Algorithmics of Large and Complex Networks, LNCS 5515, pp. 65-80, 2009

Iterative Compression for Exactly Solving NP-Hard Minimization Problems

Jiong Guo*, Hannes Moser**, and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{jiong.guo,hannes.moser,rolf.niedermeier}@uni-jena.de

Abstract. We survey the conceptual framework and several applications of the iterative compression technique introduced in 2004 by Reed, Smith, and Vetta. This technique has proven very useful for achieving a number of recent breakthroughs in the development of fixed-parameter algorithms for NP-hard minimization problems. There is a clear potential for further applications as well as a further development of the technique itself. We describe several algorithmic results based on iterative compression and point out some challenges for future research.

1 Introduction

Until the year 2004, the parameterized complexity of several important NP-hard minimization problems was open. Then, Reed, Smith, and Vetta [42] introduced in a very short paper a new technique that is now called iterative compression. Meanwhile, based on this technique, a number of the mentioned open questions could be positively answered by giving corresponding fixed-parameter algorithms. To become more specific, let us consider the NP-complete Vertex Bipartization problem, where one is given an undirected graph $G = (V, E)$ and an integer $k \geq 0$, and the question is whether there is a set of at most k vertices such that their deletion leaves a bipartite graph. Due to iterative compression, now it is known that Vertex Bipartization can be solved in $O(3^k \cdot |V||E|)$ time [42,29]. In other words, Vertex Bipartization is fixed-parameter tractable with respect to the parameter k. Since then, similar breakthroughs have been achieved, for example, for the NP-complete problems Undirected Feedback Vertex Set [11,28,8], Directed Feedback Vertex Set [10], and Almost 2-Sat [41]. Here, we review the central ideas behind iterative compression and (some of) its applications. To this end, we choose the recently studied NP-complete Cluster Vertex Deletion problem [32] as our running example for exhibiting the "essentials" of iterative compression.

* Supported by the DFG, PALG, NI 369/8.
** Supported by the DFG, projects ITKO, NI 369/5 (part of the DFG-SPP 1126 "Algorithms on Large and Complex Networks") and AREG, NI 369/9.

Notation. For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, let $G[S]$ be the subgraph of G induced by S. A parameterized problem (I, k) is *fixed-parameter tractable* with respect to the parameter k if it can be solved in $f(k) \cdot \text{poly}(|I|)$ time, where I is the input instance (see [15,19,38]). Fixed-parameter tractability can be viewed as an alternative to polynomial-time approximability in dealing with NP-hard problems. An exact algorithm showing the fixed-parameter tractability of a parameterized problem is called *fixed-parameter algorithm.* In all graph problems that follow, n denotes the number of vertices and m denotes the number of edges.

2 Illustration of the Basic Technique

In the following, we exhibit the iterative compression technique by using the NP-complete Cluster Vertex Deletion (CVD) problem, arising in graph-based data clustering, as a running example.

> **Input:** An undirected graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a vertex subset $S \subseteq V$ with $|S| \leq k$ such that deleting all vertices in S from G results in a cluster graph, that is, a graph where every connected component forms a clique?

We present a simplified version of a more general algorithm [32] that solves the vertex-weighed variant of Cluster Vertex Deletion. Note that a graph is a cluster graph if and only if it contains no *induced* path of three vertices (referred to as P_3)[1].

The central idea of iterative compression is to employ a so-called *compression routine.* A *compression routine* is an algorithm that, given a problem instance and a corresponding solution, either calculates a smaller solution or proves that the given solution is of minimum size. Using a compression routine, one finds an optimal solution to a problem by inductively building up the problem structure and iteratively compressing intermediate solutions.

Example CVD. In the following, we call a solution for CVD a *cvd-set.* Here, the problem structure is built up vertex by vertex. We start with $V' = \emptyset$ and $S = \emptyset$; clearly, S is a cvd-set for $G[V']$. Iterating over all graph vertices, step by step we add one vertex $v \in V \setminus V'$ to both V' and S. Then S is still a cvd-set for $G[V']$. In each step, if $|S| > k$, then we try to find a smaller cvd-set for $G[V']$ by applying a compression routine. It takes the graph $G[V']$ and the cvd-set S for $G[V']$, and returns a smaller cvd-set for $G[V']$, or proves that S is optimal. If S is optimal, then we can conclude that G does not have a cvd-set of size at most k. Since eventually $V' = V$, we obtain a solution for G once the algorithm returns S. Note that almost all known applications of iterative compression on graph problems with vertex subsets as solutions essentially build up the graph in this way.[2]

[1] Three vertices $u, v, w \in V$ of a graph $G = (V, E)$ form an induced path if exactly two of the three edges $\{u, v\}$, $\{u, w\}$, $\{v, w\}$ are contained in E.

[2] An alternative example where the graph is built up edge by edge is given with the NP-complete Edge Bipartization problem [28].

The main point of the iterative compression technique is that if the compression routine is a fixed-parameter algorithm, then so is the whole algorithm. The main strength of iterative compression is that it allows to see the problem from a different angle: The compression routine does not only have the problem instance as input, but also a solution, which carries valuable structural information on the input. Therefore, the design of a compression routine may be simpler than designing a fixed-parameter algorithm for the original problem.

