A Cubic Kernel for Feedback Vertex Set

Hans L. Bodlaender

Department of Information and Computing Sciences,
Utrecht University

Technical Report UU-CS-2006-042

www.cs.uu.nl
ISSN: 0924-3275

A Cubic Kernel for Feedback Vertex Set

Hans L. Bodlaender*

Abstract

The Feedback Vertex Set problem on unweighted, undirected graphs is considered. Improving upon a result by Burrage et al. [7], we show that this problem has a kernel with $O(k^3)$ vertices, i.e., there is a polynomial time algorithm, that given a graph G and an integer k, finds a graph G' and integer k', such that G has a feedback vertex set of size at most k, if and only if G' has a feedback vertex set of size at most k', and G' has $O(k^3)$ vertices. Moreover, if G' has a feedback vertex set of size at most k', then a minimum size feedback vertex set of G' directly gives a minimum size feedback vertex set of size k'. This kernelization algorithm can be used as the first step of an FPT algorithm for Feedback Vertex Set, but also as a preprocessing heuristic for Feedback Vertex Set.

1 Introduction

The Feedback Vertex Set problem is a classic and fundamental graph problem, with several applications. See e.g., [12] for an overview paper on this and related problems. In this paper, we consider the undirected and unweighted case of the problem. I.e., we are given an undirected graph $G = (V, E)$, and an integer km, and ask if there is a set of vertices S with $|S| \leq k$, such that each cycle of G contains at least one vertex from S. To facilitate the description of the algorithms, we allow G to have parallel edges and self loops.

As in [11, 18], we consider the *fixed parameter* case of this problem; i.e., k is seen as the *parameter*, and is considered to be small. For more information on fixed parameter tractability, see [11, 18]. A parameterized problem with input I and parameter k is said to be *fixed parameter tractable* (i.e., in FPT), if there is an algorithm that solves the problem in $p(|I|, k) \cdot f(k)$ time, where p is a polynomial and f an arbitrary function. Feedback Vertex Set is one of the problems, known to be fixed parameter tractable. The problem was first shown to be in FPT by Downey and Fellows [10]. In a series of papers, faster FPT algorithms were obtained [4, 11, 2, 20, 16, 21, 15, 9]. The currently best known bounds (concentrating on the function of k), are a probabilistic algorithm that finds with high

*Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands, hansb@cs.uu.nl.

probability the feedback vertex set of size at most k, if existing, and uses $O(4^k kn)$ time [2], and a deterministic algorithm that uses $O(10.567^k p(n))$ time (p a polynomial) [9] (see also [15].) An exact algorithm for FEEDBACK VERTEX SET with a running time of $O(1.8899^n)$ was recently found by Razgon [22]. This was improved to $O(1.7548^n)$ time by Fomin et al. [13].

Kernelization is a technique that yields a proof that a problem belongs to FPT (assuming it is known that the problem is decidable), and is useful in practice. A kernelization algorithm takes an input-parameter pair, and transforms it to an equivalent input-parameter pair (called the *kernel*), such that for the latter, the size of the input is a function of the (possibly new) parameter, and the new parameter is at most the old parameter. The kernelization algorithm is supposed to run in time that is both polynomial in the size of the input and the value of the parameter. If we have a kernel, then we can run any existing algorithm on the kernel, and obtain an algorithm that uses $O(p(n,k) + f(k))$ time, p a polynomial and f some function.

In a certain sense, kernelization is the very often used technique of preprocessing with in addition a mathematical guarantee on the quality of the preprocessing (the size of the input that remains after the preprocessing.) In this sense, kernelization is for preprocessing what approximation algorithms are for heuristics (i.e., an approximation algorithm can be seen as a heuristic with a guarantee for the quality.)

It was long open whether there existed a kernel or a kernel of size polynomial in k for the FEEDBACK VERTEX SET problem. This open problem was recently resolved by Burrage et al. [7], who showed that the FEEDBACK VERTEX SET problem has a kernel of polynomial size, namely one with $O(k^{11})$ vertices. In this paper, we improve on the size of this kernel, and show that FEEDBACK VERTEX SET has a kernel with $O(k^3)$ vertices. The kernelization algorithm uses the 2-approximation algorithm for FEEDBACK VERTEX SET of [1] or [3] as a first step, and then uses a set of relatively simple reduction rules. A combinatorial proof shows that if no rule can be applied, then the graph has $O(k^3)$ vertices, k the parameter in the reduced instance. It can be expected that these rules can also be of use for practical preprocessing for the undirected FEEDBACK VERTEX SET problem.

Some of the techniques in this paper were taken from, or inspired by techniques from [7].

2 Preliminaries

Throughout this paper, graphs are undirected, and can have parallel edges and self-loops. A graph is a pair (V, E), with V the set of vertices, and E the multiset of edges. Each edge is an unordered pair (of possibly equal) vertices. Here, $\{v, w\}$ represents the same pair as $\{w, v\}$. A self-loop is an edge of the form $\{v, v\}$. A pair of vertices $\{v, w\}$ is called a *double edge* in a graph $G = (V, E)$, if E contains at least two edges of the form $\{v, w\}$. $\{v, w\}$ is a *non-edge*, if there is no edge of the form $\{v, w\}$ in E, and is a *non-double edge*, if there is at most one edge of the form $\{v, w\}$ in E.

A *cycle* is a sequence of alternating vertices and edges v_0, e_1, v_1, e_1, v_2, ..., v_{r-1}, e_r,

v_r, such that $v_0 = v_r$, for each i, $1 \leq i \leq r$, $e_i = \{v_{i-1}, v_i\}$, and for all i, i', $1 \leq i < i' \leq r$, $v_i \neq v_{i'}$ and $e_i = e_{i'}$. I.e., we assume that cycles are simple. The length of a cycle is the number of edges on it. Note that if there are at least two parallel edges between vertices v and w, then we have a cycle of length two using v and w; otherwise there is no such cycle. Cycles are often also only denoted by sequences of successive vertices.

For a subset of the vertices $W \subseteq V$, the subgraph of graph $G = (V, E)$, *induced by* W is the graph $G[W] = (W, \{e \in E \mid e \subseteq W\})$. A graph G is a *forest*, if it does not contain a cycle.

