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

Hong Xu Kexuan Sun Sven Koenig T. K. Satish Kumar
University of Southern California, Los Angeles, California 90089, the 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 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 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 solution quality.

Introduction

Thanks to the advancement of technologies such as the Internet and database management systems, datasets have been growing tremendously over the past decade and have resulted in many giant datasets. Among these giant datasets, many 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.

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 with 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 (Karp 1972). 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 concept of IS is deeply connected to that of VC: 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 (Filiol et al. 2007), in crew scheduling (Sherali and Rios 1984), and in the construction of phylogenetic trees (Abu-Khzam et al. 2004). 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 (Korte and Vygen 2012). It is also a fundamental problem studied in the theory of fixed-parameter tractability (Flum and Grohe 2006).

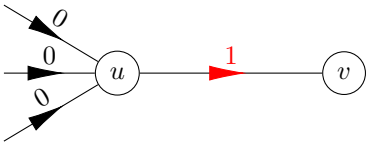
Various researchers have developed exact solvers (Niskanen and Östergård 2003; Yamaguchi and Masuda 2008; Fang, Li, and Xu 2016; Xu, Kumar, and Koenig 2016) for the MVC problem and its equivalents. However, none of these solvers work well for large problem instances of the MVC problem due to its NP-hardness. Furthermore, solving the MVC problem within any approximation factor smaller than 1.3606 is also NP-hard (Dinur and Safra 2005).

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 (Pullan 2009; Andrade, Resende, and Werneck 2012; Cai et al. 2013; Cai 2015) 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 space. Therefore, it is important to develop a linear-time-and-space algorithm to find a small VC.

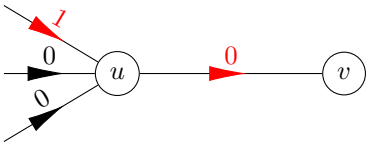
In addition, many real-world applications on giant graphs require the identification of small VCs but not necessarily the MVCs. One example of such applications is the influence maximization problem in social networks (Goyal, Lu, and Lakshmanan 2011). Here, too, linear-time-and-space algorithms for finding small VCs are imperative.

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

Citation: Hong Xu, Kexuan Sun, Sven Koenig, and T. K. Satish Kumar. A warning propagation-based linear-time-and-space algorithm for the minimum vertex cover problem on giant graphs. In Proceedings of the 15th International Symposium on Artificial Intelligence and Mathematics (ISAIM), 2018.



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



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

Figure 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 input graph $G = \langle V, E \rangle$. Only relevant parts of G are shown, i.e., u, v, and all edges incident to u.

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.

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.

Random Graph Models

The Erdős-Rényi Model An *Erdős-Rényi model* (ER model) (Erdős and Rényi 1959) is characterized by two parameters n and p. It generates random graphs with n vertices and 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) (Barabási and Albert 1999) 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*.

Warning Propagation

The *warning propagation algorithm* is a specialized *message passing algorithm* where information is processed locally and exchanged between relevant variables (Mézard and Montanari 2009). In the warning propagation algorithm, messages can only take one of two values, namely 0 or 1. To analyze properties of MVC on infinite random graphs, (Weigt and Zhou 2006) 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 until convergence: 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. The theoretical analysis in (Weigt and Zhou 2006) mainly focuses on ER graphs. They show 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 $f(x) = x \exp(x)$.

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 (Vazirani 2003): In each iteration, the algorithm 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. The algorithm terminates when all edges are marked as being covered.

ConstructVC Serving as a preprocessing step, ConstructVC is a greedy linear-time-and-space subroutine in the FastVC solver (Cai 2015), 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. After all edges are marked as being covered, ConstructVC 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 (Andrade, Resende, and Werneck 2012). 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 even after the removal. R terminates when the VC is minimal.

MVC-MPL MVC-MPL (Xu, Kumar, and Koenig 2017) is a linear-time-and-space MVC algorithm based on the intuition of warning propagation on ER graphs. It works as follows: In each iteration, MVC-MPL first arbitrarily selects a vertex, then adds it to the VC or the IS with a probability derived from theoretical results that govern warning propagation on ER graphs. It terminates when every vertex has been added to either the VC or the IS.

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

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 (Weigt and Zhou 2006, 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

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

A message $m_{u\to v}$ from vertex u to vertex v is equal to 1 iff all messages incoming 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 Equation (3) for p_0 and have

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

Here, we have $\forall \lambda > 2 : 0 \le p_0 \le 1$. Therefore, for any $\lambda > 2$, Equation (4) is always a valid solution for p_0.

