Efficient Exact Algorithms through Enumerating Maximal Independent Sets and Other Techniques ⋆

Venkatesh Raman, Saket Saurabh, and Somnath Sikdar

The Institute of Mathematical Sciences, Chennai 600 113.
{vraman|saket|somnath}@imsc.res.in

Abstract. We give substantially improved exact exponential-time algorithms for a number of NP-hard problems. These algorithms are obtained using a variety of techniques. These techniques include:

- obtaining exact algorithms by enumerating maximal independent sets in a graph,
- obtaining exact algorithms from parameterized algorithms, and
- a variant of the usual branch-and-bound technique which we call the 'colored' branch-and-bound technique.

These techniques are simple in that they avoid detailed case analyses and yield algorithms that can be easily implemented. We show the power of these techniques by applying them to several NP-hard problems and obtaining new improved upper bounds on the running time. The specific problems that we tackle are: (1) the Odd Cycle Transversal problem in general undirected graphs, (2) the Feedback Vertex Set problem in directed graphs of maximum degree 4, (3) the Feedback Arc Set problem in tournaments, (4) the 4-Hitting Set problem, (5) the Minimum Maximal Matching and the Edge Dominating Set problems. The algorithms that we present for these problems are the best known and are a substantial improvement over previous best results. For example, for the Minimum Maximal Matching we give an $O^*(1.4425^n)$ algorithm improving the previous best result of $O^*(1.4422^m)$ [35]. For the Odd Cycle Transversal problem, we give an $O^*(1.62^n)$ algorithm which improves the previous time bound of $O^*(1.7724^n)$ [3].

1 Introduction

An important goal of the theory of algorithms is to design efficient algorithms that solve computationally difficult problems. When considering the class of NP-hard combinatorial optimization problems, the goal "appears" beyond our reach. As these problems need to be solved *exactly* for various applications in theory and practice routinely, exact exponential-time algorithms become unavoidable. In recent years, there has been a growing interest in designing exact algorithms for NP-complete problems. This has led to the development of fast exponential-time algorithms for various problems including Satisfiability [20, 23, 40], Graph Coloring [4, 9], Maximum Independent Set [37] and many others. These exponential-time algorithms lead to practical algorithms for at least moderate instance sizes. See recent surveys by Woeginger [41, 42] and Fomin et al. [15] for an overview and recent developments.

In this paper, we identify three techniques that have been known to obtain efficient exact algorithms. We modify these techniques and show the power of these modifications by obtaining substantially improved bounds on several NP-hard problems.

In Section 2, we illustrate the idea of designing exact algorithms by *enumerating maximal independent sets (MIS)* in a graph. Algorithms for enumerating maximal independent sets in a graph have been at the heart of many exact algorithms for the Graph Coloring problem [9, 4, 2]. In this paper, we extend the power of this technique and obtain significantly improved exact algorithms

⋆ A preliminary version of this work was presented in ICTCS 2005 and appears in the proceedings of the conference [33].

for (1) Odd Cycle Transversal, (2) Minimum Maximal Matching, (3) Minimum Edge Dominating Set and (4) Matrix Dominating Set. We also give an efficient exponential approximation algorithm with an additive factor of 1 to find the chromatic number of a graph. Though all our algorithms use the subroutine for enumerating maximal independent sets, we transform the problems or use interesting structural characterizations to make use of the MIS enumeration algorithm.

In Section 3, we develop a technique that consists of obtaining an exact algorithm for a problem by using a *parameterized algorithm* for the same problem. Before we describe our technique, we give a very brief introduction to parameterized complexity and try to bring out the basic difference between exact exponential-time and parameterized algorithms.

A parameterized problem is a decision problem and consists of a tuple $\langle \pi, k \rangle$, where π is the problem instance and k is the parameter. Typically π is an instance of an optimization problem and k is an integer. The question asked is: Does π have an optimal solution of size at least k (in case of maximization problems), or at most k (in case of minimization problems). In the framework of parameterized complexity, the run time of an algorithm is viewed as a function of two quantities: the size of the problem instance *and* the parameter.

A parameterized problem is said to be *fixed parameter tractable (FPT)* 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. The class of all fixed parameter tractable problems is denoted by FPT. Note that if the parameter k is fixed, then the run time is a polynomial in the input size.

There is also a theory of hardness or intractability known as *W-hardness* using which one can identify parameterized problems that are not amenable to such algorithms. For a comprehensive introduction to parameterized complexity, we refer readers to the book by Downey and Fellows [7]. For recent developments in the area of parameterized complexity, see the surveys by Chen [6] and Downey [8]. Some recent additions to parameterized complexity literature are the books by Flum and Grohe [13] and Niedermeier [29].

From the above discussion it is clear that both parameterized and exact exponential-time algorithms are part of the "exact algorithms" paradigm. Yet they are different paradigms in that a parameterized algorithm finds a solution whose size is governed by the *parameter* that is supplied as part of the input whereas an exact exponential algorithm solves the *optimization version* of the problem. From the viewpoint of exact exponential-time algorithms, Vertex Cover and Independent Set problems are identical as we can obtain an optimal solution to one from an optimal solution to the other. On the other hand, the parameterized complexity of these two problems are very different. Vertex Cover is known to be FPT whereas Independent Set is known to be $W[1]$-hard, evidence that it does not have an algorithm of time complexity $O(f(k)\cdot|\pi|^{O(1)})$.

We develop a technique by which we can use a parameterized algorithm of time complexity $O^*((4-\epsilon)^k)$,[1] where k is the parameter and $\epsilon > 0$, to obtain an exact algorithm of time complexity $O^*((2-\eta)^n)$, $\eta > 0$. This technique is based on a careful use of the parameterized algorithm for certain values of the parameter and brute-force for other values. In Section 3.1, we give several applications of this technique.

[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[41] for a detailed discussion on this.

Finally in Section 4, we present the *colored branch-and-bound* technique. The 'colored' branch-and-bound technique is a generalization of the usual branch-and-bound technique. When we apply the branch-and-bound technique to a graph theoretic problem, we typically choose a vertex $v \in V$ of the input graph $G = (V, E)$ and recurse on two cases: in one case, we include v in our partial solution and look for an optimal solution extending this partial solution; in the other case, we exclude v from the solution and look for an optimal solution not containing v.

In a typical scenario, we are able to delete a few vertices from the graph in each branch of the recursion and so the size of the graph that we work on in each branch decreases. The reduction in the number of vertices in each branch dictates the recurrence for running time. The larger the number of vertices deleted, the better the running time. But for the problems that we consider, we cannot delete a vertex when we exclude it from the solution. Thus if we apply the usual branch-and-bound technique, we will end up making no progress along one branch. To overcome this, we label the vertices good and bad and argue that in both branches of the recursion, we make progress as the number of good vertices decreases. The idea of coloring vertices has been applied extensively on the Dominating Set problem [1, 32]. We give a detailed description of the colored branch-and-bound technique in Section 4 and illustrate it by applying it to the Feedback Vertex Set problem in directed graphs of maximum degree 4.

In Section 5, we conclude with some remarks and open problems. For the rest of the paper, given a graph $G = (V, E)$ (undirected or directed), we let n represent the number of vertices and m the number of edges or arcs. For a subset of vertices $V' \subseteq V$, we let $G[V']$ represent the subgraph of G induced by V'.

2 Exact Algorithms via Enumerating Maximal Independent Sets

In this section, we describe exact algorithms that are based on a structural characterization of the optimal solution in terms of maximal independent sets. We first list out some known results that we make use of. We then describe exact algorithms for the following problems (1) Odd Cycle Transversal, (2) Minimum Maximal Matching, (3) Minimum Edge Dominating Set. Finally, we give an exponential-time approximation algorithm for the Chromatic Number problem with an additive error of 1.

2.1 Preliminaries

The first two theorems below provide bounds on the total number of maximal independent sets of size k and the third bounds the total number of maximal independent sets in a graph.

Theorem 1. [10] *The number of maximal independent sets of size k in a graph is at most $3^{4k-n}4^{n-3k}$.*

Theorem 2. [4] *Let $G = (V, E)$ be a graph on n vertices. The maximum number of maximal independent sets of size k in G is*

$$N(n, k) = \lfloor n/k \rfloor^{(\lfloor n/k \rfloor + 1)k - n}(\lfloor n/k \rfloor + 1)^{n - \lfloor n/k \rfloor k}.$$

In fact, all maximal independent sets of size at most k can be listed in time proportional to $N(n, k)$.

