FPT is P-Time Extremal Structure I

VLADIMIR ESTIVILL-CASTRO[1] , MICHAEL R. FELLOWS[2] , MICHAEL A. LANGSTON[3] , FRANCES A. ROSAMOND

ABSTRACT. We describe a broad program of research in parameterized complexity, and hows this plays out for the MAX LEAF SPANNING TREE problem.

1 Introduction

The program we describe here, using MAX LEAF as a case study, is concerned with the development of a mathematical monster machine (in the sense of Grothendieck) for dealing with hard computational problems. In a nutshell, parameterization is all about structure. In our case study example of MAX LEAF one can ask:

What is the structure of graphs that exclude a tree subgraph with k leaves?

Grothendieck had a point of view about methodology that we adopt, that a mathematical project formulated in the correct perspective should unfold as a series of small steps in an overall trajectory that is described by the appropriate "mathematical machine" [Jac04]. That is what we are exploring here for the design of FPT algorithms: the *right* way to do it, and the recipes that constitute the appropriate mathematical machine for this task.

The design of FPT algorithms, in the right way, is only the first of five deliverables associated to our extremal structure approach to algorithm design for hard problems:

(A) FPT algorithms.
(B) Powerful and efficient pre-processing and data-reduction rules.
(C) Gradients and transformation rules for local search heuristics.
(D) Polynomial-time approximation algorithms.

[1]Research supported by the Australian Research Council.
[2]Research supported by the Australian Research Council, by the Australian Centre for Bioinformatics and by the New Zealand Marsden Fund for Basic Research.
[3]Research supported by the U.S. National Science Foundation under grant CCR–0075792, by the U.S. Office of Naval Research under grant N00014–01–1–0608, and by the U.S. Department of Energy under contract DE–AC05–00OR22725.

(E) Structure to exploit in solving other problems.

We will illustrate these objectives for the MAX LEAF problem as a first case study. Our concern is with *the right way* to pursue these goals in a unified way.

1.1 Relevant Structure

There are plenty of reasons to be interested in parameterized complexity and FPT algorithmics. Contemporary sources of introductory material can be found in [Ra97, DF99, DFS99, Fe01, Fe02, Nie02, Dow03, Fe03, Nie04, Fer05, Nie05].

Some background in a nutshell. Parameterized complexity is basically a two-dimensional generalization of the familiar P versus NP framework. In addition to the overall input size n we consider the effects on complexity of a declared secondary "measurement" k (the *parameter*) that generally is used to capture some structure of the input or other aspect of our computational objective (for example, $k = 1/\epsilon$ turns out to be a useful parameterization in the analysis of approximation complexity). *The Good* is defined to be solvability in time $f(k)n^c$, termed *fixed-parameter tractability*, where f is some function (usually exponential) and c is a constant independent of k. After that, you just follow your nose. Define a notion of reducibility in a way that respects good behavior. *The Bad* (probable unavailability of FPT) is measured by a strong two-dimensional analog of NP-hardness, termed $W[1]$-hardness. A reference problem complete for $W[1]$ is the k-step halting problem for nondeterministic Turing machines of unlimited nondeterminism. This is obviously solvable by brute force in time $O(n^{O(k)})$. The increasingly substantial theory of parameterized complexity and algorithmics unfolds from this small start. The positive toolkit of FPT turns out to be technically quite rich, and the negative toolkit of $W[1]$-hardness turns out to be widely applicable.

The main motivation for parameterized complexity is that in almost all settings and for almost all purposes of computing, the input has "extra structure" that we are able relevantly to capture with the mathematical device of the parameter. To put the case more sharply, but negatively: When does it happen that you know *nothing* about your input apart from the size of the input file? Outside of cryptography: "Never!"

Historically, a key motivating source for parameterized complexity has been the monumental graph minors project of Robertson and Seymour [RS85]. Our realm of concern here is with computer science rather than pure mathematics, and this leads to *the correct way to view the graph minors project within this realm of concern*. The parameterized computational decision problem GRAPH MINOR takes as input graphs G and H and asks whether H is a minor of G (that is, whether a graph isomorphic to H can be obtained from G by contracting edges of a subgraph of G). This is a fundamental problem, naturally parameterized by H. The GRAPH MINOR problem inevitably commands attention within the realm of theoretical computer science. The point is: all of the beautiful structure theory of the graph

minors project, *pathwidth*, *treewidth*, etc., is necessary (as far as we know) in order to show that the GRAPH MINOR problem, parameterized by H, is fixed-parameter tractable.

In this perspective, the many-layered structure theory of the graph minors project can be viewed as naturally associated to the parameterized GRAPH MINOR problem. The point for our monster machine program is that *every sensible parameter (in the sense of parameterized complexity) should lead to a structure theory project, naturally associated to the parameterized problem* in a "similar" manner. Where the efforts of FPT algorithm design lead, in this point of view, is to:

> One, two, ..., a thousand analogs of the graph minors project: one for every fundamental parameterized problem.

1.2 An Ecology of Relevant Parameters

The extent to which the structure theory of the graph minors project has turned out to be of practical relevance to computing is quite striking. For one example, most naturally occuring databases have bounded treewidth, a matter of immense significance to the realistic assessment of the complexity of database problems [GM99, FFG01, Gr01].

Another example is the problem of TYPE CHECKING of programs written in high-level logic-based programming languages such as ML. This problem has been shown to be complete for EXP, and thus "extremely" intractable from the classical point of view. Nevertheless, the ML compilers generally work just fine. The explanation is that most naturally occuring programs have a maximum type-declaration nesting depth k of no more than 5. The FPT type-checking algorithm that runs in time $O(2^k n)$ is entirely adequate in practice. The reason why naturally occuring programs have small nesting depth is that otherwise the programs quickly become incomprehensible to the programmer.

A perspective on this quoted from the survey [DFS99]:

> We feel that the parametric complexity notions, with their implicit ultrafinitism, correspond better to the natural landscape of computational complexity, where we find ourselves overwhelmingly among hard problems, dependent on identifying and exploiting thin zones of computational viability. Many natural problem distributions are generated by processes that inhabit such zones themselves (e.g., computer code that is written in a structured manner so that it can be comprehensible to the programmer), and these distributions then inherit limited parameter ranges because of the computational parameters that implicitly govern the feasibility of the generative processes, though the relevant parameters may not be immediately obvious. [1]

[1] For a philosophically similar discussion see [Gur89].

This leads to a programmatic point of view about practical deliverable (E). We want to know how all these various parameterized structural notions interact with all the other computational objectives one might have. In other words, the familiar paradigm of efficiently solving various problems for graphs of bounded treewidth just represents one row of a matrix of algorithmic questions that arise from the relevant parameterized structure theories. In the case of MAX LEAF, we investigate how to solve the INDEPENDENT SET problem, etc., on graphs bounded "max leaf number," exploiting the structure that bounding this parameter yields.

Such a complete "ecological" study of bounded computational parameters may turn up more useful surprises akin to the remarkable algorithmic gold mine of bounded treewidth structure. The structure theories associated to parameterized problems can also be expected to interact in interesting ways.

That completes an overall sketch of the FPT monster machine program. In the next sections we will discuss the five objectives one at a time, in each case describing a canonical recipe for the objective, and discussing how the recipe unfolds for the MAX LEAF problem.

2 Objective A: Fixed-Parameter Tractability

Our objective here is a better FPT algorithm. After a series of improvements based on various approaches [FL92, Bod93, DF99, FMRS00], the current best FPT algorithm for MAX LEAF is due to Bonsma, Brueggemann and Woeginger [BBW03]. The running time of the algorithm is $O(n^3 + 9.49^k k^3)$. In the convenient notation introduced by Woeginger [Woe03] that ignores polynomial time factors, this is $O^*(9.49^k)$.

But what is a *better* FPT algorithm? A straightforward answer could be $O^*(f(k))$ for a more slowly growing function $f(k)$. We achieve this for MAX LEAF, but this is *not* the sense of "better FPT algorithm" for the monster machine program. There are now two well-established "races" in FPT algorithm design. There is the "$f(k)$ race." And there is the completely general "kernelization race" set up as follows.

LEMMA 1. *A parameterized problem Π is in FPT if and only if there is a polynomial-time (polynomial in both n and k) transformation that takes (x, k) to (x', k') such that:*
(1) (x, k) is a yes-instance of Π if and only if (x', k') is a yes-instance of Π,
(2) $k' \leq k$, and
(3) $|x'| \leq g(k)$ for some fixed function g.

In the situation described by Lemma 1, we say that we can *kernelize* to instances of size at most $g(k)$. The proof is trivial. Suppose Π is solvable by an FPT algorithm with a running time of $f(k)n^c$. If $|x| = n \geq f(k)$ then use the algorithm to solve Π in time $O(n^{c+1})$. If $|x| < f(k)$ then take $x' = x$ and $g = f$.

Lemma 1 is the charter for our program. If Π is an FPT parameterized problem,

then we are *necessarily* interested in finding a polynomial-time pre-processing (or *kernelization*) algorithm where $g(k)$ is as small as possible. This has a direct payoff in terms of program objective (B). In the case of MAX LEAF, the best previous P-time kernelization is to $5.75k$, due to Elena Prieto [Pr05], following closely a sketch in [FMRS00]. There is an implicit $34k$ kernelization in [BBW03]. We will improve this to $3.75k$. Our improved kernelization also yields an FPT algorithm for MAX LEAF with an improved parameter function $f(k)$, by exploring all k-element subsets of the $3.5k$ problem kernel. Figure 1 shows some of the reduction rules employed in [BBW03].

