Iterative Compression and Exact Algorithms

Fedor V. Fomin[1]*, Serge Gaspers[1], Dieter Kratsch[2],
Mathieu Liedloff[2] and Saket Saurabh[1]

[1] Department of Informatics, University of Bergen,
N-5020 Bergen, Norway.
{fomin|serge|saket}@ii.uib.no
[2] Laboratoire d'Informatique Théorique et Appliquée,
Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France.
{kratsch|liedloff}@univ-metz.fr

Abstract. Iterative Compression has recently led to a number of breakthroughs in parameterized complexity. The main purpose of this paper is to show that iterative compression can also be used in the design of exact exponential time algorithms. We exemplify our findings with algorithms for the Maximum Independent Set problem, a counting version of k-Hitting Set and the Maximum Induced Cluster Subgraph problem.

1 Introduction

Iterative Compression is a tool that has recently been used successfully in solving a number of problems in the area of Parameterized Complexity. This technique was first introduced by Reed et al. to solve the Odd Cycle Transversal problem, where one is interested in finding a set of at most k vertices whose deletion makes the graph bipartite [20]. Iterative compression was used in obtaining faster FPT algorithms for Feedback Vertex Set, Edge Bipartization and Cluster Vertex Deletion on undirected graphs [6, 12, 14]. Recently this technique has led to an FPT algorithm for the Directed Feedback Vertex Set problem [4], one of the longest open problems in the area of parameterized complexity.

The main idea behind iterative compression for parameterized algorithms is an algorithm which, given a solution of size $k+1$ for a problem, either compresses it to a solution of size k or proves that there is no solution of size k. This is known as the compression step of the algorithm. Based on this compression step, iterative (and incremental) algorithms for minimization problems are obtained. The most technical part of an FPT algorithm based on iterative compression is to show that the compression step can be carried out in time $f(k) \cdot n^{O(1)}$, where f is an arbitrary computable function, k is a parameter and n is the length of the input.

The presence of a solution of size $k + 1$ can provide important structural information about the problem. This is one of the reasons why the technique of

* Partially supported by the Research Council of Norway.

iterative compression has become so powerful. Structures are useful in designing algorithms in most paradigms. By seeing so much success of iterative compression in designing fixed parameter tractable algorithms, it is natural and tempting to study its applicability in designing exact exponential time algorithms.

The goal of the design of moderately exponential time algorithms for NP-complete problems is to establish algorithms for which the worst-case running time is provably faster than the one of enumerating all prospective solutions, or loosely speaking, *algorithms better than brute-force enumeration*. For example, for NP-complete problems on graphs on n vertices and m edges whose solutions are either subsets of vertices or edges, the brute-force or trivial algorithms basically enumerate all subsets of vertices or edges. This mostly leads to algorithms of time complexity 2^n or 2^m, modulo some polynomial factors, based on whether we are enumerating vertices or edges. Almost all the iterative compression based FPT algorithms with parameter k have a factor of 2^{k+1} in the running time, as they all branch on all partitions (A, D) of a $k+1$ sized solution S and look for a solution of size k with a restriction that it should contain all elements of A and none of D. This is why, at first thought, iterative compression is a quite useless technique for solving optimization problems because for $k = \Omega(n)$, we end up with an algorithm having a factor 2^n or 2^m in the worst-case running time, while a running time of 2^n or 2^m (up to a polynomial factor) often can be achieved by (trivial) brute force enumeration. Luckily, our intuition here appears to be wrong and with some additional arguments, iterative compression can become a useful tool in the design of moderately exponential time algorithms as well. We find it interesting because despite of several exceptions (like the works of Björklund et al. [1, 2, 16]), the area of exact algorithms is heavily dominated by branching algorithms, in particular, for subset problems. It is very often that an (incremental) improvement in the running time of branching algorithm requires an extensive case analysis, which becomes very technical and tedious. The analysis of such algorithms can also be very complicated and even computer based.

The main advantage of iterative compression is that it provides combinatorial algorithms based on problem structures. While the improvement in the running time compared to (complicated) branching algorithms is not so impressive, the simplicity and elegance of the arguments allow them to be used in a basic algorithm course.

To our knowledge, this paper is the first attempt to use iterative compression outside the domain of FPT algorithms. We exemplify this approach by the following results:

1. We show how to solve Maximum Independent Set for a graph on n vertices in time $\mathcal{O}(1.3196^n)$. While the running time of our iterative compression algorithm is slower than the running times of modern branching algorithms [10, 21], this simple algorithm serves as an introductory example to more complicated applications of the method.
2. We obtain algorithms counting the number of minimum hitting sets of a family of sets of an n-element ground set in time $\mathcal{O}(1.7198^n)$, when the size of each set is at most 3 (#Minimum 3-Hitting Set). For #Minimum

4-Hitting Set we obtain an algorithm of running time $\mathcal{O}(1.8997^n)$. For Minimum 4-Hitting Set similar ideas lead to an algorithm of running time $\mathcal{O}(1.8704^n)$. These algorithms are faster than the best algorithms known for these problems so far [9, 19].