Theorem 3. [24, 28] *Let $G = (V, E)$ be a graph on n vertices, then G contains at most $3^{n/3} = (1.44225)^n$ maximal independent sets. All the maximal independent sets can be enumerated in $O^*(1.44225^n)$ time.*

We also need the following two theorems.

Theorem 4. [37] *Let $G = (V, E)$ be a graph on n vertices, then a maximum independent set of G can be found in time $O^*(2^{n/4})$.*

Theorem 5. [2] *Given a graph $G = (V, E)$ on n vertices, the chromatic number of G can be computed in $O^*(2.3236^n)$ time and $O^*(2^n)$ space. In fact, an optimal coloring can also be produced in this time.*

2.2 Minimum Odd Cycle Transversal

The Odd Cycle Transversal problem is defined below:

> Odd Cycle Transversal (OCT): Given a graph $G = (V, E)$, find a subset $O \subseteq V$ of minimum size such that $G[V - O]$ is bipartite.

Byskov [3] gave an algorithm that enumerates all minimal OCT's in a graph with running time $O^*(1.7724^n)$. We give an improved algorithm to find a minimum sized OCT which can be modified to count all minimum sized OCT's. We first give a characterization of a minimum odd cycle transversal (OCT) in terms of maximal independent sets and then use it to obtain an improved exact algorithm.

Theorem 6. *Let $G = (V, E)$ be a connected undirected graph and let O be a minimum odd cycle transversal of G. Then $V - O$ can be partitioned into V_1 and V_2 such that V_1 is a maximal independent set of G and V_2 is a maximum independent set of $G[V - V_1]$.*

Proof. Since O is an odd cycle transversal of G, $G[V - O]$ is bipartite. Let K and L be a bipartition of $G[V - O]$. Add isolated vertices (if any) from either K to L or from L to K such that one of them, say K, becomes a maximal independent set of $G - O$. Note that since O is a minimum odd cycle transversal of G, for every vertex $x \in O$ there exists a witness odd cycle C_x such that $C_x \cap O = \{x\}$. Hence every vertex $x \in O$ has at least one neighbor in K and one in L. This shows that K is also maximal in G. Set $V_1 = K$ and $V_2 = L$. It remains to show that V_2 is a maximum independent set of $G[V - V_1]$. Suppose not. Let W be a maximum independent set of $G[V - V_1]$. Then $|W| > |V_2|$. Observe that $(V - V_1 - W)$ is an odd cycle transversal of G. Now

$$\begin{aligned} |V - V_1 - W| &= |V| - |V_1| - |W| && (\text{since } V_1 \cap W = \emptyset) \\ &= |O| + |V_2| - |W| && (\text{since } |V| = |V_1| + |V_2| + |O|) \\ &< |O| && (\text{since } |V_2| < |W|) \end{aligned}$$

This contradicts the fact that O is a minimum odd cycle transversal of G. □

Observe that any other *maximum* independent set of $G - V_1$, say T, also gives us a minimum odd cycle transversal. This is because $(V - V_1 - T)$ is an OCT of size $|V - V_1 - T| = |V - V_1 - V_2|$. This observation is the basis of the algorithm in Figure 1.

The correctness of the algorithm follows from Theorem 6. Theorems 3 and 4 give, respectively, a bound on the total number of maximal independent sets enumerated in Step 2 and the time taken to find a maximum independent set in Step 3*a*. Hence, the time complexity of the algorithm is upper bounded by

$$\sum_{k=1}^{n} |MIS_k| 2^{(n-k)/4} \;\leq\; 3^{n/3} 2^{n/4} \;\leq\; 1.7152^n, \tag{1}$$

Algorithm $\mathsf{Min\ OCT}(G)$
Input: A connected undirected graph $G = (V, E)$.
Output: A minimum odd cycle transversal of G.

Step 1 Set $X \leftarrow V$ (X stores a minimum odd cycle transversal.)
Step 2 Enumerate all maximal independent sets of G, and for each MIS I,
  Step 3a Find a maximum independent set of $G - I$, say I'.
  Step 3b If $|X| > |V - I - I'|$ then set $X = V - I - I'$.
Step 4 Return X.

Fig. 1. Algorithm for finding a minimum odd cycle transversal of a graph.

where MIS_k represents the set of all maximal independent sets of size k. Thus this algorithm takes $O^*(1.7152^n)$ which improves the $O^*(1.7724^n)$ time algorithm of Byskov [3].

We next present a refined time analysis of the algorithm using tighter bounds on the number of maximal independent sets of small size.

A Refined Timing Analysis We break up the summation in equation (1) as follows

$$\sum_{k=1}^{n} |MIS_k| 2^{(n-k)/4} = \sum_{k=1}^{\lfloor 0.33n \rfloor} |MIS_k| 2^{(n-k)/4} + \sum_{k=\lfloor 0.33n \rfloor + 1}^{n} |MIS_k| 2^{(n-k)/4} \qquad (2)$$

We will bound each term on the right side of (2) individually. To bound the first term, we use the bound in Theorem 1 on the number of maximal independent sets of size k: $N(k) \leq 3^{4k-n}4^{n-3k}$ for $1 \leq k \leq \lfloor 0.33n \rfloor$. It is easy to verify that $N(k)$ is an increasing function of k. Thus,

$$\begin{aligned}
\sum_{k=1}^{\lfloor 0.33n \rfloor} |MIS_k| 2^{(n-k)/4} &\leq \sum_{k=1}^{\lfloor 0.33n \rfloor} 3^{4k-n} 4^{n-3k} 2^{(n-k)/4} \\
&\leq \left(\frac{4.2^{1/4}}{3} \right)^n \cdot \sum_{k=1}^{0.33n} \left(\frac{3^4}{4^3.(2^{1/4})} \right)^k \\
&\leq (0.33n)(1.58561)^n \left(\frac{3^4}{4^3.(2^{1/4})} \right)^{0.33n} \quad \leq O^*(1.62^n)
\end{aligned}$$

To bound the second term, we use the bound on the total number of maximal independent sets in a graph, which is $3^{n/3}$. This gives,

$$\begin{aligned}
\sum_{k=\lfloor 0.33n \rfloor + 1}^{n} |MIS_k| 2^{(n-k)/4} &\leq 3^{n/3} \sum_{k=\lfloor 0.33n \rfloor + 1}^{n} 2^{(n-k)/4} \\
&\leq (0.67n) 3^{n/3} 2^{(0.67n)/4} \quad \leq O^*(1.62^n)
\end{aligned}$$

This gives us following theorem.

Theorem 7. *Let $G = (V, E)$ be an undirected graph with n vertices. A minimum odd cycle transversal of G can be found in $O^*(1.62^n)$ time.*

Counting all Minimum Odd Cycle Transversals The algorithm for ODD CYCLE TRANSVERSAL presented in the previous section can be generalized to the *counting version* of the problem (#OCT) where the objective is to count all minimum sized odd cycle transversals. To do this, the only modification we need is to use an algorithm to *count* all maximum independent sets in Step $3a$ of the minimum odd cycle transversal algorithm in Figure 1. Using the $O^*(1.2461^n)$ time algorithm developed by Fürer and Kasiviswanathan [17] to count all maximum independent sets in a graph, we obtain the following theorem

Theorem 8. *Let $G = (V, E)$ be an undirected graph on n vertices. Then we can count all minimum odd cycle transversals in $O^*(1.6713^n)$ time.*

2.3 Minimum Maximal Matching

A MINIMUM MAXIMAL MATCHING (abbreviated MMM) of a graph $G = (V, E)$ is a maximal matching of minimum cardinality. In this section, we first give an algorithm which finds a minimum maximal matching of G in $O^*(2^n)$ time and then improve this to obtain an $O^*(1.44225^n)$ time algorithm. This improves an $O^*(1.4422^m)$ time algorithm of Randerath and Schiermeyer [35] for the problem. We need a lemma.