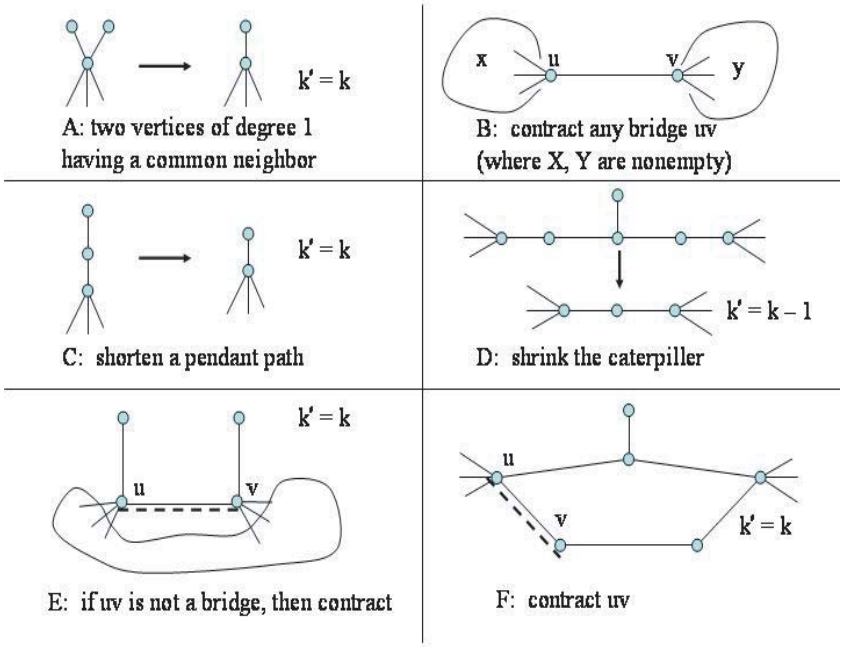

Figure 1. A sampler of reduction rules for MAX LEAF.

Canonically, in the sense of our program, an FPT algorithm has the following form:

Step 1: Kernelize the input by exhaustively applying some set of data-reduction rules to obtain a *reduced* instance of the problem.

Step 2: If the reduced instance is large, then output the answer.
Step 3: If the reduced instance is small, then analyze the kernel by brute force or table lookup or (preferably) more clever means.

Step 2 is accomplished by means of a Kernelization Lemma such as:

LEMMA 2. *If (G, k) is a reduced instance of* MAX LEAF *and the graph G has more than $3.75k$ vertices, then we can answer YES. The graph G necessarily has a spanning tree with at least k leaves.*

In Step 2, it does not matter whether the answer output is YES or NO. For example, the VERTEX COVER problem (parameterized by the size of the vertex cover) supports a number of quite surprising reduction rules [DFS99, Fe03, ACFLSS04, CFJ04]. A large reduced instance for VERTEX COVER is automatically a NO instance, contrasting with MAX LEAF.

Results from extremal combinatorics can often be used to construct Kernelization Lemmas that yield pretty good FPT algorithms. In the case of MAX LEAF Kleitman and West [KW91] showed:

PROPOSITION 3. *Every (simple) connected graph $G = (V, E)$ with minimum degree at least 3 has a spanning tree with at least $\frac{1}{4}|V| + 2$ leaves.*

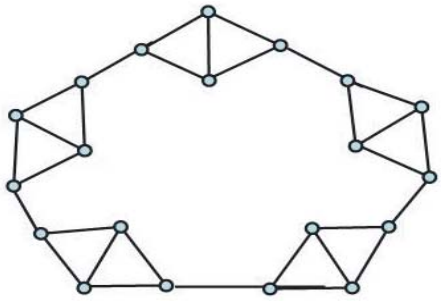

Figure 2. The Kleitman-West bound is best possible for minimum degree 3.

The Kleitman-West extremal is used in the following way in [BBW03]. Their FPT algorithm has the canonical form described above. They kernelize in polynomial time to an instance (G', k') where G' can be described as obtained from a simple graph H of minimum degree at least three, with edges of H replaced by *caterpillar paths* (the details are not important). In particular, G' contains a subdivision of H. If the underlying graph H has more than $4k$ vertices, then G' is automatically a yes-instance by the Kleitman-West result. The reduced instance

kernel is analyzed by exploring all k-subsets of the vertices of the underlying graph H, yielding a complexity of $O^*(\binom{4k}{k}) = O^*(9.49^k)$.

Improving on a long series of various approaches to FPT design for the MAX LEAF problem and powered by a strong result from extremal graph theory, how complain? But we do complain, because even though this FPT result has the canonical form justified by Lemma 1, it proves the Kernelization Lemma in the *wrong way*. The *right way* to establish the Kernelization Lemma is to take up the challenge of proving *a kind of extremal theorem native to the realm of concern of algorithms and complexity*. We next describe the canonical recipe of our monster machine methodology for the right way to do it.

REMARK 4. The reduction rules shown in Figure 1 are some of the main ones employed by Bonsma, Brueggemann and Woeginger. One can observe that the graph shown in Figure 2 is irreducible under this set of rules, and therefore we cannot hope to prove a kernelization bound better than $4k$ without more powerful reduction rules than those shown in Figure 1. But this raises an important question: *What is the right way to find and prove reduction rules?* One of the subsidiary aims of the monster machine methodology is to systematize the search for polynomial-time reduction rules.

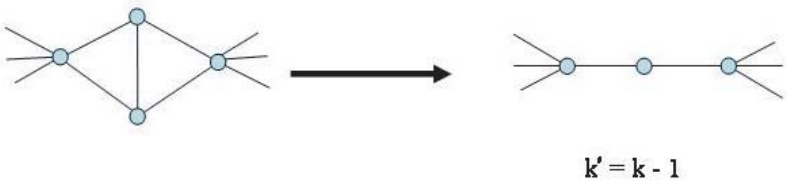

Figure 3. The K-W Dissolver Rule.

REMARK 5. The form of the graph in Figure 2 suggests looking for the reduction rule shown in Figure 3.

2.1 The Right Way to Prove a Kernelization Lemma

According to our methodology, the right way to prove the Kernelization Lemma is to "study the boundary between YES and NO." In general, an instance of a parameterized problem consists of a pair (x, k), so the boundary to which we refer is located by holding x fixed and varying k and regarding the outcome of the

decision problem. We are interested in what things look like at the "boundary" when x is reduced (under whatever polynomial-time reduction rules we may be able to find).

Thus the canonical approach is to prove a Boundary Lemma of the form indicated by this example in our study of the MAX LEAF problem:

LEMMA 6. *Suppose (G, k) is a reduced instance of* MAX LEAF*, with (G, k) a yes-instance of the problem and $(G, k + 1)$ a no-instance. Then $|G| \leq ck$. (Here c is a small constant that we will clarify below.)*

When we start off to prove such a Boundary Lemma, we do not initially know what bound we will eventually succeed in proving and we don't know exactly what we mean by *reduced*. The methodological situation is that in the course of an attempted proof, both of these details are worked out. The methodology serves to organize our efforts. As our arguments unfold, we will be confronted with structural situations that suggest new reduction rules.

There is a canonical way to prove the Boundary Lemma that involves two crucial strategic choices:
(1) A choice of *witness structure* for the hypothesis that (G, k) is a yes-instance.
(2) A choice of inductive priorities.

The overall structure of the argument is "by minimum counterexample" according to the priorities established by (2), which generally make reference to (1). Given these choices, the proof then proceeds by a series of small steps consisting of structural claims that lead to a detailed structural picture at the "boundary" — and thereby to the bound on the size of G that is the conclusion of the lemma.

We will illustrate the recipe with proofs of three increasingly stronger forms of a Boundary Lemma for the MAX LEAF problem. We will refer to these as Boundary Lemmas I,II, and III. The point of giving three different proofs is to illustrate the methodology, and in particular, to illustrate the effects of different choices of witness structure and inductive priorities.

2.2 Boundary Lemma I

LEMMA 7. **Boundary Lemma I.** *Suppose (G, k) is a reduced instance of* MAX LEAF*, with (G, k) a yes-instance of the problem and $(G, k + 1)$ a no-instance. Then $|G| \leq 7.75k$.*

Proof. The proof is by minimum counterexample. If (G, k) is a yes-instance, $G = (V, E)$, then we can assume we are given as a witness structure a tree subgraph $T = (V', E')$ of G that has k leaves, and we can also assume that G is connected.

We do not assume that T is a spanning subgraph. (If T is not spanning, then it clearly extends to a spanning tree T' for G that has at least k leaves.)

A counterexample to the lemma would be a graph $G = (V, E)$ such that: (1) (G, k) is a reduced instance of MAX LEAF, (2) (G, k) is a yes-instance of MAX

LEAF, (3) $(G, k+1)$ is a no-instance, and (4) $|G| > 7.75k$.

Among all such counterexamples, consider one where the witness subgraph tree T is as small as possible.

Let $O = V - V'$ be the set of vertices not in the witness subtree T, which we will refer to as *outsiders*. Let L denote the leaves of T, I the internal (non-leaf) vertices of T, $B \subseteq I$ the *branch vertices* of T (the non-leaf, internal vertices of T that have degree at least 3 with respect to T), and let J denote the *subdivider vertices* of T (the non-branch internal vertices of T that have degree 2 with respect to T). See Figure 4 for an illustration of these sets of vertices.

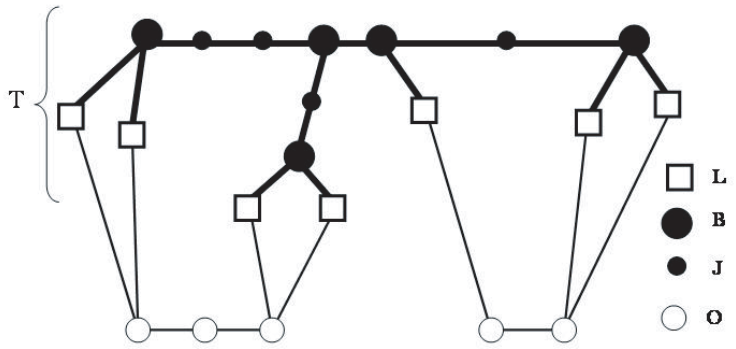

Figure 4. The witness tree and various sets of vertices.

We will also need to discuss the structure of T in more detail, so we introduce the following further terminology. A path $\langle b, j_1, ..., j_r, b' \rangle$ in T where $b, b' \in B$ are branch vertices of T, and the vertices j_i for $i = 1, ..., r$ are subdivider vertices of T is termed a *topological edge* or *topo-edge* of the tree T. In this situation we will say that b and b' are *topologically adjacent* in T, and in order to be able to refer to the length of the path $\langle b, j_1, ..., j_r, b' \rangle$, we may say that b and b' are joined by an r*-topo-edge* in T. We will eventually be interested in structures that arise by considering the subtrees of T induced by 0-topo-edges and 1-topo-edges of T. (Note that a 0-topo-edge is just an ordinary edge of T.)

Claim 1. No internal vertex of T is adjacent in G to a vertex of O.