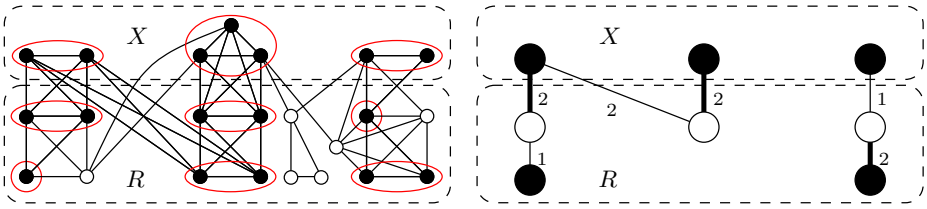
While embedding the compression routine into the iteration framework is usually straightforward, finding the compression routine itself is not. It is not even clear that a compression routine with useful running time exists even when we already know a problem to be fixed-parameter tractable. Therefore, the art of iterative compression typically lies in the design of the compression routine. In many applications of iterative compression, the compression routine first exhaustively considers all possible intersection sets of the given solution and a potentially smaller solution; elements in the intersection can be "discarded" from the problem instance.[3] Then, the remaining task is to find a smaller *disjoint* solution.

Example CVD continued. For Cluster Vertex Deletion the compression routine works as follows. Consider a smaller cvd-set S' as a modification of the larger cvd-set S for the graph $G = (V, E)$. This modification retains some vertices $Y \subsetneq S$ as part of the solution set (that is, the vertices to be deleted), while the other vertices $X := S \setminus Y$ are replaced by new vertices from $V \setminus S$. The idea is to try by brute force all $2^{|S|} - 1$ nontrivial partitions of S into these two sets Y and X. For each such partition, the vertices from Y are immediately deleted, since we already decided to take them into the cvd-set. In the resulting instance $G' = (V', E') := G[V \setminus Y]$, it remains to find a smaller cvd-set that is disjoint from X. This turns out to be a much easier task than finding a cvd-set in general; in fact, it can be done in polynomial time using data reduction and maximum matching.

The idea to try by brute force all nontrivial partitions of a given solution S into two sets Y and X is applied for all the problems in this survey; thus, we always describe the corresponding compression routines by showing how to compute a smaller *disjoint* solution.

Compression for CVD. Recall that X is a cvd-set for G', and the task is to find a smaller cvd-set for G' disjoint from X. First, we discard partitions where X does not induce a cluster graph; these partitions cannot lead to a solution since we fixed that none of the vertices in X would be deleted. Further, the set $R := V' \setminus X$ also induces a cluster graph since $R = V \setminus S$ and S is a cvd-set. Therefore, the following computational problem remains:

[3] It depends on the concrete problem of how we "discard" these elements. For instance, for a vertex deletion problem we may just remove the corresponding vertices.



(a) CVD Compression instance (b) The assignment problem

Fig. 1: (a) Data reduction in the compression routine. The white vertices in the input instance are removed by the three data reduction rules. Each group of encircled vertices corresponds to a vertex in the assignment problem (b). If a group of encircled vertices is in X or if no vertex in this group has a neighbor in X, then the corresponding vertex is black, otherwise, it is white.

CVD Compression
Instance: An undirected graph $G = (V, E)$ and a vertex set $X \subseteq V$ such that $G[X]$ and $G[V \setminus X]$ are cluster graphs.
Task: Find a vertex set $X' \subseteq V \setminus X$ such that $G[V \setminus X']$ is a cluster graph and $|X'| < |X|$.

An example for a CVD Compression instance is shown in Figure 1a. In a first step, this instance can be simplified by a series of simple data reduction rules; their correctness is easy to see [32]:

1. Delete all vertices in $R := V \setminus X$ that are adjacent to more than one clique in $G[X]$.
2. Delete all vertices in R that are adjacent to some, but not all vertices of a clique in $G[X]$.
3. Remove connected components that are cliques.

After these data reduction rules have been exhaustively applied, the instance is much simplified: In each clique of $G[R]$, we can divide the vertices into equivalence classes according to their neighborhood in X; each class then contains either vertices adjacent to all vertices of a particular clique in $G[X]$, or the vertices adjacent to no vertex in X (see Figure 1a). This classification is useful because of the following.

Lemma 1. *If there exists a solution for* CVD Compression*, then in the cluster graph resulting by this solution, for each clique in $G[R]$ the vertices of at most one equivalence class are present.*

Proof. Clearly, inside a clique, it is never useful to delete only some, but not all vertices of an equivalence class, since if that led to a solution, we could always re-add the deleted vertices without introducing new induced P_3's (which are the forbidden substructure characterizing cluster graphs). Further, assume that for a clique C in $G[R]$ the vertices of two equivalence classes are present. Let $u \in C$

and $v \in C$ be a vertex from each equivalence class, respectively. Since u and v are in different equivalence classes, they must have a different neighborhood with respect to the cliques in $G[X]$. Assume without loss of generality that v is adjacent to all vertices of a clique C' in $G[X]$. Since u is in an other equivalence class than v, u is not adjacent to any vertex of C'. Let $w \in C'$. The path uvw forms an induced P_3, contradicting our assumption. □

