Improved Exact Exponential Algorithms for Vertex Bipartization and Other Problems

VENKATESH RAMAN, SAKET SAURABH, SOMNATH SIKDAR

The Institute of Mathematical Sciences, Taramani, Chennai 600113.
{vraman|saket|somnath}@imsc.res.in

Abstract. We study efficient exact algorithms for several problems including VERTEX BIPARTIZATION, FEEDBACK VERTEX SET, 3-HITTING SET, MAX CUT in graphs with maximum degree at most 4. Our main results include:

1. an $O^*(1.9526^n)$ [1] algorithm for VERTEX BIPARTIZATION problem in undirected graphs;
2. an $O^*(1.8384^n)$ algorithm for VERTEX BIPARTIZATION problem in undirected graphs of maximum degree 3;
3. an $O^*(1.945^n)$ algorithm for FEEDBACK VERTEX SET and VERTEX BIPARTIZATION problem in undirected graphs of maximum degree 4;
4. an $O^*(1.9799^n)$ algorithm for 4-HITTING SET problem;
5. an $O^*(1.5541^m)$ algorithm for FEEDBACK ARC SET in tournaments.

To the best of our knowledge, these are the best known exact algorithms for these problems. In fact, these are the first known exact algorithms with the base of the exponent < 2. En route to these algorithms, we introduce two general techniques for obtaining exact algorithms. One is through parameterized complexity algorithms, and the other is a 'colored' branch-and-bound technique.

1 Introduction

In recent years there has been a growing interest in designing exact algorithms for NP-Complete problems. Fast exponential-time algorithms lead to practical algorithms for at least moderate instance sizes. Furthermore, there is a wide variation in the time complexities of exact algorithms for NP-complete problems. Classical complexity theory cannot explain these differences. The study of exact algorithms may lead to a finer classification, and hopefully a better understanding, of NP-complete problems. For a recent survey on exact algorithms see Woeginger [14].

Parameterized complexity is a recently developed approach devised by Downey and Fellows for dealing with hard computational problems arising from industry and applications. The theory of parameterized complexity is based on the observation that many hard problems are associated with a parameter that

[1] The O^* notation suppresses polynomial terms. Thus we write $O^*(T(x))$ for a time complexity of the form $O(T(x) \cdot \text{poly}(|x|))$ where $T(x)$ grows exponentially with $|x|$, the input size. See the survey by Woeginger[14] for a detailed discussion on this.

varies within a small or moderate range. By taking advantage of small parameter values many hard problems can be solved practically. A parameterized problem consists of a tuple (π, k) where π is the problem instance and k is the parameter. A parameterized problem is said to be *fixed parameter tractable* if there exists an algorithm for the problem with time complexity $O(f(k) \cdot |\pi|^{O(1)})$, where f is a function of k alone and $|\pi|$ represents the size of the input instance. For an introduction to parameterized complexity see the book by Downey and Fellows [2]. For recent developments see the survey by Downey and Fellows [3].

The Vertex Bipartization problem is to find, given an undirected graph G on n vertices, the minimum number of vertices whose removal makes the graph bipartite. This problem has numerous applications, for instance, in VLSI design [1], computational biology [11], and register allocation [15]. The Vertex Bipartization problem is known to be NP-Complete even for graphs with maximum degree 3 [1]. This problem has been studied extensively from different algorithmic paradigms. An approximation algorithm with factor $\log n$ is known for the problem [4]. The parameterized version of this problem has been recently shown to fixed parameter tractable by Reed et al [10]. Their algorithm runs in time $O(3^k \cdot kmn)$, where k is the parameter, n is the number of vertices and m is the number of edges. The (optimization) problem can be solved exactly by exhaustively looking at all possible sets of vertices in time $O^*(2^n)$. So far there have been no exact algorithms that are better than this trivial brute-force algorithm.

In this paper we describe a generic technique that allows us to construct efficient exact algorithms for a problem using a parameterized algorithm for the same problem. More specifically, we show that if a (parameterized) problem (π, k) has a fixed parameter algorithm with time complexity $c^k \cdot |\pi|^{O(1)}$ then its optimization version has an exact algorithm with time complexity $O^*(d^{|\pi|})$, where $d < c$, and $|\pi|$ is the size of the input instance. Using this technique we obtain an exact algorithm for the Vertex Bipartization problem that runs in time $O^*(1.9526^n)$, where n is the number of vertices. For maximum degree 3 graphs we can do better. We devise another technique which is a modified form of the branch-and-bound method and obtain an exact algorithm with time complexity $O^*(1.8384^n)$ for the Vertex Bipartization problem in maximum degree 3 graphs. We extend this technique to obtain exact algorithms for the Feedback Vertex Set and Vertex Bipartization problems in maximum degree 4 graphs. Note that the Feedback Vertex Set problem is polynomial time solvable in maximum degree 3 graphs [12], but is NP-complete for maximum degree 4 graphs.

The paper is organised as follows. In Section 2, we present exact algorithms for the Vertex Bipartization problem in graphs with maximum degrees 3 and 4, and the Feedback Vertex Set problem in graphs with maximum degree 4. In Section 3, we develop a general technique by which we can convert a parameterized algorithm of time complexity $O^*((4-\epsilon)^k),\ \epsilon > 0$ to an exact algorithm of time complexity $O^*((2-\eta)^k),\ \eta > 0$. This technique is based on a careful use of the parameterized algorithm for certain values of the parameter and a brute-force algorithm for other values. In Section 3.1, we give several applications of Algorithm Exact() and Theorem 4. In particular, we give the best known exact algorithms for the following problems:

1. Vertex Bipartization in general undirected graphs;
2. 4-Hitting Set;
3. Feedback Arc Set in tournaments;

Apart from this, we also give simple efficient exact algorithms based on Theorem 4 for the Max Cut problem in graphs with average vertex degree 3 and 4 and for the 3-Hitting Set problem; these are not the best known exact algorithms for these problems but are stated in this paper to highlight the applicability of Theorem 4. Finally in Section 4, we conclude with some remarks and open problems. All graphs in this paper are undirected unless stated otherwise.