Proof of Claim 1. Otherwise, we could augment T to a subgraph tree with $k+1$ leaves, contradicting that $(G, k+1)$ is a no-instance. (See Figure 5.)

Claim 2. A leaf vertex of T is adjacent to at most one outsider.

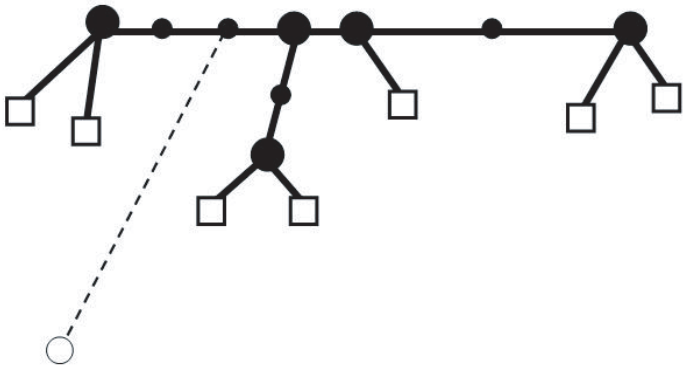

Figure 5. No internal vertex of T is adjacent to an outsider, else $k+1$ leaves.

Proof of Claim 2. Otherwise we contradict that $(G, k+1)$ is a no-instance.

Claim 3. The subgraph $\langle O \rangle$ induced by the outsiders is acyclic.

Proof of Claim 3. Otherwise, since G is connected, we contradict that $(G, k+1)$ is a no-instance. See Figure 6.

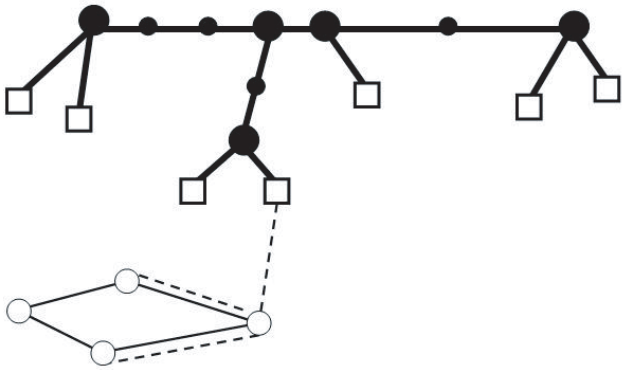

Figure 6. $\langle O \rangle$ is acyclic, else $k+1$ leaves.

Claim 4. The subgraph $\langle O \rangle$ induced by the outsiders has maximum degree at most 2.

Proof of Claim 4. Otherwise we contradict that $(G, k+1)$ is a no-instance. See Figure 7.

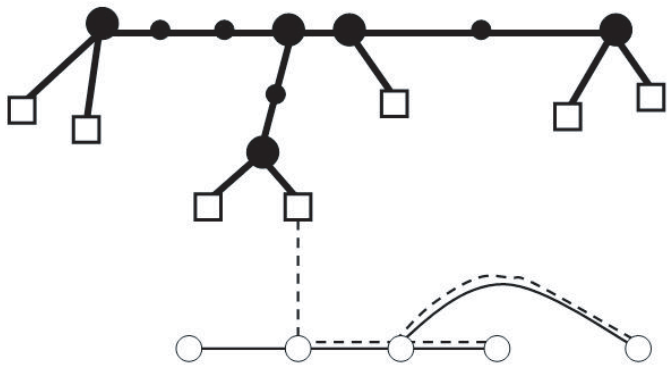

Figure 7. $\langle O \rangle$ has maximum degree at most 2, else $k+1$ leaves.

By Claims 3 and 4, the subgraph $\langle O \rangle$ induced by the outsiders consists of a union of paths.

Claim 5. A leaf of G cannot be adjacent to an interior vertex of a path of $\langle O \rangle$.

Proof of Claim 5. Otherwise we contradict that $(G, k+1)$ is a no-instance.

Claims 1-5 show that $\langle O \rangle$ consists of a disjoint union of paths, and that the interior vertices of these paths have degree 2 in G. This motivates us to look for a reduction rule that addresses this structure. A reduction rule first reported in [FMRS00] was discovered in this way. The *Two Adjacent Degree 2 Rule* says that for $k' = k$, if u and v are two adjacent vertices of degree 2, then the edge uv can either be deleted or contracted, depending on whether or not it is a bridge.

Claim 6. $\langle O \rangle$ consists of a union of paths, where each path has at most 3 vertices.

Proof of Claim 6. Otherwise the "Two Adjacent Degree 2" reduction rule applies.

The evident structure of $\langle O \rangle$ suggests looking for a reduction that applies to vertices of degree 1 in G. In fact, there is a simple but not at all obvious rule to eliminate vertices of degree 1. (The reader might want to take a moment to try to discover it, in order to appreciate some of the depth of the subject of reduction rules. A degree one rule: how hard can that be?)

Claim 7. $|O| \leq .75k$

Proof of Claim 7. This follows from the fact that T has k leaves, and from Claims 2,5 and 6, inducting on the number of path components in $\langle O \rangle$. The "worst case"

for the induction is where a path component of $\langle O \rangle$ has 3 vertices (a path of length 2). In this case each endpoint of the path must be adjacent to at least two leaves of T, else G is reducible either by the "Degree 1" or the "Two Adjacent Degree 2" reduction rules. Thus the number of leaves is $k \geq 4|O|/3$.

The picture that has emerged through Claims 1-7 is starting to give us a handle on how big G can be. Next we must bound the size of T.

Claim 8. $|B| \leq k - 2$

Proof of Claim 8. A straightforward induction on the number of leaves.

Claim 9. The subgraph induced by the vertices of a topological edge of T contains no further edges.

Proof of Claim 9. This is trivially true for an r-topo-edge where $r = 0$, so suppose $r \geq 1$. But then we can re-engineer T to have $k + 1$ leaves, as shown in Figure 8.

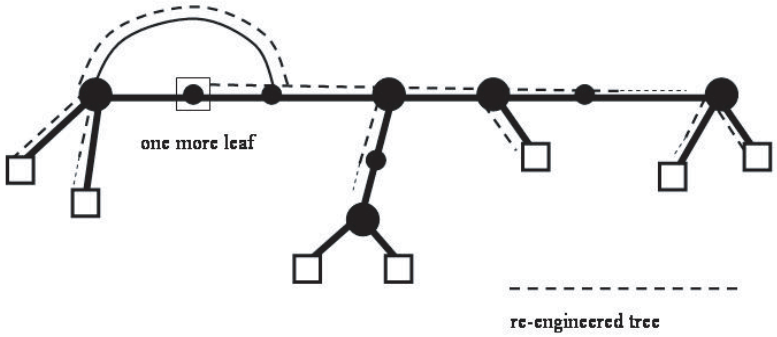

Figure 8. Topo-edges are induced subgraphs.

Claim 10. There are no r-topological edges in T for $r \geq 6$.

Proof of Claim 10. Suppose we have a path $\langle b, j_1, ..., j_r, b' \rangle$ in T where $b, b' \in B$ are branch vertices of T, and the vertices j_i for $i = 1, ..., r$ are subdivider vertices of T, where $r \geq 6$. let T_b denote the subtree of T "to the left" of b, and let $T_{b'}$ denote the subtree of T "to the right" of b'. The vertex j_3 cannot be adjacent in G to a vertex of T_b, otherwise we can re-engineer T to have $k + 1$ leaves as shown in Figure 9.

The vertex j_3 cannot be adjacent to a vertex of $T_{b'}$ for similar reasons. By Claim 9, and by symmetry, the vertices j_3 and j_4 must have degree 2 in G. But then G

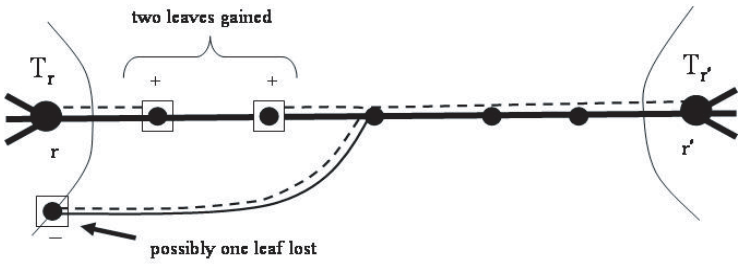

Figure 9. Long topo-edges of T have middle vertices of degree 2 in G.

is reducible, either by Rule B of Figure 1 (contracting a bridge) or by the "Two Adjacent Degree 2 Rule."

Claim 11. Each leaf in T is adjacent in T to a branch vertex.

Proof of Claim 11. Otherwise, we would contradict that T is as small as possible. See Figure 10.

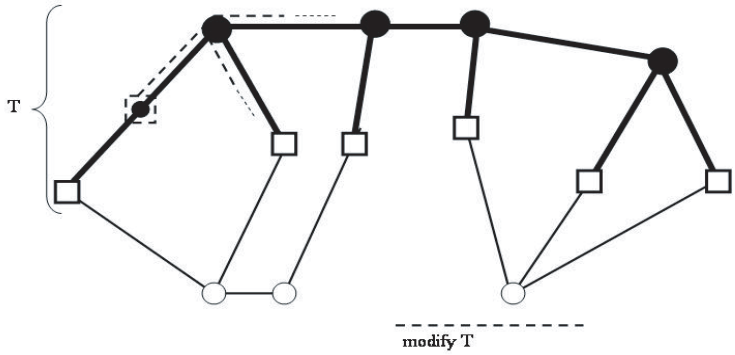

Figure 10. A leaf is adjacent to a branch, else smaller T.

Claim 12. $|J| \leq 5(k-3)$

Proof of Claim 12. This follows from Claims 9 and 10.

We can now conclude the proof of Boundary Lemma I on the basis of Claims 7,8 and 12. ■

2.3 Boundary Lemma II

LEMMA 8. **Boundary Lemma II.** *Suppose* (G, k) *is a reduced instance of* MAX LEAF*, with* (G, k) *a yes-instance of the problem and* $(G, k+1)$ *a no-instance. Then* $|G| \leq 5.75k$.

Proof. The proof is by minimum counterexample. Witnessing that (G, k) is a yes-instance we have a tree T with k leaves, as in the proof of Boundary Lemma I. Here we will have a little more structure and another inductive priority. We consider that the tree T is equipped with a root vertex $r \in V'$. The possible counterexample that we entertain in our argument is one where:
(1) T is as small as possible, and among all counterexamples satisfying this requirement, one where
(2) the sum over all leaves $l \in L$ of the distance in T from r to l is minimized.