3. We provide an algorithm to solve the Maximum Induced Cluster Subgraph problem in time $\mathcal{O}(1.6181^n)$. The only algorithm for this problem we were aware of before is the use of a very complicated branching algorithm of Wahlström [23] for solving 3-Hitting Set (let us note that Maximum Induced Cluster Subgraph is a special case of 3-Hitting Set, where every subset is a set of vertices inducing a path of length 3), which results in time $\mathcal{O}(1.6278^n)$.

2 Maximum Independent Set

Maximum Independent Set (MIS) is one of the well studied problems in the area of exact exponential time algorithms and many papers have been written on this problem [10, 21, 22]. It is customary that if we develop a new method then we first apply it to well known problems in the area. Here, as an introductory example, we consider the NP-complete problem MIS.

> Maximum Independent Set (MIS): Given a graph $G = (V, E)$ on n vertices, find a maximum independent set of G. An *independent set* of G is a set of vertices $I \subseteq V$ such that no two vertices of I are adjacent in G. A *maximum independent set* is an independent set of maximum size.

It is well-known that I is an independent set of a graph G iff $V \setminus I$ is a vertex cover of G, i.e. every edge of G has at least one end point in $V \setminus I$. Therefore Minimum Vertex Cover (MVC) is the complement of MIS in the sense that I is a maximum independent set of G iff $V \setminus I$ is a minimum vertex cover of G. This fact implies that when designing exponential time algorithms we may equivalently consider MVC. We proceed by defining a compression version of the MVC problem.

> Comp-MVC: Given a graph $G = (V, E)$ with a vertex cover $S \subseteq V$, find a vertex cover of G of size at most $|S| - 1$ if one exists.

Note that if we can solve Comp-MVC efficiently then we can solve MVC efficiently by repeatedly applying an algorithm for Comp-MVC as follows. Given a graph $G = (V, E)$ on n vertices with $V = \{v_1, v_2, ..., v_n\}$, let $G_i = G[\{v_1, v_2, ..., v_i\}]$ and let C_i be a minimum vertex cover of G_i. By V_i we denote the set $\{v_1, v_2, ..., v_i\}$. We start with G_1 and put $C_1 = \emptyset$. Suppose that we already have computed C_i for the graph G_i for some $i \geq 1$. We form an instance of Comp-MVC with input graph G_{i+1} and $S = C_i \cup \{v_{i+1}\}$. In this stage we either *compress* the solution S which means that we find a vertex cover S' of G_{i+1} of size $|S| - 1$ and put $C_{i+1} = S'$, or (if there is no S') we put $C_{i+1} = S$.

Our algorithm is based on the following lemma.

Lemma 1. [⋆][3] *Let G_{i+1} and S be given as above. If there exists a vertex cover C_{i+1} of G_{i+1} of size $|S|-1$, then it can be partitioned into two sets A and B such that*
(a) $A \subset S$, $|A| \leq |S|-1$ and A is a minimal vertex cover of $G_{i+1}[S]$.
(b) $B \subseteq (V_{i+1} \backslash A)$ is a minimum vertex cover of the bipartite graph $G_{i+1}[V_{i+1} \backslash A]$.

Lemma 1 implies that the following algorithm solves Comp-MVC correctly.

Step 1: Enumerate all minimal vertex covers of size at most $|S|-1$ of $G_{i+1}[S]$ as a possible candidate for A.
Step 2: For each minimal vertex cover A find a minimum vertex cover B of the bipartite graph $G_{i+1}[V_{i+1} \setminus A]$ (via the computation of a maximum matching in this bipartite graph [13]).
Step 3: If the algorithm finds a vertex cover $A \cup B$ of size $|S|-1$ in this way, set $C_{i+1} = A \cup B$, else set $C_{i+1} = S$.

Steps 2 and **3** of the algorithm can be performed in polynomial time, and the running time of **Step 1**, which is exponential, dominates the running time of the algorithm. To enumerate all maximal independent sets or equivalently all minimal vertex covers of a graph in **Step** 1, one can use the polynomial-delay algorithm of Johnson et al. [15].

Proposition 1 ([15]). *All maximal independent sets of a graph can be enumerated with polynomial delay.*

For the running time analysis of the algorithm we need the following bounds on the number of maximal independent sets or minimal vertex covers due to Moon and Moser [17] and Byskov [3].

