B-03

最小重み頂点被覆問題に対する線形時間の発見的手法の提案

A linear-time heuristic for the weighted vertex cover problem

田中 智之 † 山口 一章 † 斎藤 寿樹 † 増田 澄男 †
Tomoyuki Tanaka Kazuaki Yamaguchi Toshiki Saitoh Sumio Masuda

1 まえがき

無向グラフ $G=(V,E)$ において，任意の辺の少なくとも一つの端点が V の部分集合 S に含まれるとき，S は頂点被覆と呼ばれる．G の各頂点に重み $w(\cdot)$ が与えられているとき，V の部分集合 V' に対し $w(V')=\sum_{v\in V'} w(v)$ と定める．無向グラフ G と頂点の重み $w(\cdot)$ が与えられたときに $w(C)$ の値ができるだけ小さな頂点被覆 C を求める問題は最小重み頂点被覆問題と呼ばれる．以降，無向グラフ $G=(V,E)$ に対し，頂点数を n，辺数を m，頂点 v の隣接頂点集合と次数をそれぞれ $N(v), deg(v)$ と記す．

無向グラフと自然数 K が与えられたときに要素数 K 以下の頂点被覆が存在するか否かを判定する問題は頂点被覆問題と呼ばれており，古くから研究が行われている．頂点被覆問題は NP 完全性が証明されている [2]．また，最小重み頂点被覆問題については，P=NP でない限り，任意の $\varepsilon>0$ に対し，$(2-\varepsilon)$–近似アルゴリズムが存在しないことが証明されている [3]．一方，最小重み頂点被覆問題に対する近似アルゴリズムはいくつか提案されている．文献 [1] では時間計算量が $O(m+n)$ の 2-近似アルゴリズムが提案されている（以降，L2A 法と呼ぶ）．また，文献 [4] では，解の 2-近似性を保証しつつ，かつ平均的に良好な解が得られることを目指した方法（以降，CLA 法と呼ぶ）が提案されている．CLA 法の時間計算量は $O(m\log n)$ である．

本研究では，最小重み頂点被覆問題に対する線形時間の発見的手法を提案する．計算機実験により，多くの場合に CLA 法よりも短い計算時間で良い解が得られることを示す．本稿の構成を以下に示す．2 節では提案法の詳細を説明する．3 節で計算機実験による性能比較を行う．4 節で本論文のまとめと今後の課題について述べる．

2 提案法

無向グラフ $G=(V,E)$ において，V の部分集合 S の任意の二頂点が隣接していないとき，S は独立頂点集合と呼ばれる．G の任意の独立頂点集合 S に対し，$V-S$ は頂点被覆となる．また，C が頂点被覆ならば，$V-C$ は独立頂点集合となる．この性質から，できるだけ重みの大きな独立頂点集合を求める問題（以降，最大重み独立頂点問題と呼ぶ）と最小重み頂点被覆問題は同等な問題であることは容易に示せる．提案法では，以下の手順のように，最大重み独立頂点集合問題に対する解を求めることで最小重み頂点被覆問題の解を求める．

1. 頂点をおおよそ重みの降順に並べる（2.1 節）．
2. 前段階で得られた頂点の並びを並べ替える（2.2 節）．
3. 独立頂点集合 S を求め，$V-S$ を出力する（2.3 節）．

以下，各手順の詳細を説明する．

2.1 重みによる頂点の並べ替え

最大重み独立頂点集合に対する貪欲算法として，重みの大きい頂点を優先して選び，重みが同じ頂点については次数が小さい方を優先するという方法が考えられる．頂点を重みの降順にソートすればこの貪欲算法が実行できるが，ソートの時間計算量は $O(n\log n)$ である．提案法ではソートする代わりに下記のような性質を持つように頂点を並べることにする（各頂点に名前 $v_1, v_2, \ldots, v_n$ と名前を付ける）．

> $2\le i\le n$ なる i に対し，下記 1. または 2. のいずれかが成り立つ．ただし，$j=\lfloor (i/2)\rfloor$ である．
>
> 1. $w(v_j)>w(v_i)$
> 2. $w(v_j)=w(v_i)$ かつ $deg(v_j)\le deg(v_i)$

上記のように定めた $v_1, v_2, \ldots, v_n$ は，v_j を v_i の親とみたとき，v_1 が重み最大の根であるような 2 分ヒープとみなせる（重みが同じときは次数を考慮）．つまり，2 分ヒープを構成するのと同じ方法が使えるので，$O(n)$ 時間で実行可能である．