2 Exact Algorithms for Vertex Bipartization and Feedback Vertex Set in Graphs with Maximum Degree 4

In this section, we give improved exact algorithms for the Vertex Bipartization problem in graphs of maximum degrees 3 and 4 and the Feedback Vertex Set problem in graphs of maximum degree 4.

The main idea behind these algorithms is to use the techniques of *preprocessing* and *branching*. Typical branch-and-bound algorithms (for Independent Set, Vertex Cover) build a solution by either picking a vertex or excluding it from the solution. When they exclude a vertex from the solution they typically delete it and work on the resulting smaller graph. For the problems we work on, you cannot delete a vertex because removing a vertex that is not part of the solution may kill cycles passing through it.

To overcome this, we resort to coloring the vertices. All vertices are colored good initially. When we branch on a good vertex, we either include it in our solution and delete it, or exclude it and color it bad. Coloring a vertex bad *decreases the number of good vertices.* As we always branch on good vertices we end up reducing the graph size in both cases. This gives us an $O^*(2^n)$ algorithm. Our main contribution is in pushing this idea to get an $O^*(c^n)$ algorithm where $c < 2$.

2.1 Vertex Bipartization in Graphs with Maximum Degree 3

The algorithm is recursive and is depicted in Figure 1. It makes use of a preprocessing routine P described in Figure 2.
Correctness Step $2a$ of the algorithm VBP-D3 simply does a brute-force enumeration. In Step $2b$, the algorithm branches on a good vertex and constructs a solution containing that vertex and one not containing that vertex and returns the solution with minimum size. Both these steps do not need any further justification. We only need to justify the steps of the preprocessing algorithm P.

```
Algorithm VBP-D3($G = (V, E)$, $S$, Col)
Input: An undirected multigraph $G = (V, E)$ with maximum degree 3, whose vertices
have been colored. Here $n$ is the number of vertices in the input graph. Initially the
algorithm is called with $S \leftarrow \emptyset$ and $\mathsf{Col}(v) = \mathsf{good}$ for all vertices $v \in G(V)$.
Output: A minimum odd cycle transversal of $G$.
Step 1 Call $P(G, S, \mathsf{Col})$. If $P$ returns NO then return NO.
Step 2 Apply the first step which is applicable:
    Step 2a Let $n'$ be the number of vertices in the current graph. If $n' \leq 0.6n$ or if
        every path of length 2 has at least one bad vertex then try all possible solutions
        $S \cup T$, where $T$ is a some subset of the remaining good vertices, and return the
        one with minimum size.
    Step 2b Pick a path $xyz$ in $G$ where all of $x, y,$ and $z$ are good. Call the algorithm
        on the following instances and return the smaller solution.
          – $S \leftarrow S \cup \{z\}$ and call VBP-D3($G - \{z\}$, $S$, Col).
          – Set $\mathsf{Col}(z) = \mathsf{bad}$ and call VBP-D3($G$, $S$, Col).
```

Fig. 1. Algorithm VBP-D3()

In Step 1 of the preprocessing algorithm, we recursively remove vertices of degree ≤ 1. Such vertices cannot be part of any minimum solution and can be safely removed. In Step 2, we check whether the subgraph $G[B]$ induced by the bad vertices contains an odd cycle. If this is the case then there cannot be a minimum solution containing good vertices only. Thus P returns NO. The correctness of Step 3 is obvious.

We next consider Step 4 in detail. In this step, we look for a vertex v_i of degree 2 which does not have self-loop. Such a vertex must be part of some path $uv_1 \ldots v_i \ldots v_k w$, where u and w are of degree 3 and are possibly identical. There are two broad cases to handle:

Case 1: At least one of u or w is colored good. Without loss of generality assume that $\mathsf{Col}(u) = \mathsf{good}$. Every odd cycle that passes through v_i also passes through u and w. Thus if there is a good vertex v_j that is part of some minimum solution S, then $(S \setminus \{v_j\}) \cup \{u\}$ is also a minimum solution. Therefore we can label all the vertices v_j bad, $(1 \leq j \leq k)$. We would also like to maintain the parity of all cycles passing through u (and w). Thus we retain only one of the vertices v_j if the path length is even and none if the path length is odd.

Case 2: Both u and w are colored bad. In this case, we remove vertices from the path $v_1 \ldots v_k$ taking care to maintain parity of cycles passing through u and w, and if there exists one good vertex among the v_j's to retain it. This is necessary because these good vertices could be the only good ones on the odd cycles passing through u and w.

