC. It terminates when all edges are marked as covered.

ConstructVC Serving as a preprocessing step, this algorithm is a greedy linear-time-and-space subroutine in the FastVC solver [5], that constructs a minimal VC[1]. It works as follows: In each iteration, ConstructVC first arbitrarily selects an uncovered edge, then adds its endpoint vertex v with the larger degree to the VC, and finally marks all edges incident to v as being covered. When all edges are marked as covered, it removes all redundant vertices in the VC to construct a minimal VC.

R This algorithm is used as the preprocessing step to produce a maximal IS (complement of a minimal VC) in the local search algorithm for solving the maximum IS problem developed by [2]. R can be easily adjusted to produce a minimal VC, and the adapted algorithm works as follows: R first adds all vertices into the VC. In each iteration, R randomly removes a vertex v from the VC if it continues to be a VC after the removal. It terminates when the VC is minimal.

MVC-MPL and MVC-L MVC-MPL is a linear-time-and-space MVC algorithm based on some theoretical results of warning propagation on ER graphs. It works as follows [28]: In each iteration, MVC-MPL first arbitrarily selects a vertex v, then adds it to the IS with probability $(1 - W(c)/c)^{\kappa(v)}$, where $\kappa(v)$ is the degree of v, and otherwise to the VC. It terminates when every vertex has been added to either the VC or the IS. MVC-L is a variant of MVC-MPL with the probability of adding a vertex v to the IS replaced by $1/(\kappa(v) + 1)$ [28].

3 Warning Propagation on Scale-Free Graphs

Assuming that the warning propagation algorithm is applied on an SF graph, we derive the approximate message distribution upon convergence by following a method similar to that in [26, section IV.B]. We use p_0 and p_1 to denote the fractions of all messages with values 0 and 1 upon convergence, respectively. Clearly, we have

$$p_0 + p_1 = 1. \tag{2}$$

[1] A minimal VC is a VC such that no proper subset thereof is also a VC.

A message $m_{u \to v}$ from vertex u to vertex v is equal to 1 iff all incoming messages to u from its other neighbors are equal to 0, i.e., $\forall w \in \partial u \setminus v : m_{w \to u} = 0$, where ∂u is the set of vertices adjacent to u. Assuming that all messages incoming to u are independent and using the fact that the probability distribution of the number of such messages follows a power law on an SF graph, we have

$$1 - p_0 = p_1 = \sum_{d=1}^{\infty} \frac{d^{-\lambda}}{\zeta(\lambda)} p_0^{d-1} = \frac{\mathrm{Li}_\lambda(p_0)}{p_0 \zeta(\lambda)}, \tag{3}$$

where $\mathrm{Li}_\lambda(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^\lambda}$ is the polylogarithm function. After making the approximation $\mathrm{Li}_\lambda(p_0) \approx p_0 + \frac{p_0^2}{2^\lambda}$, we solve Eq. (3) for p_0 and have

$$p_0 = \frac{\zeta(\lambda) - 1}{\zeta(\lambda) + \frac{1}{2^\lambda}}, \tag{4}$$

where $\forall \lambda > 2 : 0 \leq p_0 \leq 1$. Therefore, for any $\lambda > 2$, Eq. (4) is always a valid solution for p_0.

4 The Algorithm

Our algorithm MVC-WP (Algorithm 1) is based on the analytical results that govern the warning propagation algorithm for the MVC problem [26]. MVC-WP first uses Algorithm 2, an algorithm that prunes leaves, to identify those vertices that are necessarily in some MVC and modifies the input graph accordingly. It then treats this modified graph as if it were an ER or SF graph and computes p_0 using Algorithm 3. (Although MVC-WP treats the graph as if it were an ER or SF graph, it does not impose any restrictions on the graph.) MVC-WP then assigns each message from vertex u to vertex v to be 1 with probability $p_0^{\kappa(u)-1}$, where $\kappa(u)$ denotes the degree of u. This is done under the assumption that all incoming messages of u have independent probabilities to be 0 or 1. Then, MVC-WP performs warning propagation for M iterations, where M is a given parameter. After M iterations, v is marked as being included in VC if it receives at least one message of 1; otherwise, v is marked as being excluded in VC. If v is excluded, MVC-WP marks all its adjacent vertices as being included in VC. Finally, MVC-WP uses Algorithm 4 to remove redundant vertices from VC to make it a minimal VC. This step is adapted from Lines 6 to 14 of Algorithm 2 in [5].

We note that, a warning propagation iteration in the warning propagation algorithm proposed in [26] is not linear-time due to the requirement of traversing incoming messages of vertex u when updating the message from vertex u to vertex v. To avoid this traversal and thus make each warning propagation iteration linear-time, for each vertex v, MVC-WP keeps track the number of messages incoming to v that are equal to 1 in an array *counter*. This and the value of the message from u to v provide enough information for updating the message.

We also note that, while MVC-WP is based on the analytical results from [26], it differs significantly from the warning propagation algorithm proposed in [26].