2.2 上界計算に基づく並べ替え

2.1 節では重みによって頂点を並べたが，辺の接続関係などを考慮しておらず，並び順が良い処理順序になっているとは限らない．本節では， より良い並び順を目指した並べ替えを行う Algorithm1 を示す．アルゴリズム中の k は自然数のパラメータであるが，その定め方

†神戸大学，Kobe University

については後で述べる．以降，表記を簡略化するため，$V[i,j]=\{v_i,v_{i+1},v_{i+2},\ldots,v_{j-1},v_j\}$ と定める．頂点集合 $V'\subseteq V$ による G の頂点誘導部分グラフを $G(V')$ と記す．

Algorithm1 の中に現れる $a(i,v)$ の定義は以下の通りである．

$$a(i,v)=w(v)+\max_{v_q\in V[i+1,n]\backslash N(v)} a(q,v_q)$$

ただし，$V[i+1,n]\backslash N(v)=\emptyset$ のときは $a(i,v)=w(v)$ とする．

Algorithm1

頂点を並べ替えるアルゴリズム

INPUT: An undirected graph $G=(V,E)$
OUTPUT: A new ordering of vertices
1: $a(n,v_i)\leftarrow w(v_i)$
　　for $i=n-k+1,n-k+2,\ldots,n$
2: for $i=n$ downto 2 do
3:　$h\leftarrow\max\{i-k+1,1\}$
4:　Find a vertex v_j from $\{v_h,v_{h+1},\ldots,v_i\}$
　　such that the value $a(i,v_j)$ is minimum
5:　Swap the vertices v_i and v_j
6:　Calculate $a(i-1,v_h),a(i-1,v_{h+1}),\ldots,$
　　$a(i-1,v_{i-1})$
7:　Sort $v_i,v_{i+1},\ldots,v_n$ such that
　　$a(i,v_i)\geq a(i+1,v_{i+1})\geq\cdots\geq a(n,v_n)$
8:　Calculate $a(i-1,v_{h-1})$
9: end for

$a(i,v)$ については以下の性質が成り立つ．

Lemma 1. $v\in V[1,i]$ なる頂点 v に対し，頂点誘導部分グラフ $G(V[i+1,n]\backslash N(v))$ の最大重み独立頂点集合を $S(i,v)$ とする．このとき，$w(S(i,v))\leq a(i,v)-w(v)$ が成り立つ．

Proof. i に関する帰納法で証明する．まず，$i=n$ のとき $V[i+1,n]\backslash N(v)$ は空集合なので $S(i,v)=\emptyset$ となり，$w(S(i,v))\leq a(i,v)-w(v)$ が成り立つのは明らか．

次に，$u\in V[1,i+1]$ なる u に対し $w(S(i+1,u))\leq a(i+1,u)-w(u)$ が成り立つと仮定したときに $v\in V[1,i]$ なる v に対し $w(S(i,v_i))\leq a(i,v_i)-w(v)$ が成り立つことを示す．以下では，$v_{i+1}\notin N(v)$ の場合と $v_{i+1}\in N(v)$ の場合に分けて考える．

$v_{i+1}\notin N(v)$ のとき，

$$a(i,v)=\max\{a(i+1,v),a(i+1,v_{i+1})+w(v)\}$$

である．$G(N[i+2,n]\backslash(N(v)\cup N(v_{i+1})))$ における最大重み独立頂点集合を S' とすると，

$$w(S(i,v))=\max\{w(S(i+1,v)),w(S')+w(v_{i+1})\}$$

と書くことができる．帰納法の仮定より

$$w(S(i+1,v))\leq a(i+1,v)-w(v)$$

が得られ，また，帰納法の仮定と $a(\cdot,\cdot)$ の定義より

$$\begin{aligned} w(S')+w(v_{i+1}) &\leq w(S(i+1,v_{i+1}))+w(v_{i+1})\\ &\leq a(i+1,v_{i+1})\\ &\leq a(i,v)-w(v)\end{aligned}$$

が得られるので，

$$w(S(i,v))\leq a(i,v)-w(v)$$

が導かれる．

$v_{i+1}\in N(v)$ のときは $a(i,v)=a(i+1,v)$ かつ $w(S(i,v))=w(S(i+1,v))$ なので帰納法の仮定より $w(S(i,v))\leq a(i,v)-w(v)$ は明らか． □

Algorithm1 は手順 4 において $a(i,\cdot)$ の小さな頂点，すなわち「選ぶと得られる独立頂点の重みが小さくなりそうな頂点」を選ぶ．つまり，Algorithm1 は，最大重み独立頂点集合の解に含まれる可能性が低いと思われる頂点を系列の後ろに置くことで，Algorithm 2 で得られる独立頂点集合の重みを大きくしようとしている．