In Step 5, we add vertices having self-loops in our solution. This is because vertices with self-loops represent an odd cycle in the original graph and therefore must be included in any minimum solution. A similar argument holds for triangles containing only one good vertex. A degree 2 vertex which is part of a

```
Preprocessing Algorithm $P(G, S, \mathsf{Col})$
Input: A multigraph $G$ whose vertices have been colored good or bad along with a partially constructed solution $S$.
Output: A (possibly smaller) colored multigraph along with a (possibly larger) solution or NO signifying that there does not exist a solution containing good vertices only.
Let $B$ be the set of bad vertices of $G$. Perform the following steps as long as possible.
 1. If $G$ has a vertex of degree $\leq 1$, remove it along with the incident edge.
 2. Check whether $G[B]$ is bipartite. If not then return NO.
 3. Check whether any connected component is a cycle. Remove all connected components that are even cycles; if a connected component $C$ is an odd cycle, include a good vertex of $C$ in $S$ and remove this cycle.
 4. If $G$ has a vertex $v_i$ of degree 2 (which is not a self-loop) then it must be that $v_i$ is part of some path of the form $uv_1 \ldots v_i \ldots v_k w$ where each $v_j$ $1 \leq j \leq k$ is a degree 2 vertex and degree of $u$ and $w$ are $\geq 3$. Here $u$ and $w$ could be the same vertex.
    (a) Case 1: At least one of the vertices $u$ and $w$ are colored good. If the path length ($k+1$ above) between $u$ and $w$ is odd then delete all the vertices $v_j$ and add the edge $(u, w)$ (although $u$ and $w$ might already have an edge between them). If the path length between $u$ and $w$ is even then delete vertices $v_2, v_3, \ldots, v_k$ and add an edge between $v_1$ and $w$ and color $v_1$ bad.
    (b) Case 2: Both $u$ and $v$ are colored bad. Suppose the path length ($k+1$ above) is even and there is at least one $v_i$ colored good, then replace the path $uv_1v_2 \ldots v_k w$ by $uv_i w$. If no vertex $v_i$ is colored good then replace the path $uv_1v_2 \ldots v_k w$ by $uv_1 w$. Next suppose the path length is odd. If there is a good vertex $v_i$ then replace the path $uv_1v_2 \ldots v_k w$ by $uv_i v_j w$, where $i \neq j$ and color $v_j$ bad if it is not already so. If there are no good vertices then replace the path $uv_1v_2 \ldots v_k w$ by the edge $uw$. (See Figure 3 for an explanation.)
 5. Include all vertices with self loops in $S$ and remove them from the graph. Also remove degree 2 vertices from all cycles of length 2. Find all triangles $\Delta_{uvw}$ with $\mathsf{Col}(u) = \mathsf{good}$, $\mathsf{Col}(v) = \mathsf{bad}$, $\mathsf{Col}(w) = \mathsf{bad}$. Set $S \leftarrow S \cup \{u\}$ and remove $u$ from the graph.
```

Fig. 2. The preprocessing algorithm for the VERTEX BIPARTIZATION problem

length 2 cycle can be safely removed from the graph since such a vertex cannot be part of the solution as it is not part of any odd cycle in the current graph.
Time Complexity Let $G = (V, E)$ be the input graph of maximum degree 3 with n vertices and m edges. Since the graph is of maximum degree 3, $m \leq \frac{3n}{2}$.

First we will show that if we reach Step $2a$ of the algorithm the number of good vertices n' in the current graph is at most $0.6n$. To do this we partition the set of good vertices into following three types.

Type 1: Degree 2 good vertices.
Type 2: Degree 3 good vertices with one good neighbour.
Type 3: Degree 3 good vertices with all bad neighbours.

Step 4 of the preprocessing routine ensures that any degree 2 good vertex u has both its neighbours bad. Also observe that a good vertex of degree 3 can

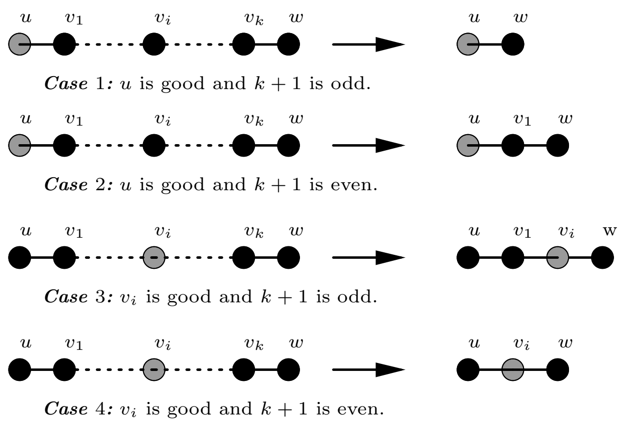
u v1 vi vk w
u w
Case 1: u is good and k + 1 is odd.
u v1 vi vk w
u v1 w
Case 2: u is good and k + 1 is even.
u v1 vi vk w
u v1 vi w
Case 3: vi is good and k + 1 is odd.
u v1 vi vk w
u vi w
Case 4: vi is good and k + 1 is even.

Fig. 3. Step 4 of the Preprocessing Algorithm P. The black vertices are bad and the shaded ones are good.

have at most one good neighbour; for if not then there exists a path of length 2 containing only good vertices. Thus any good vertex will be of one of the three types mentioned above. Let n_1, n_2, and n_3 be the number of vertices of Type 1, Type 2, and Type 3 respectively. We obtain an upper bound on the number of good vertices by counting the number of edges between good and bad vertices. Define $n_g = n_1 + n_2 + n_3$. Define a good-bad edge to be one with one end point labelled good and the other labelled bad. Similarly define a good-good edge. The number of good-bad edges in the current graph is $2n_1 + 2n_2 + 3n_3$ and the number of good-good edges is $n_2/2$. Moreover the graph has maximum degree 3. We therefore have the following inequalities.

$$\begin{aligned} 2n_1 + 2n_2 + 3n_3 &\leq \frac{3(n - n_1) + 2n_1}{2} - \frac{n_2}{2} \\ 2.5(n_1 + n_2 + n_3) &\leq \frac{3n}{2} \Rightarrow n_g \leq 0.6n. \end{aligned}$$