Proposition 2 ([17]). *A graph on n vertices has at most $3^{n/3}$ maximal independent sets.*

Proposition 3 ([3]). *The maximum number of maximal independent sets of size at most k in any graph on n vertices for $k \leq n/3$ is*

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

Moreover, all such sets can be enumerated in time $\mathcal{O}^*(N[n,k])$.[4]

Since

$$\max \left\{ \max_{0 \leq \alpha \leq 3/4} (3^{\alpha n/3}), \max_{3/4 < \alpha \leq 1} (N[\alpha n, (1-\alpha)n]) \right\} = O^*(2^{2n/5}),$$

[3] Proofs of results labeled with [⋆] will appear in the long version of the paper.
[4] Throughout this paper we use a modified big-Oh notation that suppresses all polynomially bounded factors. For functions f and g we write $f(n) = \mathcal{O}^*(g(n))$ if $f(n) = \mathcal{O}(g(n) poly(n))$, where $poly(n)$ is a polynomial. Furthermore, since $c^n \cdot poly(n) = \mathcal{O}((c+\epsilon)^n)$ for any $\epsilon > 0$, we omit polynomial factors in the big-Oh notation every time we round the base of the exponent.

we have that by Propositions 1, 2, and 3, all minimal vertex covers of $G_{i+1}[S]$ of size at most $|S|-1$ can be listed in time $\mathcal{O}^*(2^{2n/5}) = \mathcal{O}(1.3196^n)$.

Thus, the overall running time of the algorithm solving Comp-MVC is $\mathcal{O}(1.3196^n)$. Since the rounding of the base of the exponent dominates the polynomial factor of the other steps of the iterative compression, we obtain the following theorem.

Theorem 1. Maximum Independent Set *and* Minimum Vertex Cover *can be solved in time* $\mathcal{O}(1.3196^n)$ *on graphs of* n *vertices by a compression based algorithm.*

3 #k-Hitting Set

The Hitting Set problem is a generalization of Vertex Cover. Here, given a family of sets over a ground set of n elements, the objective is to hit every set of the family with as few elements of the ground set as possible. We study a version of the hitting set problem where every set in the family has at most k elements.

> Minimum k-Hitting Set (MHS$_k$): Given a universe V of n elements and a collection $\mathcal{C}$ of subsets of V of size at most k, find a minimum hitting set of $\mathcal{C}$. A *hitting set* of $\mathcal{C}$ is a subset $V' \subseteq V$ such that every subset of $\mathcal{C}$ contains at least one element of V'.

A counting version of the problem is #Minimum k-Hitting Set (#MHS$_k$) that asks for the number of different minimum hitting sets. We denote an instance of #MHS$_k$ by $(V, \mathcal{C})$. Furthermore we assume that for every $v \in V$, there exists at least one set in $\mathcal{C}$ containing it.

We show how to obtain an algorithm to solve #MHS$_k$ using iterative compression which uses an algorithm for #MHS$_{k-1}$ as a subroutine. First we define the compression version of the #MHS$_k$ problem.

> Comp-#k-Hitting Set: Given a universe V of n elements, a collection $\mathcal{C}$ of subsets of V of size at most k, and a (not necessarily minimum) hitting set $H' \subseteq V$ of $\mathcal{C}$, find a minimum hitting set $\widehat{H}$ of $\mathcal{C}$ and compute the number of all minimum hitting sets of $\mathcal{C}$.

Lemma 2. *Let* $\mathcal{O}^*(a_{k-1}^n)$ *be the running time of an algorithm solving* #MHS$_{k-1}$*, where* $a_{k-1} > 1$ *is some constant. Then* Comp-#k-Hitting Set *can be solved in time*

$$\mathcal{O}^*\left(2^{|H'|} a_{k-1}^{|V|-|H'|}\right).$$

Moreover, if $|H'|$ *is greater than* $2|V|/3$ *and the minimum size of a hitting set in* $\mathcal{C}$ *is at least* $|H'|-1$*, then* Comp-#k-Hitting Set *can be solved in time*

$$\mathcal{O}^*\left(\binom{|H'|}{2|H'|-|V|} a_{k-1}^{|V|-|H'|}\right).$$

Proof. To prove the lemma, we give an algorithm that, for each possible partition $(N, \bar{N})$ of H', computes a minimum hitting set H_N and the number h_N of minimum hitting sets subject to the constraint that these hitting sets contain all the elements of N and none of the elements of $\bar{N}$.

For every partition $(N, \bar{N})$ of H', we either reject it as invalid or we reduce the instance $(V, \mathcal{C})$ to an instance $(V', \mathcal{C}')$ by applying the following two rules in the given order.