Paths are defined similarly as cycles, but now we do not require that $v_0 = v_r$. v_0 and v_r are the *endpoints* of a path v_0, e_1, v_1, e_1, v_2, ..., v_{r-1}, e_r, v_r. $v_1, \ldots, v_{r-1}$ are the *internal vertices* of the path. A graph $G = (V, E)$ is *connected*, if there is a path between each pair of vertices in V. A tree is a forest that is connected. Two paths are said to be *vertex disjoint*, if all their internal vertices are different. (I.e., we allow that vertex disjoint paths share endpoints.)

A set of vertices $W \subseteq V$ is a *feedback vertex set* in $G = (V, E)$, if $G[V - W]$ is a forest, i.e., for each cycle in G, there is at least one vertex on the cycle that belongs to W.

The following trivial observation is regularly used in this paper.

Proposition 1 *Suppose there are at least two parallel edges $\{v, w\}$ in G. Let W be a feedback vertex set in G. Then $v \in W$ or $w \in W$.*

3 A Kernelization Algorithm

In this section, we give the main kernelization algorithm. We assume that we have as input a graph $G = (V, E)$, and an integer k. The algorithm either returns *no*, in which case we are sure that G has no feedback vertex set of size at most k, or a pair (G', k'), such that G has a feedback vertex set of size at most k, if and only if G' has a feedback vertex set of size k'. Alternatively, instead of returning *no*, the algorithm could return K_{k+3}, a clique with $k + 3$ vertices, as this graph has no feedback vertex set of size at most k.

The algorithm runs in time, polynomial in $|V| + |E|$ and in k. The number of vertices and edges in G' is bounded by $O(k^3)$. It is also possible to give a constructive version, i.e., one where we can turn a minimum size feedback vertex set of G' of size at most k' into a minimum size feedback vertex set of G in polynomial time. This will be discussed in Section 3.5.

3.1 Structure of the Algorithm

The algorithm has two phases: an initialization phase, and an improvement and reduction phase. During the algorithm, we maintain a graph G, initially the input graph, and an integer k. During both phases, we possibly could determine that the graph has no feedback vertex set of value k, and return *no*. In the improvement and reduction phase, we possibly may decrease the value of k. When this happens, we restart the initialization phase, but now with the smaller number k and the modified graph.

During the algorithm, two sets of vertices play a special role, We call these sets A and B. A will be invariantly a feedback vertex set of G.

3.2 Initialization Phase

We assume that we are given a graph $G = (V, E)$, and an integer k.

If $k = 0$, then we return *yes*, if G is a forest, and *no* otherwise. So, suppose $k \geq 1$.

The first step of the kernelization algorithm is to run the approximation algorithm of Bafna et al. [1] or the algorithm of Becker and Geiger [3]. These algorithms have a performance ratio of 2. Suppose this approximation algorithm returns a feedback vertex set A of G. If $|A| > 2k$, then from the performance ratio it follows that there is no feedback vertex set of size at most k, and we return *no*.

Otherwise, we continue with the next step, and also initialize the set B as $B = \{w \in V - A \mid \exists v \in A : \text{there are at least two edges } \{v, w\} \in E\}$. I.e., if there is a double edge between a vertex $v \in A$ and a vertex $w \notin A$, then w is added to B. Invariantly, B will be the set of vertices in $V - A$ that have a double edge to a vertex in A.

Initializing the set B can be done easily in $O(|V| + |E|)$ time using bucket sort.

3.3 Improvement and Reduction Rules

In this section, we give a number of improvement and reduction rules. Improvement rules add double edges to G; reduction rules remove edges and or vertices from G.

Each of the rules transforms the pair (G, k). We say that a rule is *safe*, if, whenever it transforms (G, k) to (G', k'), we have that G has a feedback vertex set of size k, if and only if G' has a feedback vertex set of size k'. In addition, we require that A is invariantly a feedback vertex set in the graph. We will show that each of the given rules is safe. For several rules, their safeness uses the following simple principle, earlier also used in [7].

Proposition 2 *Let $G = (V, E)$, and $v \in V$, such that there is at least one feedback vertex set W in G of minimum size with $v \in W$. Then a rule that removes v and its incident edges from G, and decreases k by one is safe.*

Proof. Safeness follows from the fact that for each $W' \subseteq V - \{v\}$, W' is a minimum feedback vertex set in $G[V - \{v\}]$, if and only if $W' \cup \{v\}$ is a minimum feedback vertex set in G. □

In our description below, we assume always that we restart the initialization phase, whenever k is decreased by one. This just simplifies some counting arguments (in particular, it ensures that $|A| \leq 2k$); it is also possible to give a variant without such restarts.

3.3.1 Simple rules

Each of the following rules makes a simple change to the graph. Many are taken from [7]. Safeness is easy to see, or follows directly from Proposition 2.

Rule 1 Islet Rule
If v is an isolated vertex, i.e., there is no edge incident to v, then remove v from G.

Rule 2 Twig Rule
If v has degree one, then remove v and its incident edge from G.

Rule 3 Triple Edge Rule
If there are three or more parallel edges of the form $\{v, w\}$, then remove all but two of these edges.

As a result of the **Triple Edge** rule, we have between each pair of vertices either 0, 1, or 2 edges when this rule cannot be applied.

Rule 4 Degree Two Rule
Suppose v has degree two. Let the edges, incident to v be $\{v, w\}$ and $\{v, x\}$. Remove v, its incident edges, and add an edge $\{w, x\}$ to G. If $\{w, x\}$ becomes a double edge, and $w \in A$, $x \notin A \cup B$, then add x to B. If $\{v, w\}$ becomes a double edge, and $x \in A$, $w \notin A \cup B$, then add w to B.

Note that the **Degree Two** rule can create a parallel edge or a self-loop.

Rule 5 Self-loop Rule
Suppose there is a self-loop $\{v, v\}$. Then remove v, all edges incident with v, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

3.3.2 Improvement

An important rule is the **Improvement** rule. It is inspired by the improvement rule, used in [5, 6, 8] in the context of algorithms to compute the treewidth of graphs.