Lemma 1. *Given a graph $G = (V, E)$, let M be a maximal matching of G. Let $U = \{v \in V : v \text{ is an end point of some edge of } M\}$. If M' is any maximum matching of $G[U]$ then M' is a maximal matching of G and $|M'| = |M|$.*

Proof. Since M is a perfect matching of $G[U]$, it is a matching of maximum size in $G[U]$, and so $|M'| = |M|$. To see that M' is maximal in G, observe that U is a vertex cover of G and that the vertex set of M' is precisely U. Thus M' cannot be augmented any further proving that it is maximal. □

Corollary 1. *Let $G = (V, E)$ be a graph and M be a minimum maximal matching of G. Let U be a subset of vertices containing the endpoints of M. If M' is any maximum matching of $G[U]$ then M' is a minimum maximal matching of G.*

Thus to find a minimum maximal matching of G, we enumerate all even-cardinality subsets $U \subseteq V$, find a maximum matching M of $G[U]$, check for M's maximality in G and finally output the one with minimum size. Note that at any point we store at most two subsets of V, so our algorithm takes polynomial space. This gives us the following theorem.

Theorem 9. *Let $G = (V, E)$ be a graph on n vertices. We can find a* MINIMUM MAXIMAL MATCHING *of G in $O^*(2^n)$ time and space polynomial in n.*

Improved Algorithm We provide a characterization of minimum maximal matchings in terms of minimal vertex covers (complement of maximal independent sets) which is used to obtain an improved algorithm.

Theorem 10. *Given any graph $G = (V, E)$, there exists a minimal vertex cover U of G such that if*

1. *M_1 is a maximum matching of $G[U]$, and*
2. *M_2 is a maximum matching of $G[V - R]$, where R is the set of endpoints of M_1,*

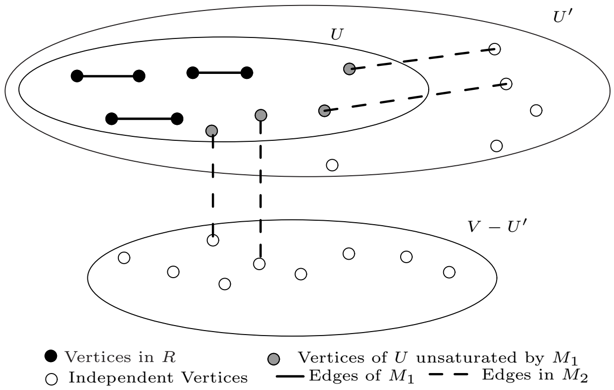

Fig. 2. Characterization of minimum maximal matching.

then $M_1 \cup M_2$ *is a minimum maximal matching of* G.

Proof. Let M be a minimum maximal matching of G and let U' be the set of endpoints of M. Since M is a maximal matching of G, U' is a vertex cover of G and hence contains a minimal vertex cover $U \subseteq U'$ of G. See Figure 2. Let M_1 and M_2 be as stated in the theorem. Note that M_1 and M_2 are disjoint and that $N = M_1 \cup M_2$ is a maximal matching of G. We will show that N is actually a minimum maximal matching by showing that $|N| \leq |M|$.

Since U is a vertex cover of G, every edge of M either has both its endpoints in U or has one endpoint in U and the other in $V - U$. Let K be the set of edges of M with both endpoints in U and L the set of edges of M with one endpoint in U and the other in $V - U$. K is a matching of $G[U]$ and since M_1 is a matching of $G[U]$ of maximum size, $|K| \leq |M_1|$. Recall that U' is the set of endpoints of M. Since $U \subseteq U'$, every vertex of U is an endpoint of an edge in either K or in L. Therefore, $|L| = |U| - 2|K|$. A similar argument shows that $2|M_1| + |M_2| \leq |U|$. Suppose $|M_1| = |K| + r$, $r \geq 0$. Then,

$$\begin{aligned}
|N| = |M_1| + |M_2| &\leq |M_1| + (|U| - 2|M_1|) \\
&= |K| + r + |U| - 2(|K| + r) \\
&= |K| + (|U| - 2|K|) - r \\
&= |K| + |L| - r && \text{(Since, } |L| = |U| - 2|K|) \\
&\leq |M| && \text{(since } r \geq 0).
\end{aligned}$$

This proves the theorem. □

We use this characterization to get an algorithm to find a minimum maximal matching of G. The algorithm is given in Figure 3. The correctness of the algorithm follows from Theorem 10 and the fact that complement of a maximal independent set is a minimal vertex cover. The time complexity of the algorithm is dominated by the enumeration of maximal independent sets which by Theorem 3 can be done in $O^*(1.44225^n)$ time. This gives us following theorem:

Theorem 11. *Let* $G = (V, E)$ *be a graph on* n *vertices. We can find a* Minimum Maximal Matching *of* G *in* $O^*(1.44225^n)$ *time.*

Algorithm MMM(G)
Input: A connected undirected graph G.
Output: A minimum maximal matching of G.

Step 0 Find a maximal matching P of G
Step 1 Set $X \leftarrow P$ (X stores a minimum maximal matching of G.)
Step 2 Enumerate all maximal independent sets of G, and for each MIS I,
  Step 3a Find a maximum matching of $G[V - I]$, say M'.
  Step 3b Find a maximum matching M'' of $G[V - R]$, where R is the set of vertices containing the end points of M'.
  Step 3c If $|X| > |M \cup M''|$ then set $X = M \cup M''$.
Step 4 Return X.

Fig. 3. Algorithm for finding a minimum maximal matching of a graph.

2.4 Minimum Edge Dominating Set

A minimum edge dominating set (MEDS), D, of a graph $G = (V, E)$ is a set of edges of minimum size such that every edge of G is either in D or is adjacent to some edge in D. An algorithm of time complexity $O^*(1.4422^m)$ appears in [35]. Our algorithm depends on a classical result which shows that every minimum maximal matching is a MEDS [18].

Lemma 2. [18] *Let $G = (V, E)$ be a graph. Then every minimum maximal matching of G is a minimum edge dominating set.*

Lemma 2 and Theorem 11 give us the following theorem.

Theorem 12. *Given a graph G on n vertices, the* Minimum Edge Dominating Set *problem on G can be solved in time $O^*(1.44225^n)$.*

Matrix Domination Set

Problem: Matrix Domination (MMD)
Instance: An $m \times n$ matrix M with entries from $\{0, 1\}$.
Question: Find a minimum set of non-zero entries in M that dominates all others, i.e., a minimum size subset $C \subseteq \{1, 2 \dots m\} \times \{1, 2 \dots n\}$ such that $M_{i,j} = 1$ for all $(i, j) \in C$ and such that, whenever $M_{i,j} = 1$, there exists an $(i', j') \in C$ for which either $i = i'$ or $j = j'$.

It is well known that this problem can be reduced to finding a minimum edge dominating set in a bipartite graph. Given an $m \times n$ matrix M construct a bipartite graph $B = (U \uplus V, E)$ with $U = \{u_1, u_2, \cdots, u_m\}$ and $V = \{v_1, v_2, \cdots, v_n\}$ such that there is an edge between u_i and v_j if and only if $M[i, j] = 1$. Then we can easily show that a minimum edge dominating set of B is a minimum matrix dominator of M and vice versa.

Observe that the run time of the algorithms for the Minimum Maximal Matching and the Minimum Edge Dominating Set problems are upper bounded by the *total number of maximal independent sets* in a graph. For triangle-free graphs, there are better bounds known for the total number of maximal independent sets. For such graphs we know the following

Theorem 13. [22] *A triangle-free graph on n vertices contains at most $2^{n/2} = 1.4142^n$ maximal independent sets. All maximal independent sets can be enumerated in $O^*(1.4142^n)$ time.*

The reduction from the MATRIX DOMINATION SET to the MINIMUM EDGE DOMINATING SET, transforms an $m \times n$ matrix M into a bipartite (and hence triangle-free) graph B. Theorem 13 then gives us

Theorem 14. *Let M be an $m \times n$ matrix. A minimum size matrix dominator for M can be found in time $O^*(1.4142^{n+m})$.*

2.5 Approximation Algorithm For Chromatic Number

The chromatic number of a graph $G = (V, E)$, $\chi(G)$, is the minimum number of colors required to color the vertices of G such that the endpoints of every edge get different colors. In this section, we give an approximation algorithm which, given a graph G, outputs either $\chi(G)$ or $\chi(G) + 1$ in time $O^*(2.29^n)$. We combine existing algorithms for computing the chromatic number of a graph developed by Lawler [25] and Björklund and Husfeldt [2] and make use of the bounds on the number of maximal independent sets.

```
Algorithm LawCHM(G)
Input: An Undirected graph G.
Output: Chromatic number of a graph.

Let X be an array indexed from 0 to 2^n − 1, initialized to 0 in all cells.
for S ⊆ V enumerated in increasing order of cardinality do
        for all maximal independent sets (MIS) I of G[S] do
                X[S] ← min(X[S], X[S − I] + 1).
return X[V].
```

