Crown Structures for Vertex Cover Kernelization[*][†]

Faisal N. Abu-Khzam,[‡] Michael R. Fellows,[§] Michael A. Langston[¶] and W. Henry Suters[||]

Abstract

Crown structures in a graph are defined and shown to be useful in kernelization algorithms for the classic vertex cover problem. Two vertex cover kernelization methods are discussed. One, based on linear programming, has been in prior use and is known to produce predictable results, although it was not previously associated with crowns. The second, based on crown structures, is newer and much faster, but produces somewhat variable results. These two methods are studied and compared both theoretically and experimentally with each other and with older, more primitive kernelization algorithms. Properties of crowns and methods for identifying them are discussed. Logical connections between linear programming and crown reductions are established. It is shown that the problem of finding an induced crown-free subgraph, and the problem of finding a crown of maximum size in an arbitrary graph, are solvable in polynomial time.

1 Introduction

The computational challenge posed by $\mathcal{NP}$-hard problems has inspired the development of a wide range of algorithmic techniques. Due to the seemingly intractable nature of such problems, approaches have historically emphasized the design of polynomial-time algorithms that deliver only approximate solutions. Unfortunately, obtaining near-optimal polynomial-time approximations has proved to be equally hard for a large class of these problems. We now know, however, that a wide variety of otherwise-intractable problems are polynomial-time solvable when relevant parameters are fixed. This tenet was pivotal to the early work of Fellows and Langston on applications of well-quasi order theory [18, 19], and has lead to powerful new hierarchies of complexity classes as pioneered by Downey and Fellows [17]. Our main focus here is on the class of problems termed *fixed-parameter tractable* (henceforth FPT). Such a problem is solvable in $O(f(k)n^c)$ time, where n denotes the problem size, k denotes the parameter of interest, and c is an arbitrary constant. Observe that if a problem is FPT, then it can be solved in polynomial time for any fixed parameter value.

[*]This research has been supported in part by the U.S. National Science Foundation under grant CCR-0311500 and by the U.S. National Institutes of Health under grants 1-R01-MH-074460-01, 1-P01-DA-015027-01 and U01-AA13512.

[†]A preliminary version of a portion of this paper was presented at the ACS/IEEE International Conference on Computer Systems and Applications, held in Cairo, Egypt, in January, 2005.

[‡]Division of Computer Science and Mathematics, Lebanese American University, Chouran, Beirut, Lebanon

[§]School of Electrical Engineering and Computer Science, University of Newcastle, Calaghan NSW 2308, Australia

[¶]Department of Computer Science, University of Tennessee, Knoxville, TN 37996–3450, USA

[||]Department of Mathematics and Computer Science, Carson-Newman College, CN Box 71958, Jefferson City, TN 37760, USA

The best known example of an FPT problem is almost certainly the parameterized vertex cover problem. Given an undirected graph G of order n, we seek to determine whether there exists a set C with k or fewer vertices such that every edge of G has at least one endpoint in C. Vertex cover can be solved in $O(1.2738^k + kn)$ time [8] by employing the bounded search tree technique, which restricts the search space of the problem to a tree whose size is bounded by a function of the parameter. Vertex cover has a host of realizations in practice, and is especially well-known for its utility in computational biology. There it has been used in quantitative trait analysis [10], construction of phylogenetic trees [2], and the analysis of microarray data [5], to name just a few of its applications.

In this paper, we are primarily concerned with identifying certain *crown structures*, to be defined later, that can be used to reduce the size of both n and k. Original work on these structures can be found in [3, 12]. We shall investigate two methods for isolating crowns. One is based on linear programming, the other on direct reduction. They can be used alone or in combination. We study the properties of the crowns found by each method, as well as their relative performance on real data. We also examine connections between these methods, paralleling a similar exploration independently proposed in [11]. One of our main goals is the synthesis of a polynomial-time algorithm for decomposing an arbitrary graph into an induced crown and a crown-free subgraph.

2 Kernelization

Kernelization transforms a graph G of order n with parameter k into another graph G' of order $n' \leq n$ with parameter $k' \leq k$. The usual goal is to bound n' by some small function of k'. G' is called the *kernel*. It should have a vertex cover of size at most k' if and only if G has a vertex cover of size at most k. It is kernelization that distinguishes an arbitrary problem from one that is FPT. After kernelization, the process reverts to *branching*, an exhaustive examination of the kernel via tree search techniques. Empirical studies of branching are of related interest [4, 7].

For the purpose of comparison, we describe three vertex cover kernelization methods. The first is based on elementary preprocessing techniques. The second, which we term “LP kernelization,” reformulates vertex cover first as an integer programming problem, and then relaxed into a linear programming framework. The resultant instance can be solved either with standard linear programming techniques or with network flow algorithms such as those described in [16, 20]. The third method is the most recent, and is dubbed *crown reduction*. It is based on finding a special independent set so that both the set and its neighborhood can be efficiently removed from the graph.

2.1 Elementary Kernelization

The techniques we cover in this paper assume that input graphs have been preprocessed using a variety of rules described in previous publications. We refer the reader to [3] for a detailed study. These rules focus mainly on the elimination of low and high degree vertices. Thus, for example, we shall assume that any vertex of degree one or two has been eliminated. Similarly, we shall assume that any vertex whose degree exceeds k has been removed [6]. Such preprocessing is computationally inexpensive. All require at most $O(n^2)$ time with modest constants of proportionality. Moreover, they ensure a kernel whose size is at most quadratic in k.

Theorem 1 [1] *If G has a vertex cover of size k, and if the degree of each vertex in G lies in $[3, k]$, then $n \leq \frac{k^2}{3} + k$.*

2.2 Kernelization by Linear Programming

The optimization version of vertex cover can be stated in the following manner. Assign a value $X_u \in \{0, 1\}$ to each vertex u of the graph $G = (V, E)$ so that the following conditions hold.

(1) Minimize $\sum_u X_u$.

(2) Satisfy $X_u + X_v \geq 1$ whenever $\{u, v\} \in E$.

This is an integer programming formulation of the optimization problem. In this context the objective function is the size of the vertex cover, and the set of all feasible solutions consists of functions from V to $\{0, 1\}$ that satisfy condition (2). We relax the integer programming problem to a linear programming problem by replacing the restriction $X_u = \{0, 1\}$ with $X_u \geq 0$. The value of the objective function returned by the linear programming problem is a lower bound on the objective function returned by the related integer programming problem [20, 21, 23].

The solution to the linear programming problem can be used to simplify the related integer programming problem in the following manner. Let $N(S)$ denote the neighborhood of S, and define $P = \{u \in V | X_u > 0.5\}$, $Q = \{u \in V | X_u = 0.5\}$ and $R = \{u \in V | X_u < 0.5\}$. We employ the following modification by Khuller [21] of a theorem originally due to Nemhauser and Trotter [23].