Rule 6 Improvement Rule
Suppose $v \in A$, $w \in V$, $v \neq w$. Suppose there is no double edge between v and w, and that there are at least $k + 2$ vertex disjoint paths from v to w in G. Then add two edges $\{v, w\}$ to G. If $v \notin A \cup B$, then put w in B.

For a given pair of vertices, $v, w \in V$, one can compute in polynomial time the maximum number of vertex disjoint paths from v to w, using standard flow techniques. Nagamochi and Ibaraki [17] gave an algorithm that uses $O(k^2 n)$ time for checking if there are k vertex disjoint paths between a given pair of vertices v, w. See also [23, Chapter 9].

Lemma 3 *Suppose there are at least $k + 2$ vertex disjoint paths from v to w in G. Then in each feedback vertex set of size S at most k, we have that $v \in S$ or $w \in S$.*

Proof. Suppose S is a feedback vertex set of size at most k with $v, w \notin S$. There are at least two paths from v to w that do not contain a vertex in S. These form, with v and w a cycle that does not intersect S, so S is not a feedback vertex set, contradiction. □

Lemma 4 *The* **Improvement** *rule is safe.*

3.3.3 Many Cycles Through a Vertex

Another important rule is the following.

Rule 7 Flower Rule
Let $v \in A$. Suppose there is a collection of at least $k+1$ cycles in G, such that each pair of cycles has exactly $\{v\}$ as intersection. Then remove v and all its incident edges from G, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

I.e., we look for a set of cycles, that are vertex disjoint, except that they intersect in the vertex v. The following lemma formulates the **Flower** rule as a special case of *generalized matching*, hence it can be checked in polynomial time, see e.g., [14]. Let $w : V \to \mathbf{N}$ be a function, giving each vertex a weight. A set of edges $F \subseteq E$ is a perfect generalized matching with respect to w, if each vertex $v \in V$ is incident to exactly $w(v)$ edges in F.

Lemma 5 *Let $G' = (V', E')$ be the graph obtained from G by adding to each vertex $x \neq v$ two new vertices x_1 and x_2, with edges $\{x_1, x_2\}$, $\{x_1, x\}$, and $\{x_2, x\}$. Set $w(v) = 2k + 2$, and for each $x \in V$, $x \neq v$, $w(x_1) = w(x_2) = 1$, $w(x) = 2$. There is a collection of $k + 1$ cycles in G such that each pair of cycles intersects only in v, if and only if there is a perfect generalized matching with respect to w in G'.*

Proof. If we have the collection of cycles in G, then construct set F as follows. First put the edges in the cycles in F. Further, if $w \neq v$ is on a cycle in the collection, add the edge $\{w_1, w_2\}$ to F, otherwise, add the edges $\{w, w_1\}$ and $\{w, w_2\}$ to F'. The resulting set F is a generalized perfect matching with respect to w.

Suppose we have a generalized perfect matching F in G'. For each $x \in V - \{v\}$, the vertices x_1 and x_2 are either mapped to each other, or both are mapped to x. In the former case, x is incident in F to two other vertices in V; in the latter case, x is incident in F to no vertices in V. Moreover, v is incident in F to $2k + 2$ vertices in V. Thus, $F \cap E$ forms a collection of $k + 1$ cycles, that are disjoint, except that they intersect in v. □

It can be checked in polynomial time whether a generalized perfect matching exists in a given graph, see e.g., [14]. We now argue safeness of the rule.

Lemma 6 *Let $v \in A$. Suppose there is a collection of at least $k + 1$ cycles in G such that no two different cycles in the collection share another vertex except v. Then v belongs to each feedback vertex set S in G of size at most k.*

Proof. Consider a feedback vertex set S in G with $v \notin S$. Then, each cycle in the collection contains a vertex in S, and these are all different vertices, so $|S| \geq k + 1$. □

Lemma 7 *The* **Flower** *rule is safe.*

Proof. This follows directly from Lemma 6 and Proposition 2. □

A simple special case of the **Flower** rule is the following.

Rule 8 Large Double Degree Rule
Suppose $v \in V$, such that there are at least $k+1$ vertices w with $\{v, w\}$ a double edge. Then remove v and its incident edges, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

Safeness of the **Large Double Degree** rule follows directly from the safeness of the **Flower** rule.

3.3.4 Abdication

To describe the two abdication rules, we introduce some additional terminology. This terminology will also be used in the counting phase.

A *piece* is a connected component of $G[V - A - B]$. Let X be the set of vertices of a piece. The *border* of this piece is the set of vertices in $A \cup B$ that is adjacent to a vertex in X. A vertex v in the border of a piece *governs* the piece, if it has a double edge to each other vertex $w \neq v$ in the border of the piece.

Rule 9 First Abdication Rule
Suppose $v \in A \cup B$ governs a piece with vertex set X. If there is exactly one edge with one endpoint v and one endpoint in X, (i.e., one edge of the form $\{v, w\}$ with $w \in X$), then remove this edge $\{v, w\}$ with $w \in X$ from G.

As a result of the **First Abdication** rule, v will no longer belong to the border of the piece.

Lemma 8 *The* **First Abdication** *rule is safe.*

Proof. Let v, w, X be as in the **First Abdication** rule. Let G' be the graph, obtained by removing the edge $\{v, w\}$.

We claim that for each set $S \subseteq V$, S is a feedback vertex set in G, if and only if S is a feedback vertex set in G'. If S is a feedback vertex set in G, then, as G' is a subgraph of G, S is also a feedback vertex set in G'. Suppose S is a feedback vertex set in G'. Each cycle in G that is not a cycle in G' uses the edge $\{v, w\}$. Hence, if $v \in S$, S is also a feedback vertex set in G. Suppose $v \notin S$. As v governs the piece X, all vertices in the border of the piece except v must belong to S. Each cycle that uses the edge $\{v, w\}$ uses besides v one other border vertex of the piece (as v has only one edge to the piece, and the vertex set of a piece induces a tree), and thus contains a vertex in S. So, again S is a feedback vertex set.

As removing the edge $\{v, w\}$ does not change the collection of feedback vertex sets, the rule is safe. □

Rule 10 Second Abdication Rule
Suppose $v \in A \cup B$ governs a piece with vertex set X. If there are at least two edges with one endpoint v and one endpoint in X, then remove v and all its incident vertices from G, and decrease k by one. Restart the initialization phase with the new graph and new value of k.