The Algorithm

Our algorithm MVC-WP (Algorithm 1) is based on the analytical results that govern the warning propagation algorithm for the MVC problem (Weigt and Zhou 2006). It first uses Algorithm 2 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 thereafter assigns each message from vertex u to vertex v to be 0 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 iff it receives at least one message of 1. 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 (Cai 2015).

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

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

Proof. We first prove that all edges removed from G by Algorithm 2 are covered by the VC that it outputs. 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, IS is an independent set since only leaves are added to it. In Algorithm 2, $VC \cup IS$ is the set of all removed vertices, since each removed vertex is added to either VC or IS.

In Algorithm 1, the M iterations of warning propagation do not affect the correctness of VC.

We now prove that Lines 27 to 33 in Algorithm 1 add

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: User-specified number of iterations of the
              warning propagation algorithm.
       Output: A minimal VC of G.
2      VC, IS := Prune-Leaves(G);
3      p0 := Compute-p0(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 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 ≤ p0^(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 u to VC;
34     VC := Remove-Redundancy(G, VC);
35     return 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 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 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 Equation (4);
8      return p0;
```

new vertices to VC to cover all edges in G. Since Line 31 guarantees that no two adjacent vertices are added to IS, IS must be an independent set of G. In addition, Line 28 guarantees that IS and VC are complementary, i.e., $IS \cap VC = \emptyset$ and $IS \cup VC = V$. Therefore, VC must be a vertex cover.

(Cai 2015) has proved that `Remove-Redundancy` produces a minimal vertex cover provided that VC is a vertex cover. Therefore, MVC-WP produces a minimal vertex cover. □

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

Proof. We first prove that `Prune-Leaves` terminates in $\mathcal{O}(|V| + |E|)$ time. This can be done 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

Algorithm 4: Remove redundant vertices from a given VC.