Theorem 2 *If P, Q, and R are defined as above, then there is an optimal vertex cover that is a superset of P and that is disjoint from R.*

Proof. Let A be the set of vertices of P that are not in the optimal vertex cover and let B be the set of vertices of R that are in the optimal cover, as selected by the solution to the integer programming problem. Notice that $N(R) \subseteq P$ because of condition (2). It is not possible for $|A| < |B|$, since in this case replacing B with A in the cover decreases its size without uncovering any edges ($N(R) \subseteq P$), and so it is not optimal. Additionally it is not possible for $|A| > |B|$ because then we could gain a better linear programming solution by setting $\epsilon = \min\{X_v - 0.5 : v \in A\}$ and replacing X_u with $X_u + \epsilon$ for all $u \in B$ and replacing X_v with $X_v - \epsilon$ for all $v \in A$. Thus we must conclude that $|A| = |B|$, and in this case we can replace B with A in the vertex cover (again $N(R) \subseteq P$) to obtain the desired optimal cover. ■

The graph G' is produced by removing vertices in P and R and their adjacent edges. The problem size is $n' = n - |P| - |R|$ and the parameter size is $k' = k - |P|$. Notice that because the size of the objective function for the linear programming problem provides a lower bound on the objective function for the integer programming problem, the size of any optimal cover of G' is bounded below by $\sum_{u \in Q} X_u = 0.5|Q|$. If this were not the case, then the original linear programming procedure that produced Q would not have produced an optimal result. This allows us to observe that if $|Q| > 2k'$ then the vertex cover problem has been solved as a "no" instance.

There are several techniques that can be used to make the above linear programming procedure more practical for large problems. When dealing with large dense graphs, a direct approach may not be practical because the number of constraints is the number of edges in the graph. Therefore, the code used in this paper solves the dual of the LP problem, turning the minimization problem into a maximization problem, and making the number of constraints equal to the number of vertices [13, 15]. The asymptotic time complexity of this approach is $O(n^3)$.

A second approach solves the linear programming formulation of vertex cover by reducing a graph to a network flow problem. The algorithm in the proof of the Nemhauser-Trotter theorem defines a bipartite graph B in terms of the input graph G, finds the vertex cover of B by computing a maximum matching of B, and assigns values to the vertices of G based on the cover of B. Solving the maximum matching of B can be a accomplished by turning B into a network flow problem and solving it with Dinic's maximum flow algorithm [16]. The time complexity of this algorithm is O($m\sqrt{n}$) where n is the number of vertices in the original graph G, and m is the number of edges in G. Other methods to speed LP kernelization appear in [20].

2.3 Kernelization by Crown Reduction

A crown reduction is similar to the previous algorithms in that it attempts to use the structure of the graph to recognize two sets I and H of vertices so that there is an optimal vertex cover that does not contain any vertex from I but does contain every vertex from H. This process is based on the following definition, theorem, and algorithm.

A *crown* is an ordered pair (I, H) of subsets of vertices from a graph G that satisfies the following criteria: (1) $I \neq \emptyset$ is an independent set of G, (2) $H = N(I)$, and (3) there exists a matching M on the edges connecting I and H such that all elements of H are matched. H is called the *head* of the crown. The *width* of the crown is $|H|$. A *straight crown* is a crown (I, H) that satisfies the condition $|I| = |H|$. A *flared crown* is a crown (I, H) that satisfies the condition $|I| > |H|$. These notions are depicted in Figure 1.

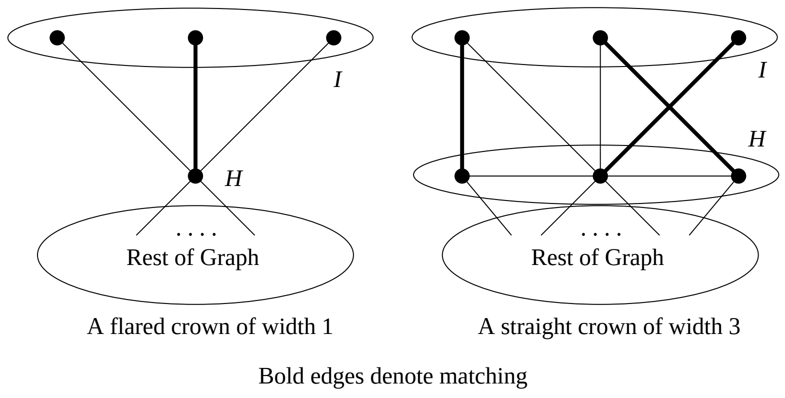
I
H
....
Rest of Graph
A flared crown of width 1
I
H
....
Rest of Graph
A straight crown of width 3
Bold edges denote matching

Figure 1: Sample crowns.

A crown that is a subgraph of another crown is called a *subcrown*. This relationship is depicted in Figure 2. Notice that if (I, H) is a crown, then I is an independent set and H is a cutset between I and the rest of the graph.

Theorem 3 *If G is a graph with a crown (I, H), then there is a vertex cover of G of minimum size that contains all the vertices in H and none of the vertices in I.*

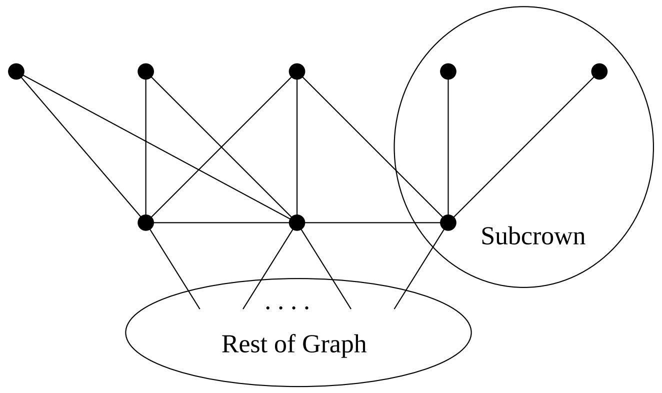

Figure 2: A flared crown of width 3 with a flared subcrown of width 1.

Proof. Since there is a matching M of the edges between I and H, any vertex cover must contain at least one vertex from each matched edge. Thus the matching will require at least $|H|$ vertices in the vertex cover. This minimum number can be realized by selecting H to be in the vertex cover. It is further noted that vertices from H can be used to cover edges that do not connect I and H, while this is not true for vertices in I. Thus, including the vertices from H does not increase, and may decrease, the size of the vertex cover as compared to including vertices from I. Therefore, there is a minimum-size vertex cover that contains all the vertices in H and none of the vertices in I. ■

The following procedure can be used to find a crown.

Step 1: Find a maximal matching M_1, and identify the set of all unmatched vertices as the set O of outsiders.
Step 2: Find a maximum auxiliary matching M_2 of the edges between O and $N(O)$.
Step 3: If every vertex in $N(O)$ is matched by M_2, then $H = N(O)$ and $I = O$ form a straight crown, and we are done.
Step 4: Let I_0 be the set of vertices in O that are unmatched by M_2.
Step 5: Repeat Steps 5a and 5b until $n = N$ so that $I_{N-1} = I_N$.
  5a. Let $H_n = N(I_n)$.
  5b. Let $I_{n+1} = I_n \cup N_{M_2}(H_n)$.
Step 6: $I = I_N$ and $H = H_N$ form a flared crown.