Due to Lemma 1, the remaining task for solving CVD Compression is to assign each clique in $G[R]$ to one of its equivalence classes (corresponding to the preservation of this class, and the deletion of all vertices from the other classes within the clique) or to nothing (corresponding to the complete deletion of the clique). However, we cannot do this independently for each clique; we must not choose two classes from different cliques in $G[R]$ such that these two classes are adjacent to the same clique in $G[X]$ since that would create an induced P_3. This assignment problem can be modelled as a weighted bipartite matching problem in an auxiliary graph H, where each edge corresponds to a possible choice. The graph H is constructed as follows (see Figure 1b):

1. Add a vertex for every clique in $G[R]$ (white vertices).
2. Add a vertex for every clique in $G[X]$ (black vertices in X).
3. For a clique C_X in $G[X]$ and a clique C_R in $G[R]$, add an edge between the vertex for C_X and the vertex for C_R if there is an equivalence class in C_R containing a vertex adjacent to a vertex in C_X. This edge corresponds to choosing this class for C_R and one assigns the number of vertices in this class as its weight.
4. Add a vertex for each class in a clique C_R that is not adjacent to a clique in $G[X]$ (black vertices outside X), and connect it to the vertex representing C_R. Again, this edge corresponds to choosing this class for C_R and is weighted with the number of vertices in this class.

Since we only added edges between black and white vertices, H is bipartite. The task is now to find a *maximum-weight bipartite matching*, that is, a set of edges of maximum weight where no two edges have an endpoint in common. To solve this matching instance, we can use an algorithm for integer weighted matching with a maximum weight of n [23], yielding a running time of $O(m\sqrt{n}\log n)$. If we apply the data reduction rules in their given order, we can execute them in $O(m)$ time. Thus, we can solve CVD Compression in $O(m\sqrt{n}\log n)$ time. The number of vertices in an intermediary solution S to be compressed is bounded from above by $k+1$, because any such S consists of a size-at-most-k solution for a subgraph of G plus a single vertex. In one iteration step, CVD Compression is thus solved $O(2^k)$ times, and there are n iteration steps, yielding a total running time of $O(2^k \cdot mn^{3/2}\log n)$. With some tricks based on problem kernelization, which will not be explained here, this can be further improved to $O(2^k k^6 \log k + nm)$.

Theorem 1 ([32]). Cluster Vertex Deletion *can be solved within a running time of* $O(2^k k^6 \log k + nm)$.

This iterative compression approach combined with matching techniques also works for the vertex-weighted version of Cluster Vertex Deletion, yielding an algorithm with a running time of $O(2^k k^9 + nm)$ [32].

As we have seen for Cluster Vertex Deletion, the iterative part of building up the graph vertex by vertex is relatively simple, but the problem-specific compression step of applying data reduction and matching techniques is more involved. In many applications of iterative compression, the iterative part is carried out in the same manner, and data reduction techniques often play an important role for the compression step.

3 Some Breakthroughs due to Iterative Compression

In this section, we survey recent breakthroughs in parameterized algorithmics that all are based on a sophisticated use of iterative compression. We mainly focus on describing the central ideas behind the respective compression routines.

3.1 Graph Bipartization

Definition and History. The NP-complete Graph Bipartization problem, also known as Vertex Bipartization, Maximum Bipartite Induced Subgraph, or Odd Cycle Transversal, is defined as follows.

> **Input:** An undirected graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a vertex subset $S \subseteq V$ with $|S| \leq k$ such that $G[V \setminus S]$ is bipartite?

Graph Bipartization has applications ranging from VLSI design to computational biology (see, e.g., [34,45]). There is a polynomial-time approximation with a factor of $O(\log n)$ [24], and there is an exact algorithm running in $O(1.62^n)$ time [40]. Mahajan and Raman [35] explicitly mentioned the fixed-parameter tractability of Graph Bipartization with respect to the parameter k as an open problem, which was settled by Reed et al. [42], thereby introducing the iterative compression technique. The running time of their approach is stated as $O(4^k \cdot knm)$, but with a better analysis and an algorithmic trick, it can been improved to $O(3^k \cdot nm)$ [29,30].

Compression Routine. In the following, we outline the basic idea of the compression routine due to Reed et al. [42]. We have as input an undirected graph $G = (V, E)$ and a vertex set $X \subseteq V$ such that $G[V \setminus X]$ is bipartite. The question is whether there exists a set X' with $X' \subseteq V \setminus X$ and $|X'| < |X|$ such that $G[V \setminus X']$ is bipartite. Let I_1 and I_2 be the two partite sets of $G[V \setminus X]$. The idea is to construct an auxiliary graph corresponding to G: Replace every vertex $v \in X$ by two vertices v_1, v_2, deleting all edges incident to v. For every edge vw in G with $w \in V \setminus X$, we add the edge $v_1 w$ if $w \in I_2$ and $v_2 w$ if $w \in I_1$. For every edge vw with $w \in X$, we arbitrarily add the edge $v_1 w_2$ or the edge $v_2 w_1$. See Figure 2 for an example. The idea behind that construction is to enforce that