(H) If there exists a set $C_i \in \mathcal{C}$ such that $C_i \subseteq \bar{N}$ then we refer to such a partition as *invalid* and reject it.

(R) For all sets C_i with $C_i \cap N \neq \emptyset$ put $\mathcal{C} = \mathcal{C} \setminus C_i$. In other words, all sets of $\mathcal{C}$, which are already hit by N, are removed.

If the partition $(N, \bar{N})$ of H' is not invalid based on rule **(R)** the instance $(V, \mathcal{C})$ can be reduced to the instance $I' = (V', \mathcal{C}')$, where $V' = V \setminus H'$ and $\mathcal{C}' = \{X \cap V' \mid X \in \mathcal{C} \text{ and } X \cap N = \emptyset\}$.

Summarizing, the instance I' is obtained by removing all the elements of V for which it has already been decided if they are part of H_N or not and all the sets that are hit by the elements in N. To complete H_N, it is sufficient to find a minimum hitting set of I' and to count the number of minimum hitting sets of I'. The crucial observation here is that I' is an instance of $\#\mathrm{MHS}_{k-1}$. Indeed, H' is a hitting set of $(V, \mathcal{C})$ and by removing it we decrease the size of every set at least by one. Therefore, we can use an algorithm for $\#\mathrm{MHS}_{k-1}$ to complete this step. When checking all partitions $(N, \bar{N})$ of H' it is straightforward to keep the accounting information necessary to compute a minimum hitting set $\widehat{H}$ and to count all minimum hitting sets.

Thus for every partition $(N, \bar{N})$ of H' the algorithm solving $\#\mathrm{MHS}_{k-1}$ is called for the instance I'. There are $2^{|H'|}$ partitions $(N, \bar{N})$ of the vertex set H'. For each such partition, the number of elements of the instance I' is $|V'| = |V \setminus H'| = |V| - |H'|$. Thus, the running time of the algorithm is $\mathcal{O}^*\left(2^{|H'|} a_{k-1}^{|V|-|H'|}\right)$.

If $|H'| > 2|V|/3$ and the minimum size of a hitting set in $\mathcal{C}$ is at least $|H'|-1$, then it is not necessary to check all partitions $(N, \bar{N})$ of H' and in this case we can speed up the algorithm. Indeed, since

- $|H'| \geq |\widehat{H}| \geq |H'| - 1$, and
- $|\widehat{H} \cap (V \setminus H')| \leq |V| - |H'|$,

it is sufficient to consider only those partitions $(N, \bar{N})$ of H' such that

$$|N| \geq |H'| - 1 - (|V| - |H'|) = 2|H'| - |V| - 1.$$

In this case, the running time of the algorithm is $\mathcal{O}^*\left(\binom{|H'|}{2|H'|-|V|} a_{k-1}^{|V|-|H'|}\right)$. □

Now we are ready to use iterative compression to prove the following theorem.

Theorem 2. *Suppose there exists an algorithm to solve* $\#\mathrm{MHS}_{k-1}$ *in time* $\mathcal{O}^*(a_{k-1}^n)$, $1 < a_{k-1} \leq 2$. *Then* $\#\mathrm{MHS}_k$ *can be solved in time*

$$\mathcal{O}^*\left(\max_{2n/3 \leq j \leq n}\left\{\binom{j}{2j-n} a_{k-1}^{n-j}\right\}\right).$$

Proof. Let $(V, \mathcal{C})$ be an instance of $\#\mathrm{MHS}_k$, where $V = \{v_1, v_2, \cdots, v_n\}$. For $i = 1, 2, \ldots, n$, let $V_i = \{v_1, v_2, \cdots, v_i\}$ and $\mathcal{C}_i = \{X \in \mathcal{C} \mid X \subseteq V_i\}$. Then $I_i = (V_i, \mathcal{C}_i)$ constitutes an instance for the ith stage of the iteration. We denote by H_i and h_i, a minimum hitting set of an instance I_i and the number of different minimum hitting sets of I_i respectively.

If $\{v_1\} \in \mathcal{C}$, then $H_1 = \{v_1\}$ and $h_1 = 1$; otherwise $H_1 = \emptyset$ and $h_1 = 0$.