Algorithm 1: MVC-WP.

```
1  Function MVC-WP(G = ⟨V, E⟩, model, M)
       Input: G: The graph to find an MVC for.
       Input: model: The random graph model to use (ER or SF).
       Input: M: Number of iterations of the warning propagation algorithm.
       Output: A minimal VC of G.
2      VC, IS := Prune-Leaves(G);
3      p_0 := Compute-p_0(G, model);
4      Convert G to a directed graph G' = ⟨V, E'⟩ by introducing ⟨u, v⟩ and ⟨v, u⟩ in E' for
         each (u, v) ∈ E;
5      Build an associative array m for the edges of G' to represent messages;
6      Build an associative array counter for the vertices of G' to record the number of
         incoming messages that are equal to 1;
7      Initialize counter to zeros;
8      • Initialize messages:
9          for each ⟨u, v⟩ ∈ E' do
10             Draw a random number r ∈ [0, 1];
11             if r ≤ p_0^{κ(u)−1} then
12                 m_{u→v} := 1;
13                 counter(v) := counter(v) + 1;
14             else
15                 m_{u→v} := 0;
16     • Run M iterations of the warning propagation algorithm:
17         for t := 1, ..., M do
18             for each ⟨u, v⟩ ∈ E' do
19                 if counter(u) − m_{v→u} = 0 then
20                     if m_{u→v} = 0 then
21                         m_{u→v} := 1;
22                         counter(v) := counter(v) + 1;
23                 else
24                     if m_{u→v} = 1 then
25                         m_{u→v} := 0;
26                         counter(v) := counter(v) − 1;
27     • Construct a VC:
28         while ∃v ∈ V \ (VC ∪ IS) do
29             v := any vertex in V \ (VC ∪ IS);
30             if counter(v) = 0 then
31                 Add v to IS and all u in ∂v to VC;
32             else
33                 Add v to VC;
34     return Remove-Redundancy(G, VC);
```

MVC-WP introduces preprocessing and postprocessing steps before and after warning propagation iterations. It also initializes messages intelligently. In addition, MVC-WP reduces the time complexity of a warning propagation iteration to linear by using a *counter* array. Most importantly, MVC-WP is specifically designed for being practically run, while the warning propagation algorithm proposed in [26] lacks many algorithmic details, since [26] only uses it as a theoretical tool to study properties of MVCs on ER graphs.

We now formally prove the correctness and time and space complexities of MVC-WP.

Theorem 1. *MVC-WP produces a minimal VC.*

Algorithm 2: Prune leaves.

```
1  Function Prune-Leaves(G = ⟨V, E⟩)
       Modified: G: The input graph.
2      Initialize vertex sets VC and IS to the empty set;
3      for each v ∈ V do
4          Prune-A-Leaf(G, VC, IS, v);
5      return VC, IS;
6  Function Prune-A-Leaf(G = ⟨V, E⟩, VC, IS, v)
       Modified: G: The input graph.
       Modified: VC: The current VC.
       Modified: IS: The current IS.
       Input: v: A vertex in V.
7      if κ(v) = 1 then
8          u := the only vertex adjacent to v;
9          VC := VC ∪ {u};
10         IS := IS ∪ {v};
11         U := ∂u \ {v};
12         Remove v and (u, v) from G;
13         for each w ∈ U do
14             Remove (u, w) from G;
15             Prune-A-Leaf (G, VC, IS, w);
16         Remove u from G;
```

Algorithm 3: Compute p_0 for different random graph models.

```
1  Function Compute-p0(G, model)
       Input: G: The input graph.
       Input: model: The random graph model to use (ER or SF).
2      c := average degree of vertices in G;
3      if model is ER then
4          p0 := 1 − W(c)/c;
5      else if model is SF then
6          λ := Z^{-1}(c);
7          Compute p0 according to Eq. (4);
8      return p0;
```

Proof. Since [5] has proved that `Remove-Redundancy` produces a minimal VC provided that variable VC in Algorithm 1 is a VC, it is sufficient to prove that, right before Line 34 in Algorithm 1, variable IS is an IS, $VC \cup IS = V$, and $VC \cap IS = \emptyset$.

In Algorithm 2, Lines 12 and 14 are the only steps that remove edges. However, these edges are covered by VC as shown on Line 9. In addition, $VC \cup IS$ is the set of all removed vertices and $VC \cap IS = \emptyset$, since each removed vertex is added to either VC or IS. Therefore, IS is an independent set.

In Algorithm 1, message initialization and the M iterations of warning propagation do not change the values of IS and VC.

In Lines 27 to 33 in Algorithm 1, since Line 31 guarantees that no two adjacent vertices are added to IS, IS must be an IS of G. In addition, Line 28 guarantees $IS \cup VC = V$ and Lines 28, 31 and 33 guarantee $IS \cap VC = \emptyset$.

Therefore, this theorem is true.

Theorem 2. *The time complexity of MVC-WP is* $\mathcal{O}(|V| + |E|)$.

Algorithm 4: Remove redundant vertices from a given VC [5].

```
1  Function Remove-Redundancy(G = ⟨V, E⟩, VC)
       Input: G: The input graph.
       Input: VC: A VC of G.
2      Build an associative array loss for vertices in VC to record whether they can be
         removed from VC;
3      Initialize loss to zeros;
4      foreach e ∈ E do
5          if only one endpoint vertex v of e is in VC then
6              loss(v) := 1;
7      foreach v ∈ VC do
8          if loss(v) = 0 then
9              VC := VC \ {v};
10             foreach v' ∈ ∂v ∩ VC do
11                 loss(v') := 1;
12     return VC;
```

Proof. We first prove that `Prune-Leaves` terminates in $\mathcal{O}(|V| + |E|)$ time by counting the number of times that `Prune-A-Leaf` is called, since the only loop in `Prune-A-Leaf` makes only one recursive call in each iteration. Line 4 in Algorithm 2 calls `Prune-A-Leaf` at most $|V|$ times. Line 15 calls `Prune-A-Leaf` iff edge (u, w) is removed from G. Therefore, Line 15 calls `Prune-A-Leaf` at most $|E|$ times.