In Step 2b, we find a path xyz of length 2 consisting of good vertices only. Here we have two situations to deal with. If we include the vertex z in the solution, we remove it from the graph which results in at least one good vertex y with degree at most 2 having a good neighbour x. But the preprocessing algorithm will either delete y or label it bad. In either case, the number of good vertices reduces by at least 2. If we don't pick z in our solution, then we label it bad, and this reduces the number of good vertices by 1. Thus the time complexity is given by the recurrence below:

$$\begin{aligned} T(n_g) &\leq T(n_g - 1) + T(n_g - 2) \\ T(0.6n_g) &= 2^{0.6n_g}. \end{aligned}$$

Here $T(n_g)$ is bounded by $(1.62)^{0.4n_g} \cdot 2^{0.6n_g}$ which is $O^*(1.8384^{n_g})$. Moreover on any path in the recursion tree, the algorithm takes polynomial space. Initially $n_g = n$ and therefore we have the following.

Theorem 1. *Let $G = (V, E)$ be an undirected graph with maximum degree 3 with n vertices and m edges. Then* Vertex Bipartization *problem on G can be solved exactly using polynomial space and in time $O^*(1.8384^n)$.*

2.2 The FVS and VBP problems in graphs with maximum degree 4

In this subsection, we extend the ideas described previously for the Vertex Bipartization problem to graphs with maximum degree 4. We also give an exact algorithm for Feedback Vertex Set problem on graphs with maximum degree 4. Again both algorithms in this subsection rely on preprocessing

```
Preprocessing Algorithm $P1(G, S, \mathsf{Col})$
Input: A multigraph $G$ whose vertices have been colored good or bad along with a partially
constructed solution $S$.
Output: A (possibly smaller) colored multigraph along with a (possibly larger) solution
or NO signifying that there does not exist a solution containing good vertices only.
Let $B$ be the set of bad vertices of $G$. Perform the following steps as long as possible.
 1. If $G$ has a vertex of degree $\leq 1$, remove it along with the incident edge.
 2. Check whether $G[B]$ is acyclic. If not then return NO.
 3. Check whether any connected component is a cycle. If a connected component $C$ is a
    cycle, include a good vertex of $C$ in $S$ and remove this cycle.
 4. If $G$ has a vertex $v_i$ of degree 2 (which is not a self-loop) then it must be that $v_i$ is
    part of some path of the form $uv_1 \ldots v_i \ldots v_k w$ where each $v_j$ $1 \leq j \leq k$ is a degree 2
    vertex and $u$ and $w$ are vertices of degree $\geq 3$. Here $u$ and $w$ could be the same vertex.
    (a) Case 1: At least one of the vertices $u$ and $w$ is colored good. Then delete all the
        vertices $v_j$ and add the edge $(u, w)$ (although $u$ and $w$ might already have an edge
        between them).
    (b) Case 2: Both $u$ and $v$ are colored bad. Suppose there is at least one $v_i$ colored
        good, then replace the path $uv_1v_2 \ldots v_k w$ by $uv_i w$. If no vertex $v_i$ is colored good
        then replace the path $uv_1v_2 \ldots v_k w$ by the edge $uw$.
 5. Include all vertices with self loops in $S$ and remove them from the graph. Find all
    cycles of length 2 with a good vertex and include it in $S$ and remove it from the
    graph. Find all triangles $\Delta_{uvw}$ with $\mathsf{Col}(u) = $ good, $\mathsf{Col}(v) = $ bad, $\mathsf{Col}(w) = $ bad. Set
    $S \leftarrow S \cup \{u\}$ and remove $u$ from the graph.
```

Fig. 4. The preprocessing algorithm for the Feedback Vertex Set problem

and branching. We will use the same preprocessing algorithm for the Vertex Bipartization problem. The preprocessing algorithm for Feedback Vertex Set is almost the same and is described in Figure 4.

We first describe the algorithm for the Feedback Vertex Set problem. The main strategy of the algorithm is to find a good vertex with a sufficient number of good neighbours so that on the branch where we include a good vertex v in the solution, we can either delete at least one good neighbour of v or color the neighbour bad without making any further branches. The algorithm achieves this by looking for a good vertex of degree 3 with at least two good neighbours or a good vertex of degree 4 with at least three good neighbours. The detailed algorithm is described in Figure 5.

```
Algorithm FVS-D4($G = (V, E)$, $S$, $\mathsf{Col}$)
Input: A multigraph $G = (V, E)$ with maximum degree 4, whose vertices have been colored.
Here $n$ is the number of vertices in the input graph. Initially the algorithm is called with
$S \leftarrow \emptyset$ and $\mathsf{Col}(v) = \mathsf{good}$ for all vertices $v \in G(V)$.
Output: A minimum feedback vertex set of $G$.
Step 1 Call $P1(G, S, \mathsf{Col})$. If $P1$ returns NO then return NO.
Step 2 Apply the first step which is applicable:
    Step 2a Let $n'$ be the size of the current graph. If $n' \leq 2n/3$ or if every good vertex $v$
        of degree 3 has at most one good neighbor or if every good vertex $v$ of degree 4 has
        at most two good neighbours, then use brute-force and try all possible solutions
        $S \cup T$, where $T$ is some subset of the good vertices of the current graph, and return
        the one with minimum size.
    Step 2b Find a vertex $u$ of degree 3 with at least two good neighbours, say $v$ and $w$.
        Call the algorithm on following instances and return the smaller solution.
          – Set $S \leftarrow S \cup \{v\}$ and call FVS-D4($G - \{v\}$, $S$, $\mathsf{Col}$).
          – Set $\mathsf{Col}(v) = \mathsf{bad}$ and call FVS-D4($G$, $S$, $\mathsf{Col}$).
    Step 2c Find a vertex $u$ of degree 4 with at least 3 good neighbours, say $v$, $w$ and
        $z$. Here we consider three cases and branch accordingly: 1. $v$ is not part of the
        solution, 2. both $v$ and $w$ are part of the solution, and 3. $v$ is part of the solution
        but $w$ isn't and return the smallest solution.
          – Set $\mathsf{Col}(v) = \mathsf{bad}$ and call FVS-D4($G$, $S$, $\mathsf{Col}$).
          – Set $S \leftarrow S \cup \{v, w\}$ and call FVS-D4($G - \{u, w\}$, $S$, $\mathsf{Col}$).
          – Set $S \leftarrow S \cup \{v\}$ and $\mathsf{Col}(w) = \mathsf{bad}$ and call FVS-D4($G - \{v\}$, $S$, $\mathsf{Col}$).
```