Consider the ith stage of the iteration. We have that $|H_{i-1}| \leq |H_i| \leq |H_{i-1}| + 1$ because at least $|H_{i-1}|$ elements are needed to hit all the sets of I_i except those containing element v_i and $H_{i-1} \cup \{v_i\}$ is a hitting set of I_i. Now, use Lemma 2 with $H' = H_{i-1} \cup \{v_i\}$ to compute a minimum hitting set of I_i. If $|H'| \leq 2i/3$, its running time is $\mathcal{O}^*\left(\max_{0 \leq j \leq 2i/3}\left\{2^j a_{k-1}^{i-j}\right\}\right) = \mathcal{O}^*\left(2^{2i/3} a_{k-1}^{i/3}\right)$ (for $a_{k-1} \leq 2$). If $|H'| > 2i/3$, the running time is $\mathcal{O}^*\left(\max_{2i/3 < j \leq i}\left\{\binom{j}{2j-i} a_{k-1}^{i-j}\right\}\right)$. Since for every fixed $j > 2i/3$, and $1 \leq i \leq n$,

$$\binom{j}{2j-i} a_{k-1}^{i-j} \leq \binom{j}{2j-n} a_{k-1}^{n-j},$$

the worst case running time of the algorithm is

$$\mathcal{O}^*\left(\max\left\{\max_{1 \leq i \leq n} 2^{2i/3} a_{k-1}^{i/3}, \max_{2n/3 \leq j \leq n}\left\{\binom{j}{2j-n} a_{k-1}^{n-j}\right\}\right\}\right).$$

Finally, $\binom{2n/3}{n/3} = 2^{2n/3}$ up to a polynomial factor, and thus the running time is $\mathcal{O}^*\left(\max_{2n/3 \leq j \leq n}\left\{\binom{j}{2j-n} a_{k-1}^{n-j}\right\}\right)$. □

Based on the $\mathcal{O}(1.2377^n)$ algorithm for $\#\mathrm{MHS}_2$ [23], the worst-case running time of the algorithm of Theorem 2 is obtained for $0.7049n < j < 0.7050n$.

Corollary 1. $\#\mathrm{MHS}_3$ *can be solved in time* $\mathcal{O}(1.7198^n)$.

The same approach can be used design an algorithm for the optimization version MHS_k, assuming that an algorithm for MHS_{k-1} is available. Based on the $\mathcal{O}(1.6278^n)$ algorithm for MHS_3 [23] this leads to an $\mathcal{O}(1.8704^n)$ time algorithm for solving MHS_4 (in that case, the maximum is obtained for $0.6824n < j < 0.6825n$).

Corollary 2. MHS_4 *can be solved in time* $\mathcal{O}(1.8704^n)$.

In the following theorem we provide an alternative approach to solve $\#\mathrm{MHS}_k$. This is a combination of brute force enumeration (for sufficiently large hitting sets) with one application of the compression algorithm of Lemma 2. For large values of a_{k-1}, more precisely for $a_{k-1} \geq 1.6553$, this new approach gives faster algorithms than the one obtained by Theorem 2.

k	$\#\mathrm{MHS}_k$	MHS_k
2	$\mathcal{O}(1.2377^n)$ [23]	$\mathcal{O}(1.2108^n)$ [21]
3	$\mathcal{O}(1.7198^n)$	$\mathcal{O}(1.6278^n)$ [23]
4	$\mathcal{O}(1.8997^n)$	$\mathcal{O}(1.8704^n)$
5	$\mathcal{O}(1.9594^n)$	$\mathcal{O}(1.9489^n)$
6	$\mathcal{O}(1.9824^n)$	$\mathcal{O}(1.9781^n)$
7	$\mathcal{O}(1.9920^n)$	$\mathcal{O}(1.9902^n)$

Fig. 1. Running times of the algorithms for $\#\mathrm{MHS}_k$ and MHS_k.

Theorem 3. *Suppose there exists an algorithm with running time $\mathcal{O}^*(a_{k-1}^n)$, $1 < a_{k-1} \leq 2$, solving $\#\mathrm{MHS}_{k-1}$. Then $\#\mathrm{MHS}_k$ can be solved in time*

$$\min_{0.5 \leq \alpha \leq 1} \max \left\{ \mathcal{O}^* \left(\binom{n}{\alpha n} \right), \mathcal{O}^* \left(2^{\alpha n} a_{k-1}^{n-\alpha n} \right) \right\}.$$

Proof. First the algorithm tries all subsets of V of size $\lfloor \alpha n \rfloor$ and identifies those that are a hitting set of I.

Now there are two cases. In the first case, there is no hitting set of this size. Then the algorithm verifies all sets of larger size whether they are hitting sets of I. It is straightforward to keep some accounting information to determine the number of hitting sets of the smallest size found during this enumeration phase. The running time of this phase is $\mathcal{O}^* \left(\sum_{i=\lfloor \alpha n \rfloor}^{n} \binom{n}{i} \right) = \mathcal{O}^* \left(\binom{n}{\alpha n} \right)$.