Obviously, `Compute-`p_0 terminates in constant time.

In Algorithm 1, Lines 8 to 15 iterate over each edge in G' exactly once, and therefore terminate in $\mathcal{O}(|E|)$ time; Lines 16 to 26 iterate over each edge in G' exactly M times, and therefore terminate in $\mathcal{O}(|E|)$ time; Lines 27 to 33 consider each vertex v in G' at least once and at most $\kappa(v)$ times, and therefore terminate in $\mathcal{O}(|V| + |E|)$ time.

[5] has proved that `Remove-Redundancy` terminates in $\mathcal{O}(|V| + |E|)$ time.

Combining the results above, MVC-WP terminates in $\mathcal{O}(|V| + |E|)$ time.

Theorem 3. *The space complexity of MVC-WP is* $\mathcal{O}(|V| + |E|)$.

Proof. [5] has proved that `Remove-Redundancy` uses $\mathcal{O}(|V| + |E|)$ space. The recursive calls of `Prune-A-Leaf` initiated in `Prune-Leaves` use $\mathcal{O}(|E|)$ stack space. The remaining steps in MVC-WP require $\mathcal{O}(|E|)$ space to store messages and $\mathcal{O}(|V|)$ space to store *counter* as well as the status of each vertex v, i.e., whether v is in VC, IS or undetermined yet. Therefore, MVC-MP uses $\mathcal{O}(|V| + |E|)$ space.

4.1 Computing Special Functions

In Algorithm 3, we are required to compute a few special functions, namely the Lambert-W function $W(\cdot)$, the Riemann zeta function $\zeta(\cdot)$ and the inverse function of $Z(\cdot)$. For some of these functions, researchers in the mathematics

Table 1: Shows the values of $\zeta(k)$ and $Z(k) = \frac{\zeta(k-1)}{\zeta(k)}$ for $k \in \{1, 2, \ldots, 9\}$. The values of $\zeta(k)$ are taken from [15, Table 23.3], and the values of $Z(k)$ are computed from the values of $\zeta(k)$.

k	1	2	3	4	5	6	7	8	9
$\zeta(k)$	$+\infty$	1.645	1.202	1.082	1.037	1.017	1.008	1.004	1.002
$Z(k)$	-	$+\infty$	1.369	1.111	1.043	1.020	1.009	1.004	1.002

community have already developed various numerical methods [7, 16]. However, they are too slow for MVC-WP, which does not critically need this high accuracy. We now present a few new approaches to quickly compute them sufficiently accurately.

4.2 The Lambert-W Function $W(\cdot)$

We approximate $W(\cdot)$ via the first 3 terms of Equation (4.19) in [7], i.e.,

$$W(c) = L_1 - L_2 + L_2/L_1 + \mathcal{O}\left((L_2/L_1)^2\right), \tag{5}$$

where $L_1 = \log c$ and $L_2 = \log L_1$.

4.3 The Riemann Zeta Function $\zeta(\cdot)$

For the SF model, we need to compute $\zeta(\lambda)$ in Eq. (4) for a given λ. To compute $\zeta(\lambda)$, we approximate $\zeta(\lambda)$ via its first 20 terms, i.e., $\zeta(\lambda) = \sum_{k=1}^{20} \frac{1}{k^\lambda} + O(\frac{1}{21^\lambda})$. This is sufficient, because $\lambda > 2$ always holds in MVC-WP due to Line 6 in Algorithm 3 since $\forall c \geq 1 : Z^{-1}(c) > 2$. In this case, the sum of the remaining terms is sufficiently small and can thus be neglected, because

$$\frac{\sum_{k=21}^{\infty} \frac{1}{k^\lambda}}{\sum_{k=1}^{\infty} \frac{1}{k^\lambda}} \leq \frac{\sum_{k=21}^{\infty} \frac{1}{k^\lambda}}{\sum_{k=1}^{\infty} \frac{1}{k^2}} \approx 0.030. \tag{6}$$

4.4 The Inverse Function of $Z(\cdot)$

- For any $x < 1.002$, we approximate $Z^{-1}(x)$ to be equal to $+\infty$ (and thus approximate p_0 to be equal to 0 in Algorithm 3).
- For any $1.002 \leq x \leq 1.369$, we approximate $Z^{-1}(x)$ via linear interpolation according to Table 1, i.e., we assume $Z^{-1}(x)$ changes linearly between two consecutive entries given in Table 1.
- For any $x > 1.369$, we have $2 < k = Z^{-1}(x) < 3$. In this case, we approximate $\zeta(k)$ via linear interpolation, i.e.,

$$\zeta(k) \approx 1.645 - 0.443 \cdot (k - 2). \tag{7}$$

  We approximate $\zeta(k-1)$ via the first three terms of the Laurent series of $\zeta(k-1)$ at $k = 2$, i.e.,

$$\zeta(k-1) = \frac{1}{k-2} + \gamma - \gamma_1(k-2) + \mathcal{O}\left((k-2)^2\right), \tag{8}$$

Table 2: Compares sizes of VCs produced by MVC-WP-ER and MVC-WP-SF, respectively, with those of alternative algorithms. The three numbers in the 3rd to 6th columns represent the numbers of benchmark instances on which MVC-WP-ER and MVC-WP-SF produce smaller/equal/larger VC sizes, respectively. The numbers in parentheses indicate the number of benchmark instances in each benchmark instance set.