Fig. 5. Algorithm FVS-D4()

Correctness The argument for correctness closely follows the one given for the Vertex Bipartization problem in the previous section and is omitted.

Time Complexity We will now analyze all cases handled by the algorithm carefully and bound the overall time taken by it. We claim that in Step $2a$, the number of good vertices is bounded by $2n/3$. If we reach Step $2a$ then either $n' \leq 2n/3$ or every good vertex of degree three has at most one good neighbor and every good vertex of degree four has at most two good neighbours. In the case when $n' \leq 2n/3$, our claim follows trivially since the number of good vertices is $\leq n'$. As for the second case, we can bound the number of good vertices by counting the number of edges between good vertices and bad vertices. We can have following types of good vertices:

Type 1 Degree 2 good vertex (with both its neighbours bad).
Type 2 Degree 3 good vertex with one good neighbour.
Type 3 Degree 3 good vertex with all its neighbours bad.
Type 4 Degree 4 good vertex with two good neighbours.
Type 5 Degree 4 good vertex with one good neighbour.
Type 6 Degree 4 good vertex with all its neighbours bad.

It should be clear that any good vertex at this stage falls in one of the types mentioned above. Let n_i represent the number of good vertices of Type i and let n_g be the total number of good vertices. Then $n_g = \sum_{i=1}^{6} n_i$.

We will now count the number of good-bad edges in the graph. The total number of good-bad edges is $2n_1 + 2n_2 + 3n_3 + 2n_4 + 3n_5 + 4n_6$. It is easy to see that the quantity $n_2 + 2n_4 + n_5$ counts every good-good edge *twice*. Thus number of good-good edges is $(n_2 + 2n_4 + n_5)/2$. Thus,

$$\begin{aligned} 2n_1 + 2n_2 + 3n_3 + 2n_4 + 3n_5 + 4n_6 \le\ & \frac{4(n - n_1 - n_2 - n_3) + 3(n_2 + n_3) + 2n_1}{2} \\ & - \frac{n_2}{2} - \frac{2n_4}{2} - \frac{n_5}{2} \end{aligned}$$

A simple calculation shows that $3n_g \le 2n$ from which it follows that $n_g \le \frac{2n}{3}$ This shows that number of good vertices in Step $2a$ is bounded by $2n/3$.

In Step $2b$, we have a good vertex u of degree 3 that has at least two good neighbors v and w. When we include v in the solution, the degree of u becomes 2 and since it has a good neighbour w it is removed by the preprocessing step. Thus we end up eliminating at least two good vertices from the graph. If we do not include v in the solution, we label it bad and end up decreasing the number of good vertices by one. Then we have

$$T(n_g) \le T(n_g - 1) + T(n_g - 2).$$

In Step $2c$, we have a vertex u of degree 4 with at least three good neighbours v, w and z. We branch on three cases:

1. v is not in the solution,
2. v and w are in the solution, and
3. v is in the solution but w isn't.

In the first case, v is labelled bad and the number of good vertices reduces by at least 1. When both v, w are part of the solution then, on removing them, u has degree 2 and since it has a good neighbour z it is removed by the preprocessing step. Thus we eliminate at least 3 good vertices in this case. In the last case, v is removed from the graph and w is labelled bad, which reduces the number of good vertices by at least 2. Thus we have the following recurrence on the number of good vertices.

$$T(n_g) \le T(n_g - 1) + T(n_g - 2) + T(n_g - 3).$$

Combining the above, we get the following recurrence for the problem in the worst case, modulo the polynomial time used at every node to find the vertex of required type.

$$\begin{aligned} T(n_g) &\leq T(n_g - 1) + T(n_g - 2) + T(n_g - 3) \\ T(2n_g/3) &= 2^{2n_g/3} \end{aligned}$$

$T(n_g)$ is bounded by $(1.8393)^{n_g/3} \cdot 2^{2n_g/3}$ which is $O^*(1.945^{n_g})$. Setting the initial value of n_g as n we get the following theorem.

Theorem 2. *Let $G = (V, E)$ be an undirected graph with maximum degree 4 with n vertices and m edges. Then the* Feedback Vertex Set *problem on G can be solved exactly using polynomial space and in time $O^*(1.945^n)$.*

We can modify our algorithm for the Feedback Vertex Set presented above for the Vertex Bipartization problem in graphs of maximum degree 4. The only modification we need to do is to call the preprocessing algorithm for Vertex Bipartization problem in Step 1. This gives us following theorem.