以下では Algorithm 1 の時間計算量の評価を行う．手順 1 と手順 4 は $O(k)$ 時間で実行できる．手順 6 では各 $v_j\in V[h,i-1]$ について $a(i-1,v_j)$ の計算を行っているが，$a(i,v_j)$ の値はこの時点では計算済みであり，$v_j\in N(v_i)$ のときは

$$a(i-1,v_j)\leftarrow a(i,v_j) \qquad (1)$$

とし，$v_j\notin N(v_i)$ のときは

$$a(i-1,v_j)\leftarrow\max\{a(i,v_j),w(v)+a(v_i)\} \qquad (2)$$

とすれば良い．$a(i,v_j)$ については古い情報は不要であり，i が小さくなり新たな値を求めれば上書きしても構わない．各 v_j について v_i と隣接するか否かを判定するのは以下の方法で行う．

1. 長さ n の整数配列を用意し，手順 2 の for ループに入る前に全ての個所に -1 を代入しておく．
2. 手順 6 において，まず v_i に隣接する全ての頂点に対応する個所に v_i の識別番号（頂点ごとに異なる値）を代入する．
3. 手順 6 の各 v_j に対する $a(i-1,v_j)$ の計算では，配列中の v_j に該当する個所に v_i の識別番号が書かれていない場合のみ代入文 (2) を実行する．

上記において 2 番目の処理の時間計算量は $O(deg(v_i))$ であり，3 番目の処理は各 v_j に対し $O(1)$ 時間である．結局，手順 6 の処理は $O(deg(v_i)+k)$ 時間で実行できる．手順 7 では頂点の並べ替えを行うが，$V[i+1,n]$ については既にソート済みであり，v_i を適切な場所に挿入するだけである．$v_{i+1}, v_{i+2}, \ldots$ と順に見ていき，v_i に隣接しない頂点が現れたとき，その頂点を v_q とする．$a(i,\cdot)$ の定義より $a(i,v_i) \geq a(q,v_q)+w(v_i)$ が成り立つので，v_i が v_q より後ろに挿入されることはない．よって，$v_{i+1}, v_{i+2}, \ldots$ と順に見ていけば挿入場所は高々 $deg(v_i)$ 回の処理で見つかるので，手順 7 は $O(deg(v_i))$ 時間で実行可能である．手順 8 では $V[i,n]\backslash N(v_{h-1})$ の頂点の中で $a(\cdot,\cdot)$ が最大のものを探す必要がある．$v_i, v_{i+1}, \ldots, v_n$ は $a(\cdot,\cdot)$ に関し降順にソートされているので，$v_i, v_{i+1}, \ldots$ と順に見ていき，v_{h-1} に隣接しない最初の頂点 v_q に対し $a(q,v_q)+w(v_{h-1})$ を計算すれば良い．この処理は，手順 7 と同様，高々 $deg(v_{h-1})$ 個の隣接頂点を見るので，この時間計算量は $O(deg(v_{h-1}))$ である．

以上より，for ループ内の各処理は $O(deg(\cdot)+k)$ 時間で実行可能である．よって，Algorithm 1 の時間計算量は $O(m+kn)$ である．$k = O(m/n)$ と定めることにより，Algorithm 1 の時間計算量は $O(m+n)$ となる．

2.3 独立頂点集合を求める

以下に示す Algorithm 2 は極大な独立頂点集合を求める．アルゴリズムの実行中，長さ n の配列によって $\cup_{v\in S}N(v)$ の値を管理しておけば手順 3 の判定が $O(1)$ 時間で行える．手順 4 で新たな頂点が v_i に加えられる度に上記の集合を更新する処理を行うが，それは $O(deg(v_i))$ 時間で実行できる．よって，Algorithm 2 は $O(m+n)$ 時間で実行可能である．

Algorithm2

極大な独立頂点集合を求めるアルゴリズム
INPUT: An undirected graph $G=(V,E)$,
　　where $V=\{v_1, v_2, \ldots, v_n\}$
OUTPUT: An maximal independent set S
1: $S \leftarrow \emptyset$
2: for $i = 1$ to n do
3:　　if $S \cap N(v_i) = \emptyset$ then
4:　　　　$S \leftarrow S \cup \{v_i\}$
5:　　end if
6: end for
7: return S

Algorithm 1 と Algorithm 2 の時間計算量はいずれも $O(m+n)$ であるが，実時間は Algorithm 2 の方がずっと短い．Algorithm 1 で得られた頂点の並び順に対し一度だけ Algorithm 2 を実行する方法でもそれなりに良い解が得られるが，