Our Algorithm	Alternative Algorithm	Misc (397)	Web (18)	Street (8)	Brain (26)
MVC-WP-ER	ConstructVC	211/39/147	12/1/5	8/0/0	0/0/26
	MVC-2	241/46/110	16/1/1	8/0/0	26/0/0
	R	376/16/5	17/1/0	8/0/0	26/0/0
	MVC-L	364/19/14	17/1/0	8/0/0	26/0/0
	MVC-MPL	317/18/62	17/1/0	1/0/7	26/0/0
MVC-WP-SF	ConstructVC	209/38/150	11/1/6	8/0/0	0/0/26
	MVC-2	249/45/103	15/1/2	8/0/0	26/0/0
	R	377/15/5	17/1/0	8/0/0	26/0/0
	MVC-L	363/21/13	17/1/0	8/0/0	26/0/0
	MVC-MPL	316/18/63	17/1/0	1/0/7	26/0/0

where $\gamma \approx 0.577$ is the Euler-Mascheroni constant and $\gamma_1 \approx -0.0728$ is the first Stieltjes constant [12, page 166]. By plugging these two equations into the definition of $Z(k)$ (i.e., $Z(k) = \frac{\zeta(k-1)}{\zeta(k)} = x$) and solving for k, we have the approximation

$$Z^{-1}(x) \approx \frac{1.645x - \gamma - \sqrt{(1.645x - \gamma)^2 - 4(0.443x - \gamma_1)}}{2 \cdot (0.443x - \gamma_1)} + 2. \tag{9}$$

5 Experimental Evaluation

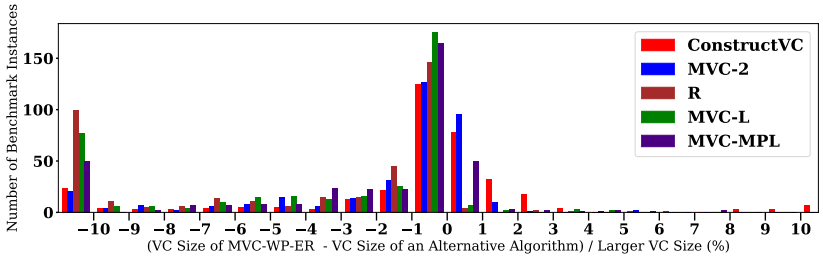
In this section, we experimentally evaluate MVC-WP. In our experiments, all algorithms were implemented in C++, compiled by GCC 6.3.0 with the "-O3" option, and run on a GNU/Linux workstation with an Intel Xeon Processor E3-1240 v3 (8MB Cache, 3.4GHz) and 16GB RAM. Throughout this section, we refer to MVC-WP using an ER model and an SF model as MVC-WP-ER and MVC-WP-SF, respectively.

We used 4 sets of benchmark instances[2]. The first 3 sets of benchmark instances were selected from the "misc networks", "web networks", and "brain networks" categories in Network Repository[3] [23]. All instances with no less than 100,000 vertices as of July 8, 2017 were used. The fourth set of benchmark instances consists of the benchmark instances in the "street networks" category in the 10th DIMACS Implementation Challenge[4] [3], in which 7 out of 8 benchmark instances have more than 1 million vertices. To obviate the influence of the orders in which the edges are specified in the input files, we shuffled the edges for each benchmark instance before applying the algorithms.

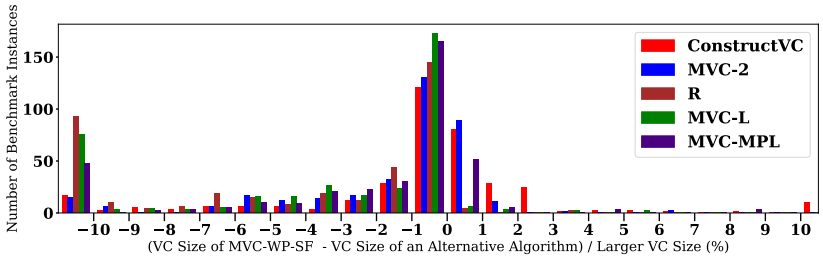
[2] We compiled these benchmark instances in the DIMACS format and made them available online at http://files.hong.me/papers/xu2018b-data.

[3] http://networkrepository.com/

[4] http://www.cc.gatech.edu/dimacs10/archive/streets.shtml



(a) MVC-WP-ER versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the misc networks benchmark instance set.



(b) MVC-WP-SF versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the misc networks benchmark instance set.

Fig. 2: Compares sizes of VCs produced by MVC-WP-ER, MVC-WP-SF, and alternative algorithms on the misc networks benchmark instance set. The x-axes show the relative suboptimality of MVC-WP-ER and MVC-WP-SF, respectively, compared with alternative algorithms. The y-axes show the number of benchmark instances for a range of relative suboptimality divided into bins of 1% (ranges beyond -10% and 10% are treated as single bins). Bars of different colors indicate different algorithms. Higher bars in the left half indicate that MVC-WP-ER and MVC-WP-SF, respectively, produce VCs of sizes smaller than the alternative algorithms.