Theorem 4 *The algorithm just described produces a crown as long as either the set I_0 of unmatched outsiders is not empty or every vertex in $N(O)$ is matched by M_2.*

Proof. First, if every vertex in $N(O)$ is matched by M_2, then it is clear that $H = N(O)$ and $I = O$ form a straight crown. Next, consider the case where I_0 is not empty. Since M_1 is a maximal matching, the set O, and consequently its subset I, are both independent. Second, because of the definition of H, it is clear that $H = N(I_{N-1})$ and since $I = I_N = I_{N-1}$ we know that $H = N(I)$. The third condition for a crown is proved by contradiction. Suppose there were an element $h \in H$ that were unmatched by M_2. Then the construction of H would produce an augmented (alternating) path of odd length. For h to be in H there must have been an unmatched vertex in O that begins the path. Then the repeated Step 5a would always produce an edge that is not in the matching while the next Step 5b would produce an edge that is part of the matching. This process repeats until the vertex h is reached. The resulting path begins and ends with unmatched vertices and alternates between matched and unmatched edges. Such a path cannot exist if M_2 is in fact a maximum matching because we could increase the size of the matching by swapping the matched and unmatched edges along the path. Therefore every element of H must be matched by M_2. The actual matching used in the crown is the matching M_2 restricted to edges between H and I. ■

The graph G' is produced by removing vertices in I and H and their adjacent edges. The problem size is $n' = n - |I| - |H|$ and the parameter size is $k' = k - |H|$. It is important to note that if a maximum matching of size greater than k is found, then there is not a vertex cover of size at most k, and the vertex cover problem has been solved as a "no" instance. Therefore if either of the matchings M_1 and M_2 is larger than k, the process can be halted. This fact also allows us to place an upper bound on the size of the graph G'.

Theorem 5 *If both the matchings M_1 and M_2 are of size less than or equal to k, then G has at most $3k$ vertices that are not in the crown.*

Proof. Since the size of the matching M_1 is less than or equal to k, it contains at most $2k$ vertices. Thus, the set O contains at least $n - 2k$ vertices. Since M_2 is less than or equal to k, there are at most k vertices in O that are matched by M_2. Thus there are at least $n - 3k$ vertices that are in O that are unmatched by M_2. These vertices are included in I_0 and are therefore in I. Thus the largest number of vertices in G that are not included in I and H is $3k$. ■

The particular crown produced by the crown reduction depends on the maximal matching M_1 that is used in its calculation. This implies that it is desirable to perform the reduction repeatedly, using a different matching each time, in an attempt to identify as many crowns as possible and consequently to reduce the size of the kernel as much as possible. One way to try to accomplish this is with the use of randomly chosen matchings. It is also desirable to perform a preprocessing procedure after each crown reduction because the reduction may result in vertices of low degree. The most computationally expensive part of the procedure is finding the maximum matching M_2, which is done in our code by recasting the maximum matching problem on a bipartite graph as a network flow problem that is then solved using the algorithm developed by Dinic [16] with a run time bounded by $O(n^{\frac{5}{2}})$ for bipartite graphs [20].

2.4 Relative Performance

The time complexity of LP kernelization is $O(n^3)$ if a general LP package is used. When a network flow approach is used instead, it is $O(m\sqrt{n})$ where m is the number of edges in G. The time complexity of crown reduction is $O(m^*\sqrt{n^*})$, where m^* is the number of edges between O and $N(O)$ and n^* is the number of vertices in O and $N(O)$. Asymptotically, the behavior of the two methods is similar. In practice, however, m^* and n^* are generally much smaller than m and n. It is difficult to say much more theoretically. Moreover, the relative sizes of n, m^* and n^* can vary greatly. With this in mind, it is helpful to conduct experiments in which both methods are applied and their performance compared. Preliminary evidence suggests that crown reduction is probably faster than LP kernelization [3], especially on large problem instances. In light of connections we will show in the sequel, we now describe recent experiments aimed squarely on a direct comparison of LP kernelization versus crown reduction.

Both LP kernelization and crown reduction result in kernels whose sizes are linear in k. The kernel that results from LP kernelization has size at most $2k$. The kernel that results from crown reduction is (perhaps loosely) bounded above by $3k$. It should be noted that the particular crown produced by crown reduction depends on the maximal matching identified in Step 1 of the algorithm. Thus, the crown reduction may be repeated, potentially resulting in smaller and smaller kernels.

We decided to run our experiments in the context of computational biology, because a common problem in that domain requires solving clique. The classic clique problem is $W[1]$-hard [17], however, and so clique is not directly amenable to a fixed-parameter algorithmic approach. Nevertheless, it is possible to find a maximum clique in a graph, G, by finding a minimum vertex cover of the complement of the graph, $\overline{G}$. Observe that a graph G has a vertex cover of size k if and only if it's complement graph $\overline{G}$ has a clique of size $n - k$. We learn, or course, that this approach tends to work best when the complement graph is not very densely connected and its covers are not huge. One of the applications to which we have applied our codes is the problem of finding phylogenetic trees based on protein domains [7]. For this the graphs we utilized were based on data obtained from well-known open-source repositories of biological data.

We now describe how our methods can be applied to practical problems. Preprocessing with low degree and high degree rules is a first step. As we then begin to iterate crown reduction, each reduction may remove several vertices, resulting in a decrease in the degree of many remaining vertices. A reapplication of the preprocessing rules may therefore be productive. This is inexpensive and performed after each iteration of crown reduction. The resulting reductions in graph sizes and run times are included in our timing results. We repeat crown reduction until it fails to reduce the graph. As a last kernelization step, LP is applied in an attempt to determine if further reductions may be possible. After kernelization, the problem is solved with branching. For the purpose of comparison, the same initial graph is reduced again using preprocessing followed only by LP kernelization and branching, without the benefit of crown reduction.

The following computational examples are derived from the work reported in [25], and were performed on a cluster of 32 500-MHz Sun Blade 100 workstations. The results in Table 1 highlight run times on graph SH2-5 derived from protein domain data. At this point, perhaps we should interject a few words about how such graphs are derived and used. Domains are common protein subunits. They are named by convention, in this case with SH2 denoting Src Homology 2 (these domains are important because they are thought to play a key role in many tyrosine-kinase mediated signal transduction pathways). Protein sequence information is downloaded from a public source, for example, the National Center for Biotechnology Information [22]. An alignment matrix is then constructed by computing pairwise sequence alignment scores with a standard tool such as ClustalW [14]. We select a threshold, and use it as a high-pass filter to build from the alignment matrix an unweighted, undirected alignment graph. The number after the domain name, in this case 5, denotes the threshold employed. A largest clique in the alignment graph represents a noise-free collection of sequences that best characterizes an underlying protein family. To compute this, we exploit fixed-parameter tractability, and solve vertex cover on the graph's complement [4]. We determine the optimum clique size by repeatedly fixing parameter sizes and, borrowing the term from operations research and artificial intelligence, employing "iterative deepening search."