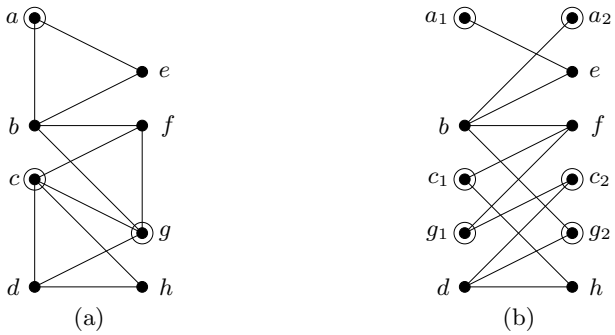


Fig. 2: (a) G with solution (encircled vertices); (b) Corresponding auxiliary graph.

if there exists an odd cycle (a cycle with an odd number of edges) in G going through a vertex a (e.g., the cycle (a, b, e) in Figure 2), then there exists a path from a_1 to a_2 in the auxiliary graph. Note that a path from a_1 to a_2 in the auxiliary graph only implies some odd cycle in G (but not necessarily containing a). Let $N_X := \{v_1, v_2 \mid v \in X\}$ be the set of the newly introduced vertices. A partition (A, B) of N_X is *valid* if for each vertex pair v_1, v_2 either $v_1 \in A$ and $v_2 \in B$ or $v_1 \in B$ and $v_2 \in A$. The intuitive idea of the compression routine is to try to find a valid partition (A, B) such that all paths between A and B can be obstructed by less than $|X|$ vertices. These vertices also obstruct every odd cycle in G (by the construction of the auxiliary graph). The following lemma, which is a variation of Lemma 1 in [42] and will not be proven here, shows that this approach is correct.

Lemma 2. *If for any valid partition (A, B) of N_X there are $|X|$ vertex-disjoint paths from A to B in the corresponding auxiliary graph, then there is no solution X' with $X' \subseteq V \setminus X$ and $|X'| < |X|$.*

Using Lemma 2, we can enumerate in $2^{|X|}$ steps all possible valid partitions (A, B) of N_X, and compute a vertex cut between (A, B) using maximum flow techniques. If the vertex cut contains less than $|X|$ vertices, then we return it as the new solution X'. The maximum flow can be computed in $O(km)$ time [22], and since $|X| \leq k+1$ the overall running time of the compression routine is $O(2^k \cdot km)$. With an algorithmic trick, which "recycles" the flow networks for each maximum flow problem, one can get rid of the factor k in the running time [29]. Together with the iteration and the partitioning needed for the compression routine, an improved analysis, not described here, yields the following.

Theorem 2 ([42,29]). Graph Bipartization *can be solved within a running time of $O(3^k \cdot nm)$.*

Further Remarks. There exists another approach using vertex colorings to describe the Graph Bipartization algorithm [30]. The Graph Bipartization

algorithm has also been heuristically improved and implemented [29,30].[4] The experiments on data from computational biology show that iterative compression can outperform other methods by orders of magnitude. For example, an instance originating from computational biology with 102 vertices and 307 edges can be solved in 6248 seconds with an ILP approach, whereas an iterative compression approach runs in 0.79 seconds, which can be further improved by algorithmic tricks [29]. The iterative compression approach also works for the variant of making a given graph bipartite by at most k edge deletions, and yields an algorithm with a running time of $O(2^k \cdot m^2)$ [28]. This algorithm can be used for Signed Graph Balancing, where experiments show that this approach has about the same running time as approximation algorithms, while producing exact solutions [31].[5]

The Almost 2-SAT problem is a generalization of Graph Bipartization. It asks whether it is possible to delete at most k clauses from a Boolean formula in conjunctive normal form with at most two literals per clause such that the remaining formula becomes satisfiable. Very recently, Razgon and O'Sullivan [41] showed that Almost 2-SAT can be solved in $O(15^k k \cdot m^3)$ time, thus proving it to be fixed-parameter tractable with respect to parameter k, which was an open question stated by Mahajan and Raman [35]. This result also implies that Vertex Cover parameterized above the size of a maximum matching M, that is, the task to find a vertex cover of size at most $|M| + k$ ("above guarantee parameterization") [37], is fixed-parameter tractable with respect to the parameter k, because it can be transformed to Almost 2-SAT [41] in $f(k) \cdot \text{poly}(n)$ time.

3.2 Undirected Feedback Vertex Set

Definition and History. The NP-complete Undirected Feedback Vertex Set (UFVS) problem is defined as follows.

> **Input:** An undirected graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a *feedback vertex set* (fvs) $S \subseteq V$ with $|S| \leq k$, that is, a set S whose deletion from G results in a forest?

UFVS has found applications in many fields, including deadlock prevention, program verification, and Bayesian inference [18]. UFVS can be approximated to a factor of 2 in polynomial time [1,4]. There is a simple and elegant randomized algorithm [3] that solves UFVS in $O(c4^k k \cdot n)$ time by finding a feedback vertex set of size k with probability at least $1 - (1 - 4^{-k})^{c4^k}$ for an arbitrary constant c. A recent exact algorithm for UFVS has running time of $O(1.7548^n)$ [21]. As to deterministic fixed-parameter algorithms, Bodlaender [7] and Downey and Fellows [14] were the first to show that the problem is fixed-parameter tractable.

[4] The source code of the Graph Bipartization solver and corresponding test data are available from `http://theinf1.informatik.uni-jena.de/occ`.