Lemma 9 *Suppose $v \in A \cup B$ governs a piece with vertex set X. Suppose there are at least two edges with one endpoint v and one endpoint in X. Then there is a minimum size feedback vertex set in G that contains v.*

Proof. Consider a minimum size feedback vertex set S with $v \notin S$. As v governs the piece, all other border vertices of the piece must belong to S. Let w_1, w_2 be two neighbors of v in the piece. v with the path in X from w_1 to w_2 forms a cycle, so there is at least one vertex in X that belongs to S, say x. Consider the set $S - \{x\} \cup \{v\}$. This is again a feedback vertex set: any cycle that contains x must contain a vertex of the border of X and hence a vertex in $S - \{x\} \cup \{v\}$. As $|S| = |S - \{x\} \cup \{v\}$, there is a feedback vertex set of minimum size that contains v. □

From Proposition 2 and Lemma 9, we directly conclude the following.

Lemma 10 *The* **Second Abdication** *rule is safe.*

Consider a piece with vertex set X with border set Y, such that for each pair of disjoint vertices in Y, there is a double edge. Consider what happens when we apply the abdication rules to this piece. Each vertex in Y governs the piece. If $v \in Y$ has one edge to the piece, this edge will be removed, and v no longer is in the border of X. If $v \in Y$ has two or more edges to the piece, then v itself is removed. Thus, after all the vertices in Y have been handled, the border of X will be empty. A piece with an empty border is a connected component of G that is a tree: it is a subgraph of $G[V - A]$ and hence does not have a cycle. Now, repeated application of the **Twig** rule, and then an application of the **Islet** rule will remove all vertices in the piece.

Above, we have seen that the given rules remove each piece for which there is a double edge between each pair of disjoint vertices in its border. A direct consequence of this is the following lemma.

Lemma 11 *Suppose none of the Rules 1 – 10 can be applied to G. Suppose $Y \subseteq V$ is the border of a piece in G. Then there are two disjoint vertices $v, w \in Y$ such that $\{v, w\}$ is not a double edge.*

It is straightforward to see that we can check in polynomial time whether an abdication rule is possible for given G, A, B.

3.4 Counting Arguments and the Counting Phase

As we will show below, Rules 1 – 10 will transform G to a kernel with $O(k^3)$ vertices, in case G has a feedback vertex set of size at most k. Thus, the algorithm ends with counting the number of vertices, and returning *no* in case the size of G is too large.

A graph $G = (V, E)$, with sets A, B, and integer k is called *a reduced instance*, if none of the rules 1 – 10 is applicable anymore.

We start with a number of lemmas.

Lemma 12 *In a reduced instance, there are at most $2k$ vertices in A and at most $2k^2$ vertices in B.*

Proof. We start with a set A of size at most $2k$. During the algorithm, we recompute A whenever k is changed.

Each vertex in B has at least one neighbor in A to which it has a double edge, but no vertex in A has more than k neighbors to which it has a double edge, otherwise the **Large Double Degree** rule can be applied. So the result follows. □

We construct an auxiliary graph, which we call the B-piece graph. In the B-piece graph, there are two types of vertices: each vertex in B is a vertex in the B-piece graph, and for each piece, there is a vertex representing the piece in the B-piece graph. The B-piece graph is bipartite, with an edge between a vertex $v \in B$ and a vertex x representing a piece, if B is in the border of the piece.

Lemma 13 *The B-piece graph is a forest.*

Proof. Suppose we have a cycle c in the B-piece graph. We transform this to a cycle in G, as follows. Consider a vertex v that represents a piece with vertex set X, and suppose it is incident to w_1 and w_2 in the cycle c. Thus, there is a path, using only vertices in X from w_1 to w_2. Replace w in c by this path. Doing this for each vertex that represents a piece in c gives a cycle c' in G. Note that there is no vertex in A on c'. Thus, A is not a feedback vertex set in G, contradiction. □

Lemma 14 *Let $v \in B$ be in the border of a piece with vertex set X. Then there is at exactly one edge from v to a vertex in X.*

Proof. By definition of border, there is at least one such edge. Suppose there are at least two edges from v to X, so to vertices w_1 and w_2. Consider the following cycle: start at v, go to w_1, then take the path using vertices in X to w_2, and then close the cycle at v. This is a cycle that does not use a vertex in A. Hence, A is not a feedback vertex set, contradiction. □

Lemma 15 *Suppose we have a reduced instance. There are at most $8k^3 + 9k^2 + k$ pieces.*

Proof. We associate each piece to a non-double edge in its border. I.e., for each piece in the reduced instance, there are two border vertices $v, w \in A \cup B$, such that $\{v, w\}$ is not a double edge; we select one such pair and associate the piece with this pair. We now count the number of pieces that can thus be associated with the different types of pairs.

Case I: Non-double edges $\{v, w\}$ with $v \in A$, $w \in A$. If $k+2$ or more pieces are associated with a pair $\{v, w\}$, $v, w \in A$, then by the **Improvement** rule, $\{v, w\}$ would be a double edge. So, at most $k+1$ pairs are associated with the pair, and hence we have at most $\frac{1}{2}|A|(|A|+1) \cdot (k+1) \leq k \cdot (2k+1) \cdot (k+1)$ pieces associated with pairs of this type.

Case II: Non-double edges $\{v, w\}$, with $v \in A$, $w \in B$. This case needs the most detailed counting argument. We consider a number of subcases.

Case IIa: Non-double edges $\{v, w\}$, with $v \in A$, $w \in B$, to which at most one piece is associated. Clearly, the number of pieces associated to these edges is at most $|A| \cdot |B| \leq 2k \cdot 2k^2 = 4k^3$.