To evaluate those algorithms that use random number generators, i.e., MVC-WP-ER, MVC-WP-SF, R, MVC-L and MVC-MPL, we ran them 10 times on each benchmark instance using different seeds. We recorded the average of the VC sizes produced by these 10 runs. For all algorithms compared in this section, we applied `Prune-Leaves` and `Remove-Redundancy` as preprocessing and postprocessing steps, respectively, since they are universally useful.

We evaluated MVC-WP-ER and MVC-WP-SF by comparing them with various other algorithms, namely ConstructVC, MVC-2, R, MVC-MPL, and MVC-L. We set $M = 3$ for both MVC-WP-ER and MVC-WP-SF, noting that $M = 3$ is a very small number of iterations of warning propagation.

Tables 2 and 4 and Fig. 2 compare these algorithms. In the misc networks and web networks benchmark instance sets, both MVC-WP-ER and MVC-WP-SF outperformed all other algorithms in terms of sizes of produced VCs. In the brain networks benchmark instance set, both MVC-WP-ER and MVC-WP-SF outperformed all other algorithms except ConstructVC. In the street networks

Table 3: Shows the number of vertices and edges of benchmark instances in the web networks, street networks, and brain networks benchmark instance sets.

	Instance	$\|V\|$	$\|E\|$
Web Networks	web-wikipedia-link-it	1,051,219	25,199,339
	web-wikipedia-growth	898,367	4,468,005
	web-BerkStan	5,121	8,345
	web-italycnr-2000	176,055	2,336,551
	web-uk-2005	127,716	11,643,622
	web-Stanford	226,733	1,612,323
	web-BerkStan-dir	552,353	5,674,493
	web-google-dir	451,765	2,434,390
	web-wikipedia2009	154,344	302,990
	web-it-2004	424,893	6,440,816
	web-wikipedia-link-fr	1,098,517	12,683,034
	web-hudong	53,799	286,998
	web-arabic-2005	102,515	1,560,020
	web-baidu-baike	56,346	104,037
	web-NotreDame	99,557	631,931
	web-sk-2005	42,237	225,932
	web-wiki-ch-internal	21,101	47,480
	web-baidu-baike-related	126,607	710,562
Street Networks	asia	2,642,989	3,032,404
	germany	2,455,500	2,673,629
	great-britain	1,284,868	1,383,591
	luxembourg	23,196	24,710
	belgium	460,536	503,521
	netherlands	726,730	815,305
	italy	1,783,377	1,942,410
	europe	12,512,346	13,711,218

	Instance	$\|V\|$	$\|E\|$
Brain Networks	0025871-session-1-bg	738,598	168,617,323
	0025872-session-2-bg	759,626	147,761,328
	0025869-session-1-bg	679,760	134,979,814
	0025876-session-1-bg	778,074	140,293,764
	0025865-session-2-bg	705,588	155,118,679
	0025868-session-1-bg	717,428	150,383,991
	0025872-session-1-bg	746,316	166,528,410
	0025864-session-2-bg	682,197	133,656,879
	0025912-session-2	771,224	147,496,369
	0025868-session-2-bg	717,420	158,562,090
	0025869-session-2-bg	705,280	151,476,861
	0025873-session-1-bg	636,430	149,483,247
	0025870-session-2-bg	799,455	166,724,734
	0025865-session-1-bg	725,412	165,845,120
	0025889-session-2	731,931	131,860,075
	0025876-session-2-bg	766,763	139,801,374
	0025867-session-1-bg	735,513	145,208,968
	0025874-session-2-bg	758,757	163,448,904
	0025873-session-2-bg	682,580	140,044,477
	0025889-session-1	694,544	144,411,722
	0025870-session-1-bg	785,719	148,684,011
	0025871-session-2-bg	724,848	170,944,764
	0025864-session-1-bg	685,987	143,091,223
	0025867-session-2-bg	724,276	154,604,919
	0025878-session-1-bg	690,012	127,838,275
	0025886-session-1	769,878	158,111,887

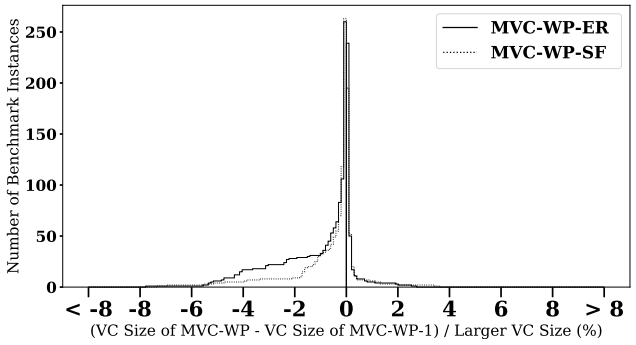


Fig. 3: Compares sizes of VCs produced by MVC-WP-ER and MVC-WP-SF with those produced by MVC-WP-1. The x-axis shows the relative suboptimality of MVC-WP compared with MVC-WP-1. The y-axis shows the number of benchmark instances. In the left half, for each point on the curve, its y coordinate shows the number of benchmark instances with relative suboptimality smaller than its x coordinate. In the right half, for each point on the curve, its y coordinate shows the number of benchmark instances with relative suboptimality larger than its x coordinate. Larger areas under the curves in the left half and smaller areas under the curves in the right half indicate that MVC-WP-ER and MVC-WP-SF, respectively, produce VCs of sizes smaller than MVC-WP-1.

benchmark instance set, both MVC-WP-ER and MVC-WP-SF outperformed all other algorithms except for MVC-MPL. The reason may be that street networks are always planar and thus in general cannot be well modeled as ER or SF graphs. Overall, MVC-WP-ER and MVC-WP-SF conclusively outperformed

Table 4: Compares sizes of VCs produced by MVC-WP-ER, MVC-WP-SF, and alternative algorithms on the web networks, street networks, and brain networks benchmark instance sets. The numbers of vertex and edge of each benchmark instance are shown in Table 3. The smallest sizes of VCs produced for each benchmark instance are highlighted.