[5] The source code of the Signed Graph Balancing solver is available from `http://theinf1.informatik.uni-jena.de/bsg`.

In 2005, Dehne et al. [11] and Guo et al. [28] independently provided the first algorithms, based on iterative compression, with running times of $O(c^k \cdot nm)$ with $c \approx 11$. Finally, Chen et al. [8], also employing iterative compression, improved the constant c to 5.

Compression Routine. In the following, we briefly describe the basic idea behind the compression routine due to Chen et al. [8]. Herein, we have as input an undirected graph $G = (V, E)$ and an fvs X. The question is whether there exists an fvs X' with $|X'| < |X|$ and $X' \subseteq V \setminus X$. Observe that G can be divided into two forests: The induced subgraph $G[V \setminus X]$ is clearly acyclic. Moreover, $G[X]$ is also a forest consisting of at most $|X|$ trees; otherwise, there would not exist an fvs X' with $X \cap X' = \emptyset$. Based on this observation, the key idea of the compression routine is to merge these two forests into one by deleting as few as possible vertices from $V \setminus X$; that is, it considers every vertex $v \in V \setminus X$ and tries to move v from $V \setminus X$ to X without introducing a cycle in $G[X]$. Since a solution X' must be disjoint from X, we have to include v into X' if its addition to X creates a cycle. If there is no cycle created by adding v to X, then make a trivial branch into two subcases, namely, (1) keeping X unchanged, adding v to X', and deleting v from G or (2) extending X by v. In the latter case, we assume that v is not part of a solution. More specifically, the compression routine follows a search tree strategy which distinguishes the following cases: First, if there is a vertex $v \in V \setminus X$ that has two neighbors from the same tree in $G[X]$, then add v to X' and delete v from G. This is clearly correct, since moving v to X introduces cycles. Second, if the first case does not apply and there is a vertex $v \in V \setminus X$ that has at least two neighbors in X, then we know that these neighbors are from different trees in $G[X]$ and, thus, $G[X \cup \{v\}]$ is acyclic; we branch the search into two subcases as described above. Finally, if the first two cases do not apply, then all vertices in $V \setminus X$ have at most one neighbor in X. We apply the following bypassing reduction rule to the leaves of the forest $G[V \setminus X]$, until one the first two cases applies or $V \setminus X = \emptyset$. We say that a graph G' is obtained from G by *bypassing* a degree-2 vertex v in G if G' is obtained by first removing v and then adding a new edge between its two neighbors. The following data reduction rule is trivially correct, because deleting degree-2 vertices to destroy cycles is never the only best solution.

Bypassing reduction rule: Bypass all degree-2 vertices in G.

To analyze the running time of the compression routine, we have to bound the search tree size. Since only the second case causes a branching into two subcases, it remains to upper-bound the number of the applications of the second case. We claim that this number is bounded by $j + l$, where j denotes the size of the input set X and l denotes the number of trees in $G[X]$. In the first subcase of the branching caused by the second case, we add a vertex to X', which can only happen at most $j - 1$ times, because we search an X' with $|X'| < |X|$. The second subcase adds vertex v to X. Since v has at least two neighbors in X which are from different trees in $G[X]$ (the precondition of the second case), the addition of v to X causes a merge of at least two trees and, thus, decreases the

number of trees in $G[X]$. Clearly, this can be done at most l times. Altogether, in the branchings caused by the second case, we either decrease the number of trees in $G[X]$ or add a vertex to X'. The second case can apply less than $j + l$ times. Since $j \leq k + 1$ and, thus, $G[X]$ has at most $k + 1$ trees, the search tree has size $O(2^{2k})$, giving an overall running time of the compression routine of $O(4^k \cdot n^2)$, where we need $O(n^2)$ time to apply the bypassing rule and to check the applicability of the three cases. Together with the iteration and the partitioning needed for the compression routine, one arrives at the following theorem.

Theorem 3 ([8]). Undirected Feedback Vertex Set *can be solved within a running time of* $O(5^k k \cdot n^2)$.

Further Remarks. The randomized algorithm by Becker et al. [3] is so elegant and simple such that it seems to be the current method of choice for practically computing an optimal undirected feedback vertex set; the current best iterative compression approach is still slower and more complicated to implement. The currently best problem kernel for UFVS has $O(k^2)$ vertices and edges [44]. Accepting a worse exponential base c, there is also a deterministic "linear-time FPT" algorithm for UFVS running in $O(c^k \cdot (m + n))$ time [28].

3.3 Directed Feedback Vertex Set

Definition and History. The NP-complete Directed Feedback Vertex Set (DFVS) problem is defined as follows.

> **Input:** A directed graph $G = (V, E)$ and a nonnegative number k.
> **Question:** Is there a *feedback vertex set* (fvs) $S \subseteq V$ with $|S| \leq k$, that is, a vertex set S whose deletion from G results in a graph with no directed cycles?