All of the structural claims from the proof of Boundary Lemma I hold here as well, since essentially all we have done is add another "further" inductive priority. This additional priority allows us to establish a strengthening of Claim 10.

Claim 13. T does not have r-topological edges for $r \geq 4$.

Proof of Claim 13. Suppose we have a path $\langle b, j_1, ..., j_r, b' \rangle$ in T where $b, b' \in B$ are branch vertices of T, and the vertices j_i for $i = 1, ..., r$ are subdivider vertices of T, where $r \geq 4$. Let T_b denote the subtree of T "to the left" of b, and let $T_{b'}$ denote the subtree of T "to the right" of b'. We can suppose that the root of T lies in $T_{b'}$, without loss of generality. The vertices j_1 and j_2 cannot be adjacent to vertices in $T_{b'}$, else we can re-engineer T to have $k+1$ leaves, as in the proof of Claim 10. By Claim 9, and since G is irreducible (in particular, j_1 and j_2 do not both have degree 2 in G), at least one of j_1 or j_2 is adjacent by an ede e to a vertex x of T_b. But then by adding e to T and removing the edge xu, where xu is the first edge on the path in T from x to b, we can obtain a modified tree with k leaves where the priority (2) has been improved.

That concludes the proof of BL II. ■

2.4 Boundary Lemma III

LEMMA 9. **Boundary Lemma III.** *Suppose* (G, k) *is a reduced instance of* MAX LEAF*, with* (G, k) *a yes-instance of the problem and* $(G, k+1)$ *a no-instance. Then* $|G| \leq 3.75k$.

Proof. The proof is by minimum counterexample. Witnessing that (G, k) is a yes-instance is the following structure:
(1) As in the proof of Boundary Lemma I and Boundary Lemma II, we assume a subgraph tree $\mathcal{T}$ of G having k leaves, which we will refer to as the *witness tree*. We consider that the sets of vertices O, I, L, B and J (the *outsiders, internal vertices, leaves, branch vertices* and *subdivider vertices*, respectively) are defined

as in the proof of Boundary Lemma I (and pictured in Figure 1).
(2) A collection $\mathcal{T}^1 = \{\mathcal{T}^1[i] : i \in \mathcal{I}\}$ of vertex disjoint subtrees $\mathcal{T}^1[i]$ of $\mathcal{T}$ (called the *1-subtrees* of $\mathcal{T}$) where each 1-subtree $\mathcal{T}^1[i]$ is a maximal subtree of T induced by the 1-topo-edges of $\mathcal{T}$ together with any adjacent leaves or J-vertices. (See Figure 12.)
(3) For each 1-subtree $\mathcal{T}^1[i]$, a collection $\mathcal{T}_i^0 = \{\mathcal{T}^0[i,j] : i \in \mathcal{I}, j \in \mathcal{J}_i\}$ of edge-disjoint subtrees of $\mathcal{T}^1[i]$ (called the 0-subtrees of $\mathcal{T}^1[i]$) where each 0-subtree $\mathcal{T}^0[i,j]$ $\mathcal{T}_i^0$ is a maximal subtree of $\mathcal{T}^1[i]$ induced by the 0-topo-edges of $\mathcal{T}$. (See Figure 13.)

DEFINITION 10. The *global structure tree* $\mathcal{G}(\mathcal{T})$ of $\mathcal{T}$ is the rooted, vertex-weighted graph, composed by making one vertex for each 1-subtree of $\mathcal{T}$, weighting each vertex x of $\mathcal{G}(\mathcal{T})$ with the positive integer $p(x)$ according to the size (number of vertices) of the 1-subtree of $\mathcal{T}$ to which it corresponds, and making two vertices of $\mathcal{G}(\mathcal{T})$ adjacent if they are joined by a topo-path in $\mathcal{T}$. The *root* $r_{\mathcal{G}(\mathcal{T})}$ of $\mathcal{G}(\mathcal{T})$ corresponds to the 1-subtree of $\mathcal{T}$ containing the root of $\mathcal{T}$.

The Inductive Priorities. The inductive priorities for the argument are listed as follows, where we include as the zeroth priority the implicit induction on the number of leaves inherent in the form of the boundary lemma:
(0) The number of leaves of $\mathcal{T}$ is maximized.
(1) The number of vertices of $\mathcal{T}$ is minimized.
(2) The total number of vertices belonging to the 1-subtrees of $\mathcal{T}$ is minimized.
(3) The total number of edges belonging to the 1-subtrees of $\mathcal{T}$ is minimized (equivalently, the number of 1-subtrees of $\mathcal{T}$ is maximized.
(4) The total number of 0-subtrees of $\mathcal{T}$ is maximized.
(5) The sum

$$\Delta = \sum_{x \in V(\mathcal{G}(\mathcal{T}))} d(r,x) \cdot p(x)$$

is minimized.

The rest of the proof proceeds in the following stages. In Part I we assemble a series of structural claims that describe what we call *The Bridge World*. In Part II we set up *The Master Inequality* that will allow us to draw the conclusion of Boundary Lemma (a bound on the size of the reduced graph G) by summing over aspects of the structural situation, where the sum is taken over all of the 1-trees of the *Bridge World*. In Part III we use the Master Inequality to complete the proof by induction. The base case for the induction involves an extensive analysis of the structural situation for small 1-trees.

Part I: The Structure of the Bridge World.

We first attempt to give some intuitive picture of the main structure, and how this structure relates to previous work on graphs that exclude leafy tree subgraphs.

In the structure that is used in the best previous FPT algorithm for MAX LEAF [BBW03], you basically have $4k$ vertices (think of these as islands) joined by "bridges" (caterpillar paths) that have a restricted form. Here we have something roughly similar, in that we will have "islands" that are 1-trees, joined by "bridges" that are quite restricted in their possible structure, and of approximately the same size as the reduced caterpillar paths of [BBW03]. (Caterpillar paths disappear from our "picture" in some sense, under the force of the polynomial-time reduction rule for degree 1 vertices, which seems to have been unknown to the authors of [BBW03].) Our "islands," the 1-subtrees of $\mathcal{T}$, make up a forest that is *exactly* the main structure employed in the best current polynomial-time approximation algorithm for MAX LEAF, due to Solis-Oba [S-Ob98]. We go deeper, here, however, engaging the internal structure of the "islands" and how this internal structure interacts with the bridges that link the islands, squaring this structure off against our inductive priorities and against our expanding arsenal of polynomial-time reduction rules. (Priority (5), one might note, is akin to priority (2) in the proof of Boundary Lemma II. Priorities (1-4) here could be viewed as a finer elaboration of priority (1) in the proof of Boundary Lemma II.)

Considering the proofs of Boundary Lemmas I and II, it is clear that "the main problem" has to do with bounding the size of the population of subdivider vertices J. Suppose that we start with our picture of $\mathcal{T}$ as it emerges from the proof of Boundary Lemma I. The subgraph $\langle O \rangle$ of G induced by the outsiders O consists of little paths of at most 3 vertices. Any r-topological edge of $\mathcal{T}$ has $r \leq 5$. If $r \leq 1$ then the topo-edge becomes part of a 1-tree. If $r \geq 2$ then "cut out the middle." This cuts $\mathcal{T}$ into various subtrees: the 1-trees, together with some left-over material. The middle parts that are cut out consist of little paths of at most 3 vertices, which remind us of the components of $\langle O \rangle$, and in fact it will turn out the pieces (the components of $\langle O \rangle$ and the middle pieces cut out from long topo-edges) all behave more-or-less the same and constitute the "bridges" between the "islands" of the Bridge World.

DEFINITION 11. A *bridge* refers either to an edge between two leaves of different 1-trees (called a *0-bridge*), or to a connected component of the subgraph of G induced by the vertices that do not belong to any 1-tree of the witness structure (these are called *serious bridges*).

DEFINITION 12. A *portal* of a 1-tree is a leaf of the 1-tree that is either adjacent (by a 0-bridge) to a leaf of a different 1-tree (called a *0-portal*), or adjacent to a vertex of a serious bridge (and in this case is called a *serious portal*).

DEFINITION 13. A *link* is a path ρ from a portal of a 1-tree to a portal of a different 1-tree of the witness structure, where this path is part of $\mathcal{T}$. That is, a link is part of the means by which the 1-trees of the witness structure are joined to make the witness tree $\mathcal{T}$.

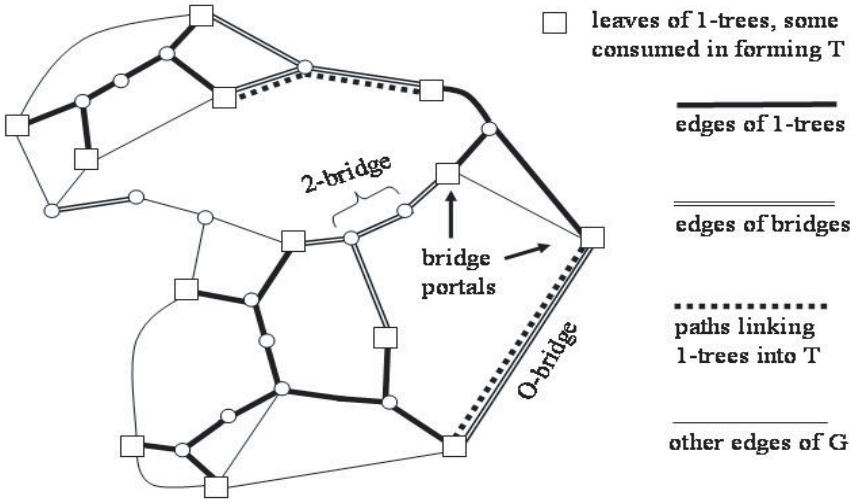

Figure 11. 1-trees, bridges, and links.

DEFINITION 14. A *gateway* vertex v of a 1-tree is a vertex that belongs to two different 0-trees in the 0-tree decomposition of the 1-tree. (Necessarily, v is a leaf of each of these 0-trees.)

