A $2k$-Kernelization Algorithm for Vertex Cover Based on Crown Decomposition

Wenjun Li[a], Binhai Zhu[b,*]

[a] *Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation, Changsha University of Science and Technology, Changsha, China.*
[b] *Gianforte School of Computing, Montana State University, Bozeman, MT 59717-3880, USA.*

Abstract

We revisit crown decomposition for the Vertex Cover problem by giving a simple $2k$-kernelization algorithm. Previously, a $2k$ kernel was known but it was computed using both crown decomposition and linear programming; moreover, with crown decomposition alone only a $3k$ kernel was known. Our refined crown decomposition carries some extra property and could be used for some other related problems.

Keywords: Vertex cover, Crown decomposition, Kernelization, FPT algorithms, NP-completeness

1. Introduction

Vertex Cover is a classic NP-complete problem which has been used to model conflicts in many applications [8, 9]. Due to its importance, a lot of research has been done on it and in this paper we focus on the parameterized version of the problem. The problem is defined as follows.

> Vertex Cover
> **Given:** A simple undirected graph $G = (V, E)$, and a positive integer k;
> **Parameter:** k;

*Corresponding Author.
Email addresses: lwjscu@163.com (Wenjun Li), bhz@montana.edu (Binhai Zhu)

Question: Decide if there is a subset $V' \subseteq V$ with $|V'| \leq k$ such that for any edge $\langle u, v\rangle \in E$ at least one of u, v is in V'.

For a parameterized problem (Π, k), we say that (Π, k) is *Fixed-Parameter Tractable* (FPT) if it can be solved in $O(f(k)n^c) = O^*(f(k))$ time, where n is the input length, c is a fixed constant and $f(-)$ is some computable function. (Up to now, the best FPT algorithm for Vertex Cover runs in $O^*(1.2738^k)$ time [4].) (Π, k) admits a *kernel* Π' if a polynomial time algorithm $\mathcal{A}$ can convert (Π, k) to an instance (Π', k') such that (1) (Π, k) is a Yes-instance if and only if (Π', k') is a Yes-instance; (2) $|\Pi'| \leq |\Pi|, k' \leq k$; and (3) $|\Pi'| \leq g(k)$ where $g(-)$ is some function. We also say that Π has a kernel of size $g(k)$. And if $g(-)$ is a polynomial function, then we say Π has a polynomial kernel. (More information on FPT algorithms can be found in [5, 7].) In this paper, we focus further on the kernelization of the Vertex Cover problem.

It is known that with the famous technique of crown decomposition, together with linear programming, one can compute a $2k$ kernel for Vertex Cover; moreover, with crown decomposition alone, a $3k$ kernel can be computed [3]. We believe that the reason why a $2k$ kernel cannot be computed only using crown decomposition is that the technique in [3] cannot induce (or, enumerate) all the crown structures (—hence linear programming must be used). In this paper, we explore Vertex Cover along this direction to obtain a $2k$ kernel with a (refined) crown decomposition. The idea is to find and delete all crowns so that the reduced graph is composed of a disjoint set of odd cycles and a subgraph admitting a perfect matching — which could induce all the crown structures. The technique might also be used to solve other problems related to Vertex Cover, like P_2-packing.

The paper is organized as follows. In Section 2 we give necessary definitions on graphs and crown decomposition. In Section 3 we give the algorithm together with the analysis and proofs. In Section 4 we conclude the paper.

2. Preliminaries

2.1. Graph Basics

Let $G = (V, E)$ be a simple undirected graph; moreover, let G contain no isolated vertices. For $u \in V$, let $N(u)$ be the neighboring vertices of u, i.e., $N(u) = \{v|(u, v) \in E\}$. For a subset $V' \subseteq V$, $N(V') = \cup_{u \in V'} N(u) \backslash V'$. For a subgraph H of G, we use $V(H)$ to denote the set of vertices of H. A matching M of G is a subset of pairwise disjoint edges of E, where $V(M)$

is the set of vertices (endpoints) of the edges in M. It is well known that if $V \backslash V(M)$ is an independent set, then M is a maximal matching. If there does not exist a matching larger than M, then M is a maximum matching of G. An M-alternating cycle is one whose edges can be arranged sequentially so that the edges in M appear alternatively on the cycle. Finally, for $u \in V(M)$, if $\langle u, v \rangle \in M$ then define $N_M(u) = v$.