DFVS has various applications such as deadlock prevention in operating systems and database systems, circuit testing, voting systems, and computational biology [18]. DFVS can be approximated to a factor of $O(\log n \log \log n)$ in polynomial time [16]. For about 15 years, it has been repeatedly stated as an open question whether or not DFVS is fixed-parameter tractable with respect to the parameter k. Very recently, Chen et al. [10] gave a fixed-parameter algorithm with a running time of $O(k!4^k k^3 \cdot n^4)$, thus answering this open question. Comparing with the result for UFVS, the particular difficulty in case of DFVS is given by the fact that destroying all directed cycles not only leaves trees as in the undirected case but leaves directed acyclic graphs, much more complicated structures than trees. Thus, the result for UFVS might be interpreted as a first "necessary" intellectual step before the result for DFVS was within reach.

Compression Routine. In the following, we briefly describe the idea behind the compression routine due to Chen et al. [10]. Herein, we have as input a directed graph $G = (V, E)$ and an fvs X. The question is whether there exists an fvs X'

with $|X'| < |X|$ and $X' \subseteq V \setminus X$. Consider an fvs X' disjoint from X in G. Since there is no directed cycle in $G[V \setminus X']$, there must be a vertex $u \in X$ such that for each vertex $v \in X$ there is no directed path from u to v in $G[V \setminus X']$ (this can be easily seen: if there is a path from every vertex in X to at least one other vertex in X, following such paths from vertex to vertex will end up visiting an already visited vertex, yielding a directed cycle in $G[V \setminus X']$). A repeated application of this argument shows that there is an ordering $u_1, \ldots, u_{|X|}$ of the vertices in X such that there is no directed path in $G[V \setminus X']$ from u_i to u_j if $i \geq j$. To find the set X', the idea is now to try all $|X|!$ orderings of the vertices in X, and, for each ordering $u_1, \ldots, u_{|X|}$, we compute X' such that there is no directed path from u_i to u_j if $i \geq j$ in $G[V \setminus X']$. To this end, we split each vertex $u_i \in X$ into two vertices s_i and t_i such that all outgoing edges of u_i are incident to s_i and all incoming edges of u_i are incident to t_i. Then, the task is to find a vertex separator X' between the vertices $s_1, \ldots, s_{|X|}$ and $t_1, \ldots, t_{|X|}$ such that there is no path from s_i to t_j if $i \geq j$. This task can be solved by an algorithm for the Skew Separator problem, which is defined as follows.

> **Instance:** A directed graph $G = (V, E)$, a parameter k, and pairwise disjoint vertex subsets $S_1, \ldots, S_l$, $T_1, \ldots, T_l$ such that there is no edge going into a set S_i for $1 \leq i \leq l$ and such that there is no edge going out from a set T_i for $1 \leq i \leq l$.
> **Task:** Find a *skew separator* $X' \subseteq V$ for G with $|X'| \leq k$, that is, a vertex set X' such that there is no directed path from any vertex in S_i to any vertex in T_j in $G[V \setminus X']$ if $i \geq j$.

Skew Separator can be solved by a clever branching strategy in a running time of $O(4^k k \cdot n^3)$ [10]. The corresponding algorithm is a directed variant of an algorithm for the Minimum Node Multiway Cut problem on undirected graphs [9], where the task is, given a graph G, a parameter k, and vertex-disjoint terminal vertex sets, to delete at most k vertices such that there is no path between any two vertices of two different terminal sets. Using the algorithm for Skew Separator, one solves the task to find a vertex separator X' by setting $S_i := \{s_i\}$ and $T_i := \{t_i\}$ for all $1 \leq i \leq |X|$ and solving Skew Separator in $O(4^k k \cdot n^3)$ time.

Next, we analyze the running time of the whole iterative compression approach. There are $|X|! \leq (k+1)!$ orderings of the vertices in X, thus the algorithm for Skew Separator is called at most $(k+1)!$ times. Altogether, the compression routine runs in $O(k!4^k k^2 \cdot n^3)$ time. With some relatively simple analysis, it is possible to show that together with the iteration and the partitioning needed for the compression routine we arrive at the following theorem.

Theorem 4 ([10]). Directed Feedback Vertex Set *can be solved within a running time of* $O(k!4^k k^3 \cdot n^4)$.

Further Remarks. The above algorithm for DFVS finds a direct application in Kemeny voting systems for the problem of computing the Kemeny score in case

of incomplete votes [5], and for the dual of the Longest Compatible Sequence problem [27]. Moreover, in the special case of DFVS where the input graph is restricted to be a tournament (that is, a directed graph whose "underlying" undirected graph is complete), an iterative compression approach yields an algorithm with a running time of $O(2^k \cdot n^2(\log \log n + k))$ [13].