- $v_1, v_2, \ldots, v_n$ という並びに対し Algorithm 2 を実行する．
- $v_2, v_3, \ldots, v_n, v_1$ という並びに対し Algorithm 2 を実行する．
- $v_3, v_4, \ldots, v_n, v_1, v_2$ という並びに対し Algorithm 2 を実行する．
  …
- $v_p, v_{p+1}, \ldots, v_n, v_1, \ldots, v_{p-1}$ という並びに対し Algorithm 2 を実行する．

のように p 個の独立頂点集合を作り最も良いものを選ぶことにする．

3 計算機実験

ランダムに作成したグラフを用いて比較実験を行う．用いたグラフの頂点数は 500, 1000, 2000 の三通り，辺密度（$= \frac{2m}{n(n-1)}$）は 0.1, 0.3, 0.5, 0.7 とし，それぞれの条件のグラフを 100 個作成する．頂点の重みは 1 から 10 の乱数とした．実験環境は次の通りである．CPU は AMD Phenom(tm) II X4 945 Processor，メモリは 1.7 GB，使用した言語は JAVA である．

提案法には k, p の二つのパラメータがあるが，まず最初に $p=1$，$k = \lfloor 2m/n \rfloor$ として実験を行った．解の重みは良い値であったが，大規模なグラフにおいては計算時間が非常に長かった．そこで，k の値を少しずつ小さくして再度実験を行った．k を小さくすれば計算時間は短くなるが，グラフの頂点数や辺密度に関わらず $k=50$ ぐらいまでは解の質はあまり変わらず，それよりも小さくすると解が悪くなった．よって提案法では入力に関わらず一律に $k=50$ とした．その後，p を様々な値にして実験を行った．p を大きくすれば解が良くなったが，p を 40 よりも大きくしても解がほとんど変わらなくなった．よって，$p=40$ とした．

提案法の Algorithm 1 の効果を確認するため，Algorithm 1 の処理を除いたものについても実験を行った．実験結果の表中では Heap と記す．従来法として CLA 法と L2A 法を紹介したが，L2A 法については，実計算時間が非常に長く，かつ解の質が悪かったため，比較実験の表には載せていない．L2A 法の時間計算量は $O(m+n)$ であるが，L2A 法は各辺に対して行われる処理が他の手法に比べて多いので，そのため実際の計算時間が長くなった．

CLA 法，Heap と提案法の $p=1$ と $p=40$ に対して行った実験での，解の重みの 100 回の平均値を表 1 に示

す．解の重みの標準偏差は，どの手法においても頂点数 500 のとき 62 ～ 67 程度，頂点数 1000 のとき 83 ～ 88 程度，頂点数 2000 のとき 121 ～ 126 程度で，辺密度による差はほとんどなかった．提案法（$p = 40$）はどの入力においても CLA 法や Heap に比べて重みが小さかった．辺密度が低いときは CLA 法と提案法の差はさほど大きくないが，辺密度が高いときはその差が大きくなった．Heap は，辺密度が高いときは提案法にかなり近い値を出しているが，辺密度が低い時は差が大きくなった．以上より，辺密度が低いときに Algorithm 1 の効果が高くなることが分かった．$p = 1$ と $p = 40$ では重みに大きな差があり，複数個の解を生成することの効果は大きかったといえる．

表 2 に各手法の計算時間を示す．計算時間は Heap が最も短く，次に提案法で，CLA 法がこの中では最も長く時間がかかった．提案法においては $p = 1$ と $p = 40$ の実行時間の差はさほど大きくなく，Algorithm 1 の実行時間が大半を占めていることが分かる．

提案法（$p = 40$）と CLA 法を 100 個のグラフに対して実行したときにより良い解を出力した回数を表 3 に示す．この表より，頂点数が多いほど，また，辺密度が高いほど，提案法は非常に多くの場合に CLA 法よりも真に優れた解を短時間で出力していることが分かる．

$p = 40$ としたときの提案法と Heap を 100 個のグラフに対して実行したときにより良い解を出力した回数を表 4 に示す．提案法の方が Heap よりも良い解を出す頻度は高いが，Heap も比較的良い解を出すことが多いことも分かる．Heap は計算時間が短いので，提案法と Heap を両方実行し，評価値の良い解を出力するというような方法による解の改良が可能であるといえる．