Case IIb: Non-double edges $\{v, w\}$, with $v \in A$, $w \in B$, to which at least two pieces are associated. Consider a vertex $v \in A$. Let $X_v = \{w \in B \mid$ at least two pieces are associated to the non-double edge $\{v, w\}\}$. For each $w \in X_v$, consider the cycle, that starts in v, moves to w through one of the pieces associated to $\{v, w\}$, and then moves back to w through another piece associated to $\{v, w\}$. This is a cycle, and for $w, w' \in W$, $w \neq w'$, the cycle of w only intersects the cycle of w' in v. So, if $|X_v| \geq k+1$, we have a collection of at least $k+1$ cycles that move through v, and pairwise have $\{v\}$ as intersection. Thus, the **Flower** rule would apply to v. As we have a reduced instance, we can assume that $|X_v| \leq k$.

So, for a fixed $v \in A$, at most k pairs $\{v, w\}$, $w \in B$, have at least two pieces associated to it. Thus, in total at most $|A| \cdot k \leq 2k^2$ such pairs have at least two pieces associated to it. The same argument as in Case I shows that each of these pairs has at most $k+1$ pieces associated to it, and this gives a maximum of $2k^2 \cdot (k+1)$ pieces for this case.

Case III: Non-double edges $\{v, w\}$, with $v \in B$, $w \in B$. If we associate two pieces to one such pair $\{v, w\}$, then these pieces and v, w form a cycle in the B-piece graph, which contradicts Lemma 13. So, to each such pair, we associate at most one piece. Moreover, if a piece is associated to such a pair, it has at least two neighbors in the B-piece graph. As the B-piece graph is a forest, and there are at most $2|A| \cdot k \leq 4k^2$ vertices in B, the number of pieces adjacent to two vertices in B can be at most $4k^2$, and hence also at most $4k^2$ pieces can be associated to pairs of this type.

The total number of pieces in the reduced instance is the sum of the number of pieces over the possible cases: $k \cdot (2k+1) \cdot (k+1) + 4k^3 + 2k^2 \cdot (k+1) + 4k^2 = 8k^3 + 9k^2 + k$. □

We now count the number of vertices and edges in the pieces. We partition the vertices that belong to a piece into two sets. C is the set of vertices in a piece (i.e., in $V - (A \cup B)$), that are adjacent to a vertex in $A \cup B$. D is the set of vertices in a piece that are adjacent to vertices in $C \cup D$ only, i.e., $D = V - (A \cup B \cup C)$. We first estimate the size of C; a simple argument then shows a bound on $|D|$.

For each $v \in A \cup B$, let $C_v = \{w \in C \mid \{v, w\} \in E\}$. I.e., C_v are the vertices in pieces, adjacent to v. The sets C_v are partitioned into two sets, $C_{v,1}$, and $C_{v,\geq 2}$. $C_{v,1}$ is the set of the vertices $w \in C_v$, such that w is the only vertex in its piece that is adjacent to v. $C_{v,\geq 2} = C_v - C_{v,1}$ is the set of vertices $w \in C_v$, such that the piece of w has at least one other vertex, also adjacent to v.

We give a number of different lemmas that give bounds on the size of these sets. In each case, we assume we have a reduced instance.

Lemma 16 *Let $v \in A$. $|C_{v,1}| \leq 3k^2 + 3k - 1$.*

Proof. Consider a vertex $w \in C_{v,1}$, and the piece containing w. As v does not govern this piece (otherwise the **First Abdication** rule would be applicable), there is a vertex $x \in (A \cup B) - \{v\}$, in the border of the piece for which the pair $\{v, x\}$ is not a double edge. For each $w \in C_{v,1}$, we associate w with such a vertex x in the border of the piece of w with $x \neq v$ and $\{v, x\}$ is not a double edge. (If there is a choice for different vertices x in the border of a piece, one of these is chosen arbitrarily.)

No vertex $x \in A \cup B$ has more than $k+1$ vertices associated to it. Suppose x has $k+2$ or more vertices in $C_{v,1}$ associated to it. For each vertex w in $C_{v,1}$, associated to x, we take the path, starting at v, and moving through the piece containing w to x. As each of these paths uses a different piece, we have $k+2$ vertex disjoint paths from v to x, and hence the **Improvement** rule will add the double edge $\{v, x\}$. However, if $\{v, x\}$ is a double edge, no vertices in $C_{v,1}$ will be associated to x, contradiction.

There are at most k vertices in $A \cup B$ that have two or more vertices associated to it. Suppose we have at least $k+1$ vertices in $A \cup B$ that have two or more vertices associated with it. For each vertex x in $A \cup B$ that has two or more vertices associated with it, we build a cycle as follows. Suppose X_1, X_2 are pieces containing vertices associated to x. Start at v, more through X_1 to x, move from x through X_2 to v. As all pieces used by these cycles are different, these cycles only intersect at v, and thus the **Flower** rule applies. This contradicts the assumption that we have a reduced instance.

Now we can count the number of vertices in $C_{v,1}$, or, equivalently, the total over all $x \in A \cup B - \{v\}$ of the number of vertices associated to x. There are at most $|A| + |B| - 1$ vertices to which one vertex is associated. There are at most k vertices in $A \cup B - \{v\}$ to which two or more vertices are associated, and to each, at most $k+1$ vertices are associated. This gives a total of at most $2k + 2k^2 - 1 + k(k+1) = 3k^2 + 3k - 1$. □

Lemma 17 *Let $v \in A$. $|C_{v,\geq 2}| \leq k^2 + 3k + 3$.*

Proof. Suppose $v \in A$, and $|C_{v,\geq 2}| > k^2 + 3k + 3$. We will show that the instance is not reduced, thus arriving at a contradiction. In particular, we will show that the **Flower** rule can be applied. To do so, a collection of cycles $\mathcal{C}$ is built. Initially, $\mathcal{C} = \emptyset$.

A *region* is a maximal connected set of vertices in pieces (i.e., a subset of $V - (A \cup B)$, that does not contain any vertex on a cycle in $\mathcal{C}$. A region is *filled*, if it contains at least two vertices in $C_{v,\geq 2}$.