4 Discussion and Future Challenges

Discussion. Iterative compression is an algorithm design principle based on induction. This elegant and simple approach is appealing and invites for further applications and extensions. On the one hand, iterative compression may strongly benefit from a combination with kernelization and corresponding data reduction rules (see, for instance, the case of Undirected Feedback Vertex Set in Subsection 3.2), and, on the other hand, has also been employed for achieving (Turing) kernelization results [12]. Iterative compression may also be used for enumerating all minimal solutions of size at most k in FPT time, as the example Undirected Feedback Vertex Set shows [28]. In many applications of iterative compression, one often occurring running time bottleneck is that all partitions of a size-k solution have to be considered in order to find a smaller-size solution (see Section 2). This typically incurs an additional running time factor of 2^k. However, in some cases this factor can be avoided by proving a guarantee that the improved solution can without loss of generality be assumed to be disjoint from the old solution. An example for this is given by the Edge Bipartization problem [28]. It appears to be fruitful to combine iterative compression with efficient polynomial-time approximation algorithms. The idea is that the compression routine may start right away with the solution provided by a constant-factor approximation, saving time otherwise needed for the iteration procedure [28]. Iterative compression is not only a tool for purely theoretical algorithm design but has already proven practical usefulness in few experimental studies [29,30,31]. One reason for this practical success also lies in the flexibility of the use of the iterative compression routine [33]. It can be started on any suboptimal initial solution to improve this solution. Moreover, it can be stopped whenever the found solution is good enough. Finally, it should be mentioned that the known applications of iterative compression have a strong focus on graph modification problems with the goal to generate graphs with hereditary properties. It would be interesting to see more applications of iterative compression outside this scenario, the case of Almost 2-Sat [41] meaning a first step.

First experimental results for iterative compression-based algorithms appear quite encouraging. An implementation of the Graph Bipartization algorithm, improved by heuristics, can solve all instances from a testbed from computational biology within minutes, whereas established methods are only able to solve about half of the instances within reasonable time. For instance, in case of Graph Bipartization instances of sizes up to 296 vertices and 1620 edges could be solved within less than 4 minutes [29]. Further, an iterative compression based approach for the Balanced Subgraph problem, which generalizes

Edge Bipartization, is able to find optimal solutions to instances for which previously only approximate solutions could be given [31].

So far, the list of successful applications of iterative compression is impressive but still clear in its range. Future research has to determine how far-ranging applications of iterative compression can be found. Clearly, this survey only sketched a few results achieved by iterative compression. One further sophisticated and technically fairly demanding application of iterative compression has been developed by Marx [36] to show that the problem to delete a minimum number of vertices in order to make a graph chordal is fixed-parameter tractable.

Future Challenges. As already mentioned, there are close connections between iterative compression and approximation algorithms as well as kernelization. In both cases, there seems to be room for further fruitful explorations. A recent paper makes a first attempt in linking iterative compression with exact algorithms [20]. Another conceptually related field of research is that of reoptimization [6]. There, the scenario is that one is given an instance of an optimization problem together with an optimal solution, and one wants to find a "high-quality" solution for a locally modified instance. Finding useful connections between this scenario and the compression scenario would be highly interesting. Similarly, exploring connections to the paradigm of local search might be promising. In addition, it would be interesting to investigate the connections between augmentation problems in the context of 0/1-integer linear programming and iterative compression. In a nutshell, in these augmentation problem one is given a vector $c \in \mathbb{Z}^n$, a feasible solution x over $\{0,1\}^n$, and the task is to find out whether there is another feasible solution y such that $c \cdot y > c \cdot x$, or to assert that no such y exists [43].

Another general issue of future research may address the inductive structure of iterative compression. So far, all applications of iterative compression have a pretty simple inductive structure, and there is no reason to believe that more sophisticated structures might not be helpful. Finally, the experiences concerning algorithm engineering results for compression-based methods are still very limited and need to be extended.

We end this survey with a list of few open questions relating to concrete computational problems. The solution of each of these would mean progress for iterative compression and parameterized algorithmics at large.

- Directed Feedback Vertex Set now having been classified in terms of parameterized complexity analysis using iterative compression, a natural next step seems to be to attack the "more general" problems of Shortest Common Supersequence and Smallest Common Supertree (see [17] for definitions) in an analogous way. The parameterized complexity of these problems is unsettled.
- Directed Feedback Vertex Set restricted to tournament graphs is solvable in $2^k \cdot n^{O(1)}$ time using iterative compression [13]. An analogous result for Directed Feedback Edge Set restricted to tournaments is missing.
- Edge Clique Cover is fixed-parameter tractable but the only known way to achieve this is based on a simple, exponential-size kernelization [26]. Can

iterative compression lead to a more efficient fixed-parameter algorithm for Edge Clique Cover?
- Can iterative compression be used to show the fixed-parameter tractability of the Correlation Clustering problem [2], a generalization of the fixed-parameter tractable Cluster Editing problem [25]?
- Contig Scaffolding problems appear in the analysis of genomic data [39]. Again, iterative compression here might become a door opener for efficient fixed-parameter algorithms, which so far are not known for these problems.

Acknowledgement: We are grateful to Christian Komusiewicz, Matthias Müller-Hannemann, and Siavash Vahdati Daneshmand for constructive comments improving the presentation of this paper.