2.2. Crown Decomposition

Definition 1. *Given a graph $G = (V, E)$ with no isolated vertices, if V can be decomposed into three components I, H and R such that the following conditions hold:*

- *I is an independent set,*
- *$N(I) = H$ (there is no edge between I and R), and*
- *there is a matching M for $G[I \cup H]$ saturating (i.e., covering all the vertices in) H;*

then (I, H) is called a crown *of G. $|H|$ is called the* width *of the crown (I, H).*

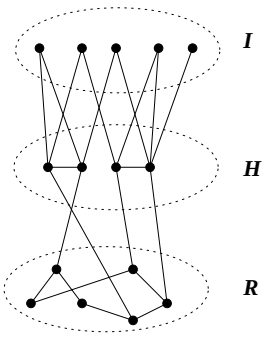

Figure 1: A crown of width 4.

An example of a crown with width 4 is given in Figure 1. The following lemma is well-known regarding crown decomposition [6]:

Lemma 1. *For Vertex Cover, given $G = (V, E)$ and a crown decomposition (I, H, R) of G, there is a vertex cover of size k for G if and only if the induced subgraph $G' = G[V \backslash (I \cup H)]$ has a vertex cover of size $k' = k - |H|$.*

Intuitively, this lemma implies that it is possible to compute an optimal vertex cover by putting H in the solution, delete H from G, recompute the crown decomposition for $G[V \backslash (I \cup H)]$ and repeat the process on $G[V \backslash (I \cup H)]$. Note that given I and $N(I)$ with $|N(I)| \leq |I|$, it is easy to compute a crown of G by starting with the maximum matching between I and $N(I)$ and gradually building up the crown (I, H) [6].

We note that crown decomposition is closely related to (but different from) the Nemhauser-Trotter theorem on Vertex Cover [10]. Besides Vertex Cover, the technique has been applied on d-hitting set, P_2-packing and r-set packing (and the special case — triangle packing) [1, 2, 11, 12].

3. The Algorithm and Its Analysis

3.1. The General Idea

The idea of the algorithm is as follows. When computing and deleting all crowns (H's), we make sure that the reduced graph is composed of a set of vertex-disjoint odd cycles and a subgraph admitting a perfect matching. This is done by first computing a maximum matching M of the graph. Let CY be the set of such odd cycles which is initially empty. Then, starting from a vertex v not in $V(M) \cup V(CY)$ we decompose the vertices of V into I_0 $(= \{v\}), H_0, I_1, H_1, \cdots, I_i, H_i, \cdots$. Finally, we try to find M-alternating odd cycles when there is an edge between nodes in I_i or when there is a matching edge in M between the nodes in H_i. Such an odd cycle cy will be identified and the procedure will be run recursively on the 'reduced' graph (with the odd cycles left intact and the corresponding maximum matching updated). When H_i is empty, the algorithm computes the crown (I, H), with $I = \bigcup_{j=0}^{i} I_j$ and $H = \bigcup_{j=0}^{i-1} H_j$. Then it reduces the graph by putting H in the solution, deleting $I \cup H$ from the graph, and updating M and CY; finally, it makes a further recursive call on the reduced graph.

3.2. The Algorithm

We now present the detailed algorithm as follows. The main steps in the recursive procedure Find-CROWN( ) are Step 5.1 (when there is an edge in M between two vertices in H_i) and Step 5.3 (when there is an edge between two nodes in I_i). The matching M and the set of odd cycles CY are then updated accordingly before the next round of recursive calls. Step 5 will only terminate under the condition $H_i = \emptyset$.