In the second case, there exists a hitting set of size $\lfloor \alpha n \rfloor$. Then count all minimum hitting sets using the compression algorithm of Lemma 2 with H' being a hitting set of size $\lfloor \alpha n \rfloor$ found by the enumeration phase. By Lemma 2, this phase of the algorithm has running time $\mathcal{O}^* \left(2^{\alpha n} a_{k-1}^{n-\alpha n} \right)$. □

The best running times of algorithms solving $\#\mathrm{MHS}_k$ and MHS_k are summarized in Figure 1. For $\#\mathrm{MHS}_{\geq 4}$ and $\mathrm{MHS}_{\geq 5}$, we use the algorithm of Theorem 3. Note that the MHS_2 problem is equivalent to MVC and MIS.

4 Maximum Induced Cluster Subgraph

Clustering objects according to given similarity or distance values is an important problem in computational biology with diverse applications, e.g., in defining families of orthologous genes, or in the analysis of microarray experiments [5, 8, 11, 14, 18]. A graph theoretic formulation of the clustering problem is called Cluster Editing. To define this problem we need to introduce the notion of a *cluster graph* . A graph is called a cluster graph if it is a disjoint union of cliques. In the most common parameterized version of Cluster Editing, given an input graph $G = (V, E)$ and a positive integer k, the question is whether the input graph G can be transformed into a cluster graph by adding or deleting

at most k edges in time $f(k) \cdot n^{O(1)}$, where f is an arbitrary computable function. This problem has been extensively studied in the realm of parameterized complexity [5, 8, 11, 18]. In this section, we study a vertex version of Cluster Editing. We study the following optimization version of the problem.

> Maximum Induced Cluster Subgraph (MICS): Given a graph $G = (V, E)$ on n vertices, find a maximum size subset $C \subseteq V$ such that $G[C]$, the subgraph of G induced by C, is a cluster graph.

Due to the following well-known observation, the MICS problem is also known as Maximum Induced P_3-free Subgraph.

Observation 1 *A graph is a disjoint union of cliques if and only if it contains no induced subgraph isomorphic to the graph P_3, the path on 3 vertices.*

Clearly, $C \subseteq V$ induces a cluster graph in $G = (V, E)$ (that is $G[C]$ is a disjoint union of cliques of G) iff $S = V \setminus C$ hits all induced paths on 3 vertices of G. Thus solving the MICS problem is equivalent to finding a minimum size set of vertices whose removal produces a maximum induced cluster subgraph of G. By Observation 1, this reduces to finding a minimum hitting set S of the collection of vertex sets of (induced) P_3's of G. Such a hitting set S is called a P_3-HS.

As customary when using iterative compression, we first define a compression version of the MICS problem.

> Comp-MICS: Given a graph $G = (V, E)$ on n vertices and a P_3-HS $S \subseteq V$, find a P_3-HS of G of size at most $|S| - 1$ if one exists.

Lemma 3. Comp-MICS *can be solved in time $\mathcal{O}(1.6181^n)$.*

Proof. For the proof we distinguish two cases based on the size of S.

Case 1: If $|S| \leq 2n/3$ then the following algorithm which uses matching techniques is applied.

Step 1: Enumerate all partitions of $(N, \bar{N})$ of S.
Step 2: For each partition, compute a maximum set $C \subseteq V$ such that $G[C]$ is a cluster graph, subject to the constraints that $N \subseteq C$ and $\bar{N} \cap C = \emptyset$, if such a set C exists.

In **Step 2**, we reduce the problem of finding a maximum sized C to the problem of finding a maximum weight matching in an auxiliary bipartite graph.[5] For completeness, we present the details of **Step 2**.

If $G[N]$ contains an induced P_3 then there is obviously no $C \subseteq V$ inducing a cluster graph that respects the partition $(N, \bar{N})$. We call such a partition *invalid*.

[5] We recently learnt about the paper by Hüffner et al. [14] where the authors obtain among others an FPT algorithm for the vertex weighted version of Cluster Vertex Deletion using iterative compression. In their compression step they use the natural idea of reduction to weighted bipartite matching that we also established independently.

Otherwise, $G[N]$ is a cluster graph, and thus the goal is to find a maximum size subset C' of $\overline{S} = V \setminus S$ such that $G[C' \cup N]$ is a cluster graph. Fortunately, such a set C' can be computed in polynomial time by reducing the problem to finding a maximum weight matching in an auxiliary bipartite graph.

First we describe the construction of the bipartite graph. Consider the graph $G[N \cup \overline{S}]$ and note that $G[N]$ and $G[\overline{S}]$ are cluster graphs. Now the following reduction rule is applied to the graph $G[N \cup \overline{S}]$.

(R) Remove every vertex $b \in \overline{S}$ for which $G[N \cup \{b\}]$ contains an induced P_3.