If we have a filled region Y, we build a cycle through v that is vertex disjoint with any cycle already in $\mathcal{C}$ in the following way. Start at v, then go to an arbitrary neighbor of v in the region. Suppose our path so far p is at vertex w. If w is adjacent to v, and is not the second vertex on the path. Then, we close the cycle by going back from w to v. Suppose now w is not adjacent to v, or is the second vertex on the path. For each neighbor x of w, except a neighbor of w that does belong to the path constructed so far, we define a number n_x. n_x is the number of vertices z in C_v for which there is a path from x to z that use only vertices in the region Y and that uses no vertex on p. (In particular, this means the path belongs to the subtree, obtained by removing the vertices on p from Y.) If there is a neighbor x of w with $n_x = 1$, x not on p, then select such a neighbor x as next vertex on the path p; set $w = x$. Otherwise, select any neighbor x of w with $n_x \geq 2$, x not on p. It is not hard to see, that selecting vertices in this way gives a cycle through the region. As the region does not contain vertices on existing cycles in $\mathcal{C}$, we invariantly have a set of cycles that only intersect at v.

After a cycle has been added, the set of regions and filled regions is changed. We consider the number of vertices in $C_{v,\geq 2}$, that belonged to a filled region, before the addition of a cycle to $\mathcal{C}$, but do no longer belong to a filled region after this addition. We claim that this number is at most $k + 2$. Clearly, the first vertex after v and the last vertex before closing the cycle at v are of this type. Suppose we go from w to x while building the cycle. Suppose y is a neighbor of w that is not used by the path that we are constructing. The new region formed by that path that contains y contains n_y vertices in C_v. Thus, if $n_y \geq 2$, this new region is filled. If $n_y = 0$, then no vertices in C_v are in this new region. So, the interesting case is when $n_y = 1$. By the algorithm we followed, we go to another neighbor x of w with $n_x = 1$. We claim that w has at most $k + 1$ neighbors y with $n_y = 1$. Suppose w has at least $k+2$ neighbors y with $n_y = 1$. Then we can build a collection of $k+2$ vertex disjoint paths from w to x, each going from w to a neighbor y of w with $n_y = 1$, and then following the path through the region to v. Thus, by the **Improvement** rule, w would have been added to B, which contradicts the assumption that w belongs to a piece. Thus, at this point, there are at most k vertices that play the role of y. Each amounts to one vertex in C_v that belonged to a filled region before the addition of the cycle, and belongs to a region with only one vertex in C_v after the addition. During the construction of the cycle, there is only one point where this can happen: after we selected an x with $n_x = 1$, all unvisited neighbors of the vertex on the path will have $n_z = 0$. So, by the addition of a cycle, two vertices in $C_{v,\geq 2}$ belong to a cycle, and at most k vertices belong to a region with only one vertex in $C_{v,\geq 2}$.

So, we can prove with induction that if there are at least $2 + \ell \cdot (k + 2)$ vertices in filled regions, we can build ℓ cycles in regions, vertex disjoint from cycles already in $\mathcal{C}$. (The base case $\ell = 0$ is trivial.)

When we start the construction, each piece containing vertices in $C_{v,\geq 2}$ is a filled region, and hence the procedure constructs ℓ cycles, when $|C_{v,\geq 2}| \geq 2 + \ell \cdot (k + 2)$. As we assumed that $|C_{v,\geq 2}| \geq k^2 + 3k + 4 = 2 + (k + 1)(k + 2)$, we can construct a collection of $k + 1$ cycles through v that intersect only at v. Thus, the **Flower** rule is applicable, which contradicts the assumption that we have a reduced instance. □

Lemma 18 $\sum_{v \in B} |C_v| \leq 8k^3 + 11k^2 + k - 1$.

Proof. By Lemma 13, if $v \in B$, then $C_{v,\geq 2} = \emptyset$. So, for all $v \in B$, $C_v = C_{v,1}$.

Suppose $w \in C_{v,1}$, $v \in B$. There is an edge from the piece, containing w to v in the B-piece graph. As w is the only vertex in this piece with an edge to v, we can associate this edge to w. In this way, each vertex in $\bigcup_{v \in B} C_v$ has at least one edge from the B-piece graph associated to it, and hence $|\bigcup_{v \in B} C_v|$ is at most the number of edges of the B-piece graph. As the B-piece graph is a forest (Lemma 13), the number of edges in this graph is smaller than the number of vertices, which equals $|B|$ plus the number of pieces. By Lemmas 12 and 15, the result follows. □

Lemma 19 $|D| \leq |C| - 1$.

Proof. Consider the graph $G[C \cup D]$, the subgraph of G, induced by $C \cup D$. As A is a feedback vertex set, this subgraph is a forest. Each vertex in D has the same degree in $G[C \cup D]$ as it has in G, by definition of D. So, in a reduced instance, each vertex in D has degree at least three in G and in $G[C \cup D]$. The number of vertices of degree at least three in a forest is less than the number of leaves in the forest, and each leaf in $G[C \cup D]$ must belong to C, hence $|D| < |C|$. □

Theorem 20 *In a reduced instance, there are $O(k^3)$ vertices and $O(k^3)$ edges.*

Proof. We use the results, shown above. We have $|A| = O(k)$, $|B| = O(k^2)$. For each $v \in A$, $|C_v| \leq |C_{v,1}| + |C_{v,\geq 2}| = O(k^2)$. Thus $|C| = O(k^3)$, and hence $|D| = O(k^3)$.

As $G[V - A]$ is a forest, there are $O(|V - A|) = O(k^3)$ edges with no endpoint in A. Thus, we only need to count the edges with at least one endpoint in A. There are $O(|A|^2) = O(k^2)$ edges with both endpoints in A and $O(|A| \cdot |B|) = O(k^3)$ edges with one endpoint in A and one endpoint in B. By definition, there are no edges between vertices in A and vertices in D. If there is an edge $\{v, w\}$, $v \in A$, $w \in C$, then $w \in C_v$, and there is no parallel edge to this one (otherwise $w \in B$, so the number of edges in $A \times C$ equals $\sum_{v \in A} |C_v| = O(k^3)$. □

Theorem 21 *A reduced instance can be obtained in polynomial time.*

Proof. From the preceding discussion, we can conclude that given G, A, B, we can verify in polynomial time if a rule can be applied. The number of times we apply a rule or a restart is also polynomially bounded. The number of restarts is bounded by the initial value of k, as for each restart, k is decreased by one. We need to check for the **Improvement** rule only at the beginning of the algorithm, or right after a restart: the other rules cannot enable an improvement. As there are $O(k|V|)$ pairs where we check for a possible improvement, the total time for checking and performing all possibilities for the **Improvement** rule is polynomial. All other rules either cause a restart, or delete at least one edge from G, so take overall polynomial time. □

Obtaining a fast implementation, e.g., with a proper choice of data structures, is an interesting issue. In this paper, we concentrated on the combinatorial aspects, and refrained from discussion such implementation issues and the precise asymptotic running time of the kernelization algorithm.