VC-KERNEL$(G=(V,E),k)$
1. Let CY be a set of M-alternating odd cycles. Initially $CY=\emptyset$.
2. Find a maximum matching M of G.
3. **Find-CROWN**(G,M,CY,k)

Find-CROWN(G,M,CY,k)
1. Delete all the isolated vertices from G.
2. **If** $V=V(CY)\cup V(M)$, **then return** (G,k);
3. Pick a vertex $v\in V\backslash(V(CY)\cup V(M))$ arbitrarily;
4. Let $I_0=\{v\}$, $H_0=N(I_0)$, $i=0$;
5. **While** $(H_i\neq\emptyset)$ {
5.1 **If** (there is an edge $e=\langle u_i,w_i\rangle\in M$ in $G[H_i]$) **then** {
Let $q=i$;
While (there are different neighbors u'_q,w'_q of u_q,w_q in I_q respectively)
$\{u_{q-1}=N_M(u'_q),w_{q-1}=N_M(w'_q),q=q-1;\}$;
Assume $\{x_q\}=N(u_q)\cap I_q=N(w_q)\cap I_q$, then $cy=\langle x_q,u_q,N_M(u_q),\ldots,N_M(u_{i-1}),$
$u_i,w_i,N_M(w_{i-1}),\ldots,N_M(w_q),w_q,x_q\rangle$ is an M-alternating odd cycle;
While $(q\neq 0)$
$\{M=M\backslash\{\langle x_q,N_M(x_q)\rangle\}\cup\{\langle N_M(x_q),x_{q-1}\rangle\}$, where
$x_{q-1}\in I_{q-1}\cap N(N_M(x_q))$; $q=q-1;\}$
Return Find-CROWN$(G,M\backslash V(cy),CY\cup\{cy\},k);\}$
5.2 **else** $\{I_{i+1}=N_M(H_i)$, $i=i+1;\}$;
5.3 **If** (there is an edge $e=\langle u_i,w_i\rangle$ in $G[I_i]$) **then** {
Let $q=i$;
While (there are different neighbors u_{q-1},w_{q-1} of $N_M(u_q),N_M(w_q)$
in I_{q-1} respectively)
$q=q-1$;
Assume $\{x_q\}=N(N_M(u_q))\cap I_{q-1}=N(N_M(w_q))\cap I_{q-1}$, then
$cy=\langle x_q,N_M(u_q),u_q,\ldots,N_M(u_i),u_i,w_i,N_M(w_i),\ldots,w_q,N_M(w_q),x_q\rangle$
is an M-alternating odd cycle;
While $(q>1)$
$\{M=M\backslash\{\langle x_q,N_M(x_q)\rangle\}\cup\{\langle N_M(x_q),x_{q-1}\rangle\}$,
where $x_{q-1}\in I_{q-2}\cap N(N_M(x_q))$; $q=q-1;\}$;
Return Find-CROWN$(G,M\backslash V(cy),CY\cup\{cy\},k);\}$
5.4 **else** $H_i=N(I_i)\backslash\bigcup_{j=0}^{i-1}H_j$; }
6. **Return Find-CROWN**$(G\backslash(I\cup H),M\backslash(I\cup H),CY,k-|H|)$, where
$H=\bigcup_{j=0}^{i-1}H_j$ and $I=\bigcup_{j=0}^{i}I_j$ form a crown.

3.3. Correctness

The correctness of the algorithm hinges on Step 5, where an M-alternating odd cycle is computed and excluded from the recursive calls at the same step. There are two cases (5.1 and 5.3, shown in Figure 2 and 3 respectively), covering the situation when there is an edge in M in $G[H_i]$ and when there is an edge in $G[I_i]$ respectively. Note that when the odd cycle is identified, M is updated accordingly, making sure that it is still a maximum matching in the 'reduced' graph not including those odd cycles found. Starting with a vertex v not in an odd cycle and not in the matching M, this procedure is run recursively until H_i is empty for some i. Then a crown (I, H) is found, with $I = \bigcup_{j=0}^{i} I_j$ and $H = \bigcup_{j=0}^{i-1} H_j$. (Note that at Step 5.4, $N(I_i)$ cannot contain a vertex which is in $V(CY)$ — as long as CY is not empty. This property is important for the correctness of the algorithm, as once an odd cycle *cy* is identified it will remain intact and the subsequent recursive calls will not touch it. We prove this separately as a lemma.) At Step 6, when H is deleted from the graph, the algorithm will run recursively on the reduced graph (starting possibly from a different v).