Getting back to Table 1, note that the two methods produce similar results. The method without crown reduction had fewer steps, so its total time was slightly shorter than with crown reduction. Also note that LP kernelization was successful in further reducing the graph after the application of repeated crown reductions. This is an unusual case, for reasons that will be explored later in this paper. Since there is an element of chance in the crown reduction algorithm, it is possible that if this experiment were repeated the results would be different. Observe that the failure of crown reduction to reduce a graph does not imply that LP kernelization will also fail.

Procedure	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6
Crown Reduction followed by LP Kernelization Total time 49.25	PP $k = 438$ $n = 810$ $t = 0.71$	CR $k = 415$ $n = 738$ $t = 1.39$	CR $k = 415$ $n = 733$ $t = 1.23$	CR $k = 415$ $n = 733$ $t = 1.19$	LP $k = 302$ $n = 496$ $t = 38.89$	BR cover found $t = 5.84$
LP Kernelization only Total time 51.28	PP $k = 438$ $n = 810$ $t = 0.70$	LP $k = 301$ $n = 494$ $t = 44.79$	BR cover found $t = 5.79$			
Times are given in seconds. PP indicates preprocessing, CR indicates crown reduction, LP indicates LP kernelization, and BR indicates branching.						

Table 1: Graph SH2-5.dim with $k = 450$, $n = 839$ and 26,612 edges.

In contrast to the SH2-5 graph, there are examples where crown reduction can reduce a graph further than LP kernelization. Such an example is the SH2-10 graph produced from the same data as SH2-5 with a different threshold value. The results for SH2-10 are shown in Table 2. This is an example in which not only did a sequence of crown reductions simplify the graph more than LP kernelization, crown reductions where able to solve the problem completely. This shows one of the strengths of crown reductions; it gives a relatively inexpensive method for attempting to identify “no” instances of the vertex cover problem.

An example with more typical behavior is the graph, SH3-5, derived from data gathered about the SH3 protein domain. A comparison of the performance of the crown reduction and LP kernelization is given in Table 3. Although both methods produce similar reductions in the graph, crown reduction is significantly faster. Because of the expense of performing LP kernelization, it is helpful to note that crown reductions could be used in place of LP kernelization or as a method to reduce the graph before attempting LP kernelization. In this case, the time for doing preprocessing and three crown reductions was 45.49 seconds. The LP kernelization that followed did not produce any further reduction and took 1847.62 seconds. If LP kernelization had been done without any preceding crown reductions the time required would have been 2416.33 seconds. Thus using crown reduction in place of LP kernelization can dramatically reduce run times. Using crown reduction prior to LP kernelization usually produces more modest reductions in run time. Despite the reductions to the graph, the kernel remains quite large. Due to the computational cost, branching was not attempted on this kernel.

Neither crown reduction nor LP kernelization is likely to work particularly well for very densely connected graphs. One example of this is the graph RMA-85-280. This graph has 1737

Procedure	Step 1	Step 2	Step 3	Step 4	Step 5
Crown Reduction Total time 4.66	PP $k = 340$ $n = 678$ $t = 1.07$	CR $k = 188$ $n = 471$ $t = 1.39$	CR $k = 188$ $n = 462$ $t = 1.61$	CR $k = 74$ $n = 295$ $t = 0.53$	CR no cover instance $t = 0.06$
LP Kernelization only Total time 64.11	PP $k = 340$ $n = 678$ $t = 1.06$	LP $k = 300$ $n = 573$ $t = 56.01$	BR no cover instance $t = 7.04$		

Times are given in seconds. PP indicates preprocessing, CR indicates crown reduction, LP indicates LP kernelization, and BR indicates branching.

Table 2: Graph SH2-10.dim with $k = 500$, $n = 839$ and 129,697 edges.

Procedure	Step 1	Step 2	Step 3	Step 4	Step 5
Crown Reduction Total time 1893.11	PP $k = 1272$ $n = 2398$ $t = 7.76$	CR $k = 1261$ $n = 2312$ $t = 12.58$	CR $k = 1261$ $n = 2305$ $t = 12.66$	CR $k = 1261$ $n = 2305$ $t = 12.49$	LP k = 1261 n = 2305 $t = 1847.62$
LP Kernelization only Total time 2424.14	PP $k = 1272$ $n = 2398$ $t = 7.81$	LP $k = 1263$ $n = 2309$ $t = 2416.33$			

Times are given in seconds. PP indicates preprocessing, CR indicates crown reduction, LP indicates LP kernelization, and BR indicates branching.

Table 3: Graph SH3-5.dim with $k = 1300$, $n = 2,466$ and 364,703 edges.

vertices and 1,011,051 edges, with over 67% of its edges in a completely connected graph with the same number of vertices. When searching for a vertex cover of size $k = 1457$, an initial preprocessing step reduced n to 1210 and k to 930. After this, neither crown reduction nor LP kernelization were successful in producing further reductions. Crown reduction took 6.84 seconds, while LP kernelization took 164.45 seconds. Thus, it may be useful to use crown reduction as a test to estimate whether LP kernelization is likely to be successful in reducing the graph.

The graph just mentioned was derived from microarray data, where a clique corresponds to a set of putatively co-regulated genes. Perhaps we should now say a few words about the use of clique in this setting. The term “putative” is important here. Like so many things in biology, the picture is more complicated than just crowns and cliques. Gene are often pleiotropic, and microarrays are frequently noisy. Thus what one really wants are dense subgraphs that point to biological pathways. Cliques are by definition the densest alternative, and provide the most stringent criteria for correlation across many experiments. The enormous effort we expend in solving clique is balanced by the ease with which we can subsequently soften thresholds, exploit neighborhood structures and find dense subgraphs. Along this vein we refer the interested reader to our work on the "paraclique" method as reported in [9, 10].

For very large dense graphs, it may not even be practical to run LP kernelization. The run time and memory requirements for LP kernelization increase rapidly with the number of edges. One example of this is the graph U74-0.5-369. This graph is based on similar data to RMA-85-280 and has 5680 vertices and 13,736,738 edges, over 85% of the edges in a complete graph with the same number of vertices. Preprocessing reduced n to 3447 and k to 3078. Subsequent attempts at crown reductions were unsuccessful in further reducing the problem, but only took 92.23 seconds. LP kernelization was rendered impractical due to its excessive memory and time requirements. Thus, particularly for large dense graphs, crown reduction may be a practical alternative when LP kernelization is too expensive.

3 LP Kernelization and Crown Isolation

Because crown reduction and LP kernelization are based on different algorithmic techniques, it seems reasonable to hope that the methods would prove to be at least somewhat orthogonal. This turns out not to be the case. The sets P and R identified by LP kernelization form, surprisingly, a crown. This is proved in the following theorem, with an example shown in Figure 3.

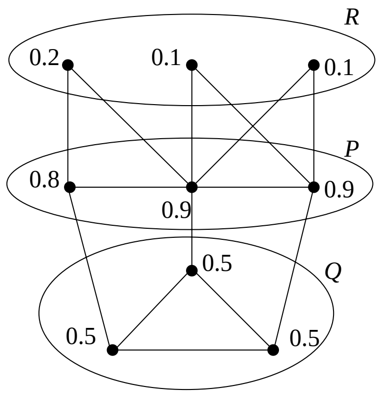

Figure 3: LP Solution used to identify a crown.