```
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 used to be in 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;
```

k	2	3	4	5	6	7	8	9
$\zeta(k)$	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

Table 1: Shows the values of $\zeta(k)$ and $Z(k) = \frac{\zeta(k-1)}{\zeta(k)}$ for $k \in \{2, 3, \ldots, 9\}$. The values of $\zeta(k)$ are taken from (Haynsworth and Goldberg 1965, Table 23.3), and the values of $Z(k)$ are computed from the values of $\zeta(k)$.

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.

(Cai 2015) has proved that `Remove-Redundancy` terminates in $\mathcal{O}(|V| + |E|)$ time.

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

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

Proof. (Cai 2015) 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. □

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 community have already developed various numerical methods (Corless et al. 1996; Hiary 2011). 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.

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

We approximate $W(\cdot)$ via the first 3 terms of Equation (4.19) in (Corless et al. 1996), i.e.,

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

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

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

For the SF model, we need to compute $\zeta(x)$ in Equation (4) for a given x. To compute $\zeta(x)$, we approximate $\zeta(x)$ via its first 20 terms, i.e., $\zeta(x) = \sum_{k=1}^{20} \frac{1}{k^x} + O(\frac{1}{21^x})$. This is sufficiency 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 to be neglected, i.e.,

$$\frac{\sum_{k=21}^{\infty} \frac{1}{k^x}}{\sum_{k=1}^{\infty} \frac{1}{k^x}} \leq \frac{\sum_{k=21}^{\infty} \frac{1}{k^2}}{\sum_{k=1}^{\infty} \frac{1}{k^2}} \approx 0.030. \quad (6)$$

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.,

  $$\zeta(k) \approx 1.645 - 0.443 \cdot (k - 2). \quad (7)$$

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

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

  where $\gamma \approx 0.577$ is the Euler-Mascheroni constant and $\gamma_1 \approx -0.0728$ is the first Stieltjes constant (Finch 2003, page 166). By plugging these two equations into the definition of $Z(k)$ (i.e., $Z(k) = \frac{\zeta(k-1)}{\zeta(k)}$) and solving for k as a function of $Z(k)$, we have the approximation

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

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-MPL	317/18/62	17/1/0	1/0/7	26/0/0
	MVC-L	364/19/14	17/1/0	8/0/0	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-MPL	316/18/63	17/1/0	1/0/7	26/0/0
	MVC-L	363/21/13	17/1/0	8/0/0	26/0/0

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.

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. The first 3 sets of benchmark instances were selected from the "misc networks", "web networks", and "brain networks" categories in Network Repository[2] (Rossi and Ahmed 2015). All instances with more than 100,000 vertices are used. The fourth set of benchmark instances consists of the benchmark instances in the "street networks" category in the 10th DIMACS Implementation Challenge[3] (Bader et al. 2013), in which 7 out of 8 benchmark instances have more than 1 million vertices.[4] 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.

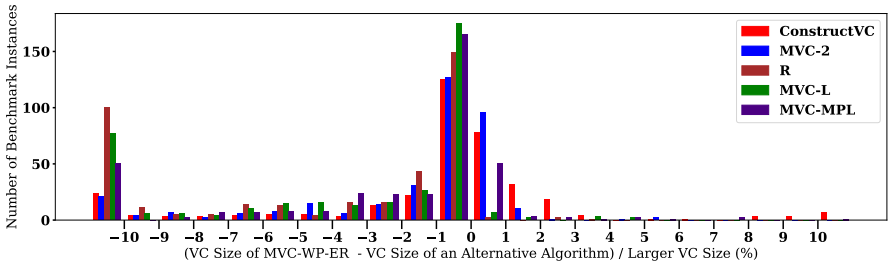
To evaluate any algorithm that uses a random number generator, we ran it 10 times on each benchmark instance using different seeds. We recorded the average of the VC sizes produced by these 10 runs. For all algorithms compared in this section, we applied `Prune-Leaves` and `Remove-Redundancy` as preprocessing 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 MVC-2, ConstructVC, R, MVC-MPL and MVC-L (a variant of

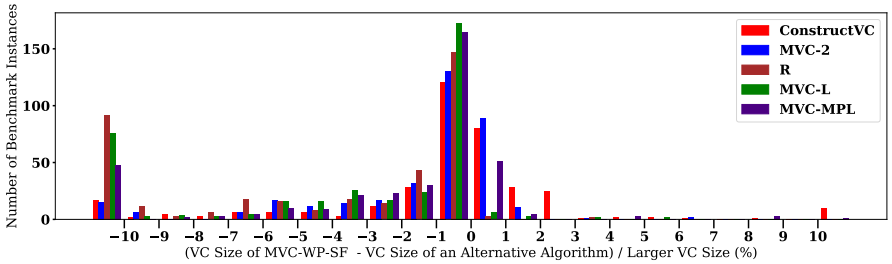
[2] `http://networkrepository.com/`

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

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

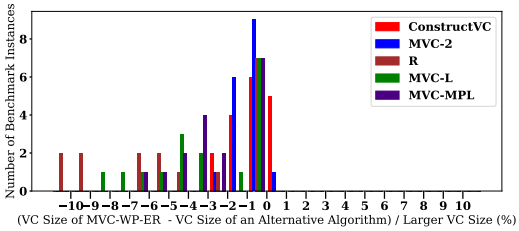