4 まとめと今後の課題

本稿では，重み付き最小頂点被覆問題の解を線形時間で求める発見的手法を提案した．計算機実験により，提案法の解の質は CLA 法より優れており，特に頂点数が多いほど，また，辺密度が高いほど，提案法は非常に多くの場合に CLA 法よりも真に優れた解を出力していることを確認した．また，提案法は L2A 法よりも実計算時間が短いことも確認した．

我々は現時点では提案法が何らかの近似率を保証しているのかどうか確認できていない．提案法の振る舞いを理論的に解析し，近似精度保証の有無を確認することが今後の課題といえる．また，本稿で提案した複数の解を作る方法は，頂点の系列の単純なローテーションを行うだけの単純な方法であったが，より良い方法について検討することも今後の課題として挙げられる．

参考文献

[1] R. Bar-Yehuda and S. Even, “A linear-time approximation algorithm for the weighted vertex cover problem,” Journal of Algorithms, vol.2, pp.198–203, 1981.

[2] M.R. Garey, and D.S. Johnson, Computers and Intractability — A Guide to the Theory of NP-completeness, W.H.Freeman, 1979.

[3] S. Khot and O. Regev, “Vertex cover might be hard to approximate to within $2-\varepsilon$,” 18th Annual IEEE Conference on Computational Complexity (CCC ’03), pp.379–386, 2003.

[4] S. Taoka, D. Takahuji and T. Watanabe, “Computing-Based Performance Analysis of Approximation Algorithms for the Minimum Weight Vertex Cover Problem of Graphs,” IEICE Transactions on Fundamentals, vol.E96-A, no.6, pp.1331–1339, 2013.

表 1: 頂点被覆の重み

頂点数	辺密度	CLA 法	Heap		提案法	
			$p = 1$	$p = 40$	$p = 1$	$p = 40$
500	0.1	2414.16	2442.77	2421.47	2428.17	2411.43
	0.3	2608.04	2617.37	2602.20	2613.72	2598.24
	0.5	2661.11	2666.39	2653.08	2664.74	2652.27
	0.7	2688.54	2693.20	2681.89	2691.38	2681.11
1000	0.1	5100.77	5133.16	5104.44	5124.79	5099.43
	0.3	5344.04	5353.16	5334.25	5349.91	5332.71
	0.5	5405.84	5412.00	5397.36	5410.65	5395.58
	0.7	5438.30	5441.65	5429.57	5439.52	5428.86
2000	0.1	10519.72	10550.07	10518.99	10547.65	10513.70
	0.3	10807.14	10814.25	10794.81	10812.52	10794.08
	0.5	10878.91	10882.73	10868.50	10882.00	10867.44
	0.7	10914.52	10917.34	10905.14	10916.42	10904.89

表 2: 100 回平均の実行時間 [ms]

頂点数	辺密度	CLA 法	Heap		提案法	
			$p = 1$	$p = 40$	$p = 1$	$p = 40$
500	0.1	0.59	0.05	0.26	0.49	0.71
	0.3	1.50	0.05	0.27	0.71	0.95
	0.5	2.46	0.09	0.31	0.90	1.13
	0.7	3.39	0.08	0.31	1.06	1.26
1000	0.1	2.16	0.13	0.61	1.18	1.66
	0.3	5.89	0.10	0.58	1.98	2.46
	0.5	9.69	0.33	0.82	2.86	3.33
	0.7	13.25	0.25	0.69	3.45	3.88
2000	0.1	8.12	0.21	1.24	3.04	4.07
	0.3	22.65	0.11	1.17	5.85	6.91
	0.5	38.22	0.81	1.82	9.80	10.81
	0.7	52.88	0.78	1.76	12.68	13.82

表 3: 提案法（$p = 40$）と CLA の比較 (より良い解が出た回数)

頂点数	辺密度	CLA 法	提案法	同じ
500	0.1	47	51	2
	0.3	13	84	3
	0.5	14	81	5
	0.7	41	58	1
1000	0.1	44	53	3
	0.3	10	86	4
	0.5	12	85	3
	0.7	6	92	2
2000	0.1	42	58	0
	0.3	12	88	0
	0.5	6	93	1
	0.7	2	96	2

表 4: $p = 40$ での提案法と Heap の比較 (より良い解が出た回数)

頂点数	辺密度	Heap	提案法	同じ
500	0.1	21	76	3
	0.3	29	69	2
	0.5	38	51	11
	0.7	36	50	14
1000	0.1	32	65	3
	0.3	37	58	5
	0.5	33	59	8
	0.7	34	52	14
2000	0.1	34	60	6
	0.3	46	51	3
	0.5	40	51	9
	0.7	40	47	13