Theorem 6 *If LP kernelization finds a set R of vertices to exclude from the vertex cover and a set P of vertices to include in the cover, then (R, P) forms a crown.*

Proof. Let us look at the requirements for a crown. Since $X_u < 0.5$ for all vertices $u \in R$, we know, because of the edge constraint $X_u + X_v \geq 1$, that there cannot be any edges $\{u, v\}$ with both u and v in R. Thus R is an independent set.

Suppose there is a vertex $u \in P$ where $u \notin N(R)$. Then every neighbor v of u has $X_v \geq 0.5$. Thus we could improve the LP solution by imposing $X_u = 0.5$ without violating any of the constraints $X_u + X_v \geq 1$. This cannot happen so we can conclude that $P \subseteq N(R)$. Similarly, suppose there is a $u \in N(R)$ where $u \notin P$. Then $X_u \leq 0.5$ while u has a neighbor $v \in R$ where $X_v < 0.5$. The edge $\{u, v\}$ must violate the constraint $X_u + X_v \geq 1$. This too cannot happen, so we conclude that $N(R) \subseteq P$ and thus $P = N(R)$.

Let M be a maximum matching on the edges between R and P. We now show that every vertex in P must be matched by contraction. Let $C_0 \subset P$ be the set of vertices in P that are unmatched by M and suppose $C_0 \neq \emptyset$. Let $D_0 = N(C_0) \cap R$ and $C_1 = N_M(D_0) \cup C_0$. Repeat this process, setting $D_n = N(C_n) \cap R$ and $C_{n+1} = N_M(D_n) \cup C_n$, until $C = C_{N+1} = C_N$ and $D = D_N$.

Since M is a maximum matching, alternating paths with an odd number of edges that begin and end at unmatched vertices are impossible. Thus any alternating path beginning with a vertex in C_0 has an even number of edges (and an odd number of vertices), beginning and ending in C. Since very vertex in D must be part of such an alternating path, this implies that C must be larger than D. This can be most easily seen by noting that the matching M gives a natural one to one association between the elements of D and $N_M(D)$. If this were not true an alternating path with an odd number of edges would result. Furthermore $C_0 \neq \emptyset$ and $N_M(D)$ are disjoint and $C = N_M(D) \cup C_0$. Thus C is larger than D.

Notice that for any set $P' \subset P$ we know that $N(P') \cap R$ must be larger than P', because otherwise we could improve the LP solution by setting $X_u = 0.5$ for all $u \in P'$ and $u \in N(P') \cap R$. However we have already shown that $C \subset P$ is larger than $N(C) \cap R = D$ so this is a contradiction. Thus $C_0 = \emptyset$ and every vertex in P must be matched. Therefore (R, P) is a crown. ■

The last theorem was proved independently in [11], where the weighted vertex cover problem was considered. Although that paper has much in common with ours, the proofs are distinct and different. We also provide experimental evidence of the comparative utility of crown reduction. More importantly, we offer a classification of crowns that allows us to combine two crowns into a possibly larger crown. With this, we shall derive the first known polynomial-time algorithm for finding a crown of maximum size.

3.1 Crown Types

We now determine the types of crowns that can be isolated by LP kernelization. To do this, it is useful to prove the following lemma that helps refine the relationship between straight and flared crowns.

Lemma 1 *If (I, H) is a flared crown then there is another crown (I', H) that is straight and where $I' \subset I$.*

Proof. Since (I, H) is a crown there is a matching M on the edges between I and H so that all elements of H are matched. Since (I, H) is flared there must be at least one unmatched vertex in I. Let I' be the set of matched vertices in I. It is clear that $I' \subset I$ and that I' is an independent set. It is also clear that M is a matching between I' and H and that every vertex in H is matched. Thus (I', H) is a crown. Finally, the M forms a one-to-one association between the vertices in I' and H. Thus $|I'| = |H|$ and the crown is straight. ■

We are now ready to prove that LP kernelization eliminates all flared crowns from the graph. It does this by either finding all of the flared crown, or by *straightening* the flared crown by identifying the flared part of the crown, but leaving the straight subcrown unrecognized. Since the LP solutions are not unique, different solution techniques may produce different results on the same graph. One solution technique may identify an entire flared crown, while another technique may only straighten the crown. An example of this is shown in Figure 4.

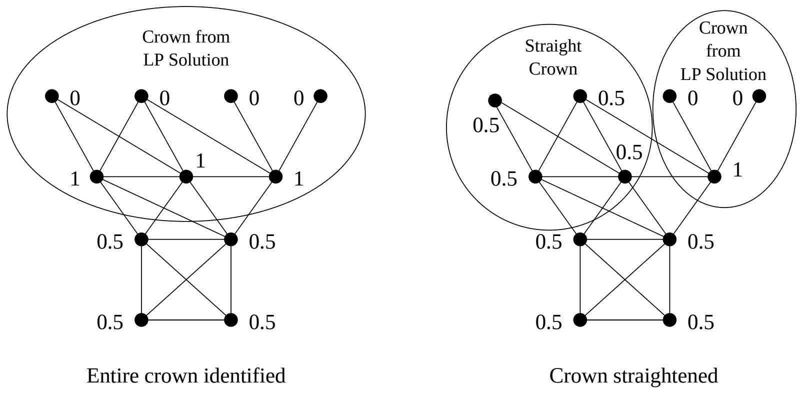

Figure 4: Two different optimal LP solutions of total weight 5.

Theorem 7 *If LP kernelization is performed, then only straight crowns remain.*

Proof. Suppose there is a crown (I, H) with $|I| > |H|$ that is not identified by LP kernelization. Vertices u not removed by LP kernelization are given weights $X_u = 0.5$. Thus $X_u = 0.5$ for every $u \in I \cup H$. Since $|I| > |H|$ we can improved the LP solution by assigning $X_u = 1$ for every $u \in H$ and $X_v = 0$ for every $v \in I$.

We must demonstrate that this new weight assignment is an LP solution by showing that it still meets the edge constraints. Since $N(I) = H$ the condition $X_u + X_v \geq 1$ is satisfied by for all edges (u, v) where either u or v is in I. Next consider edges that have an endpoint in H but not in I. These edges have had their total weight increased and so still meet the edge constraints. Finally, edges that do not connect either to I or to H are unaffected by the new weights and so still satisfy the edge constraints. ■

Although a particular straight crown may not be produced by an arbitrary LP solution, we now show that there is some LP solution that does produce that crown.

Theorem 8 *If (I, H) is a crown, then it is possible to compute an optimal LP solution that identifies a crown to which (I, H) is a subcrown.*

Proof. Suppose there is a particular optimal LP solution that does not identify a crown with (I, H) as a subcrown. This implies that the crown must be straight and $|I| = |H|$ by Theorem 7. Let us construct another LP solution by assigning $X_v = 0$ for all $v \in I$ and $X_u = 1$ for all $u \in H$ and leaving the weight of the other vertices unchanged.