Clearly all vertices removed by **(R)** cannot belong to any C' inducing a cluster subgraph of G. Let $\hat{S}$ be the subset of vertices of $\overline{S}$ which are not removed by **(R)**. Hence the current graph is $G[N \cup \hat{S}]$. Clearly $G[\hat{S}]$ is a cluster graph since $G[\overline{S}]$ is one. Further, note that no vertex of $\hat{S}$ has neighbors in two different maximal cliques of $G[N]$ and if a vertex of $\hat{S}$ has a neighbor in one maximal clique of $G[N]$ then it is adjacent to each vertex of this maximal clique. Thus, every vertex in $\hat{S}$ has either no neighbor in N or it is adjacent to all the vertices of exactly one maximal clique of $G[N]$.

Now we are ready to define the auxiliary bipartite graph $G' = (A, B, E')$. Let $\{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_r\}$ be the maximal cliques of the cluster graph $G[N]$. Let $\{\mathcal{C}'_1, \mathcal{C}'_2, \cdots, \mathcal{C}'_s\}$ be the maximal cliques of the cluster graph $G[\hat{S}]$. Let $A = \{a_1, a_2, \ldots, a_r, a'_1, a'_2, \ldots, a'_s\}$ and $B = \{b_1, b_2, \ldots, b_s\}$. Here, for all $i \in \{1, \ldots, r\}$, each maximal clique $\mathcal{C}_i$ of $G[N]$ is represented by $a_i \in A$; and for all $j \in \{1, 2, \ldots, s\}$, each maximal clique $\mathcal{C}'_j$ of $G[\hat{S}]$ is represented by $a'_j \in A$ and by $b_j \in B$.

Now there are two types of edges in G': $a_j b_k \in E'$ if there is a vertex $u \in \mathcal{C}'_k$ such that u has a neighbor in $\mathcal{C}_j$, and $a'_j b_j \in E'$ if there is a vertex $u \in \mathcal{C}'_j$ such that u has no neighbor in N. Finally we define the weights for both types of eges in the bipartite graph G'. For an edge $a_j b_k \in E'$, its weight $w(a_j b_k)$ is the number of vertices in $\mathcal{C}'_k$ being adjacent to all vertices of the maximal clique $\mathcal{C}_j$. For an edge $a'_j b_j$, its weight $w(a'_j b_j)$ is the number of vertices in $\mathcal{C}'_j$ without any neighbor in N.

This transformation is of interest due to the following claim that uses the above notation.

Claim. [$\star$] The maximum size of a subset C' of $\hat{S}$ such that $G[N \cup C']$ is a cluster subgraph of the graph $G^* = G[N \cup \hat{S}]$ is equal to the maximum total weight of a matching in the bipartite graph $G' = (A, B, E')$.

Note that the construction of the bipartite graph G', including the application of **(R)** and the computation of a maximum weighted matching of G' can be performed in time $\mathcal{O}(n^3)$ [7]. Thus, the running time of the algorithm in Case 1 is the time needed to enumerate all subsets of S (whose size is bounded by $2n/3$) and this time is $\mathcal{O}^*(2^{2n/3}) = \mathcal{O}(1.5875^n)$.

Case 2: If $|S| > 2n/3$ then the algorithm needs to find a P_3-HS of G of size $|S| - 1$, or show that none exists.

The algorithm proceeds as in the first case. Note that at most $n-|S|$ vertices of $V \setminus S$ can be added to N. Therefore, the algorithm verifies only those partitions $(N, \bar{N})$ of S satisfying $|N| \geq |S| - 1 - (n - |S|) = 2|S| - n - 1$. In this second case, the worst-case running time is obtained for $0.7236 < \alpha < 0.7237$, and it is

$$\mathcal{O}^* \left(\max_{2/3<\alpha\leq 1} \left\{ \binom{\alpha n}{(2\alpha - 1)n} \right\} \right) = \mathcal{O}(1.6181^n). \qquad \square$$

Now we are ready to prove the following theorem using iterative compression.

Theorem 4. MICS *can be solved in time* $\mathcal{O}(1.6181^n)$ *on a graph on* n *vertices.*

Proof. Given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$. Let $G_i = G[\{v_1, \ldots, v_i\}]$ and let C_i be a maximum induced cluster subgraph of G_i. Let $S_i = V_i \setminus C_i$.