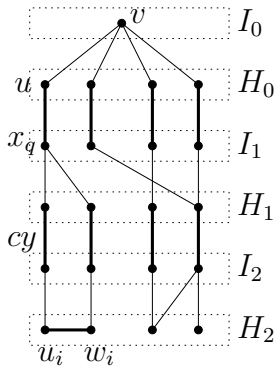

Figure 2: Illustration for case 5.1 (with $i = 2$), where bold edges are edges in the matching M. Here *cy* is the low-left cycle. $N_M(x_q) = u$ and $\langle u, v\rangle$ will be swapped with the edge $\langle x_q, u\rangle$ in the (initial) matching M.

Lemma 2. *In the algorithm* **Find_Crown**, *after some odd cycle cy is identified at Step 5.3, in the subsequent recursive calls $N(I_i)$ (at Step 5.4) cannot contain any node of the cycle cy.*

Proof. We prove this lemma by contradiction. Assume that after some odd cycle *cy* is identified, a recursive call of **Find_Crown** selects some node w not

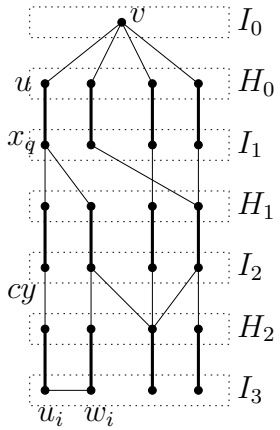

Figure 3: Illustration for case 5.3 (with $i = 3$), where bold edges are edges in the matching M. Here cy is the low-left cycle. $N_M(x_q) = u$ and $\langle u, v\rangle$ will be swapped with the edge $\langle x_q, u\rangle$ in the (initial) matching M.

in $V(CY) \cup V(M)$; moreover, for some node $a \in I_i$ (computed at Step 5.4) it connects to $b \in V(cy)$. Then, following the odd path from such a vertex b to w, we could update edges in the matching (by putting $\langle a, b\rangle$ in the matching, then updating the remaining ones alternatively) to have a matching whose size is larger than the maximum matching. This gives us the contradiction. (Note that the matching edges in cy can be updated accordingly, e.g., moving $\langle b, c\rangle$ out of the matching and putting $\langle c, d\rangle$ in the matching, etc. Hence the matching in the odd cycle cy will maintain its size.) □

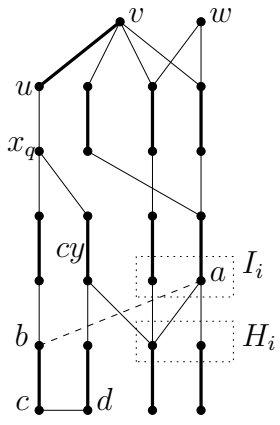

Figure 4: Illustration for the proof of Lemma 2.

3.4. Time Complexity

Let $|V| = n, |E| = m$. Computing the maximum matching takes $O(n^{2.5})$ time. In the **Find_Crown** algorithm, within Step 5, computing the neighbors of vertices takes $O(n + m)$ time and dominates the whole cost. Hence for one run (starting with a v not in $V(CY \cup V(M))$) the total running time is $O(n+m)$. As there could be $O(n)$ such v's, we could have $O(n)$ recursive calls. Hence, the total time the algorithm takes is $O(n(n + m))$, which is $O(n^3)$ in the worst case.