Fig. 4. Lawler's Algorithm for Finding Chromatic Number

Lawler's Algorithm The algorithm is based on the idea that

$$\chi(G[S]) = \begin{cases} 0 & \text{when } S = \emptyset \\ 1 + \min_I\{\chi(G[S-I]) : I \text{ is a maximal} \\ \qquad\qquad \text{independent set of } G[S]\} & \text{otherwise.} \end{cases}$$

We refer readers to [25, 41] for further details of this algorithm. Let $I(G)$ denote the collection of maximal independent sets of G. Then the time complexity of Lawler's Algorithm (Figure 4) is bounded by

$$O^*\Big(\sum_{S \subseteq V} |I(G[S])|\Big) = O^*\left(\sum_{i=1}^{n} \binom{n}{i} 3^{i/3}\right) = O^*(2.4422^n). \tag{3}$$

This follows from Theorem 3.

Algorithm ApxCHM(G)
Input: A connected undirected graph G.
Output: Either $\chi(G)$ or $\chi(G)+1$.

Step 1: Construct a maximum independent set I of G.
Step 2: If $|I| \geq (0.04486n)$, use Björklund and Husfeldt's algorithm (see Theorem 5) to compute $\chi(G-I)$; output $\chi(G-I)+1$ and halt.
Step 3: Use Lawler's algorithm to compute $\chi(G)$; output $\chi(G)$ and halt.

Fig. 5. Approximation algorithm for finding the chromatic number.

Approximation Algorithm Our approximation algorithm for computing the chromatic number of a graph is described in Figure 5. The correctness of the algorithm follows from the fact that for any maximum independent set I of a graph G, $\chi(G-I) \leq \chi(G) \leq \chi(G-I)+1$.

We show that the time complexity of the algorithm is $O^*(2.2374^n)$. Step 1 takes $O^*(2^{n/4})$ time by Theorem 4. Björklund and Husfeldt's algorithm (see Theorem 5) is called in Step 2 with input $G-I$ with $|G-I| \leq 0.95514n$; the time taken in this step is upper bounded by $O^*(2.3236^{0.95514n}) = O^*(2.2374^n)$. In Step 3, any maximal independent set has size at most $0.04486n$. Let $N(p,j)$ denote the upper bound on the number of maximal independent sets of size j in a p-vertex graph as given in Theorem 2 and let $\gamma = 0.04486$. By equation (3), the time taken in Step 3 is

$$O^*\left(\sum_{S\subseteq V} |I(G[S])|\right) \leq O^*\left(\sum_{p=0}^{\lceil 0.2243n\rceil} \binom{n}{p} 3^{p/3} + \sum_{p=\lceil 0.2243n\rceil+1}^{n} \binom{n}{p}\left(\sum_{j=1}^{\gamma n} N(p,j)\right)\right). \tag{4}$$

The first half can be crudely upper bounded by $2^n \cdot 3^{0.2243n/3} = 2.1712^n$. One can easily verify that for a fixed p, $N(p,j)$ is an increasing function of j. Setting $p = \alpha n$, $j = \beta n$ and $d = \lfloor \alpha/\beta \rfloor$

$$N(p,j) = \left(\frac{d+1}{d}\right)^{\alpha n} \left(\frac{d^{d+1}}{(d+1)^d}\right)^{\beta n}.$$

In Equation (4), as the value of p increases, the value of d used to bound $N(p,j)$ ranges from 5 to 22. We upper bound the above summation as follows

$$\sum_{p=d\gamma n+1}^{(d+1)\gamma n} \binom{n}{p}\left(\sum_{j=1}^{\gamma n} N(p,j)\right) \leq \gamma n \binom{n}{p'} N((d+1)\gamma n, \gamma n). \tag{5}$$

Here p' takes the value in the range $[d\gamma n, (d+1)\gamma n]$ such that $\binom{n}{p'}$ is maximized. A simple calculation shows that the maximum value in any range in Equation (5) is $O^*(2.2374^n)$. This bounds the time taken in Step 3 by $O^*(2.2374^n)$.

Theorem 15. *Given a graph $G=(V,E)$ on n vertices, we can approximate $\chi(G)$ with an additive factor of 1 in $O^*(2.2374^n)$ time and $O^*(2^n)$ space.*

3 Using FPT algorithms to design exact algorithms

In this section, we give a general technique of designing exact algorithms using parameterized algorithms as a subroutine and apply it to several problems including the Odd Cycle Transversal and Feedback Vertex Set. Let Q be an NP-optimization problem and suppose that its parameterized version (Q, k) is fixed parameter tractable. An instance of the parameterized version is a tuple $\langle I, k\rangle$, where I is an instance of Q and k is an integer. The question is to decide whether I has a solution of size $\geq k$, if Q is a maximization problem, and of size $\leq k$, if Q is a minimization problem. 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 $O^*(c^n)$, where n is an upper bound on the optimum solution size. What is interesting is that the FPT algorithm can actually give us 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, where n is the size of the vertex set. 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 the optimization version of 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 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 Odd Cycle Transversal. Had we used brute-force throughout, then the time complexity would have been $O^*(\sum_{i=0}^{n} \binom{n}{i}) = O^*(2^n)$. 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 $\mathsf{Exact}(Q,\mathscr{A},c)$
(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 an upper bound on the optimum solution size.)
Compute the largest λ such that $c^{\lfloor n\lambda\rfloor} \leq \sum_{i=\lfloor n\lambda\rfloor+1}^{n} \binom{n}{i}$.
for $i = 1$ to $\lfloor \lambda n\rfloor$
  use the FPT algorithm $\mathscr{A}$ for Q to check whether there is solution S of size i; if yes output S and halt.
for $i = \lfloor \lambda n\rfloor + 1$ to n
  use brute-force to check whether there exists a solution S of size i; if yes, then output S 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:

$$O^*\left(c^{\lfloor \lambda n \rfloor}\right) = O^*\left(\sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}\right) \tag{6}$$

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 3. *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

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

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. □

Now it is easy to see that

$$O^*\left(\sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}\right) = \begin{cases} \Omega\left(2^n\right) & \text{if } \lambda \leq 1/2 \\ O^*\left(\binom{n}{n-\lambda n}\right) & \text{if } \lambda > 1/2 \end{cases}$$

If $c = 4 - \epsilon$, for some $0 < \epsilon < 4$, then we must have $\lambda > 1/2$. To see this assume that $\lambda \leq 1/2$. We would then have:

$$(4-\epsilon)^{\lfloor \lambda n \rfloor} \approx \sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i} = \Omega(2^n).$$

This is clearly impossible since $(4-\epsilon)^{\lambda n} = o(2^n)$. We thus have the following theorem.

Theorem 16. *Let Q be an NP-optimization problem such that for every instance I of Q there is a polynomial time computable universe U of size, say n, such that an optimum solution of I is a subset of U. Suppose that the parameterized version of Q is FPT with time complexity $O^*(c^k)$, then there is an exact algorithm for Q with time complexity $O^*(d^n)$, where $d = c^{\lambda}$ and λ (< 1) is the largest value such that $c^{\lfloor n\lambda \rfloor} \leq \sum_{i=\lfloor n\lambda \rfloor+1}^{n} \binom{n}{i}$. In particular, $c < 4$ implies $d = c^{\lambda} < 2$.*

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 Odd Cycle Transversal problem in general undirected graphs, the 3 and 4-Hitting Set problems, the Feedback Arc (Vertex) Set problem in tournaments. Some of these results are new and some of them are given here to show the applicability of Theorem 16 and Algorithm Exact().

Odd Cycle Transversal in General Graphs The ODD CYCLE TRANSVERSAL problem can be solved exactly in $O^*(2^{|V|})$ time. Reed, Kaleigh, and Vetta [36] 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 optimization 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 $c = 3$, we obtain $\lambda = 0.6091$ and get a running time of $O^*(1.9526^n)$. We therefore have the following theorem.