3.5 A Constructive Version

It is not hard to transform the algorithm to a constructive one. Suppose the kernelization algorithm transforms the pair (G, k) to a pair (G', k'). The constructive algorithm would ensure that when we have a minimum size feedback vertex set of G' of size at most k', then we can obtain (in polynomial time) a minimum size feedback vertex set of G. We need to modify the reduction rules as follows.

In the **Degree Two** rule, if we contract a vertex v of degree two to a neighbor w, we give the new vertex the name of w. We keep a set S, which initially is empty. Each vertex, removed by the **Self-loop** rule, **Flower** rule, **Second Abdication** rule, or **Large Double Degree** rule is placed in S.

Now, for any minimum size feedback vertex set W in G' with $|W| \leq k'$, $S \cup W$ is a minimum size feedback vertex set in G.

4 Variants

In this section, we describe a few alternatives to the kernelization algorithm described in Section 3.

4.1 Without Restarts

To speed up, we can refrain from doing restarts after k has been decreased. The size of the kernel may become somewhat larger, and is a function of both the initial as the final value of k. One could also choose to only do a restart when $|A| > ck$, for some $c \geq 2$.

4.2 Without an Initial Value of k

It is also possible to run the algorithm while we do not have an initial value of k as input. In such a way, the algorithm could be used as a preprocessing heuristic. We first start with setting k to the size of the feedback vertex set A, returned by the approximation algorithm of Bafna et al. [1] or the algorithm of Becker and Geiger [3].

4.3 With a Different Algorithm to Find the Initial Feedback Vertex Set

Of course, we could use another approximation algorithm or heuristic for the feedback vertex set problem in the initialization phase. If we use, instead of, or in addition to, the algorithm of Bafna et al. [1] or Becker and Geiger [3] some other algorithms that finds close to optimal feedback vertex sets, and this algorithm finds a feedback vertex set that is smaller, then the guaranteed bound on the size of the kernel is also smaller. For instance, we could try to improve upon a solution found by the algorithms of [1, 3] with a local search algorithm.

4.4 Strongly Forced Vertices

The approximation algorithms of Bafna et al. [1] and of Becker and Geiger [3] also can solve weighted versions of the feedback vertex set problem. This can also be of help for preprocessing and kernelization of the unweighted problem.

Lemma 22 *Let $v \in V$. Suppose an approximation algorithm for* Weighted Feedback Vertex Set *with performance ratio 2 returns a solution of weight at least $2k+1$ on the weighted instance of G, obtained by setting the weight of v to $2k+1$, and the weight of all other vertices to 1. Then v belongs to any feedback vertex set in G with at most k vertices.*

The correctness of this lemma is trivial. It directly implies the correctness of the following rule.

Rule 11 Strongly Forced Vertex Rule
Let $v \in V$. Build the following weighted instance of the problem. Set the weight of v to $2k+1$, and the weight of all other vertices to 1. If the algorithm of [1] or [3] returns a feedback vertex set of weight at least $2k+1$, then remove v and its incident vertices from G, decrease k by one, and restart with the initialization phase.

The **Strongly Forced Vertex** rule could be used instead of (or, in addition to) the **Flower** rule. We would run this check for each $v \in A$. It still gives an $O(k^3)$ bound on the kernel size, but with a slightly higher constant. Namely, if there are at least $2k+1$ cycles that are vertex disjoint except that they may share v, then any feedback vertex set that does not use v has size at least $2k+1$, so the rule will remove v. Note that the **Many Cycles** rule has here a value of $k+1$ instead of $2k+1$.

4.5 Strongly Forced Pairs

In a similar way, we have a new rule that is a counter part to the **Improvement** rule.

Lemma 23 *Let $v, w \in V$. Suppose an an approximation algorithm for* Weighted Feedback Vertex Set *with performance ratio 2 returns a solution of weight at least $2k+1$ on the weighted instance of G, obtained by setting the weights of v and of w to $2k+1$, and the weight of all other vertices to 1. If W is a feedback vertex set in G of size at most k, then $v \in W$ or $w \in W$.*

This lemma, whose correctness is again trivial, implies the correctness of the following rule.

Rule 12 Strongly Forced Pair Rule
Let $v, w \in V$, $v \neq w$. Build the following weighted instance of the problem. Set the weight of v to $2k+1$, and the weight of all other vertices to 1. If the algorithm of [1] or [3] returns a feedback vertex set of weight at least $2k+1$, then add two edges $\{v, w\}$ to G, and if $v \in A$ and $w \notin A \cup B$, then add w to B; if $w \in A$ and $v \notin A \cup B$, then add v to B.

We can use this as an additional rule, as a heuristic for stronger preprocessing, or we can replace the **Improvement** rule by this rule. In the latter case, a similar proof as in Section 3.4 can be used to show that we obtain a kernel of size $O(k^3)$, although the constant factor is somewhat higher.

4.6 Improved Flowers

The **Flower** rule can be heuristically improved as follows. If we have a collection of $\ell < k+1$ cycles through a vertex v, then we can heuristically try to add $k+1-\ell$ cycles that are disjoint from the cycles so far in the collection and each other. If we succeed finding such a collection of cycles, then we still know that v belongs to each feedback vertex set of size at most k, and thus we remove v and its incident edges and decrease k by one.

5 Discussion

In this paper, we showed that the Feedback Vertex Set problem on undirected graphs has a kernel of size $O(k^3)$. In this paper, we concentrated on the combinatorial part of this result. We showed that the kernelization algorithm uses polynomial time. A precise analysis of the running time was not given. Such analysis would bring up several issues, e.g.,: what data structures do we use (e.g., for the **Triple Edge** rule, a representation of G with only an adjacency matrix seems inadequate), in what order do we perform the checks for rules, and do we need all checks, or can we speed up by omitting some checks, unnecessary for the guarantee on the kernel size?