Instance	ConstructVC	MVC-2	R	MVC-L	MVC-MPL	MVC-WP-ER	MVC-WP-SF
web-wikipedia-link-it	991,272	987,621	1,039,011	1,018,672	1,020,827	972,275	972,670
web-wikipedia-growth	914,746	926,530	966,410	950,741	940,302	909,910	909,989
web-BerkStan	5,542	5,469	5,605	5,567	5,726	5,463	5,469
web-italycnr-2000	99,645	99,609	110,559	104,272	103,153	97,844	97,932
web-uk-2005	127,774	127,774	127,774	127,774	127,774	127,774	127,774
web-Stanford	126,603	126,960	136,048	130,248	128,296	123,540	123,890
web-BerkStan-dir	290,206	291,277	320,384	310,483	304,623	285,593	285,934
web-google-dir	350,676	355,930	387,462	379,508	365,427	351,241	351,343
web-wikipedia2009	650,888	654,152	657,343	656,131	655,813	652,241	652,363
web-it-2004	415,408	415,083	415,915	415,533	415,042	414,835	414,972
web-wikipedia-link-fr	1,574,973	1,558,998	1,626,052	1,598,021	1,597,887	1,538,658	1,538,960
web-hudong	503,373	504,025	506,598	505,903	504,839	503,335	503,359
web-arabic-2005	114,504	114,743	115,161	114,999	115,004	114,721	114,727
web-baidu-baike	637,805	638,538	640,537	639,811	639,935	637,796	637,815
web-NotreDame	76,468	76,257	80,341	79,013	77,893	75,735	75,953
web-sk-2005	58,238	58,300	58,669	58,443	58,370	58,347	58,349
web-wiki-ch-internal	260,354	260,476	261,571	261,244	260,927	260,213	260,231
web-baidu-baike-related	144,388	146,588	151,689	149,957	148,749	145,272	145,249
asia	6,087,218	6,099,227	6,130,265	6,104,699	6,018,875	6,053,077	6,049,489
germany	5,822,566	5,834,966	5,864,005	5,841,507	5,768,621	5,792,165	5,789,947
great-britain	3,837,647	3,843,980	3,857,098	3,844,972	3,804,317	3,821,741	3,820,618
luxembourg	58,168	58,267	58,456	58,230	57,417	57,823	57,810
belgium	739,185	741,647	747,374	742,968	729,190	733,264	732,877
netherlands	1,133,606	1,141,977	1,147,202	1,141,786	1,131,859	1,127,358	1,125,978
italy	3,425,723	3,434,538	3,452,907	3,434,830	3,374,512	3,401,824	3,400,005
europe	25,903,178	25,968,573	26,104,371	25,983,279	25,589,132	25,743,670	25,730,398
0025871-session-1-bg	688,391	695,636	701,228	698,818	699,234	694,616	694,625
0025872-session-2-bg	706,691	714,407	720,128	717,692	718,017	713,557	713,599
0025869-session-1-bg	629,715	637,159	642,647	640,327	641,147	636,057	636,050
0025876-session-1-bg	712,322	721,476	728,763	725,679	726,560	720,332	720,404
0025865-session-2-bg	656,483	663,359	669,205	666,846	666,996	662,446	662,564
0025868-session-1-bg	662,749	670,415	676,703	674,029	674,729	669,473	669,452
0025872-session-1-bg	692,476	700,076	705,979	703,495	704,039	699,263	699,284
0025864-session-2-bg	631,361	638,668	644,392	641,875	642,737	637,833	637,831
0025912-session-2	716,154	724,000	730,201	727,577	727,982	723,070	723,165
0025868-session-2-bg	662,025	669,661	676,133	673,415	673,972	668,845	668,784
0025869-session-2-bg	651,656	659,116	665,348	662,796	662,931	658,201	658,171
0025873-session-1-bg	595,166	601,378	605,916	604,042	604,270	600,488	600,467
0025870-session-2-bg	735,348	744,205	751,736	748,478	749,706	743,052	743,103
0025865-session-1-bg	676,416	683,459	689,108	686,788	687,193	682,528	682,526
0025889-session-2	674,887	683,034	689,378	686,679	687,462	682,117	682,056
0025876-session-2-bg	701,544	710,627	717,965	714,784	715,793	709,579	709,604
0025867-session-1-bg	682,118	689,846	695,806	693,294	693,749	688,861	688,829
0025874-session-2-bg	701,559	709,752	716,330	713,553	714,320	708,796	708,786
0025873-session-2-bg	635,363	641,971	647,566	645,325	645,449	641,349	641,441
0025889-session-1	644,858	651,812	657,483	655,226	655,525	651,102	651,113
0025870-session-1-bg	721,906	730,773	738,123	734,922	736,193	729,761	729,805
0025871-session-2-bg	674,478	681,551	687,309	684,955	685,328	680,690	680,686
0025864-session-1-bg	634,702	642,146	647,884	645,431	646,130	641,119	641,058
0025867-session-2-bg	673,075	680,577	686,239	683,853	684,641	679,654	679,647
0025878-session-1-bg	636,617	644,044	650,351	647,661	648,109	643,237	643,190
0025886-session-1	713,597	721,617	728,101	725,332	726,009	720,641	720,638

their competitors. We also conducted further experiments to demonstrate the usefulness of various individual steps of MVC-WP-ER and MVC-WP-SF.