Theorem 3. *Let $G = (V, E)$ be an undirected graph with maximum degree 4 with n vertices and m edges. Then the* Vertex Bipartization *problem on G can be solved exactly using polynomial space and in time $O^*(1.945^n)$.*

3 Using FPT algorithms to design exact algorithms

In the last section, we gave efficient algorithms for Vertex Bipartization and Feedback Vertex Set, but they critically used the fact that the maximum degree of the graphs is 3 or 4. Here we give a general technique of designing exact algorithms using parameterized algorithms as a subroutine and apply it to several problems. Let Q be an NP-optimization problem and suppose that it's parameterized version (Q, k) is fixed parameter tractable. Let us also suppose that the FPT algorithm $\mathscr{A}$ for (Q, k) has a time complexity of the form $O^*(c^k$, where k is the parameter, and c is a constant. This algorithm $\mathscr{A}$ immediately gives us an exact algorithm for Q with time complexity $O^*(c^n)$. What is interesting is that there actually exists an exact algorithm for Q with time complexity $O^*(d^n)$, where $d < c$. Moreover, if $c < 4$ then we will show that $d < 2$.

This fact has an interesting consequence. There are many optimization problems such as Max Independent Set, Min Vertex Cover, Min Feedback Vertex Set which have trivial brute-force enumeration algorithms of time complexity 2^n. If the parameterized versions of any of these problems is solvable in time $O^*(c^k)$, where $c < 4$ then we immediately obtain exact algorithms for these problems which are better than the trivial brute-force algorithms. We will show that this technique simplifies exact algorithms for many optimization problems and for some (e.g. Vertex Bipartization) gives the best known exact algorithm.

Our algorithm makes clever use of the FPT algorithm $\mathscr{A}$ and brute-force enumeration. Consider a problem such as VERTEX BIPARTIZATION. Had we used brute-force throughout, then the time complexity would have been $O^*(\sum_{i=0}^{i=n} \binom{n}{i})$ $= O^*(2^n)$. It is well known that the function $\binom{n}{i}$ increases with increasing i, attains a maximum at $i = n/2$, and then decreases. Also, it is symmetric in that $\binom{n}{i} = \binom{n}{n-i}$. Brute-force pushes the time complexity to $O^*(2^n)$ because it is costlier to search exhaustively when i is near $n/2$, since $\binom{n}{n/2} \approx 2^n$. Therefore, if we adopt the strategy of using brute-force only for those values of i which are far removed from $n/2$ and using the FPT algorithm $\mathscr{A}$ for the remaining i values (that is, those near $n/2$), then we might end up with an exponential time complexity better than that of $\mathscr{A}$. And indeed we do. Our algorithm is given in Figure 6. For simplicity the algorithm considers minimization problems only. For maximization problems we can modify the algorithm to output the largest i for which there exists a solution.

```
Algorithm Exact($Q$,$\mathscr{A}$,$c$)
$Q$ is a minimization problem and $\mathscr{A}$ is the FPT algorithm that solves its parameterized version in time $O^*(c^k)$, where $c$ is a constant and $k$ is a parameter.
Here $n$ is the input size.
Compute $\lambda$ from the equation $c^{n\lambda} = \binom{n}{n-\lambda n}$.
 for $i = 1$ to $\lambda n$
          use the FPT algorithm $\mathscr{A}$ for $Q$ to check whether there is solution of
          size $i$; if yes output $i$ and halt.
 for $i > \lambda n$
          use brute-force to check whether there exists a solution of size $i$; if yes,
          then output $i$ and halt.
```

Fig. 6. Algorithm Exact()

Suppose the FPT algorithm $\mathscr{A}$ for Q takes $O^*(c^k)$ time, where c is some constant. Then from the description of Algorithm Exact, it is easy to observe that its time complexity is upper bounded by following:

$$O^*\left(\max\left\{c^{\lambda n}, \binom{n}{n-\lambda n}\right\}\right) \tag{1}$$

Now suppose that the trivial brute-force algorithm for Q has time complexity $O^*(2^n)$. We show that if we want Algorithm Exact to beat this trivial time bound then we must have $c < 4$. We need a lemma.

Lemma 1. *Let* $\frac{1}{2} < \lambda < 1$. *Then* $\binom{n}{n-\lambda n}$ *is bounded by* d^n, *where* d *is some constant* < 2.

Proof. We know that

$$\binom{n}{n-\lambda n} = \binom{n}{\lambda n} \le \frac{n^n}{(\lambda n)^{\lambda n}((1-\lambda)n)^{(1-\lambda)n}} = \left(\left(\frac{1}{\lambda}\right)^{\lambda}\left(\frac{1}{1-\lambda}\right)^{1-\lambda}\right)^n$$

One can easily verify using calculus that the function

$$h(\lambda) = \left(\frac{1}{\lambda}\right)^{\lambda}\left(\frac{1}{1-\lambda}\right)^{1-\lambda} \quad (0 < \lambda < 1)$$

attains a maximum of 2 at $\lambda = 1/2$. At other points in the interval $(\frac{1}{2}, 1)$ it has a value less than 2. This proves the claim. $\square$

Equating c^{λ} and $h(\lambda) = d$ we get $c = \{h(\lambda)\}^{1/\lambda} < 2^{1/\lambda}$. Since $1/2 < \lambda < 1$ we see that $c < 4$. Also note that $d = c^{\lambda} < c$. We thus have the following theorem.