References

1. V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discrete Math.*, 3(2):289–297, 1999.
2. N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1–3):89–113, 2004.
3. A. Becker, R. Bar-Yehuda, and D. Geiger. Randomized algorithms for the Loop Cutset problem. *J. Artificial Intelligence Res.*, 12:219–234, 2000.
4. A. Becker and D. Geiger. Approximation algorithms for the Loop Cutset problem. In *Proc. 10th UAI*, pages 60–68. Morgan Kaufmann, 1994.
5. N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny scores. In *Proc. 4th AAIM*, volume 5034 of *LNCS*, pages 60–71. Springer, 2008.
6. H.-J. Böckenhauer, J. Hromkovič, T. Mömke, and P. Widmayer. On the hardness of reoptimization. In *Proc. 34th SOFSEM*, volume 4910 of *LNCS*, pages 50–65. Springer, 2008.
7. H. L. Bodlaender. On disjoint cycles. *Int. J. Found. Computer Science*, 5:59–68, 1994.
8. J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for the feedback vertex set problems. In *Proc. 10th WADS*, volume 4619 of *LNCS*, pages 422–433. Springer, 2007. To appear in *J. Comput. System Sci.*
9. J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. In *Proc. 10th WADS*, volume 4619 of *LNCS*, pages 495–506. Springer, 2007.
10. J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008. Article 21, 19 pages.
11. F. K. H. A. Dehne, M. R. Fellows, M. A. Langston, F. A. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Theory Comput. Syst.*, 41(3):479–492, 2007.
12. F. K. H. A. Dehne, M. R. Fellows, F. A. Rosamond, and P. Shaw. Greedy localization, iterative compression, and modeled crown reductions: New FPT techniques, an improved algorithm for set splitting, and a novel $2k$ kernelization for Vertex Cover. In *Proc. 1st IWPEC*, volume 3162 of *LNCS*, pages 271–280. Springer, 2004.

13. M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Proc. 6th CIAC*, volume 3998 of *LNCS*, pages 320–331. Springer, 2006.
14. R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congr. Numer.*, 87:161–187, 1992.
15. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
16. G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.
17. M. R. Fellows, M. T. Hallett, and U. Stege. Analogs & duals of the MAST problem for sequences & trees. *J. Algorithms*, 49(1):192–216, 2003.
18. P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization,* supplement volume A, pages 209–259. Kluwer Academic Publishers, 1999.
19. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.
20. F. V. Fomin, S. Gaspers, D. Kratsch, M. Liedloff, and S. Saurabh. Iterative compression and exact algorithms. In *Proc. 33rd MFCS*, volume 5162 of *LNCS*, pages 335–346. Springer, 2008.
21. F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a minimum feedback vertex set in time $O(1.7548^n)$. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 184–191. Springer, 2006.
22. D. R. Fulkerson and L. R. Ford, Jr. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.
23. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.
24. N. Garg, V. V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proc. 21st ICALP*, volume 820 of *LNCS*, pages 487–498. Springer, 1994.
25. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory Comput. Syst.*, 38(4):373–392, 2005.
26. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *ACM J. Exp. Algorithmics*, 13, 2008. Article 2.2, 15 pages.
27. S. Guillemot. Parameterized complexity and approximability of the SLCS problem. In *Proc. 3rd IWPEC*, volume 5018 of *LNCS*, pages 115–128. Springer, 2008.
28. 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. System Sci.*, 72(8):1386–1396, 2006.
29. F. Hüffner. Algorithm engineering for optimal graph bipartization. In *Proc. 4th WEA*, volume 3503 of *LNCS*, pages 240–252. Springer, 2005.
30. F. Hüffner. *Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2007.
31. F. Hüffner, N. Betzler, and R. Niedermeier. Optimal edge deletions for signed graph balancing. In *Proc. 6th WEA*, volume 4525 of *LNCS*, pages 297–310. Springer, 2007.
32. F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *Proc. 8th LATIN*, volume 4957 of *LNCS*, pages 711–722. Springer, 2008.
33. F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008.

34. A. B. Kahng, S. Vaya, and A. Z. Zelikovsky. New graph bipartizations for double-exposure, bright field alternating phase-shift mask layout. In *Proc. Asia and South Pacific Design Automation Conference*, pages 133–138. ACM Press, 2001.
35. M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *J. Algorithms*, 31(2):335–354, 1999.
36. D. Marx. Chordal deletion is fixed-parameter tractable. In *Proc. 32nd WG*, volume 4271 of *LNCS*, pages 37–48. Springer, 2006.
37. S. Mishra, V. Raman, S. Saurabh, S. Sikdar, and C. R. Subramanian. The complexity of finding subgraphs whose matching number equals the vertex cover number. In *Proc. 18th ISAAC*, volume 4835 of *LNCS*, pages 268–279. Springer, 2007.
38. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
39. M. Pop, D. S. Kosack, and S. L. Salzberg. Hierarchical scaffolding with Bambus. *Genome Research*, 14(1):149–159, 2004.
40. V. Raman, S. Saurabh, and S. Sikdar. Efficient exact algorithms through enumerating maximal independent sets and other techniques. *Theory Comput. Syst.*, 41(3):563–587, 2007.
41. I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed-parameter tractable. In *Proc. 35th ICALP*, volume 5125 of *LNCS*, pages 551–562. Springer, 2008.
42. B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.
43. A. S. Schulz, R. Weismantel, and G. M. Ziegler. 0/1-integer programming: Optimization and augmentation are equivalent. In *Proc. 3rd ESA*, volume 979 of *LNCS*, pages 473–483. Springer, 1995.
44. S. Thomassé. A quadratic kernel for feedback vertex set. In *Proc. 20th SODA*, pages 115–119. ACM/SIAM, 2009.
45. X.-S. Zhang, R.-S. Wang, L.-Y. Wu, and L. Chen. Models and algorithms for haplotyping problem. *Current Bioinformatics*, 1(1):104–114, 2006.