To demonstrate the effectiveness of the message initialization step in MVC-WP-ER and MVC-WP-SF, i.e., assigning messages to be zero with probability p_0 computed from random graph models, we compared MVC-WP-ER and MVC-WP-SF with variants thereof in which p_0 is always set to 1 in order to mimic the message initialization in the standard warning propagation algorithm [20]. We refer to this variant as MVC-WP-1.

Figure 3 compares MVC-WP-ER and MVC-WP-SF with MVC-WP-1 on the misc networks benchmark instance set. Both MVC-WP-ER and MVC-WP-SF

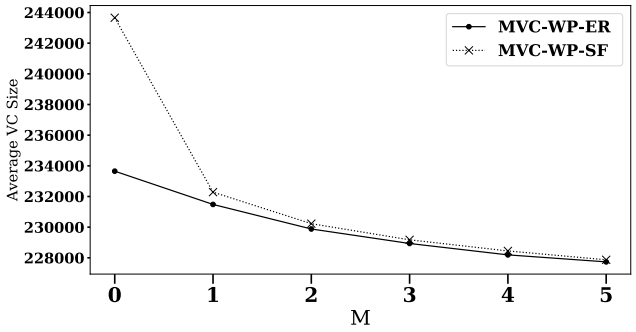


Fig. 4: Compares sizes of VCs produced by MVC-WP-ER and MVC-WP-SF for different values of M on the misc networks benchmark instance set.

significantly outperformed MVC-WP-1 in terms of sizes of produced VCs. These results demonstrate the importance of our message initialization step.

To study the effect of M on MVC-WP-ER and MVC-WP-SF, we ran them for different values of M. For both MVC-WP-ER and MVC-WP-SF with $M \in \{0, 1, \ldots, 5\}$, Fig. 4 shows the sizes of the VC averaged over all benchmark instances in the misc networks benchmark instance set. The average VC size decreases with increasing M. The results demonstrate the usefulness of warning propagation iterations in MVC-WP-ER and MVC-WP-SF.

To demonstrate the effectiveness of Algorithm 2, we compared MVC-WP-ER and MVC-WP-SF with and without the use of it on the web networks benchmark instance set. MVC-WP-ER and MVC-WP-SF produced VCs of sizes that are on average 0.51% and 1.0% smaller than their counterparts without the use of Algorithm 2. These results demonstrate the importance of Algorithm 2.

Due to the fact that all algorithms are linear-time, all of them terminated very quickly. Despite that MVC-WP-ER and MWVC-WP-SF are slower than alternative algorithms, over 80% of their runs terminated within 300ms, which makes it difficult to measure the algorithms' running times on a single benchmark instance. In addition, it took much longer time (a few hundred times longer) to read input files from the hard disk than running these algorithms, which makes it difficult to reliably count the numbers of benchmark instances solved within a certain amount of time. For these reasons, it is difficult to have reliable comparisons of running times of all algorithms. Therefore, we skip the detailed comparison here, while this may be an interesting future work.

It is also interesting to compare the VCs produced by these linear-time-and-space algorithms with the sizes of MVCs. Since the MVC problem is NP-hard, it is elusive to find MVCs on the giant graphs in our previous used benchmark instances. Therefore, we ran all algorithms on the benchmark instances with provided solutions from the Second DIMACS Implementation Challenge[5] [17]. Since the given solutions are for the maximum clique problem, we ran all algo-

[5] http://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/

Table 5: Compares sizes of MVCs and produced VCs by ConstructVC, MVC-2, MVC-R, MVC-L, MVC-MPL, MVC-WP-ER, and MVC-WP-SF, respectively, on benchmark instances with solutions from the Second DIMACS Implementation Challenge.

Graph			Algorithm						
Instance	$\|V\|$	MVC	ConstructVC	MVC-2	R	MVC-L	MVC-MPL	MVC-ER	MVC-SF
c-fat200-1	200	188	188	188	189	189	189	188	190
c-fat200-2	200	176	176	176	177	177	177	177	177
c-fat200-5	200	142	142	142	143	142	143	142	142
c-fat500-10	500	374	374	374	374	375	375	374	374
c-fat500-1	500	486	486	486	487	487	487	487	488
c-fat500-2	500	474	474	474	474	475	474	474	474
c-fat500-5	500	436	436	436	437	437	437	437	436
hamming10-2	1024	512	760	674	787	741	738	593	651
hamming6-2	64	32	45	40	44	40	39	34	41
hamming6-4	64	60	60	60	60	60	60	60	60
hamming8-2	256	128	155	185	188	182	178	141	160
hamming8-4	256	240	250	248	248	246	246	245	248
johnson16-2-4	120	112	112	112	112	112	112	112	112
johnson8-2-4	28	24	24	24	24	24	24	24	24
johnson8-4-4	70	56	56	60	61	60	60	58	56
keller4	171	160	162	164	163	163	163	163	161
p-hat300-1	300	292	295	294	295	294	294	295	295
p-hat300-2	300	275	279	285	286	286	284	280	282
p-hat300-3	300	264	270	273	278	278	277	273	271
p-hat500-1	500	491	494	494	494	494	494	493	495
p-hat500-2	500	464	470	479	482	480	481	472	473
san200-0	200	170	184	185	185	185	185	185	185
san200-0	200	130	155	155	159	156	154	154	155
san200-0	200	140	163	163	168	167	166	164	165
san200-0	200	156	169	169	173	172	171	171	172
sanr200-0	200	182	187	188	187	188	187	187	188