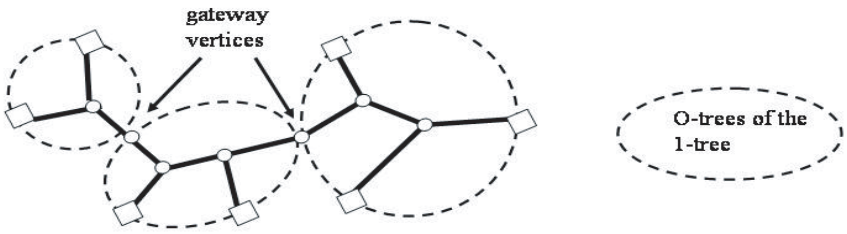

Figure 12. O-tree structure of 1-trees and gateway vertices.

The Bridge World has a huge amount of structure under our indicated inductive priorities. Space limitations for this abstract allow only a brief summary of some

of the main aspects. Of key importance is that the bridges are neatly separated in the sense that if p is a portal for a bridge B, then B is the only bridge for which p is a portal, unless the bridges for which p is a portal are all 0-bridges. In this case, all of the 0-bridges link p to the same 1-tree. A bridge B connects to at most two 1-trees. (A bridge B that connects to only one 1-tree is termed a *loop bridge*, and its portals are termed *loop portals*.) The fact that bridges are between at most two 1-trees establishes a clean global structure that we can model as a graph, where two 1-trees are adjacent if they have bridges between them. This fact is crucially used in the "two-sided counting strategies" of the Master Inequality, and the neat separation of the bridges is used in the induction on portals that establishes the Master Inequality. The detailed 0-tree substructure of 1-trees is important for this latter inductive argument, as it is this detailed structure that provides positive estimates for the quantities $L_i(p)$ and $W_i(p)$ discussed below in the context of this induction.

Part II: The Master Inequality.

Let t denote the number of 1-trees in the bridge world decomposition of G. We will ultimately conclude a bound on the size of G by summing over t *structural pictures*, one for each 1-tree. We will use N to denote the total number of vertices in G. Our goal is to obtain a bound $N \leq \alpha k$ for a small constant α.

DEFINITION 15. In the *structural picture* of the 1-tree $\mathcal{T}^1[i]$ of the bridge world decomposition of G we have the following elements:
(1) The 1-tree $\mathcal{T}^1[i]$ said to be *on the left side* of the picture.
(2) The other 1-trees of the bridge world decomposition, said to be *on the right side* of the picture.
(3) The serious non loop bridges that link $\mathcal{T}^1[i]$ on the left to 1-trees on the right. The vertices of these serious non loop bridges are said to be *in the middle* of the picture.
(4) The serious loop bridges that attach only to $\mathcal{T}^1[i]$. The vertices of these serious loop bridges are said to be *on the left side* of the picture.
(5) The 0-bridges that attach vertices of $\mathcal{T}^1[i]$ on the left side to vertices of the other 1-trees on the right side of the picture.

We will use the following variables that inventory aspects of the structure of G that appear in the ith structural picture. For the remainder of the discussion of the Master Inequality, we will use $\mathcal{T}[i]$ for short to refer to $\mathcal{T}^1[i]$. It is important to note that some of the inventoried quantities are counted on one side of the picture only, and some are counted on both sides.

DEFINITION 16. Variables for a census of the i^{th} structural picture.
X_i denotes the number of leaves of $\mathcal{T}[i]$ (on the left).
J_i denotes the number of internal vertices of $\mathcal{T}[i]$ (on the left).
B_i denotes the number of serious non loop bridge vertices (in the middle).

C_i denotes the number of serious loop bridge vertices (on the left).
$P(i)$ denotes the total number of portals of non loop serious bridges (on both sides).
$P_i^{\leftarrow}$ denotes the number of portals of $\mathcal{T}[i]$ of non loop serious bridges (on the left).
$P_i^{\rightarrow}$ denotes the number of portals of non loop serious bridges on the right side.
Q_i denotes the number of portals of serious loop bridges (on the left).
Z_i denotes the number of portals of 0-bridges (on both sides).
$Z^{\leftarrow}(i)$ denotes the number of portals of 0-bridges (on the left side only).
$Z^{\rightarrow}(i)$ denotes the number of portals of 0-bridges (on the right side only).
L_i denotes the number of non portal leaves of $\mathcal{T}[i]$ (on the left).

DEFINITION 17. S_i denotes the number of *saved vertices* of $\mathcal{T}[i]$, defined to be the difference between the maximum possible number of internal vertices of a 1-tree with X_i leaves, and the actual number J_i of internal vertices of the i^{th} 1-tree. Thus $S_i = (2X_i - 5) - J_i$.

DEFINITION 18. The *link degree* of $\mathcal{T}[i]$ denoted r_i is defined to be the degree of the vertex corresponding to $\mathcal{T}[i]$ in the global structure graph.

The following variables are used to give the overall inventory of the vertices of G.

DEFINITION 19. X denotes the total number of leaves of the 1-trees.
J denotes the total number of vertices of the 1-trees.
B denotes the total number of serious non loop bridge vertices.
C denotes the total number of serious loop bridge vertices.
P denotes the total number of portals of non loop serious bridges.
Q denotes the total number of portals of serious loop bridges.
Z denotes the total number of portals of 0-bridges.
L denotes the total number of non portal leaves of the 1-trees.

LEMMA 20. *Suppose:*

$$\sum_{i=1}^{t} (B_i + C_i + 2X_i + 2J_i) \leq \sum_{i=1}^{t} \alpha\,(P_i + 2Q_i + Z_i + 2L_i - 2r_i)$$

then $N \leq \alpha k$.

Proof. By the definition of the total quantities, and noting that in summing over all structural pictures, vertices of bridges are counted twice (relevant to the sum on the left side of the inequality) and portals of 0-bridges and non loop serious bridges are counted twice (relevant to the sum on the right side of the inequality), we have

$$2B + 2C + 2X + 2J \leq \alpha\,(2P + 2Q + 2Z + 2L - 4(t-1))$$

Since

$$N = B + C + X + J$$

and

$$X = P + Q + Z + L$$

this can be rewritten as

$$2N \leq \alpha(2X - 4(t-1)) = 2\alpha k$$

using the fact that while the total number of leaves of the 1-trees is X, $2(t-1)$ of these are used up in linking the 1-trees together to form $\mathcal{T}$. ■

LEMMA 21. *The inequality of Lemma 7 holds if*

$$\sum_{i=1}^{t} B_i + 2C_i \leq \sum_{i=1}^{t} (\alpha - 3)\left(P_i + Z_i - \left(4 + \frac{2}{\alpha - 3}\right)\right) + (2\alpha - 6)(Q_i + L_i) + 2S_i$$

Proof. Arguing backwards, the inequality of Lemma 7 is true if:

$$\sum_{i=1}^{t} (B_i + 2C_i + 2X_i + +4X_i - 10 - 2S_i) \leq \sum_{i=1}^{t} \alpha\,(P_i + 2Q_i + Z_i + 2L_i - 2r_i)$$

where we rewrite the left side using the fact that the maximum number of internal vertices of a 1-tree with X_i leaves is $(X_i - 2) + (X_i - 3) = 2X_i - 5$, and using the definition of the savings S_i as the difference between J_i and the maximum possible. This inequality holds if

$$\sum_{i=1}^{t} (B_i + 2C_i + 6P_i^{\leftarrow} + 6Q_i + 6L_i + 6Z^{\leftarrow}(i))$$

$$\leq \sum_{i=1}^{t} [2S_i + 10 + \alpha(P_i^{\leftarrow} + P_i^{\rightarrow} + 2Q_i + Z^{\leftarrow}(i) + Z^{\rightarrow}(i) + 2L_i - 2r_i)]$$

where in formulating this inequality we use the fact that

$$X_i = P_i^{\leftarrow} + Q_i + L_i + Z^{\leftarrow}(i)$$

and break up P_i and Z_i into left side and right side populations in the i^{th} structural picture. This bound holds if we have

$$\sum_{i=1}^{t}(B_i + 2C_i + (6-\alpha)P_i^{\leftarrow} + (6-\alpha)Z^{\rightarrow}(i))$$
$$\leq \sum_{i=1}^{t}((2\alpha-6)(Q_i+L_i) + \alpha(P_i^{\rightarrow} + Z^{\rightarrow}(i)) + 2S_i + 10 + 2\alpha r_i)$$

where terms have simply been rearranged. This is true if:

$$\sum_{i=1}^{t} B_i + 2C_i \leq \sum_{i=1}^{t}(\alpha-3)\left(P_i + Z_i - \left(4 + \frac{2}{\alpha-3}\right)\right) + (2\alpha-6)(Q_i+L_i) + 2S_i$$

The last step is justified by two small arguments. The first is that

$$\sum_i(\alpha P_i^{\rightarrow} + (\alpha-6)P_i^{\leftarrow}) = \left(\sum_i P_i^{\rightarrow}\right) + \left(\sum_i(\alpha-6)P_i^{\leftarrow}\right) = \alpha P + (\alpha-6)P$$

and

$$(2\alpha-6)P = \sum_i(\alpha-3)P_i$$

and similarly for the Z_i terms. The second small justification is that

$$\sum_{i=1}^{t} r_i = 2(t-1)$$

■

Our ultimate goal in Boundary Lemma III is to prove: $N \leq \alpha k$ for a small constant α, given as $\alpha = 3.75k$ in the statement of BL III. For convenience in handling the Master Inequality for 1-trees, define

$$\delta = 8 + \frac{2}{\alpha-3}$$

LEMMA 22. *If the inequality:*

$$\sum_{i=1}^{t} B_i + 2C_i \leq \sum_{i=1}^{t}(\alpha-3)(P_i + Z_i + 2r_i - \delta) + +(2\alpha-6)(Q_i+L_i) + 2S_i$$

holds, then $N \leq \alpha k$.

Proof. This follows from the previous lemmas, and by a small rearrangement, noting that

$$\sum_{i}^{t}(2r_i - 4) = (2\sum_{i}^{t} r_i) - (\sum_{i}^{t} 4) = 4(t-1) - 4t = -4$$

■

Our basic approach is to prove the required inequality separately for each i. Considering the i^{th} structural picture, we will use the well-behaved structure of the Bridge World to prove the i^{th} inequality by induction on the serious portals of the i^{th} 1-tree $\mathcal{T}^1[i]$. The case analysis for this induction will use the local structure in the vicinity of the portal, and it becomes important to "credit" the local contribution to the quantities on the right side of the inequality. The quantity of the *savings* S_i of the i^{th} 1-tree is defined "globally" for the 1-tree. We next describe a crediting system that indexes the savings of a 1-tree to local structure.