(a) MVC-WP-ER versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the misc networks benchmark instance set.

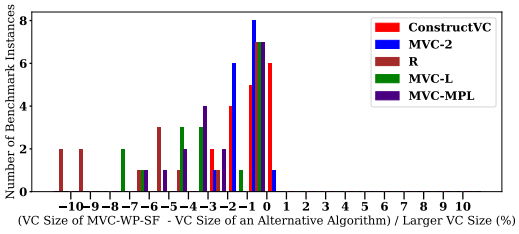


(b) MVC-WP-SF versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the misc networks benchmark instance set.

Figure 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-SF and MVC-WP-ER, 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 smaller sizes.

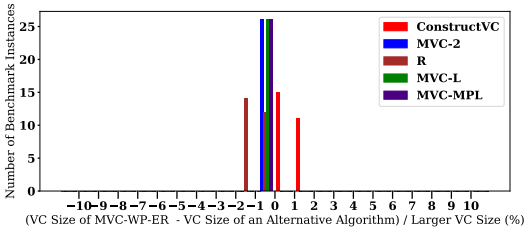


(a) MVC-WP-ER versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the web networks benchmark instance set.

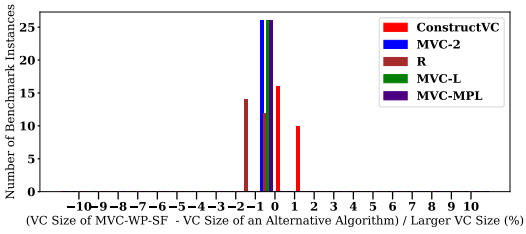


(b) MVC-WP-SF versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the web networks benchmark instance set.

Figure 3: Compares sizes of VCs produced by MVC-WP-ER, MVC-WP-SF, and alternative algorithms on the web networks benchmark instance set. The meanings of the x-axes, y-axes, and legends are the same as those in Figure 2.

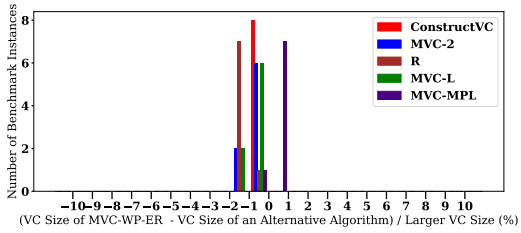


(a) MVC-WP-ER versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the brain networks benchmark instance set.

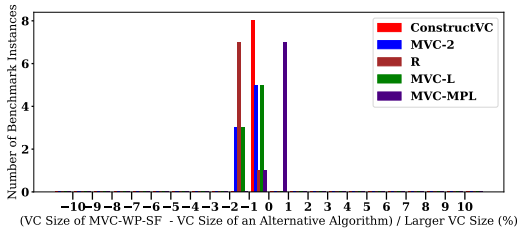


(b) MVC-WP-SF versus ConstructVC/MVC-2/R/MVC-L/MVC-MPL on the brain networks benchmark instance set.

Figure 4: Compares sizes of VCs produced by MVC-WP-ER, MVC-WP-SF, and alternative algorithms on the brain networks benchmark instance set. The meanings of the x-axes, y-axes, and legends are the same as those in Figure 2.



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



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

Figure 5: Compares sizes of VCs produced by MVC-WP-ER, MVC-WP-SF, and alternative algorithms on the street networks benchmark instance set. The meanings of the x-axes, y-axes, and legends are the same as those in Figure 2.

MVC-MPL introduced by (Xu, Kumar, and Koenig 2017)). 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.

Table 2 and Figures 2 to 5 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 the brain networks benchmark instance set, both MVC-WP-ER and MVC-WP-SF outperformed all alternative algorithms except ConstructVC. In the street networks benchmark instance set, both MVC-WP-ER and MVC-WP-SF outperformed all other algorithms except MVC-MPL. Overall, MVC-WP-ER and MVC-WP-SF conclusively outperformed 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 a probability of 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 (Mézard and Montanari 2009). We refer to this variant as MVC-WP-1.

Figure 6 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 significantly outperformed MVC-WP-1 in terms of solution quality. These results demonstrate the importance of setting p_0 according to

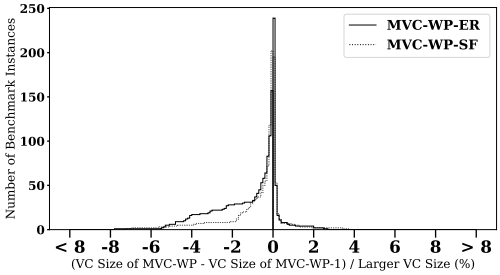


Figure 6: Compares sizes of VCs produced by MVC-WP-ER and MVC-WP-SF with those produced by MVC-WP-1. The x-axis shows the relative suboptimality of MVC-WP compared with MVC-WP-1. The y-axis shows the number of benchmark instances. In the left half, for each point on the curve, its y coordinate shows 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 smaller sizes than MVC-WP-1.

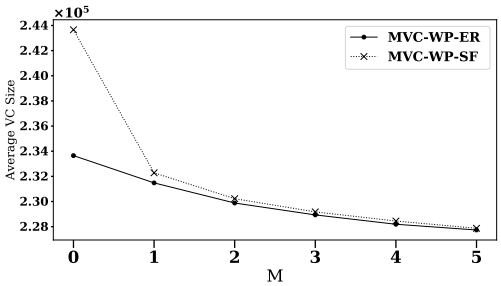


Figure 7: Compares MVC-WP-ER and MVC-WP-SF for different values of M.

random graph models.

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 $M = \{0, 1, \ldots, 5\}$, Figure 7 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. This indicates the usefulness of warning propagation in MVC-WP-ER and MVC-WP-SF.

Due to space limitations and given that all algorithms are linear-time, we do not show running times in the experimental results. Over 80% of runs terminated within 300ms.

Conclusions and Future Work

We developed MVC-WP, a warning propagation-based linear-time-and-space algorithm that finds a small minimal VC for giant graphs. We empirically showed that MVC-WP outperforms several other linear-time-and-space algorithms in terms of solution quality. 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