Theorem 17. *The* ODD CYCLE TRANSVERSAL *problem can be solved in time* $O^*(1.9526^n)$ *on a graph on* n *vertices.*

An algorithm for this problem with time complexity $O^*(1.62^n)$ was already presented in Section 2.2.

Odd Cycle Transversal in 3-Colorable and Max Degree 3 Graphs In this section, we relate the notions of odd cycle transversal and chromatic number of a graph. We use this to show that $n/3$ is an upper bound on the size of a minimum odd cycle transversal in 3-colorable and maximum degree 3 graphs. This ensures that when we use Algorithm $\mathsf{Exact}()$, it never uses brute-force.

Lemma 4. *Let* G *be a graph on* n *vertices with chromatic number* $k \geq 2$. *Then* G *has an odd cycle transversal of size at most* $\frac{(k-2)n}{k}$.

Proof. Consider a proper k-coloring of G. Let $C_1, C_2, \ldots, C_k$ be the k color classes ordered so that $|C_i| \leq |C_{i+1}|$, $1 \leq i \leq k$. Clearly, if we remove the vertices in the first $k-2$ color classes from G, we will be left with a bipartite graph. We claim that $\sum_{i=1}^{k-2} |C_i| \leq \frac{n(k-2)}{k}$. Suppose not. Then $\sum_{i=1}^{k-2} |C_i| > \frac{n(k-2)}{k}$. Since

$$\frac{1}{k-2} \sum_{i=1}^{k-2} |C_i| \leq |C_{k-1}| \leq |C_k|,$$

we have $\frac{n}{k} < |C_{k-1}| \leq |C_k|$ and so $\sum_{i=1}^{k} |C_i| > \frac{n(k-2)}{k} + \frac{2n}{k} = n$, a contradiction. This proves the lemma. □

By Brook's theorem [39], for any graph G, $\chi(G) \leq \Delta(G)+1$, where $\Delta(G)$ denotes the maximum degree of a vertex in G. If in addition, G is not an odd cycle or a complete graph then $\chi(G) \leq \Delta(G)$. In particular, if G is maximum degree 3 it is 3-colorable except when $G = K_4$. This immediately gives us the following corollary.

Corollary 2. *Let* G *be a graph of maximum degree 3 on* n *vertices other than a complete graph. Then* G *has an odd cycle transversal of size at most* $n/3$.

We obtain an improved algorithm for finding a minimum odd cycle transversal in 3-colorable graphs by applying the parameterized algorithm due to Reed, Smith and Vetta [36] which has a running time of $O(3^k kmn)$, with the parameter k running from 1 to $n/3$, the upper bound on the size of a minimum odd cycle transversal.

Theorem 18. *Let* G *be a graph on* n *vertices that is 3-colorable. Then an odd cycle transversal of* G *can be found in time* $O^*(3^{n/3} = 1.4423^n)$.

Since a maximum degree 3 graph is 3-colorable except when it is K_4, we obtain the following corollary to Theorem 18.

Corollary 3. *Let G be a graph on n vertices with maximum degree 3. Then a minimum odd cycle transversal of G can be found in time $O^*(3^{n/3} = 1.4423^n)$.*

We further improve the time complexity of finding a minimum odd cycle transversal in graphs with maximum degree 3 based on a reduction from Odd Cycle Transversal to Edge Bipartization Set in graphs with maximum degree 3. An instance of the Edge Bipartization Set problem is a graph $G = (V, E)$ and the goal is to find a minimum set of edges whose deletion makes G bipartite.

Lemma 5. *Let $G = (V, E)$ be a graph on n vertices and m edges with maximum degree 3. Then G has an odd cycle transversal of size at most k if and only if G has an edge bipartization set of size at most k.*

Proof. ($\Longrightarrow$) Let $U \subseteq V$ be an odd cycle transversal of size k. Then $G[V - U]$ is bipartite. Let V_1 and V_2 be a bipartition of $G[V - U]$. Since G is maximum degree 3, for every vertex $u \in U$ there exists a set V_i of the bipartition $\{V_1, V_2\}$ such that u has at most one edge e incident with the vertices of V_i. Remove this edge e and include u in V_i. Repeating this for every vertex in U results in a bipartite graph G' which has at most k edges lesser than G.

($\Longleftarrow$) Let $F \subseteq E$ be an edge bipartization set of G of size k. Construct a set U by adding a vertex from each of the edges in F. Clearly the size of U is at most k and $G[V - U]$ is a bipartite graph. This proves the lemma. □

Note that Lemma 5 proves that in graphs of maximum degree 3, a minimum edge bipartization set and a minimum odd cycle transversal have the same size. Lemma 5 also gives us a method to obtain a minimum odd cycle transversal (U) from a minimum edge bipartization set (F). U is obtained from F by adding exactly one vertex from each of the edges in F as in the proof of Lemma 5. By Corollary 2, we also know that the size of a minimum odd cycle transversal in graphs with maximum degree 3 is at most $n/3$. We apply the parameterized algorithm for Edge Bipartization Set mentioned in Theorem 19, with k running from 1 to $n/3$, and obtain an improved algorithm for finding a minimum odd cycle transversal in graphs of maximum degree 3.

Theorem 19. [21] *Let $G = (V, E)$ be an undirected graph on n vertices and m edges. We can determine whether G has an edge bipartization set of size at most k in time $O^*(2^k)$. This algorithm takes polynomial space.*

Theorem 20. *Let G be a graph on n vertices that is not a complete graph. If G has maximum degree 3 then a minimum odd cycle transversal and a minimum edge bipartization set of G can be found in $O^*(2^{n/3}) = O^*(1.26^n)$ time.*

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, $|S_i| \leq 3$ $(1 \leq i \leq m)$ and in the 4-HS problem, $|S_i| \leq 4$ $(1 \leq i \leq m)$. The parameterized versions of these problems have been shown to be fixed parameter tractable by Niedermeier et al [30]. Their result was improved by Fernau in [11], the main results of which can be summarized in the following theorem.

Theorem 21. [11] *The parameterized version of the* 3*-HS and the* 4*-HS problem can be solved in time* $O^*(2.179^k)$ *and* $O^*(3.115^k)$ *respectively.*

Using the values of c from Theorem 21, we obtain the following values of λ for the 3- and 4-HS problems: 3-HS: $c = 2.179$ and $\lambda = 0.738$; 4-HS: $c = 3.115$ and $\lambda = 0.5943$. Applying algorithm Exact() with the above values of λ and the parameterized algorithms by Fernau gives us the following theorem.

Theorem 22. *The* 3*- and* 4-Hitting Set *problems can be solved exactly in time* $O^*(1.7768^n)$ *and* $O^*(1.9646^n)$*, where* $n = |U|$.

Recently Wahlström [38] proposed an exact algorithm for the 3-HS problem with time complexity $O^*(1.6316^n)$. The algorithm for the 3-HS problem in [38] 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.

In [31], Raman and Saurabh describe parameterized algorithms for the Feedback Arc Set and the Feedback Vertex Set problems in tournaments with running times $O^*(2.415^k)$ and $O^*(2.27^k)$ respectively. Using these algorithms, we get $\lambda = 0.696$ with $c = 2.415$ for Feedback Arc Set in tournaments and $\lambda = 0.72$ with $c = 2.27$ for Feedback Vertex Set in tournaments. Then using Algorithm Exact() we obtain the following theorem.

Theorem 23. *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 ensures that Algorithm Exact() will never use brute-force as $\frac{m}{2} \leq 0.696m$. Hence the time complexity of the algorithm is bounded by $O^*((2.415)^{m/2})$ and therefore we get the following theorem.

Theorem 24. *Let* $G = (V, E)$ *be a tournament with* n *vertices and* m *arcs. Then the minimum feedback arc set can be found in time* $O^*(1.5541^m)$.

We give several other applications of Theorem 16 in Figure 7. In Figure 7, *CPA* denotes the constant of the parameterized algorithm of the problem, λ is the cut value computed by the algorithm Exact() and *CEA* represents the constant of the exact algorithm obtained by applying Theorem 16. See reference in last column for problem definitions.

4 'Colored' Branch-and-Bound Technique

In this section, we illustrate the 'colored' branch-and-bound technique by applying it to the Feedback Vertex Set problem in directed graphs with maximum degree 4.