Theorem 4. *Let Q be an* NP*-optimization problem with input size n such that a trivial brute force algorithm for Q takes time $O^*(2^n)$. Suppose also that the parameterized version of Q is* FPT *with time complexity $O(c^k poly(n))$ for some $c < 4$. Then there is an exact algorithm for Q with time complexity $O^*(d^n)$ for some $d < c$ and $d < 2$.*

3.1 Applications

In this section, we apply the algorithm developed in Section 3 to various problems and obtain exact algorithms with nontrivial worst-case time bounds. The problems for which we give efficient exact algorithms include the Vertex Bipartization problem in general undirected graphs, the 3 and 4-Hitting Set problems, the Feedback Set problem in tournaments and the Max Cut problem in graphs with average degree 3 and 4. Some of these results are new and some of them are given here to show the applicability of the theorems and algorithms developed in the Section 3.

Vertex Bipartization Problem We apply the algorithm developed in Section 3 to the Vertex Bipartization problem in general undirected graphs. This problem can be solved exactly in $O^*(2^{|V|})$ time. Reed, Kaleigh, and Vetta [10] have recently given an FPT algorithm for the parameterized version of this problem with running time $O(3^k kmn)$. If we use their FPT algorithm directly to solve the optimum version of the problem we will take time $O^*(3^n)$ which is worse than that taken by the trivial exponential time algorithm. However, if we use the algorithm in Figure 6 then with $c = 3$, $\lambda = 0.6091$ and we get a running time of $O^*(1.9526^n)$. The time taken by the first for-loop is $O^*(3^{0.6091n})$ which is $O^*(1.9526^n)$. The second step takes $O^*(\binom{n}{0.6091n})$ time which works out to $O^*(1.9526^n)$. We therefore have the following theorem.

Theorem 5. *Let $G = (V, E)$ be an undirected graph with n vertices then* Vertex Bipartization *problem can be solved in time $O^*(1.9526^n)$.*

3- and 4-Hitting Set Problems The HITTING SET (HS) problem is defined as follows:

Instance	A finite family of sets $S_1, S_2, \ldots, S_m$ comprised of elements from a universal set U.
Goal	Find a minimum sized subset $T \subseteq U$ such that $S_i \cap T \neq \emptyset$ for all i.

The 3- and 4-HS problems are special cases of the HITTING SET problem. In the 3-HS problem $|S_i|$ $(1 \leq i \leq m)$ is bounded by 3 and in the 4-HS problem by 4. The parameterized versions of these problems have been shown to be fixed parameter tractable by Neidermeier et al [7]. The main results in [7] can be summarized in the following theorem.

Theorem 6. *[7] The parameterized version of the* 3*-HS and the* 4*-HS problem can be solved in time* $O^*(2.27^k)$ *and* $O^*(3.3^k)$ *respectively.*

Recently Wahlström [13] proposed an exact algorithm for the 3-HS problem with time complexity $O^*(1.6316)$. This algorithm makes use of the parameterized algorithm of Neidermeier et al [7] but only when there is an upper bound on the size of the optimum solution set.

We can apply Algorithm Exact() to solve the 3- and 4-HS problems; the parameterized algorithm we use is the one by Neidermeier et al [7] with $\lambda = 0.72$ for the 3-HS problem and $\lambda = 0.5721$ for the 4-HS problem. This gives us the following.

Theorem 7. *The* 3*- and* 4*-*HITTING SET *problems can be solved exactly in time* $O^*(1.80933)^n$ *and* $O^*(1.9799^n)$*, where* $n = |U|$.

The algorithm for the 3-HS problem in [13] does not directly generalize to the 4-HS problem. To the best of our knowledge our algorithm is the first exact algorithm for the 4-HS problem with the base of the exponent less than 2.

Feedback Set Problems in Tournaments The FEEDBACK ARC (VERTEX) SET problem in directed graphs is defined as follows:

Instance	A directed graph $G = (V, E)$.
Goal	Find a minimum sized subset $F \subseteq E$ $(F \subseteq V)$ such that $G' = (V, E - F)$ $(G' = (V - F, E'))$ is acyclic.

We will use the parameterized algorithms developed by Raman and Saurabh [9] for feedback set problems in tournaments. They give an $O^*(2.415^k)$ and $O^*(2.27^k)$ algorithm for the FEEDBACK ARC SET and the FEEDBACK VERTEX SET problem respectively. For the FEEDBACK ARC SET problem in tournaments we get $\lambda = 0.696$; for the FEEDBACK VERTEX SET we get $\lambda = 0.72$. Then using Algorithm Exact() we obtain the following theorem.

Theorem 8. *Let* $G = (V, E)$ *be a tournament with* n *vertices and* m *arcs. Then the minimum feedback arc set and feedback vertex set can be found in time* $O^*(1.84821^m)$ *and* $O^*(1.80933^n)$ *respectively.*

Observe that in any directed graph, the size of the minimum feedback arc set is at most $m/2$. This fact combined with Algorithm Exact() gives us the following theorem.

Theorem 9. *Let $G = (V, E)$ be a tournament with n vertices and m arcs. Then the minimum feedback arc set can be found in time $O^*(1.5541^m)$.*

Max Cut in graphs of average degree 3 or 4 We will solve the MAX CUT problem using the parameterized algorithm for EDGE BIPARTIZATION developed by Niedermeier et al [6] and a lower bound on the size of the maximum cut. The parameterized algorithm for EDGE BIPARTIZATION presented in [6] has time complexity $O(2^k \cdot n^{O(1)})$. Recall that an instance of the EDGE BIPARTIZATION problem is an undirected graph $G = (V, E)$ and the question is to find a minimum set of edges that needs to be deleted from G to make it bipartite. The relationship between this problem and the MAX CUT problem is straightforward: the maximum cut in a graph is $E-$the minimum set of edges to make the graph bipartite. Thus solving the EDGE BIPARTIZATION problem is equivalent to solving the MAX CUT problem.