It would also be interesting to perform an experimental evaluation of this, and similar algorithms for Feedback Vertex Set kernelization. What techniques will give fast algorithms and/or small kernel sizes. How well do these algorithms perform on data obtained from real-life applications?

It is to be expected that with small (or larger) changes, improvements to the constant in the $O(k^3)$ bound on the kernel size are possible. It is also to be expected that in a similar way, we can obtain a kernel for the case where vertices have integer weights with the parameter k an upper bound on the sum of the weights of the vertices in the vertex set.

We end the paper with mentioning some open problems.

- Is there a kernel for Feedback Vertex Set of size $o(k^3)$? See e.g., the discussion in [7].

- Is it possible to carry over the techniques to the related Loop Cut Set problem? In this problem, we are given a directed (acyclic) graph $G = (V, A)$. A loop is a cycle in the underlying undirected graph, (i.e., in a loop, we are allowed to follow edges against their direction.) A *loop cut set* is a set of vertices $S \subseteq V$, such that the graph, obtained by removing all arcs whose tail is in S has no loops. (In other words, a vertex v on a loop cuts the loop in not both edges on the loop point towards v.) The

Loop Cut Set problem is to find a minimum size loop cut set in a given directed acyclic graph. This problem is clearly strongly related to Feedback Vertex Set, as also can be seen from the fact that many techniques carry over from one problem to the other (see e.g., [3, 2].) Finding a small loop cut set is necessary when one wants to apply the algorithm of Pearl [19] for computing inference in probabilistic networks.

- The Directed Feedback Vertex Set problem and Feedback Arc Set problems, where we look for a minimum size set of vertices or arcs in a directed graph G, such that each cycle in G contains a vertex or arc in the set, are still not known to be in FPT. Membership in FPT of these problems is a long outstanding open problem. As having a kernel for these problems would imply membership in FPT, it is probably very hard to find a kernelization algorithm for these problems.

References

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Disc. Math.*, 12:289–297, 1999.

[2] A. Becker, R. Bar-Yehuda, and D. Geiger. Randomized algorithms for the loop cutset problem. *J. Artificial Intelligence Research*, 12:219–234, 2000.

[3] A. Becker and D. Geiger. Optimization of Pearl's method of conditioning and greedy-liker approximation algorithms for the vertex feedback set problem. *Acta Informatica*, 83:167–188, 1996.

[4] H. L. Bodlaender. On disjoint cycles. *Int. J. Found. Computer Science*, 5(1):59–68, 1994.

[5] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[6] H. L. Bodlaender. Necessary edges in k-chordalizations of graphs. *Journal of Combinatorial Optimization*, 7:283–290, 2003.

[7] K. Burrage, V. Estivill-Castro, M. R. Fellows, M. A. Langston, S. Mac, and F. A. Rosamond. The undirected feedback vertex set problem has a poly(k) kernel. In H. L. Bodlaender and M. A. Langston, editors, *Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*, pages 192–202. Springer Verlag, Lecture Notes in Computer Science, vol. 4169, 2006.

[8] F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre. New lower and upper bounds for graph treewidth. In J. D. P. Rolim, editor, *Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003*, pages 70–80. Springer Verlag, Lecture Notes in Computer Science, vol. 2647, 2003.

[9] 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. In L. Wang, editor, *Proceedings 11th International Computing and Combinatorics Conference COCOON 2005*, pages 859–869. Springer Verlag, Lecture Notes in Computer Science, vol. 3595, 2005.

[10] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–178, 1992.

[11] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1998.

[12] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In *Handbook of Combinatorial Optimization, Vol. A.*, pages 209–258, Amsterdam, the Netherlands, 1999. Kluwer.

[13] F. V. Fomin, S. Gaspers, and C. Knauer. Finding a minimum feedback vertex set in time $O(1.7548^n)$. In H. L. Bodlaender and M. A. Langston, editors, *Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*, pages 183–191. Springer Verlag, Lecture Notes in Computer Science, vol. 4169, 2006.

[14] A. M. H. Gerards. Matching. In M. O. Ball et al., editor, *Handbooks in Operations Research and Management Sciences, Volume 7, Network Models*, chapter 3, pages 135–224. Elsevier Science, Amsterdam, 1995.

[15] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Improved fixed-parameter algorithms for two feeback set problems. In *Proc. 9th Int. Workshop on Algorithms and Data Structures WADS 2004*, pages 158–168. Springer Verlag, Lecture Notes in Computer Science, vol. 3608, 2004.

[16] I. A. Kanj, M. J. Pelmajer, and M. Schaefer. Parameterized algorithms for feedback vertex set. In *Proc. 1st Int. Workshop on Parameterized and Exact Computation, IWPEC 2004*, pages 235–248. Springer Verlag, Lecture Notes in Computer Science, vol. 3162, 2004.

[17] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7:583–596, 1992.

[18] R. Niedermeier. Invitation to fixed-parameter algorithms. Universität Tübingen, 2002. Habilitation Thesis.

[19] J. Pearl. *Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Palo Alto, 1988.

[20] V. Raman, S. Saurabh, and C. R. Subramanian. Faster fixed parameter tractable algorithms for undirected feedback vertex set. In *Proceedings 13th International Symposium on Algorithms and Computation, ISAAC 2002*, pages 241 – 248. Springer Verlag, Lecture Notes in Computer Science, vol. 2518, 2002.

[21] V. Raman, S. Saurabh, and C. R. Subramanian. Faster algorithms for feedback vertex set. *Proceedings 2nd Brazilian Symposium on Graphs, Algorithms, and Combinatorics, GRACO 2005, Electronic Notes in Discrete Mathematics*, 19:273–279, 2005.

[22] I. Razgon. Exact computation of maximum induced forest. In L. Arge and R. Freivalds, editors, *10th ScandinavianWorkshop on Algorithm Theory, SWAT 2006*, pages 160–171. Springer Verlag, Lecture Notes in Computer Science, vol. 4059, 2006.

[23] A. Schrijver. *Combinatorial Optimization. Polyhedra and Efficiency.* Springer, Berlin, 2003.