Problem	*Universe*	*CPA*	λ	*CEA*	*Reference*
Cluster Vertex Deletion	Vertex Set	1.53	0.878	1.452	[19]
Cograph Vertex Deletion	Vertex Set	3.115	0.595	1.965	[19]
Constrained Minimum Vertex Cover in Bipartite Graphs	Vertex Set	1.26	0.942	1.25	[5]
Constraint Bipartite Vertex Cover	Vertex Set	1.3999	0.908	1.36	[12]
Set Splitting	Universe	2.6494	0.66	1.9	[26]
Steiner Tree	Vertex Set	$(2+\epsilon)$	0.5	1.414	[27]
Triangle Edge Deletion	Edge Set	2.179	0.738	1.7768	[11]
Triangle Vertex Deletion	Vertex Set	2.179	0.738	1.7768	[11]

Fig. 7. Results obtained by applying Algorithm Exact to some optimization problems.

The 'colored' branch-and-bound technique is a generalization of the usual branch-and-bound technique. This technique has been used before for the Dominating Set problem in [1, 32]. To understand this technique, consider the usual branch-and-bound technique when applied to a problem such as Independent Set. Given a graph $G = (V, E)$, typically, we choose a vertex $v \in V$ and recurse on two cases: in one case, we include v in the partial solution we are constructing and recurse on $G - N[v]$, where $N[v] = \{v\} \cup \{u : (u, v) \in E\}$; in the other, we exclude v from the solution and recurse on $G - v$. Note that in either case, we are free to delete the vertex from the graph and as a result we recurse on smaller subgraphs in both cases.

For problems such as Feedback Vertex Set we cannot delete a vertex when we choose to exclude it from a solution. Nevertheless once we have decided to exclude a vertex, we need to record this information. This is done by labelling vertices good or bad. On any branch of the recursion tree, a bad vertex denotes one that has already been considered and has been left out of the solution that is being constructed in that branch. We will argue that whenever the algorithm recurses, the number of good vertices decreases no matter which branch it takes, and this allows it to make progress along each branch of the recursion. We also show that when we are unable to branch, the number of good vertices remaining is small, in which case we do a brute-force enumeration.

In the recent past, the Feedback Vertex Set problem for undirected graphs has received a lot of attention. The first nontrivial exact algorithm for the Feedback Vertex Set problem is the $O^*(1.945^n)$ time algorithm in undirected graphs of maximum degree 4 [33] which appeared in the conference version of this paper. At about the same time, Fomin [16] developed an $O^*(1.8621^n)$ time algorithm for bipartite graphs. Very recently, Razgon [34] broke the 2^n time barrier with an $O^*(1.9053^n)$ time algorithm for Feedback Vertex Set in general undirected graphs. This algorithm is based on the branch-and-bound technique and uses very a clever measure in its time analysis. Building on Razgon's paper, Fomin et al. [14] have obtained the fastest known exact algorithm for Feedback Vertex Set running in time $O^*(1.7548^n)$. But unfortunately, these techniques cannot be extended to the directed graphs case. Here, we give an algorithm for Feedback Vertex Set in directed graphs with maximum degree (indegree + outdegree) 4 using the 'colored' branch-and-bound technique.

4.1 The FVS problem in colored directed graphs with maximum degree 4

We consider a slight generalization of the directed Feedback Vertex Set problem where each vertex of the input graph is marked either good or bad. We denote the set of good vertices by A and the set of bad vertices by B. The problem is defined below.

Colored Feedback Vertex Set (CFVS): Given a colored directed graph $G = (A \cup B, E)$ with vertices in A colored good and those in B colored bad, find a minimum sized subset $F \subseteq A$ such that $G - F$ is acyclic.

Recall that a *source* in a digraph is a vertex with in-degree 0 and a *sink* is a vertex with out-degree 0. Clearly sources and sinks are not part of any minimum feedback vertex set and can be safely removed.

If v is a degree 2 vertex with in-neighbor u and out-neighbor w, then any directed cycle passing through v must pass through both u and w. Hence if v is part of an optimal feedback vertex set F, then there exists an optimal feedback vertex set not containing v but containing either u or w. Therefore, we can remove v from the graph and add the arc (u, w). These two observations form the basis of the preprocessing algorithm which is described in Figure 8.

<u>*Preprocessing Algorithm*</u> $P2(G, S, \mathsf{Col})$
<u>*Input:*</u> A multi-digraph G whose vertices have been colored good or bad along with a partially constructed solution S.
<u>*Output:*</u> A (possibly smaller) colored multi-digraph 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. Recursively remove the 'sources' and 'sinks' of G.
2. If $G[B]$ has a di-cycle return NO.
3. If a connected component of G is a di-cycle C, include a good vertex from C in S and remove C from the graph.
4. If G has a vertex v_i with exactly one in- and one out-neighbor then it must be that v_i is part of some di-path of the form $u \to v_1 \to \ldots \to v_i \to \ldots \to v_k \to w$ where each v_j $1 \leq j \leq k$ is a vertex with exactly one in- and one out-neighbor, and u and w are either vertices of degree (in-degree + out-degree) ≥ 3 or have exactly one in- and one out-neighbor. Here u and w could be the same vertex.
   (a) Case 1: At least one of u and w is colored good. Then delete all the vertices v_j $(1 \leq j \leq k)$ and add the arc $(u \to w)$ (although u and w might already have an arc in the same direction 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 $u \to v_1 \to \ldots \to v_k \to w$ by $u \to v_i \to w$. If no vertex v_i is colored good then replace the path $u \to v_1 \to \ldots \to v_k \to w$ by the arc $u \to w$.
5. Include all vertices with self loops in S and remove them from the graph. Find all length 2 di-cycles containing a good vertex of degree 2. Include this good vertex in S and remove it from the graph. Find all triangles $\Delta_{u \to v \to w}$ 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. 8. The preprocessing algorithm for the Feedback Vertex Set problem in directed graphs.

After preprocessing, we choose appropriate structures in the graph so that when we branch on a good vertex, the resulting graph has a substructure that is removed by the preprocessing algorithm without any further branching. We next describe the branching strategies of our algorithm.

Branching Strategies At any stage, the algorithm has a partially constructed solution, say S. It branches on a good vertex v and either includes v in S and deletes it from the graph or excludes it by coloring it bad. In each of the cases described below, S denotes the partially constructed solution and the term *reduction(good)* means the reduction in the number of good vertices. For

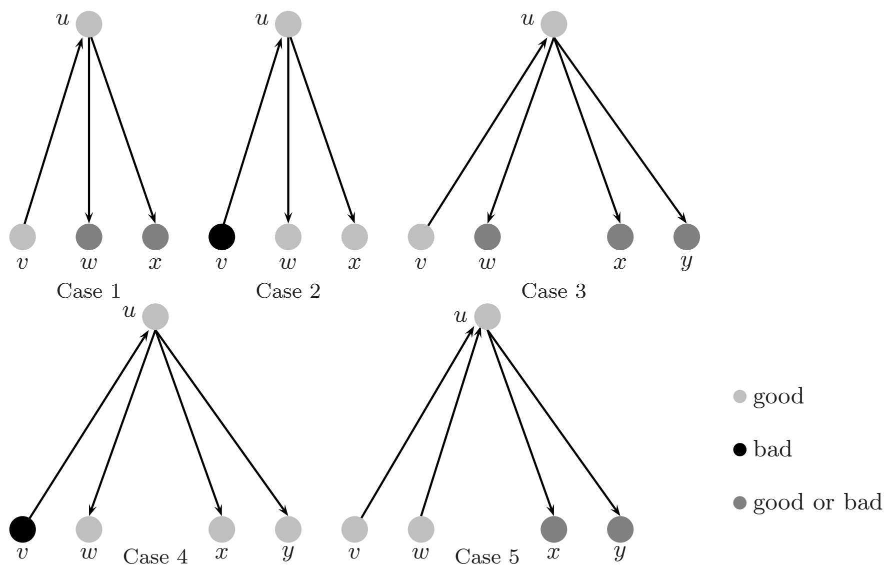

Fig. 9. Branching strategies for the Feedback Vertex Set problem in directed graphs.

each case that we describe below, there exists a symmetric case where the directions of all the arcs are reversed. We let $T(n, g)$ denote the time taken by the algorithm to compute the minimum feedback vertex set of a colored digraph on n vertices of which g are good.

Case 1: u is a good vertex of degree 3 with one good in-neighbor v and two out-neighbors $\{w, x\}$.