Poljak et al in [8] give a lower bound of $\frac{m}{2} + \frac{1}{2}\left\lceil \frac{n-c}{2} \right\rceil$ for maximum cut, where c is number of connected components and m and n are, respectively, the number of edges and vertices in the graph. If G is a connected graph on n vertices and m edges with average degree 3, then $m = 1.5n$. The minimum number of edges to be removed from G to make it bipartite is then

$$m - |\text{max cut}| \le m - \left(\frac{m}{2} + \frac{n-1}{4}\right) = \frac{3n}{2} - \left(\frac{3n}{4} + \frac{n-1}{4}\right) = \frac{n}{2} + \frac{1}{4}.$$

We now use the algorithm in [6] to solve the EDGE BIPARTIZATION problem. Because of the upper bound on the the number of edges needed to be removed, we achieve a time complexity of $O^*(2^{n/2}) = O^*(1.414^n)$. Had G been of average degree 4, then the upper bound on the number of edges to be deleted would be $3n/4+1/4$ and the time complexity of solving the EDGE BIPARTIZATION problem would be $O^*(2^{3n/4}) = O^*(1.6818^n)$. We thus have the following theorem.

Theorem 10. *The Max Cut problem can be solved exactly in time $O^*(1.4141^n)$ in graphs with average degree 3 and in time $O^*(1.6818^n)$ in graphs with average degree 4. Both of these algorithms take polynomial space.*

Neidermeir et al [5] achieve the same time bound for graphs with maximum degree 3 and a better time bound of $O^*(1.5871^n)$ for graphs with maximum degree 4.

4 Conclusion

In this paper, we have obtained improved exact algorithms for several problems including VERTEX BIPARTIZATION, 4-HITTING SET, FEEDBACK VERTEX SET in graphs with maximum degree at most 4, FEEDBACK ARC SET in tournaments. We introduced two general techniques to obtain efficient exact algorithms. One of these is a modified version of the general branch-and-bound technique and the other one is based on parameterized complexity algorithms. Further reduction in the base of the exponent of all these algorithms remains open.

We leave it open to explore the practical performance of these algorithms. Another major open problem is to devise an exact algorithm with time complexity less than $O^*(2^n)$ for the FEEDBACK VERTEX SET problem in general undirected graphs.

References

1. H. CHOI, K. NAKAJIMA AND C. S. RIM. *Graph Bipartization and Via Minimization.* SIAM Journal of Discrete Mathematics 2 (1): 38-47, 1989.
2. R. DOWNEY AND M. FELLOWS. *Parameterized Complexity.* Springer-Verlag (1998).
3. R. DOWNEY AND M. FELLOWS. *Parameterized Complexity for the Skeptic.* In Proc. of 18th CCC : 147-169, 2003.
4. N. GARG, V. VAZIRANI AND M. YANNAKAKIS. *Approximate Max-Flow Min-(Multi) Cut Theorems and Their Applications.* SIAM Journal on Computing 25 (2): 235-251, 1996.
5. J. GRAMM, E. A. HIRSCH, R. NIEDERMEIER, AND P. ROSSMANITH *Worst-case upper bounds for MAX-2-SAT with an application to MAX-CUT.* Discrete Applied Mathematics 130 (2): 139-155, 2003.
6. J. GUO, J. GRAMM, F. HÜFFNER, R. NEIDERMEIER AND S. WERNICKE. *Improved Fixed-Parameter Algorithms for Two Feedback Set Problems.* Manuscript.
7. R. NIEDERMEIER AND P. ROSSMANITH. *An effiecient fixed parameter algorithm for 3-Hitting Set.* Journal of Discrete Algorithms 1 (1): 89-102, 2003.
8. S. POLJAK AND D. TURZIK. *A Polynomial Algorithm for Constructing a Large Bipartite Subgraph, with an Application to a Satisfiability Problem.* Canad. J. Math 34 (3): 519-524, 1982.
9. V. RAMAN, AND S. SAURABH. *Parameterized Algorithms for Feedback Set Problems and Their Duals in Tournaments.* To appear in TCS.
10. B. REED, K. SMITH AND A. VETTA. *Finding Odd Cycle Transversals*, Operations Research Letters 32 (2004) 299-301.
11. R. RIZZI, V. BAFNA, S. ISTRAIL AND G. LANCIA. *Practical Algorithms and Fixed-Parameter Tractability for the Single Individual SNP Haplotyping Problem.* WABI: 29-43, 2002.
12. S. UENO, Y. KAJITANI AND S. GOTOH. *On the Nonseparating Indepedent Set Problem and Feedback Set Problem for Graphs with no Vertex Degree Exceeding Three.* Discrete Mathematics 72 (1988) 355-360.
13. M. WAHLSTRÖM. *Exact algorithms for finding minimum transversals in rank-3 hypergraphs.* Journal of Algorithms 51(2): 107 - 121, 2004.
14. G. WOEGINGER. *Exact algorithms for NP-hard problems: A survey.* In *Combinatorial Optimization—Eureka! You shrink!* Springer LNCS 2570: 185-207, 2003.
15. X. ZHUNAG AND S. PANDE. *Resolving Register Bank Conflicts for a Network Processor.* In Proc. of 12th PACT: 260-278, 2003.