We first show that it is an LP solution by showing that it still meets the edge constraints. Since $N(I) = H$, the condition $X_u + X_v \geq 1$ is satisfied for all edges (u, v) where either u or v is in I. Next consider edges that have an endpoint in H but not in I. These edges have had their total weight increased and so still meet the edge constraints. Finally, edges that do not connect either to I or to H are unaffected by the new weights and so still satisfy the edge constraints.

Finally, we need to show that the solution is optimal. Since $|I| = |H|$ the value of the objective function $\sum_{u \in V} X_u$ is unchanged. Therefore this is a new solution to the LP problem that identifies the crown.■

3.2 Isolating All Crowns

Although each crown has an LP solution that would identify it, we seek to determine if there is a particular LP solution that would eliminate all crowns from the graph. We present a lemma and series of theorems that can be used to design a procedure for identifying all crowns in a graph.

First, we need to show that if a crown is isolated and removed from a graph, and if a second crown is isolated among the remaining vertices, then these two crowns can be combined to form a single crown. To do this it is useful to prove the following lemma that allows us to restrict the number of values that must be considered in the LP solution. It is previously known that there are optimal solutions to the LP problem that only use weights 0, 0.5, and 1 [21, 23]. Nevertheless, it is useful to recast this result here in term of crowns, using Theorem 6, and then to demonstrate a method for modifying any LP solution to use only these weights.

Lemma 2 *If there is an optimal solution to the LP kernelization problem that assigns weight X_u to each vertex $u \in V$ and we define $R = \{u \in V | X_u < 0.5\}$, $Q = \{u \in V | X_u = 0.5\}$, and $P = \{u \in V | X_u > 0.5\}$, then there is another optimal solution to the LP kernelization problem that assigns weights $X'_u = 0$ if $u \in R$, $X'_u = 0.5$ if $u \in Q$, and $X'_u = 1$ if $u \in P$.*

Proof. By Theorem 6 we know that (R, P) forms a crown and so there is a matching M between R and P so that every element in P is matched. The total weight contribution of $R \cup P$ must be at least $|M| = |P|$ in any LP solution, because the edges in M must have total edge weight ≥ 1. We can achieve this lower bound by making the weight assignments $X'_u = 0$ if $u \in R$ and $X'_u = 1$ if $u \in P$. The weights of the remaining vertices are unchanged by the assignment $X'_u = 0.5$ if $u \in Q$, so the total edge weight of the graph is either unchanged or reduced by these assignments.

We now show that this new assignment is in fact an LP solution by considering the edge constraints. The edge constraints are met for all edges $\{u, v\}$ with $u \in P$, since in this case $X'_u = 1$. Edges $\{u, v\}$ with $u \in R$ must have $v \in P$ and so we have already met the edge constraint. This is because (R, P) is a crown and so $N(R) = P$. Finally, we consider edges $\{u, v\}$ with $u \in Q$. Such an edge cannot have $v \in R$ since $N(R) = P$, and so we only need to examine cases where $v \in P$ or $v \in Q$. If $v \in P$ then we have already met the edge constraint. If $v \in Q$ then $X'_u = 0.5$ and $X'_v = 0.5$ and so the edge constraint is met in this final case. ■

Theorem 9 *Suppose a graph $G = (V, E)$ has a crown (I, H) identified by LP kernelization, and that when (I, H) is removed the induced subgraph $G' = (V', E')$ has another crown (I', H'). Then $(I \cup I', H \cup H')$ forms a crown in G.*

Proof. Let S be the optimal LP solution for G where $X_v = 0$ for every $v \in I$, $X_u = 1$ for every $u \in H$, and $X_w = 0.5$ for every $w \notin H \cup I$. We know that such an optimal LP solution exists by Lemma 2.

We construct a new optimal LP solution S'. For any $u \in V$ we set X'_u as follows:

(1) If $u \in H \cup H'$ then $X'_u = 1$.

(2) If $u \in I \cup I'$ then $X'_u = 0$.
(3) If $u \notin H \cup H' \cup I \cup I'$ then $X'_u = 0.5$.

Notice that all of the vertices in $I' \cup H'$ remain when (I, H) is removed so $I' \cup H'$ and $I \cup H$ are disjoint. We already know that I and H are disjoint and that I' and H' are disjoint. This implies that I, H, I', and H' are all mutually disjoint. Thus there are no contradictions in the definition of S'.

We now show that S' is in fact an LP solution by verifying the edge constraints for an arbitrary edge $(u, v) \in E$.

Case 1: One of the endpoints is in $H \cup H'$. Without loss of generality assume $v \in H \cup H'$, then $X'_v = 1$ and the edge constraint is met.

Case 2: One of the endpoints is in I. Without loss of generality assume $u \in I$. We know that $v \in H$ because $N(I) = H$. Thus the problem reduces to case 1 and the edge constraint is met.

Case 3: One of the endpoints is in I'. Without loss of generality assume $u \in I'$. Since (I, H) is removed before (I', H') is identified, there are two possibilities $v \in I \cup H$ or $v \notin I \cup H$. If $v \in I \cup H$, we know that $N(I) = H$ and $u \notin H$ so we can restrict our attention to the case were $v \in H$. Thus the problem reduces to case 1 and the edge constraint is met. If $v \notin I \cup H$ then v is a vertex in G' and in this graph $N(I') = H'$. Thus $v \in H'$, the problem reduces to case 1 and the edge constraint is met.

Case 4: Neither u nor v is in $H \cup H' \cup I \cup I'$. In this case $X'_u = 0.5$ and $X'_v = 0.5$ and the edge constraint is met.

Finally, we show that S' is an optimal LP solution for G. We know that S is an optimal LP solution for G. Notice that the total edge weight in S is $|H| + 0.5|V - (H \cup I)| = |H| + 0.5|V'|$ and that an optimal LP solution for G' has total edge weight $0.5|V'|$. Also notice that because (I', H') is a crown, there is another optimal LP solution that identifies this crown and uses weights 0, 1, and 0.5. We know this is possible by the proof of Lemma 2. The total edge weight of this new LP solution for G' is $|H'| + 0.5|V' - (H' \cup I')|$. However, since these are both optimal solutions for G', we know that $0.5|V'| = |H'| + 0.5|V' - (H' \cup I')|$.

Now consider the total edge weight for S', which is $|H \cup H'| + 0.5|V - (H \cup H' \cup I \cup I')| = |H \cup H'| + 0.5|V - (H \cup I) - (H' \cup I')| = |H \cup H'| + 0.5|V' - (H' \cup I')|$. Since H, H' are disjoint, we know that $|H \cup H'| = |H| + |H'|$. Thus the total edge weight for S' is $|H| + |H'| + 0.5|V' - (H' \cup I')| = |H| + 0.5|V'|$, which is the total edge weight for S. Thus, both S and S' are optimal LP solutions for G. Thus by Theorem 6 we know that the sets identified by S', namely $I \cup I'$ and $H \cup H'$ must form a crown. ■

Now we need to show that identifying two different crowns cannot result in any serious conflicts. That is, if a graph has two different crowns, then the two crowns can be combined to form a single crown. This is not a matter of simply taking the union of the two crowns. There may be vertices in the independent set of one crown that are included in the cutset of the other crown. Such conflicts always occur in straight crowns that are subcrowns of the two original crowns and that can be reversed without disrupting the crown properties. An example is shown if Figure 5.