- $v \in S$ (reduction(good) ≥ 2)
- $v \notin S$ (reduction(good) ≥ 1)

In the first case, two good vertices are deleted from the graph. To see this, note that by including v in the solution we delete it from the graph, making u a 'source' vertex which is then removed by the next application of the preprocessing algorithm. In the second case, we label v bad and reduce the number of good vertices by at least 1. The recurrence for run time is therefore

$$T(n, g) \leq T(n, g-1) + T(n-2, g-2) + O(n+m).$$

Case 2: u is a good vertex of degree 3 with one bad in-neighbor v and two good out-neighbors $\{w, x\}$.

- $w \in S$ (reduction(good) ≥ 2)
- $w \notin S$ (reduction(good) ≥ 1)

If $w \in S$, then after deleting it from the graph, u becomes a degree 2 vertex on a di-path $v \to u \to x$ with x colored good. The preprocessing algorithm removes u and places an arc between v and x. If $w \notin S$, it is colored bad thus reducing the number of good vertices by 1. The recurrence for run time is therefore

$$T(n, g) \leq T(n, g-1) + T(n-2, g-2) + O(n+m).$$

Case 3: u is a good vertex of degree 4 with one good in-neighbor v and three out-neighbors $\{w, x, y\}$.

- $v \in S$ (reduction(good) ≥ 2)
- $v \notin S$ (reduction(good) ≥ 1)

The recurrence for run time is

$$T(n, g) \leq T(n, g-1) + T(n-2, g-2) + O(n+m).$$

Case 4: u is a good vertex of degree 4 with one bad in-neighbor v and three good out-neighbors $\{w, x, y\}$.

- $w \notin S$ (reduction(good) ≥ 1)
- $w \in S$, $x \notin S$ (reduction(good) ≥ 2)
- $w, x \in S$ (reduction(good) ≥ 3)

If $w \notin S$, then we color it bad and reduce the number of good vertices by at least 1. Otherwise, given that $w \in S$ we have either $x \in S$ or $x \notin S$. If $x \notin S$, then we color x bad and reduce the number of good vertices by at least 2 but delete only one vertex form the graph. Else, we delete both w and x from the graph after which u has exactly one in-neighbor v and exactly one (good) out-neighbor y. The preprocessing algorithm then deletes u and adds the arc (v, y). We thus delete at least 3 good vertices in this case. The recurrence for run time is therefore

$$T(n, g) \leq T(n, g-1) + T(n-1, g-2) + T(n-3, g-3) + O(n+m).$$

Case 5: u is a good vertex of degree 4 with two good in-neighbors $\{v, w\}$ and two out-neighbors $\{x, y\}$.

- $v \notin S$ (reduction(good) ≥ 1)
- $v \in S$, $w \notin S$ (reduction(good) ≥ 2)
- $v, w \in S$ (reduction(good) ≥ 3)

The recurrence for run time is again

$$T(n, g) \leq T(n, g-1) + T(n-1, g-2) + T(n-3, g-3) + O(n+m).$$

This completes the description of branching strategies. The algorithm is described in Figure 10 and its correctness is easy to see. Now we bound the time complexity of the algorithm.

Observe that when we cannot apply the branching strategies and the preprocessing algorithm mentioned above, then any good vertex in the remaining graph must belong to one of the following classes.

- *Degree* 2 *class.* A vertex with exactly one in-neighbor and one out-neighbor both of which are bad.
- *Degree* 3 *class.* A vertex of degree 3 with at most one good neighbor. A vertex in this class with one good neighbor also has one bad neighbor of the same 'parity' (that is, both vertices are either in-neighbors or out-neighbors).

Algorithm FVS-DIGRAPH-D4($G = (V, E)$, S, Col)
Input: A multi-digraph $G = (V, E)$ with maximum degree 4, whose vertices have been colored good or bad.
Output: A minimum feedback vertex set of G.
Comment: 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 $v \in G(V)$.

Step 1 Call $P2(G, S, \mathsf{Col})$. If $P2$ 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 there exists no good vertex of the types described in Cases 1 to 5 (and their symmetric analogues), 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 Branch on a good vertex of type described in Cases 1 to 5 (and their symmetric analogues) in Section 4.1 and recurse on the resulting smaller graph. Return a solution with smallest size.

Fig. 10. Algorithm FVS-DIGRAPH-D4()

– *Degree* 4 *class.* A vertex of degree 4 with at most two good neighbors. A vertex in this class with exactly two good neighbors has one good in-neighbor and one good out-neighbor.

We bound the number of good vertices in the graph by counting the number of good-bad edges in the underlying undirected graph $\overline{G}$. Every good vertex in $\overline{G}$ is of one of the following types:

Type 1 Degree 2 good vertex with all neighbors bad.
Type 2 Degree 3 good vertex with one good neighbour.
Type 3 Degree 3 good vertex with all 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 neighbours bad.

Let n_i represent the number of good vertices of Type i and let $g = \sum_{i=1}^{6} n_i$. 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 subgraph of $\overline{G}$ induced by the good vertices consists of a matching of size $(n_2 + 2n_4 + n_5)/2$ and possibly some isolated vertices. If $n = |V(\overline{G})|$ then

$$2n_1 + 2n_2 + 3n_3 + 2n_4 + 3n_5 + 4n_6 \leq \frac{4(n - n_1 - n_2 - n_3) + 3(n_2 + n_3) + 2n_1}{2} - \frac{n_2}{2} - \frac{2n_4}{2} - \frac{n_5}{2}$$

A simple calculation shows that $3g \leq 2n$ from which it follows that $g \leq \frac{2n}{3}$.

The recurrence for running time is upper bounded by:

$$\begin{aligned} T(n, g) &\leq T(n, g-1) + T(n-1, g-2) + T(n-3, g-3) + O(n+m) && \text{if } g > \tfrac{2n}{3} \quad (7)\\ T(n, g) &\leq 2^g && \text{if } g \leq \tfrac{2n}{3}. \end{aligned}$$

The solution of this recurrence and hence the time complexity of the algorithm is bounded by $(1.8393)^{g - \frac{2n}{3}} \cdot 2^{\frac{2n}{3}}$, a fact which can be proved by induction on g. We thus have the following theorem and its corollary as earlier.

Theorem 25. *Let $G = (A \cup B, E)$ be a colored directed graph with maximum degree 4 with n vertices of which g are good. Then the* Colored Feedback Vertex Set *problem on G can be solved exactly using polynomial space and time $O^*((1.8393)^{g-\frac{2n}{3}} \cdot 2^{\frac{2n}{3}})$.*

Setting the initial value of g as n, we obtain the following corollary.

Corollary 4. *Let $G = (V, E)$ be a directed graph with maximum degree 4. Then the* Feedback vertex Set *problem on G can be solved in time $O^*(1.945^n)$ time, where n is the number of vertices in G.*

5 Conclusion and Discussions

In this paper, we have described three techniques of designing exact algorithms, namely:

1. exact algorithms by enumerating maximal independent sets in a graph,
2. exact algorithms from parameterized algorithms,
3. 'colored' branch-and-bound technique,

and have obtained improved exact algorithms for several problems including Odd Cycle Transversal in general undirected graphs, Feedback Vertex Set in directed graphs of maximum degree at most 4, 4-Hitting Set, and Feedback Arc Set in tournaments. Here is a summary of the bounds on the problems that we addressed in this paper.

Problem	*Previous Best*	*Bound in this Paper*
Odd Cycle Transversal	$O^*(1.7724^n)$ [3]	$O^*(1.9526^n)$ (Sec. 3.1)
		$O^*(1.62^n)$ (Sec. 2.2)
# Odd Cycle Transversal	$O^*(1.7724^n)$ [3]	$O^*(1.6713^n)$ (Sec. 3.1)
Feedback Vertex Set in di-graphs with total degree ≤ 4	$O^*(2^n)$	$O^*(1.945^n)$ (Sec. 4.1)
Feedback Arc Set in Tournaments	$O^*(2^m)$	$O^*(1.5541^m)$ (Sec. 3.1)
Minimum Maximal Matching	$O^*(1.4422^m)$ [35]	$O^*(1.44225^n)$ (Sec. 2.3)
Minimum Edge Dominating Set	$O^*(1.4422^m)$ [35]	$O^*(1.44225^n)$ (Sec. 2.4)
4-Hitting Set	$O^*(2^n)$	$O^*(1.9646^n)$ (Sec. 3.1)

Fig. 11. A survey of results in this paper.