The Bookkeeping for 1-Tree Savings S_i.

The i^{th} 1-tree $\mathcal{T}^1[i]$ has an internal structure consisting of 0-subtrees. These are *adjacent* if they share a gateway, and according to this notion of adjacency, the 0-subtrees are naturally organized into a tree (of the 0-subtrees of $\mathcal{T}^1[i]$). We describe an assignment of a *savings value* $v(l)$ to each leaf l of the 1-tree. This assignment has the property that

$$\sum_{l} v(l) = S_i$$

This assignment is calculated in four phases:
(1) The first phase consists of calculating an initial *net savings value* for each 0-tree.
(2) The second phase consists in transferring net savings value between 0-trees. At the end of the second phase, only 0-trees that have at least one leaf of $\mathcal{T}^1[i]$ retain positive net savings value.
(3) In the third phase, the net savings value of each 0-tree is divided and assigned equally to the leaves of the 0-tree that are leaves of $\mathcal{T}^1[i]$.
(4) In the fourth phase, some of the assigned leaf values are transfered from non-portal leaves of $\mathcal{T}^1[i]$ to portal leaves of $\mathcal{T}^1[i]$.

Due to space limitations for this extended abstract, details of the savings assignment scheme are omitted.

DEFINITION 23. The total amount of savings value assigned to portals (of $\mathcal{T}^1[i]$) on the left side of the i^{th} structural picture is denoted $W_i^{\leftarrow}$. The total amount of savings value assigned to portals on the right side of the i^{th} structural picture is

denoted $W_i^{\rightarrow}$. We define $W_i = W_i^{\leftarrow} + W_i^{\rightarrow}$. The difference $S_i - W_i^{\leftarrow}$ is denoted U_i.

LEMMA 24. **The Master Inequality for 1-Trees.** *If for each i we have:*

$$B_i + 2C_i \leq (\alpha - 3)(P_i + Z_i + 2r_i - \delta) + (2\alpha - 6)(Q_i + L_i) + 2U_i + W_i$$

then $N \leq \alpha k$.

The Master Inequality is proved for small 1-trees by carefully analyzing their (entire) structure. This provides a base step. For large 1-trees we can essentially amortize the δ term (which can be thought of as a kind of "startup cost"). Note that the left side of the Master Inequality can be indexed as contributed by bridges connected to serious portals p. The structure of the Bridge World allows us straightforwardly to define these contributions as $B_i(p)$ and $C_i(p)$. On the left side, these quantities provide a complete breakdown of B_i and C_i. For the quantities on the right side of the Master Inequality, we can define contributions $P_i(p)$, $Z_i(p)$, $Q_i(p)$, $L_i(p)$ and from the savings assignment scheme we have $W_i(p)$. In defining these, it is of course important that for portals $p \neq p'$ the credited contributions are disjoint. Because our goal is an inequality, it is not necessary that these contributions credited to portals p provide a complete breakdown of the total quantities. These credit assignments are defined in terms of local structure. The details have to be omitted from this extended abstract.

Supposing that the base step holds for all 1-trees having fewer than m leaves, we prove the following lemma that gives us the inductive approach.

LEMMA 25. **The Portal Inequality.** *Suppose that the 1-tree has at least m leaves, and that for every serious portal p*

$$\begin{aligned} B_i(p) + 2C_i(p) \leq (\alpha - 3)(1 - (\delta/m))(P_i(p) + Z_i(p)) \\ +(2\alpha - 6)(Q_i(p) + L_i(p)) + W_i(p) \end{aligned}$$

then the Master Inequality holds.

Part III: The Induction. The induction is completed by making a generic analysis of the local structure in the vicinity of portals, according to cases that address the various ways that a leaf of the 1-tree can serve as a serious portal. Based on Claim 6 of the proof of BL I (this carries over to BL III) one can scout the difficulties ahead. Making completely pessimistic assumptions about the p-contributions to the more difficult terms to "see" of the right side (that is, assuming $L_i(p) = 0$ and the like), it is straightforward to verify that the Portal Inequality will hold "easily" for $\alpha \leq 3.8$, for moderate m. Many details must here be omitted.

2.5 A Summary of the Recipe for Objective A

The monster machine recipe for the objective of designing an FPT algorithm involves the following steps:

(1) Determine the polarity of the boundary, and set up the boundary lemma.
(2) Choose a witness structure.
(3) Set inductive priorities.
(4) Develop a series of structural claims that describe the situation at the boundary.
(5) Discover reduction rules that can act in polynomial time on relevant structural situations at the boundary.
(6) As the structure at the boundary becomes clear, fill in the blank regarding the kernelization bound.

We have to emphasize that while this recipe is *the right way* to design an FPT algorithm, and it is *necessary* in the sense that a parameterized problem is FPT *if and only if* it is polynomial-time kernelizable to a problem kernel of size bounded by a function of k, which inevitably leads to the challenge of developing extremal structure theory (modulo polynomial time data-reduction and pre-processing), this is not the *only* way to develop FPT algorithms. In particular, the above recipe is silent on what to do with the kernel. We say nothing here about how to develop efficient branching strategies, although it is conceivable that general recipes for employing "parameter-appropriate structure theory" in branching strategies for sophisticated problem kernel analysis may be possible.

One can point to at least three other important issues at the level of general methodology that would lead beyond what we offer here:

1. Turing kernelizability. The charter for the monster machine program is Lemma 1, that tells us that a parameterized problem is fixed-parameter tractable if and only if it is polynomial-time kernelizable. The notion of instance reduction that is operative in this lemma, the polynomial-time transformation of (x, k) to the simpler reduced instance (x', k') is a many:1 transformation. One can generalize the notion of many:1 reduction to Turing reduction, and this actually turns up quite importantly in the practice of FPT algorithm design, for example in [RSV04, DFRS04, Ma04, DFLRS05, GGHNW05]. How should the quest for polynomial-time extremal structure theory unfold under this "more generous" characterization of FPT? What is the standard recipe for establishing Turing kernelization bounds?

2. Problem annotation. Combinatorial problems ought to become more complex when the inputs are annotated. For example, we might consider a generalized MAX LEAF problem where vertices and edges have various annotations as to whether they *must* be leaves (or internal vertices) in a solution, etc. Such a generalized form of the problem would generally be expected to be "more difficult" than the vanilla form of the problem. It is striking, however, that several of the "best known" FPT algorithms for various problems, are based on these generalized, annotated forms of the problems. Examples include PLANAR DOMINATING SET [AFFFNRS04] and FEEDBACK VERTEX SET [DFLRS05]. Should annotation be part of the recipe for the best possible polynomial-time kernelization?

3. Algorithmic forms of the boundary lemma approach. The hypothesis of the

Boundary Lemma that (G, k) is a yes-instance implies that there exists (according to our strategic choice of what this means) a witness structure to this fact. There is no assumption that we have algorithmic access to this structure, and when we discover reduction rules, these have to be transformations that can be applied to (G, k) and structure that can be discovered in (G, k) in polynomial time. In other words, reduction rules cannot be *defined* with respect to the witness structure. Can we describe more general approaches to kernelization where the witness structure used in the proof of the Boundary Lemma is polynomial-time computable, and this structure provides a conditional context for some reduction rules? How would this change the monster machine recipe?

3 Objective B: Powerful Pre-processing and Data-Reduction Routines

Three important points:

• Pre-processing can have a very powerful effect on natural input distributions. The "gold standard" successes in software development for hard problems (e.g., CPLEX for integer linear programming) depend heavily on sophisticated pre-processing routines.

• Pre-processing routines have a permanent and universal significance for practical computing. In practical computing you will almost *always* pre-process your input (and generally re-kernelize on every branch of a backtracking effort). You will pre-process your input even if your main strategy is simulated annealing or roaming ants, genetic, memetic or the kitchen sink, and regardless of whether the seemingly relevant parameters really are *small* (maybe not — for effective pre-processing *this doesn't matter*).

• Pre-processing reduction rules can be entirely nontrivial and surprising.

In the area of algorithms and complexity there has been a kind of myth that we design good algorithms, and then others will come along and implement what we have designed. Generally, what practitioners will do, if they even read theory papers at all, is ransack them for ideas that they might be able to use in designing practical algorithms. The work of algorithm theorists is destined for the most part to be digested and deconstructed, not implemented. Algorithm theorists should think ahead about this and make it easy. Our goal, after all, is to serve those who want to compute. Reduction rules are clearly a ready-to-go deliverable to implementors with practical concerns.

3.1 The Unexpected Power of Pre-processing

A crucial question is: *What are the actual inputs that practical computing implementations have to deal with?* Karsten Weihe has given a thought-provoking account in [Wei00].

Weihe's Train Problem. Weihe describes a problem concerning the train systems of Europe [Wei98]. Consider a bipartite graph $G = (V, E)$ where V is bipartitioned into two sets S (stations) and T (trains), and where an edge represents that a train t stops at a station s. The relevant graphs are huge, on the order of 10,000 vertices. The problem is to compute a minimum number of stations $S' \subseteq S$ such that every train stops at a station in S'. It is easy to see that this is a special case of the HITTING SET problem, and is therefore *NP*-complete. Moreover, it is also $W[1]$-hard [DF99], so the straightforward application of the parameterized complexity program seems to fail as well.

However, the following two reduction rules can be applied to simplify (preprocess) the input to the problem. In describing these rules, let $N(s)$ denote the set of trains that stop at station s, and let $N(t)$ denote the set of stations at which the train t stops.

1. If $N(s) \subseteq N(s')$ then delete s.
2. If $N(t) \subseteq N(t')$ then delete t'.

Applications of these reduction rules cascade, preserving at each step enough information to obtain an optimal solution. Weihe found that, remarkably, these two simple reduction rules were strong enough to "digest" the original, huge input graph into a *problem kernel* consisting of disjoint components of size at most 50 — small enough to allow the problem to be solved optimally by brute force.

Note that we also have here a polynomial-time constant factor approximation algorithm, getting us a solution within a factor of 50 of optimal in $O(n^2)$ time by taking all the station vertices in the kernel components.

The surprising power of cascading data-reduction rules on real input distributions is why our deliverable (B) is probably, in the grand scheme of things, more important than deliverable (A).