Theorem 10 *If (I_1, H_1) and (I_2, H_2) are crowns of a graph G that are identified by two different LP solutions, then there is a crown (I, H) that contains all the vertices in $I_1 \cup I_2 \cup H_1 \cup H_2$ and where $I_1 \subseteq I$ and $H_1 \subseteq H$.*

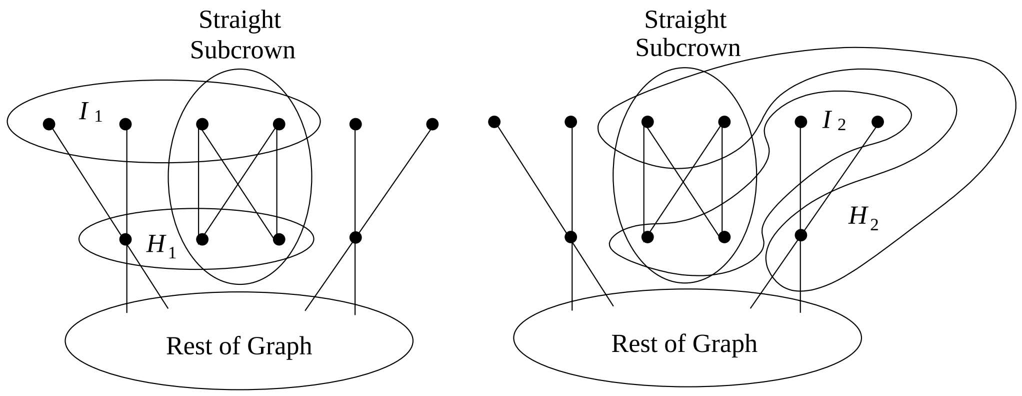

Figure 5: Two crowns have different orientations for a straight subcrown.

Proof. For each vertex u in G, let X_u^1 designate the weight of u in an optimal LP solution that identifies (I_1, H_1) so that $X_u^1 = 0$ if and only if $u \in I_1$, $X_u^1 = 1$ if and only if $u \in H_1$, and $X_u^1 = 0.5$ otherwise. We know such a solution exists by Lemma 2. Similarly find X_u^2 for an optimal LP solution that identifies (I_2, H_2).

We now create another optimal LP solution by defining, for each vertex u, $X_u^* = \frac{X_u^1 + X_u^2}{2}$. Notice that the total weight $\sum_u X_u^* = \sum_u X_u^1 = \sum_u X_u^2$ is still optimal. If $\{u, v\}$ is an edge in G, then $X_u^* + X_v^* = \frac{X_u^1+X_u^2}{2} + \frac{X_v^1+X_v^2}{2} \geq 1$ since $X_u^1 + X_v^1 \geq 1$ and $X_u^2 + X_v^2 \geq 1$. Thus X^* defines an optimal LP solution. By Lemma 2 we can modify these values so that $X_u^* = 0$, 0.5, or 1 for all vertices u in G.

The only vertices in the original two crowns that do not have X^* weights of 1 or 0 are those in $I_1 \cap H_2$ and $I_2 \cap H_1$. Let G' be the graph that remains when all vertices with X^* weights of 0 or 1 are removed from the original graph. In this graph let $u \in I_1 \cap H_2$ and v be a neighbor of u. Since $u \in I_1$ and $H_1 = N(I_1)$ we know $v \in H_1$. Thus $X_v^1 = 1$ and since we know $X_v^* = 0.5$ we also know $X_v^2 = 0$. Therefore $v \in I_2$ and $v \in H_1 \cap I_2$. Thus $N(I_1 \cap H_2) \subseteq (I_2 \cap H_1)$ in graph G'. A similar argument shows that $N(I_2 \cap H_1) \subseteq (I_1 \cap H_2)$ in G'.

Finally, we show that $(I_1 \cap H_2, I_2 \cap H_1)$ forms a straight crown that can be reversed. Notice that if $|I_1 \cap H_2| \leq |I_2 \cap H_1|$ we would not increase the size of the LP solution by assigning weight 1 to the vertices in $I_1 \cap H_2$ and 0 to the vertices in $I_2 \cap H_1$. On the other hand if $|I_2 \cap H_1| \leq |I_1 \cap H_2|$ then we would not increase the size of the LP solution by assigning weight 1 to the vertices in $I_2 \cap H_1$ and 0 to the vertices in$I_1 \cap H_2$. In either case we have a new LP solution that identifies $(I_1 \cap H_2, I_2 \cap H_1)$ as a crown but X^* does not. By Theorem 7, this implies that $(I_1 \cap H_2, I_2 \cap H_1)$ is a straight crown. Since $N(I_1 \cap H_2) \subseteq (I_2 \cap H_1)$ and $N(I_2 \cap H_1) \subseteq (I_1 \cap H_2)$ in G' we know that this crown has no neighbors. Thus it can be reversed without loss of generality.

We select the independent set of the straight crown to be $I_1 \cap H_2$ and the cutset of the straight crown to be $I_2 \cap H_1$ so that the weight agrees with the crown (I_1, H_1). Notice that all of the remaining vertices in the two original crowns were identified by the LP solution defined by X^*. Thus, by Theorem 9, we can union the crown identified by X^* and the straight crown to obtain (I, H) where $I = I_1 \cup (I_2 - I_1)$ and $H = H_1 \cup (H_2 - I_1)$. ■

4 A Polynomial-Time Algorithm for Crown Decomposition

We can now use the results we have just proved to produce a polynomial-time algorithm that will find all possible maximal crowns in an arbitrary graph. In particular, we prove that finding a crown of maximum size is solvable in polynomial time.

Theorem 11 *There is a polynomial-time algorithm for processing a graph that will produce an induced subgraph with no crowns.*

Proof. We present such an algorithm.

Step 1: Perform LP kernelization. By Theorem 7, this can be used to eliminate all flared crowns by either removing the entire crown, or by removing enough vertices so that all the crowns are straight. Let $G_1 = (V_1, E_1)$ be the graph induced by the set of vertices that remain in the kernel. Since these vertices were not removed by LP kernelization, we know that $X_u = 0.5$ for all $u \in V_1$. We also know that the total weight in the optimal LP solution for G_1 is $0.5|V_1|$.

Step 2: Pick a vertex $w \in V_1$ and test it to see if it is in the independent set of some crown by finding the optimal solution of the following LP problem.

Assign a value $X_u \geq 0$ to each vertex $u \in V_1$ so that the following conditions hold.

(1) Minimize $\sum_{u \in V_1} X_u$.
(2) Satisfy $X_u + X_v \geq 1$ whenever $uv \in E_1$.
(3) $X_w = 0$

Step 3: If the total weight is still $0.5|G_1|$, then this too is an optimal solution of the original LP problem and we have identified a straight crown. We remove the straight crown from the graph in the usual manner producing an induced subgraph $G_2 = (V_2, E_2)$ where we know that $X_u = 0.5$ for all $u \in V_2$ and the total weight in the optimal LP solution for G_2 is $0.5|V_2|$. If the total weight is larger, then we have not identified a crown and we let $G_2 = G_1$.