Any further reduction in the base of the exponent of all the algorithms presented in the paper remains open. It would also be interesting to investigate the practical performance of these algorithms and find other applications of the techniques presented in the paper. Another major open problem is to devise an exact algorithm with time complexity less than $O^*(c^n)$, $c < 2$ for the Feedback Vertex Set problem in general directed graphs.

6 Acknowledgements

We thank anonymous referees of *ICTCS 2005* and *Theory of Computing Systems* for several useful comments. The second author would also like to thank Gerhard Woeginger for suggesting an investigation of exact algorithms for the Minimum Maximal Matching problem and C. R. Subramanian for several discussions on the Odd Cycle Transversal and other problems.

References

1. J. Alber, H. Fan, M. R. Fellows, H. Fernau, R. Niedermeier, F. Rosamond and U. Stege. *A refined search tree technique for Dominating Set on planar graphs.* Journal of Computer and System Sciences 71(4): 385-405 (2005).
2. A. Björklund and T. Husfeldt. *Exact algorithms for exact satisfiability and number of perfect matchings.* To appear in the Proceedings of 33rd International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science, Springer Verlag, (2006).
3. J. M. Byskov. *Exact Algorithms for Graph Colouring and Exact Satisfiability.* PhD Dissertation, (2004).
4. J. M. Byskov. *Enumerating Maximal Independent Sets with Applications to Graph Colouring.* Operations Research Letters 32(6): 547-556 (2004).
5. J. Chen and I. A. Kanj. *Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms.* Journal of Computer and System Sciences 67(4): 833-847 (2003).
6. J. Chen. *Parameterized Computation and Complexity: A New Approach Dealing with NP-Hardness.* Journal of Computer Science and Technology 20(1): 18-37 (2005).
7. R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag (1999).
8. R. Downey. *Parameterized Complexity for the Skeptic.* In the Proceedings of of 18th IEEE Conference on Computational Complexity: 147-169 (2003).
9. D. Eppstein. *3-coloring in time* $O(1.3289^n)$. Journal of Algorithms 54 (2): 168-204 (2005).
10. D. Eppstein. *Small Maximal Independent Sets and Faster Exact Graph Coloring.* In the Proceedings of 7th Workshop on Algorithms and Data Structures (WADS). Lecture Notes in Computer Science 2125: 462-470 (2001).
11. H. Fernau. *Parameterized Algorithmics for* d-Hitting Set. Manuscript.
12. H. Fernau and R. Niedermeier. *An Efficient Exact Algorithm for Constraint Bipartite Vertex Cover.* Journal of Algorithms 38(2): 374-410 (2001).
13. J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer EATCS Series, 2006.
14. F. V. Fomin, S. Gaspers, and A. V. Pyatkin. *Finding a Minimum Feedback Vertex Set in time* $O(1.7548^n)$. Manuscript, (2006).
15. F. V. Fomin, F. Grandoni, and D. Kratsch. *Some new techniques in design and analysis of exact (exponential) algorithms.* Bulletin of the European Association for Theoretical Computer Science (EATCS) 87: 47-77 (2005).
16. F. V. Fomin and A. V. Pyatkin. *Finding Feedback Vertex Set in a Bipartite Graph.* Reports in Informatics. Department of Informatics, University of Bergen. Report No. 291, (2005).
17. M. Fürer and S. P. Kasiviswanathan. *Algorithms for Counting 2-SAT Solutions and Colourings with Applications.* Electronic Colloquium on Computational Complexity (ECCC) 33 (2006).
18. F. Gavril and M. Yannakakis. *Edge Dominating Sets in Graphs.* SIAM Journal on Applied Mathematics 38(3): 364-372 (1980).
19. J. Gramm, J. Guo, F. Hüffner and R. Niedermeier. *Automated Generation of Search Tree Algorithms for Hard Graph Modification Problems.* Algorithmica 39(4): 321-347 (2004).
20. 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).
21. J. Guo, J. Gramm, F. Hüffner, R. Neidermeier and S. Wernicke. *Improved Fixed-Parameter Algorithms for Two Feedback Set Problems.* In the Proceedings of 9th Workshop on Algorithms and Data Structures (WADS). Lecture Notes in Computer Science 3608: 158-168 (2005).
22. M. Hujter and Z. Tuza. *The number of Maximal Independent Sets in Triangle-Free Graphs.* SIAM Journal on Discrete Mathematics 6(2): 284-288 (1993).
23. K. Iwama and S. Tamaki. *Improved upper bounds for 3-SAT.* In the Proceedings of 15th ACM-SIAM Symposium on Discrete Algorithms (SODA): 328 (2004).
24. D. S. Johnson, M. Yannakakis and C. H. Papadimitriou. *On Generating all Maximal Independent Sets.* Information Processing Letters 27: 119-123 (1988).
25. E. L. Lawler. *A Note on the Complexity of the Chromatic Number.* Information Processing Letters 5 (3): 66-67 (1976).
26. D. Lokshtanov and C. Sloper. *Fixed Parameter Set Splitting, Linear Kernel and Improved Running Time.* In Proceedings of Algorithms and Complexity in Durham (ACID), 105-113, (2005).
27. D. Mölle, S. Richter and P. Rossmanith. *A Faster Algorithm for the Steiner Tree Problem.* In the Proceedings of 23rd Symposium on Theoretical Aspects of Computer Science. Lecture Notes in Computer Science 3884: 561-570 (2006).
28. J. W. Moon and L. Moser. *On Cliques in Graphs.* Israel Journal of Mathematics 3: 23-28 (1965).

29. R. Niedermeier. *Invitation to Fixed Parameter Algorithms.* Oxford University Press, (2006).
30. R. Niedermeier and P. Rossmanith. *An efficient fixed parameter algorithm for 3-Hitting Set.* Journal of Discrete Algorithms 1 (1): 89-102 (2003).
31. V. Raman and S. Saurabh. *Parameterized Algorithms for Feedback Set Problems and their Duals in Tournaments.* Theoretical Computer Science 351(3): 446-458 (2006).
32. V. Raman and S. Saurabh. *Triangles, 4-cycles and Parameterized (In-)Tractability.* To appear in the proceedings of 10th Scandinavian Workshop on Algorithm Theory (SWAT). Lecture Notes in Computer Science (2006).
33. V. Raman, S. Saurabh and S. Sikdar. *Improved Exact Exponential Algorithms for Vertex Bipartization and Other Problems.* In the Proceedings of 9th Italian Conference on Theoretical Computer Science (ICTCS). Lecture Notes in Computer Science 3701: 375-389 (2005).
34. I. Razgon. *Exact Computation of Maximum Induced Forest.* To appear in the Proceedings of 10th Scandinavian Workshop on Algorithm Theory (SWAT). Lecture Notes in Computer Science (2006).
35. B. Randerath and I. Schiermeyer. *Exact Algorithms for Minimum Dominating Set.* Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln, (2004).
36. B. Reed, K. Smith and A. Vetta. *Finding Odd Cycle Transversals.* Operations Research Letters 32: 299-301 (2004).
37. J. M. Robson. *Finding a Maximum Independent Set in time $O^*(2^{n/4})$?* Technical Report 1251-01, LaBRI, Université Bordeaux I, (2001).
38. M. Wahlström. *Exact algorithms for finding minimum transversals in rank-3 hypergraphs.* Journal of Algorithms 51(2): 107 - 121 (2004).
39. D. B. West. *Introduction to Graph Theory.* Prentice Hall, Second Edition (2001).
40. R. Williams. *A new algorithm for optimal 2-constraint satisfaction and its implications.* In the Proceedings of International Colloquium on Automata, Languages and Programming (ICALP). Lecture Notes in Computer Science 3142: 1227-1237 (2004).
41. G. Woeginger. *Exact algorithms for NP-hard problems: A survey.* In *Combinatorial Optimization—Eureka! You shrink!.* Lecture Notes in Computer Science 2570: 185-207 (2003).
42. G. Woeginger. *Space and Time Complexity of Exact Algorithms: Some Open Problems.* In the Proceedings of the 1st International Workshop on Exact and Parameterized Algorithms (IWPEC). Lecture Notes in Computer Science 3162: 281-290 (2004).