3.2 Polynomial-Time Pre-processing Is Mathematically Nontrivial

Kernelization rules are frequently surprising in character, laborious to prove, and nontrivial to discover. Once found, they are small gems of *data reduction* that remain permanently in the heuristic design file for hard problems.

With a moment's thought the reader can easily observe a reduction rule for the VERTEX COVER problem that eliminates vertices of degree 1. It has to be regarded as quite surprising, however, that in polynomial time it is possible to kernelize, for the VERTEX COVER problem, to graphs of minimum degree 4, and "almost" to minimum degree 5!

The following is one of the reduction rules for VERTEX COVER that can be used to eliminate vertices of degree 3, taken from [DFS99]. Suppose G has a vertex x of degree 3 that has three mutually nonadjacent neighbors a, b, c. Then G can be simplified by: (1) deleting x, (2) adding edges from c to all the vertices in $N(a)$,

(3) adding edges from a to all the vertices in $N(b)$, (3) adding edges from b to all the vertices in $N(c)$, and (4) adding the edges ab and bc. The resulting (smaller) graph G' has a vertex cover of size k if and only if G has a vertex cover of size k. Moreover, an optimal or good approximate solution for G' lifts constructively to an optimal or good approximate solution for G. Note that this reduction rule is not symmetric!

Crown type reduction rules provide another recent example of highly nontrivial polynomial-time reduction rules that are of great practical value (see for example [ACFLSS04]) as well as being a significant development in terms of FPT theory. For examples of crown-type kernelizations see [DFRS04, FHRST04, PS04, Pr05]. (As an historical aside, crown reductions were first discovered in an attempt better to understand the monster machine methodology by trying it out on the classic FPT example of the VERTEX COVER problem [ACFLSS04, CFJ04].)

3.3 A general perspective on reduction rules

To date, reduction rules have been treated in the FPT literature in an *ad hoc* way. Because of their importance both theoretically and in practical terms, the subject deserves to be better organized. What exactly *is* a reduction rule? (What different kinds of reduction rules are there?) Can they be systematically computed or mechanically verified?

In keeping with the monster machine mission to articulate *the right way* to design FPT algorithms, we report here on a very general perspective that one can take on reduction rules that we term "*the oplus perspective*" based on the algebraic Myhill-Nerode machinery adapted to graph theory. (For basic references and exposition of this somewhat technical area see [FL89, AF93, MP94, BEF96, DF99, CDDFL00].)

First of all, we can make a few observations that point to a variety of mathematical phenomena in the matter of reduction rules.

• Some reduction rules that transform (G, k) to (G', k') depend for their definition on the specific value of k, and some do not. For example, the *degree 1 reduction rule* for VERTEX COVER, for which $k' = k - 1$ is defined on G in the same way regardless of the value of k. The *high degree reduction rule*, on the other hand, says to take into the solution any vertex of degree greater than k, and thus the effect on G is sensitive to the specific value of k.

• Some reduction rules, such as the *K-W Dissolver Rule* for MAX LEAF depicted in Figure 3, have a fixed "boundary size" (in this case: 2), whereas crown type reduction rules [ACFLSS04, CFJ04] do not have a fixed boundary size. (Crown decompositions of a specified boundary size are in fact $W[1]$ and NP-hard to compute! [Slo04].)

• The usual decision criterion for the soundness of a reduction rule is that (G, k) is a yes-instance if and only if (G', k') is a yes-instance. For the purposes of con-

structing polynomial-time approximation algorithms, or in using reduction rules in the context of the *iterative compression* strategy for FPT algorithm design (see [RSV04, DFRS04, DFLRS05, GGHNW05, Ma04] for recent examples of this) it is necessary that a k'-solution for G' can be constructively lifted to a k-solution for G in polynomial time.

Our Myhill-Nerode perspective does not capture all of these issues.

DEFINITION 26. A *t-boundaried graph*, is just a graph with t distinguished vertices labeled $1, ..., t$.

DEFINITION 27. If X and Y are t-boundaried graphs, then $X \oplus Y$ is defined to be the graph obtained from X and Y by identifying the boundaries. (For more detailed definitions see [DF99].

A key point of a mathematical machine in the sense of Grothendieck is to break things down into systematic series of small steps. The "oplus perspective" on reduction rules allows us to do this for the error-prone and tedious matter of proving correctness of reduction rules, and it also allows us to describe algorithms for computing "all" reduction rules (although we do not pursue this here). The table below records information that shows correctness for the "K-W Dissolver Rule" for MAX LEAF. (See Figure 3 of §2.1.) This reduction rule can be thought of a specifying the replacement of a boundaried subgraph (denoted A), of boundary size 2, by a smaller subgraph (denoted B) and adjusting the parameter to $k' = k - 1$. What is important is that for any 2-boundaried graph X and for any l, $X \oplus A$ has an l-leaf spanning tree if and only if $X \oplus B$ has a spanning tree with $l - 1$ leaves.

The table records, for each possible *boundary state* for a possible solution, the maximum number of leaves that can be arranged (respecting that state information) inside of A (respectively, B). Because both A and B are symmetric with respect to the boundary vertices, it is only necessary to inventory the possible boundary states up to symmetry. The boundary state notation "SR" denotes that the first boundary vertex is a non-leaf, and that the second boundary vertex is connected "on the right" to an internal vertex of the spanning tree. (The state notation "L" indicates similarly "on the left.") The asterisk indicates that the boundary vertices, which are specified to be internal vertices of the tree, are also to be connected in the tree by a path "on the right."

state	A leaves	B leaves
SR	1	0
SL	2	1
SS	2	1
SS*	1	0

Table 1. Data for the K-W Dissolver Rule of Figure 3.

4 Objective C: Gradients and Solution Transformations for Local Search

This deliverable of the structure theory associated to the parameterization is more speculative than the other objectives, but still seems worth exploring and may have significant potential.

We use MAX LEAF, with the structure theory of Boundary Lemma II to illustrate the idea.

The typical form that a local search heuristic for the MAX LEAF problem would take would have the following core elements:
(1) A *current solution* consisting of a spanning tree T for G would be maintained.
(2) The *value* of T would be defined as the number of leaves of T, as per the definition of the optimization form of the problem.

The local search routine would consist of repeated rounds of:
• For every possible way of removing r edges from T (where r is a small fixed number typically $r \leq 3$), and for every possible way of using r edges to link the fragments to form a new spanning tree T', see if the value of T' is greater than the value of T. If so, update the current solution to T'.

The search would continue until no r-change of the current solution T yields an improved solution T'.

The proof of Boundary Lemma II essentially employs a more refined notion of a representation of a solution (or maybe we should say *proto-solution*). That is, the proof works with a *witness structure* that is not a spanning tree of G with k leaves, but rather a (possibly non spanning) tree subgraph that has k leaves. It is a witness to the fact that (G, k) is a yes-instance because it can be extended to a spanning tree having at least k leaves. The first idea, then, is that local search might be conducted based on maintaining a "current witness structure" rather than a full solution (spanning tree).

Moreover, the improved kernelization outcome of Boundary Lemma II (over Boundary Lemma I, and with Boundary Lemma III being even stronger) is essentially due to the fact that the argument employs a more refined notion of "better solution," where "more leaves" is only the zeroth priority. The second idea is to use the list of inductive priorities to define a "better solution" gradient for the local search. Most structural claims (as in the proof of Boundary Lemma I) essentially describe situations where a solution (represented as a witness structure) can be transformed into a better one.

This generalization of the usual setup for local search is suggested by the mathematical power of the more complicated gradient in obtaining superior kernelization bounds. Whether this leads to something useful in local search heuristics seems worthy of exploration, but remains to be seen. It could be that the best practical results might come from using *some* of the priorities to refine the local search gradient, but not all of them, just as in practice *some* reduction rules appear to

worth implementing, and some seem introduce too much overhead actually to be worthwhile. In terms of the general lessons of "structure theory," this perhaps mirrors some phenomena of graph minors obstruction sets, where usually a few obstructions capture almost all of the information (e.g., there are two Kuratowski obstructions to planarity, but $K_{3,3}$ does almost all the work). For some discussion of such issues in FPT implementations, see [ACFLSS04, Alb02].

5 Objective D: Polynomial-Time Approximation Algorithms

Empirically, polynomial-time constant factor approximation and FPT seem to be closely associated. As in the case of FPT algorithm design, polynomial-time constant-factor approximation algorithms have come in a wide variety that also calls out for some general perspective and systematization. Some very different approaches have previously been taken to polynomial-time approximation for MAX LEAF [LR98, S-Ob98].

Our goal in this section is to describe *the right way* to design polynomial-time constant-factor approximation algorithms based on polynomial-time extremal structure theory. This connection is an important part of the monster machine program, and it plays out beautifully in the case of MAX LEAF.

A Polynomial-time Approximation Algorithm for MAX LEAF

Step 1. Reduce G using the kernelization rules. (It is easy to verify that the rules are approximation-preserving.)

Step 2. Take *any* tree T (not necessarily spanning) in G.

- If all of the structural claims hold, then by our arguments for Boundary Lemma III, the tree T must have at least n/c leaves, for $c = 3.75$. Therefore, lifting T back along the reduction path of Step 1, we obtain a c-approximation. (This bound can obviously be improved by a more careful argument.)

- If at least one of the structural claims does not hold, then the tree T can be improved against one of the inductive priorities. Notice that each structural claim is proved by an argument that can be interpreted as a polynomial-time routine that improves T, when the claim is contradicted.

An examination of the inductive priorities shows that T (and its successors) can be improved in Step 2 only a polynomial number of times (determined by the list of inductive priorities) until we arrive at a tree T' for which all of the various structural claims hold. At that point, we must have a tree that yields a c-approximate solution.

A careful analysis of the performance of this approximation algorithm would be similar in structure to the proof of Boundary Lemma III. In Boundary Lemma

III we are concerned with proving a bound on the "total population." To prove this bound, we set a Master Inequality, and sum over the structural pictures, one for each 1-tree of the witness structure. Here we would entertain an annotation of G, for example, indicating which vertices are leaves in an "optimal solution." (The complement of the annotated set must be a connected dominating set.) The task is to bound the population of annotated vertices.

6 Objective E: Structure to Exploit in the Ecology of Complexity

The point of view expressed in §1.2 leads to a vital but still relatively unexplored research program first suggested by Leizhen Cai [Cai01] in its proper generality. The program is to understand (in the sense of *FPT versus* $W[1]$-hard) how every input-governing problem-parameter affects the complexity of every other problem.