We repeatedly apply Steps 2 and 3 until all vertices have been checked or eliminated, producing the graph G'. Theorem 9 guarantees that removing a crown does not create new crowns from vertices not previously in a crown. Theorem 10 guarantees that the order in which crowns are identified can change the result only up to the direction of a reversible crown. ■

We therefore need only to check each vertex once, and when this process is complete, there can be no crowns and we will have identified all possible crowns in the graph. The total run time of the LP solution procedure is $O(mn^{3/2})$, where m is the number of edges and n is the number of vertices in G if the network flow approach is used. This is a worst case scenario, in which the original graph has no crowns, and the process is repeated n times, once for each vertex.

Theorems 9 and 10 assert that the union of the crowns found in the algorithm of Theorem 11 forms a single large crown and is, up to reversals of straight crowns, unique. Moreover, that algorithm allows us to find a single large crown that breaks any input graph into a crown and a subgraph without any crowns. We state this in the form of the following corollary.

Corollary 1 *Any graph can be decomposed into two subgraphs, C and K, where C is a crown and K has no crowns.*

Thus, we can obtain a crown-free subgraph, K, from any arbitrary graph. What if we want K to be of minimum order? Is there another decomposition that produces a crown-free graph K'

whose order is smaller than $|K|$? The answer turns out to be "no." To see this, first note that if we have two crowns $C = (I, H)$ and $C' = (I', H')$ such that $C \subset C'$ and crown C is identified by our LP-based algorithm, then $C' - C$ is also contained in a crown that will be identified by the same run of the algorithm (possibly in subsequent iterations). In other words, any crown identified by the algorithm is maximal (the algorithm does not deliver proper subcrowns). Now assume C and C' are two distinct pairs produced by independent runs of the algorithm just described. Also assume C' is a maximum crown. Then, by Theorem 10, we can combine C and C' to obtain an even larger crown. But this is impossible, because C and C' are both maximal. We have therefore (constructively) proved the final result of this paper.

Theorem 12 *Isolating a crown of maximum order is solvable in polynomial time.*

5 Concluding Remarks

Crown reduction runs much faster in practice than linear programming, and often reduces the graph just as well even though its worst-case bound on kernel size is somewhat larger. Given what we have observed, an effective approach seems to be preprocessing followed immediately by crown reduction. If the resultant kernel is dense, one should proceed directly to branching. If the graph is sparse, however, one should apply either network flow or linear programming before continuing.

Decomposing a graph into crowns and subgraphs that are crown free seems likely to have uses other than just reducing vertex cover kernel size. Exactly where this line of research may lead is hard to predict. We are hopeful that the study of crown structures will be helpful in problems such as $n - k$ coloring and a variety of $\mathcal{NP}$-hard packing problems. Finally, we note the significance of Theorem 12 in the context of the recent work of Sloper.

Theorem 13 [24] *Given a graph G and an integer k, it is $\mathcal{NP}$-complete to decide whether G contains a crown whose order is exactly k.*

As of this writing, the complexity of isolating a crown of minimum order is open.

Acknowledgments

We wish to thank Henning Fernau for many helpful discussions. We also thank two anonymous referees, whose detailed comments on the original version of this paper have helped us to improve the presentation.

References

[1] F. N. Abu-Khzam. *Topics in Graph Algorithms: Structural Results and Algorithmic Techniques, with Applications*. PhD thesis, Dept. of Computer Science, University of Tennessee, 2003.

[2] F. N. Abu-Khzam, J. Cheetham, F. Dehne, M. A. Langston, S. Pitre, A. Rau-Chaplin, P. Shanbhag, and P. J. Taillon. *ClustalXP, An eXtended and Parallel version of Clustal*. See `http://www.scs.carleton.ca/ dehne/proj/clustalxp/bottom.htm`.

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

[4] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons. Scalable parallel algorithms for FPT problems. *Algorithmica*, 2006. Accepted for publication.

[5] N. E. Baldwin, E. J. Chesler, S. Kirov, M. A. Langston, J. R. Snoddy, R. W. Williams, and B. Zhang. Computational, integrative and comparative methods for the elucidation of genetic co-expression networks. *Journal of Biomedicine and Biotechnology*, 2:172–180, 2004.

[6] J. F. Buss and J. Goldsmith. Nondeterminism within $\mathcal{P}$. *SIAM Journal on Computing*, 22:560–572, 1993.

[7] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large FPT problems on coarse grained parallel machines. *Journal of Computer and System Sciences*, 67:691–706, 2003.

[8] J. Chen, I. A. Kanj, and G. Xia. Simplicity is beauty: Improved upper bounds for vertex cover. Technical report, Texas A&M University, 2005.

[9] E. J. Chesler and M. A. Langston. Combinatorial genetic regulatory network analysis tools for high throughput transcriptomic data. In *Proceedings, First Annual RECOMB Satellite Workshop on Systems Biology and Second Annual RECOMB Satellite Workshop on Regulatory Genomics*, 2005.

[10] E. J. Chesler, L. Lu, S. Shou, Y. Qu, J. Gu, J. Wang, H. C. Hsu, J. D. Mountz, N. E. Baldwin, M. A. Langston, J. B. Hogenesch, D. W. Threadgill, K. F. Manly, and R. W. Williams. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. *Nature Genetics*, 37:233–242, 2005.

[11] M. Chlebík and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Electronic Colloquium on Computational Complexity, Report No. 101*, 2004.

[12] B. Chor, M. R. Fellows, and D. W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proceedings, International Workshop on Graph Theoretic Concepts in Computer Science*, pages 257–269, 2004.

[13] Vasek Chv´tal. *Linear Programming*. W.H.Freeman, New York, 1983.

[14] ClustalW. See `http://helix.nih.gov/apps/bioinfo/clustalw.html`.

[15] W. Cook. Private communication, 2003.

[16] E. A. Dinic. Algorithm for solution of a problem of maximum flows in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280, 1970.

[17] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[18] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM*, 35:727–739, 1988.

[19] M. R. Fellows and M. A. Langston. On search, decision and the efficiency of polynomial-time algorithms. *Journal of Computer and Systems Sciences*, 49:769–779, 1994.

[20] D. Hochbaum. *Approximation Algorithms for $\mathcal{NP}$-hard Problems*. PWS, 1997.

[21] S. Khuller. The vertex cover problem. *ACM SIGACT News*, 33:31–33, June 2002.

[22] NCBI. See `http://www.ncbi.nlm.nih.gov/`.

[23] G.L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

[24] C. Sloper. *Techniques in Parameterized Algorithm Design*. PhD thesis, Department of Computer Science, University of Bergen, Norway, 2005.

[25] W. H. Suters. Crown decomposition: Theoretical results and practical methods. Master's thesis, Department of Computer Science, University of Tennessee, 2004.