3.5. Kernel Size Analysis

Lemma 3. *Given any Vertex Cover instance $\langle G, k\rangle$, let $G' = \langle G', k'\rangle$ be the reduced instance returned by the algorithm* **VC-Kernel***, then G' is composed of two parts: (1) a vertex-disjoint set of odd cycles, and (2) a subgraph with a perfect matching.*

Proof. According to the algorithm, given a maximum matching M, starting with a vertex v not in $V(CY) \cup V(M)$, **FIND_CROWN** keeps finding M-alternating odd cycles and updating M accordingly. When this is done **FIND_CROWN** returns a crown at Step 6 (and deletes it from G before the next round of recursive call). The recursive algorithm terminates at Step 2, when all vertices are either in the set of M-alternating odd cycles or in the maximum matching (i.e., when such a v cannot be found). This completes the proof. □

Theorem 1. *Given any Vertex Cover instance $\langle G, k\rangle$, let $G' = \langle (V', E'), k'\rangle$ be the reduced instance returned by the algorithm VC-Kernel. If $\langle G, k\rangle$ is a Yes-instance, then $|V'| \leq 2k'$.*

Proof. By the previous lemma, G' is partitioned into two parts: (1) a disjoint set of odd cycles, and (2) a subgraph admitting a perfect matching. For any odd cycle cy, it contains $|cy|$ edges. Hence the vertex cover for the induced subgraph $G[cy]$ has size at least $(|cy| + 1)/2$. On the other hand, for a graph G_M with a perfect matching M, its vertex cover has size at least $|M|$.

Hence, the vertex cover for G' has size at least $|V'|/2$. Moreover, if $\langle G, k\rangle$ is a Yes-instance (i.e., G has a vertex cover of size k), then $\langle G', k'\rangle$ is also a Yes-instance, with $k' \geq |V'|/2$ or $|V'| \leq 2k'$. □

4. Concluding Remarks

In this paper, we give a $2k$ kernel for Vertex Cover, by only using the (refined) crown decomposition method. Previously, a $2k$ kernel is known, but the method is a combination of crown decomposition and linear programming. (With crown decomposition alone, only a $3k$ kernel is known for the problem before this work.) The method is to enforce that the reduced graph maintains some special property which could induce or enumerate all possible crown structures for the problem (in this case, Vertex Cover). We believe that similar methods could be used on some problems related to Vertex Cover, like P_2-packing.

Acknowledgments

This research is supported by National Natural Science Foundation of China under grants 61502054 and 61628027. WL is also supported by the Natural Science Foundation of Hunan Province (grant no. 2017JJ3333). We also thank the anonymous referee whose comments greatly improve the presentation of the paper.

References

[1] F. Abu-Khzam. An improved kernelization algorithm for r-Set Packing. *Information Processing Lett.*, 110(6):621-624, 2010.

[2] F. Abu-Khzam. A kernelization algorithm for d-Hitting Set. *J. of Comput. and System Sci.*, 76(7):524-531, 2010.

[3] F. Abu-Khzam, M. Fellows, M. Langston and W. Suters. Crown structures for vertex cover kernelization. *Theory Comput. Sys.*, 41(3):411-430, 2007.

[4] J. Chen, I. Kanj and G. Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736-3756, 2010.

[5] R. Downey and M. Fellows. *Parameterized complexity.* Springer New York, 1999.

[6] M. Fellows. Blow-ups, win/win's, and crown rules: some new directions in FPT. *Proc. 29th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'03)*, pp. 1-12, 2003.

[7] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[9] S. Khuller. Algorithms column: the vertex cover problem. *ACM SIGACT News*, 33(2):31-33, 2003.

[10] G. Nemhauser and L. Trotter Jr. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232-248, 1975.

[11] J. Wang, D. Ning, Q. Feng and J. Chen. An improved kernelization for P_2-packing. *Information Processing Lett.*, 110(5):188-192, 2010.

[12] Y. Yang. Towards optimal kernel for edge-disjoint triangle packing. *Information Processing Lett.*, 114(7):344-348, 2014.