The algorithm starts with G_1, $C_1 = \{v_1\}$ and $S_1 = \emptyset$. At the ith iteration of the algorithm, $1 \leq i \leq n$, we maintain the invariant that we have at our disposal C_{i-1} a maximum set inducing a cluster subgraph of G_{i-1} and S_{i-1} a minimum P_3-HS of G_{i-1}. Note that $S_{i-1} \cup \{v_i\}$ is a P_3-HS of G_i and that no P_3-HS of G_i has size smaller than $|S_{i-1}|$. Now use the algorithm of Lemma 3 to solve Comp-MICS on G_i with $S = S_{i-1} \cup \{v_i\}$. Then the worst-case running time is attained at the nth stage of the iteration and the run time is $\mathcal{O}(1.6181^n)$. $\square$

5 Conclusion

Iterative compression is a technique which is succesfully used in the design of FPT algorithms. In this paper we show that this technique can also be used to design exact exponential time algorithms. This suggests that it might be used in other areas of algorithms as well. For example, how useful can iterative compression be in the design of approximation algorithms?

Carrying over techniques from the design of FPT algorithms to the design of exact exponential time algorithms and vice-versa is a natural and tempting idea. A challenging question in this regard is whether Measure and Conquer, a method that has been succesfully used to improve the time analysis of simple exponential-time branching algorithms, can be adapted for the analysis of FPT branching algorithms.

References

1. A. Björklund and T. Husfeldt, Inclusion–Exclusion Algorithms for Counting Set Partitions, in the proceedings of FOCS'06, 575–582 (2006).
2. A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto, Fourier meets Möbius: Fast Subset Convolution, in the proceedings of STOC'07, 67–74 (2007).
3. J. M. Byskov, Enumerating maximal independent sets with applications to graph colouring, Oper. Res. Lett. 32(6), 547–556 (2004).
4. J. Chen, Y. Liu, S. Lu, I. Razgon, B. O'Sullivan, A Fixed-Parameter Algorithm for the Directed Feedback Vertex Set Problem, in the proceedings of STOC'08, 177–186 (2008).

5. F. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw and Y. Zhang, The Cluster Editing Problem: Implementations and Experiments, in the proceedings of IWPEC'06, 13-24 (2006).
6. F. Dehne, M. Fellows, M. Langston, F. Rosamond, and K. Stevens, An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem, in the proceedings of COCOON'05, 859–869 (2005).
7. J. Edmonds and R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM 19(2), 248–264 (1972).
8. M. R. Fellows, M. A. Langston, F. A. Rosamond and P. Shaw, Efficient Parameterized Preprocessing for Cluster Editing, in the proceedings of FCT'07, 312-321 (2007).
9. H. Fernau, Parameterized Algorithms for Hitting Set: The Weighted Case, in the proceedings of CIAC'06, 332-343 (2006).
10. F. V. Fomin, F. Grandoni, and D. Kratsch, Measure and conquer: A simple $O(2^{0.288\,n})$ independent set algorithm, in the proceedings of SODA'06, 18–25 (2006).
11. J. Guo, A More Effective Linear Kernelization for Cluster Editing, in the proceedings of ESCAPE'07, 36-47 (2007).
12. J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization, J. Comput. Syst. Sci. 72(8), 1386–1396 (2006).
13. J. E. Hopcroft and R.M. Karp, An $n^{5/2}$ algorithm for maximum matching in bipartite graphs, SIAM J. Computing 2(4), 225–231 (1973).
14. F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier, Fixed-parameter algorithms for cluster vertex deletion, in the proceedings of LATIN'08, 711-722 (2008).
15. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, On Generating All Maximal Independent Sets, Inf. Process. Lett. 27(3), 119–123 (1988).
16. M. Koivisto, An $O(2^n)$ Algorithm for Graph Colouring and Other Partitioning Problems via Inclusion-Exclusion, in the proceedings of FOCS'06, 583–590 (2006).
17. J. W. Moon and L. Moser, On Cliques in Graphs, Israel J. Mathematics 3, 23–28 (1965).
18. S. Rahmann , T. Wittkop , J. Baumbach , M. Martin , A. Trub and S. Böcker, Exact and Heuristic Algorithms for Weighted Cluster Editing, in the proceedings of Comput. Syst. Bioinformatics Conference'07, 6 (1), 391-401(2007).
19. V. Raman , S. Saurabh, and S. Sikdar, Efficient Exact Algorithms through Enumerating Maximal Independent Sets and Other Techniques, Theory Comput. Syst. 41(3), 1432-4350 (2007).
20. B. A. Reed, K. Smith, and A. Vetta, Finding odd cycle transversals, Oper. Res. Lett. 32 (4), 299-301 (2004).
21. J. M. Robson, Algorithms for maximum independent sets, J. Algorithms 7, 425–440 (1986).
22. R. Tarjan and A. Trojanowski, Finding a maximum independent set, SIAM J. Computing 6(3), 537–546 (1977).
23. M. Wahlström, Algorithms, measures and upper bounds for satisfiability and related problems, PhD thesis, Linköping University, Sweden, (2007).