Abu-Khzam, F. N.; Collins, R. L.; Fellows, M. R.; Langston, M. A.; Suters, W. H.; and Symons, C. T. 2004. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *the Workshop on Algorithm Engineering and Experiments*.

Andrade, D. V.; Resende, M. G. C.; and Werneck, R. F. 2012. Fast local search for the maximum independent set problem. *Journal of Heuristics* 18(4):525–547.

Bader, D. A.; Meyerhenke, H.; Sanders, P.; and Wagner, D., eds. 2013. *Graph Partitioning and Graph Clustering*. American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science.

Barabási, A.-L., and Albert, R. 1999. Emergence of scaling in random networks. *Science* 286(5439):509–512.

Cai, S.; Su, K.; Luo, C.; and Sattar, A. 2013. NuMVC: An efficient local search algorithm for minimum vertex cover. *Journal of Artificial Intelligence Research* 46(1):687–716.

Cai, S. 2015. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *the International Joint Conference on Artificial Intelligence*, 747–753.

Corless, R. M.; Gonnet, G. H.; Hare, D. E. G.; Jeffrey, D. J.; and Knuth, D. E. 1996. On the LambertW function. *Advances in Computational Mathematics* 5(1):329–359.

Dinur, I., and Safra, S. 2005. On the hardness of approximating minimum vertex cover. *Annals of Mathematics* 162(1):439–485.

Erdős, P., and Rényi, A. 1959. On random graphs I. *Publicationes Mathematicae* 6:290–297.

Fang, Z.; Li, C.-M.; and Xu, K. 2016. An exact algorithm based on MaxSAT reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research* 55:799–833.

Filiol, E.; Franc, E.; Gubbioli, A.; Moquet, B.; and Roblot, G. 2007. Combinatorial optimisation of worm propagation on an unknown network. *International Journal of Computer, Electrical, Automation, Control and Information Engineering* 1(10):2931–2937.

Finch, S. R. 2003. *Mathematical Constants*, volume 94 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press.

Flum, J., and Grohe, M. 2006. *Parameterized Complexity Theory*. Springer.

Goyal, A.; Lu, W.; and Lakshmanan, L. V. S. 2011. SIMPATH: An efficient algorithm for influence maximization under the linear threshold model. In *the IEEE International Conference on Data Mining*, 211–220.

Haynsworth, E. V., and Goldberg, K. 1965. Bernoulli and Euler polynomials—Riemann zeta function. In Abramowitz, M., and Stegun, I. A., eds., *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover Publications, Inc. 803–819.

Hiary, G. A. 2011. Fast methods to compute the Riemann zeta function. *Annals of Mathematics* 174(2):891–946.

Karp, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Plenum Press, New York. 85–103.

Korte, B., and Vygen, J. 2012. *Combinatorial Optimization: Theory and Algorithms*. Springer, 5th edition.

Mézard, M., and Montanari, A. 2009. *Information, Physics, and Computation*. Oxford University Press.

Niskanen, S., and Östergård, P. R. J. 2003. Cliquer user's guide, version 1.0. Technical Report T48, Communications Laboratory, Helsinki University of Technology, Espoo, Finland.

Pullan, W. 2009. Optimisation of unweighted/weighted maximum independent sets and minimum vertex covers. *Discrete Optimization* 6(2):214–219.

Rossi, R. A., and Ahmed, N. K. 2015. The network data repository with interactive graph analytics and visualization. In *the AAAI Conference on Artificial Intelligence*, 4292–4293.

Sherali, H. D., and Rios, M. 1984. An air force crew allocation and scheduling problem. *The Journal of the Operational Research Society* 35(2):91–103.

Vazirani, V. V. 2003. *Approximation Algorithms*. Springer.

Weigt, M., and Zhou, H. 2006. Message passing for vertex covers. *Physical Review E* 74(4):046110.

Xu, H.; Kumar, T. K. S.; and Koenig, S. 2016. 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*, 392–405.

Xu, H.; Kumar, T. K. S.; and Koenig, S. 2017. A linear-time and linear-space algorithm for the minimum vertex cover problem on giant graphs. In *the International Symposium on Combinatorial Search*, 173–174.

Yamaguchi, K., and Masuda, S. 2008. A new exact algorithm for the maximum weight clique problem. In *the International Technical Conference on Circuits/Systems, Computers and Communications*, 317–320.