rithms on the complements of the graphs in these benchmark instances, since the maximum clique problem on a graph is equivalent to the MVC problem on the complement of the graph. The solutions are shown in Table 5. From the table, we see that all linear-time-and-space algorithms produced VCs of similar results. We also see that, the produced VCs have sizes very close to the sizes of MVCs on all benchmark instances except hamming6-2 and hamming10-2.

6 Conclusions and Future Work

We developed MVC-WP, a warning propagation-based linear-time-and-space algorithm that finds small minimal VCs for giant graphs. We empirically showed that MVC-WP outperforms several other linear-time-and-space algorithms in terms of sizes of produced VCs. We also empirically showed that the theoretical underpinnings of MVC-WP significantly contribute to its success. These include both the way in which MVC-WP performs message initialization by computing p_0 and the iterations of warning propagation. We also made secondary contributions in computing various special functions efficiently with numerical accuracy sufficient for many AI applications. Future directions include applying similar techniques to solving other fundamental combinatorial problems on giant graphs.

Acknowledgment The research at the University of Southern California was supported by the National Science Foundation (NSF) under grant numbers 1724392, 1409987, and 1319966. 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. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: Theory and experiments. In: the Workshop on Algorithm Engineering and Experiments (2004)
2. Andrade, D.V., Resende, M.G.C., Werneck, R.F.: Fast local search for the maximum independent set problem. Journal of Heuristics 18(4), 525–547 (2012)
3. Bader, D.A., Meyerhenke, H., Sanders, P., Wagner, D. (eds.): Graph Partitioning and Graph Clustering. American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science (2013)
4. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509–512 (1999)
5. Cai, S.: Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In: the International Joint Conference on Artificial Intelligence. pp. 747–753 (2015)
6. 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)
7. Corless, R.M., Gonnet, G.H., Hare, D.E.G., Jeffrey, D.J., Knuth, D.E.: On the LambertW function. Advances in Computational Mathematics 5(1), 329–359 (1996)
8. Dinur, I., Safra, S.: On the hardness of approximating minimum vertex cover. Annals of Mathematics 162(1), 439–485 (2005)
9. Erdős, P., Rényi, A.: On random graphs I. Publicationes Mathematicae 6, 290–297 (1959)
10. Fang, Z., Li, C.M., Xu, K.: An exact algorithm based on MaxSAT reasoning for the maximum weight clique problem. Journal of Artificial Intelligence Research 55, 799–833 (2016)
11. Filiol, E., Franc, E., Gubbioli, A., Moquet, B., Roblot, G.: Combinatorial optimisation of worm propagation on an unknown network. International Journal of Computer, Electrical, Automation, Control and Information Engineering 1(10), 2931–2937 (2007)
12. Finch, S.R.: Mathematical Constants, Encyclopedia of Mathematics and its Applications, vol. 94. Cambridge University Press (2003)
13. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer (2006)
14. Goyal, A., Lu, W., Lakshmanan, L.V.S.: SIMPATH: An efficient algorithm for influence maximization under the linear threshold model. In: the IEEE International Conference on Data Mining. pp. 211–220 (2011)
15. Haynsworth, E.V., Goldberg, K.: Bernoulli and Euler polynomials—Riemann zeta function. In: Abramowitz, M., Stegun, I.A. (eds.) Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables, pp. 803–819. Dover Publications, Inc. (1965)
16. Hiary, G.A.: Fast methods to compute the Riemann zeta function. Annals of Mathematics 174(2), 891–946 (2011)
17. Johnson, D.J., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society (1996)
18. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972)
19. Korte, B., Vygen, J.: Combinatorial Optimization: Theory and Algorithms. Springer, 5th edn. (2012)

20. Mézard, M., Montanari, A.: Information, Physics, and Computation. Oxford University Press (2009)
21. 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)
22. Pullan, W.: Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. Discrete Optimization 6(2), 214–219 (2009)
23. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: the AAAI Conference on Artificial Intelligence. pp. 4292–4293 (2015), http://networkrepository.com
24. Sherali, H.D., Rios, M.: An air force crew allocation and scheduling problem. The Journal of the Operational Research Society 35(2), 91–103 (1984)
25. Vazirani, V.V.: Approximation Algorithms. Springer (2003)
26. Weigt, M., Zhou, H.: Message passing for vertex covers. Physical Review E 74(4), 046110 (2006)
27. Xu, H., Kumar, T.K.S., Koenig, S.: A new solver for the minimum weighted vertex cover problem. In: the International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming. pp. 392–405 (2016)
28. Xu, H., Kumar, T.K.S., Koenig, S.: A linear-time and linear-space algorithm for the minimum vertex cover problem on giant graphs. In: the International Symposium on Combinatorial Search. pp. 173–174 (2017)
29. Yamaguchi, K., Masuda, S.: A new exact algorithm for the maximum weight clique problem. In: the International Technical Conference on Circuits/Systems, Computers and Communications. pp. 317–320 (2008)