As a small example of this program for graph problems, we can make the following table. We use here the shorthand: TW is TREEWIDTH, BW is BANDWIDTH, VC is VERTEX COVER, DS is DOMINATING SET, G is GENUS and ML is MAX LEAF. The entry in the 2nd row and 4th column indicates that there is an *FPT* algorithm optimally to solve the DOMINATING SET problem for a graph G of bandwidth at most k. The entry in the 4th row and second column indicates that it is unknown whether BANDWIDTH can be solved optimally by an *FPT* algorithm when the parameter is a bound on the domination number of the input.

	TW	BW	VC	DS	G	ML
TW	*FPT*	$W[1]$-hard	*FPT*	*FPT*	?	*FPT*
BW	*FPT*	$W[1]$-hard	*FPT*	*FPT*	?	*FPT*
VC	*FPT*	?	*FPT*	*FPT*	?	*FPT*
DS	?	?	$W[1]$-hard	$W[1]$-hard	?	?
G	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	*FPT*	?
ML	*FPT*	?	*FPT*	*FPT*	*FPT*	*FPT*

Table 2. The Complexity Ecology of Parameters

Our attention so far has mostly been concerned with:
(1) The diagonal — TREEWIDTH is *FPT* and BANDWIDTH is $W[1]$-hard — as stand-alone problems, and
(2) The first row.
But if the natural world of complexity "runs" on a commerce of (sometimes rather hidden) structural parameters, then it is important to understand how different sources of parameterization interact in contributing to complexity.

The classification given in the above table is crude: FPT vs. $W[1]$-hard. When a problem is FPT we are naturally interested in the best possible algorithm, and

— the theme of this paper — the right way to pursue this objective. We next illustrate how the monster machine program applies to the last row of the table. In the interests of simplicity of exposition, we do not attempt to use the full power of the structure theory uncovered by Boundary Lemma III for MAX LEAF, but rather use only the structure given by the the proof of Boundary Lemma II.

THEOREM 28. *For graphs of max leaf number bounded by k, the minimum domination number can be computed in time $O^*(103^k)$ based on a polynomial-time reduction to a kernel of size at most $7k$.*

Proof. (*Sketch.*) Since this is an FPT result, we are necessarily (by Lemma 1) interested in polynomial-time extremal structure and effective kernelization for *this* problem, as prescribed by the monster machine. We must therefore develop a polynomial-time extremal account of the boundary — but what exactly *is* the boundary in this case?

We take the following hypotheses:
(1) (G, k) is a yes-instance of MAX LEAF.
(2) $(G, k + 1)$ is a no-instance of MAX LEAF.
(3) There is a witness structure for (1) that satisfies the inductive priorities of the proof of Boundary Lemma II for MAX LEAF.
(4) G is *reduced*.

Here, however, we are obliged to employ a completely different interpretation of *reduced* than that used in the proof of Boundary Lemma II. (The reason for this is that we have no idea what the reduction rules used there might do to the domination number — we need to use here reduction rules that are specific to our computational objective of computing the minimum domination number.)

The reduction rules shown in Figure 13 below can be used in this situation.

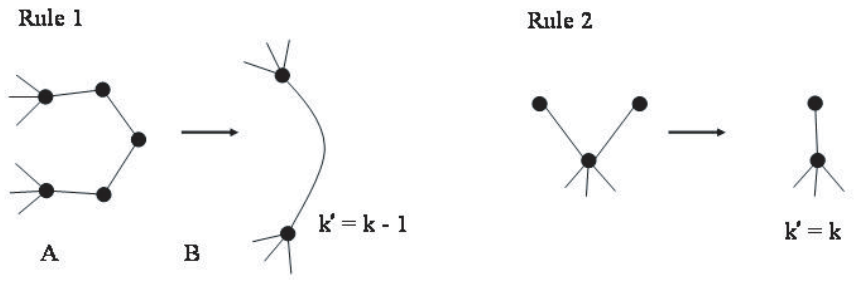

Figure 13. Reduction rules for minimum domination.

The table below displays the data that shows that reduction rule 1 of Figure 13

is sound. The entry $\emptyset$ denotes that the boundary state is "impossible." The "A-coercion" column indicates an *alternate state*. Suppose for a state σ the alternate state indicated is σ'. The meaning of the entry is that for any 2-boundaried graph X and for any d, $X \oplus A$ has a d-dominating set with a boundary state of σ if and only if $X \oplus A$ has a d-dominating set with boundary state of σ'. Here "S" denotes that the boundary vertex is in the dominating set, "R" denotes that it is not in the dominating set and is dominated from the left, and "L" denotes that it is not in the dominating set and is dominated from the right.

state	A-cost	A-coercion	B-cost
LL	1		0
LR	2	SL	$\emptyset$
RR	2	SR	$\emptyset$
SR	1		0
SL	1		0
SS	1		0

Table 3. Reduction rule 1 data for DOMINATING SET

Almost all of the structural claims in the proof of Boundary Lemma II carry over (with a few requiring slight modification). The reader can easily check that we obtain a problem kernel of size at most $7k$. The kernel can be analyzed by means of the algorithm due to Fomin, Kratsch and Woeginger [FKW04], yielding the running time stated for our algorithm. Knowing the domination number of the problem kernel allows us to compute the domination number of the input graph by retracing this information backwards along the kernelization path in polynomial time. ■

What was the best previous result for this problem? Implicit in some previous results would be a bound $g(k)$ on the pathwidth (or treewidth) of graphs that do not have a k-leaf tree subgraph. Using the structure theory of Boundary Lemma II we can show that a path decomposition of width at most $g(k) = 20k/3$ can be computed in polynomial time for graphs whose max leaf number is bounded by k. Combining this with the carefully engineered dynamic programming algorithm for DOMINATING SET in this setting of Telle and Proskurowsky [TP93] (refined by Alber and Niedermeier [AN02]) one would get a "best previous" running time of around $O^*(4^{20k/3})$ or $O^*(10322^k)$.

THEOREM 29. *For graphs of max leaf number bounded by k, the maximum size of an independent set can be computed in time $O^*(2.972^k)$ based on a polynomial-time reduction to a kernel of size at most $7k$.*

Proof. (*Sketch.*) The proof is quite similar to the previous theorem. The reduction

rules shown in Figure 14 below can be used, and again (coincidentally) obtain a bound on the kernel size of $7k$. Analysis of the kernel with Robson's algorithm [Rob86] yields a running time of $O^*(3.82^k)$. This can be improved, however, by closer attention to the structure of the kernel. Using the structure revealed in the proof of Boundary Lemma II, it is not hard to show that the kernel has a vertex cover of size at most $4.5k$. We can then use the FPT algorithm of Chen, Kanj and Xia [CKX05] for VERTEX COVER to obtain the claimed running time. ■

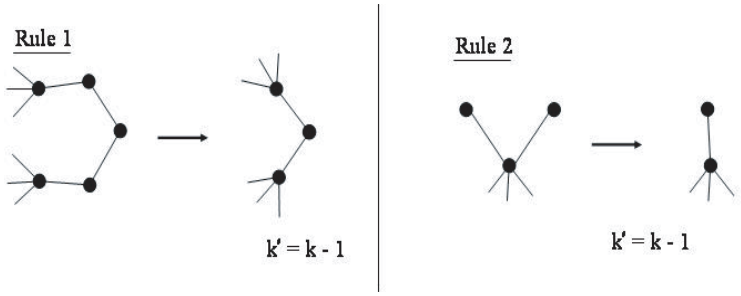

Figure 14. Reduction rules for maximum independent set.

REMARK 30. As a sequel to the above theorems, the reader might wish to consider, as an exercise, why the fifth entry in row six of Table 3 is "FPT."

The outcome in the above theorems seems quite good, and can no doubt be much further improved by more careful attention to the extremal structure theory of graphs of bounded max leaf number, and in particular by employing the deeper structure of Boundary Lemma III. In general, parameterized algorithmics should broaden its attention beyond the well-trodden "row" of bounded treewidth. It will be interesting to see how often other bounded parameter structures turn up in applications, once we start looking for them.

6.1 A Summary of the Recipe for Objective E

The monster machine recipe for objective E is a variation on the recipe for objective A:
(1) Determine the polarity of the boundary, and set up the boundary lemma.
(2) Choose a witness structure.
(3) Set inductive priorities.
(4) Develop a series of structural claims that describe the situation at the boundary.
Note. Much of (1-4) can be imported from the Boundary Lemma for objective A.

(5) Discover reduction rules that can act in polynomial time on relevant structural situations at the boundary, *where these reduction rules are relevant to the new computational objective*.
(6) As the structure at the boundary becomes clear, fill in the blank regarding the kernelization bound.

In the proof of Theorem 30, we noted that more careful attention to the boundary structure shows that the problem kernel admits a vertex cover of size at most $4.5k$. This can no doubt be improved. The task of obtaining this bound is formally very similar to the task of proving a performance bound for a polynomial-time approximation algorithm as discussed at the end of §5.

7 Summary

If the only purpose of this paper were to report on an improved FPT kernelization for the MAX LEAF problem from "something like" $4k$ [BBW03, Pr05] to $3.75k$ — you'd really have to wonder why anyone would bother, for all the effort involved. It would seem that "life is too short" for that!

Our main point has been programmatic: *every reasonable parameter (in the sense of parameterized complexity) leads to a structure theory project*. We have offered a retrospective interpretation of the graph minors project of Robertson and Seymour along these lines. One would like some further examples. Unfortunately, this source of inspiration suggests the possibility of a rather intensive effort.

One can point to some "historical periods" of FPT algorithm design. In the "Naive Era" various unsystematized and sometimes ad hoc approaches were employed. Frequently these either re-invented earlier results (for example, the $O^*(2^k)$ FPT algorithm for VERTEX COVER was actually first described in [Mehl84]), or even sometimes achieved results that were inferior to earlier work.

The field of FPT algorithm design is now firmly in the Second Era, where the subject pays better attention to pre-"parameterized complexity" results, and in particular, where for a number of problems, including VERTEX COVER, MAX LEAF and NONBLOCKER, the best current FPT algorithms employ deep, but off-the-shelf, extremal combinatorics results, such as the Kleitman-West result employed by Bonsma, Brueggemann and Woeginger for MAX LEAF [BBW03].

Our intention here is to help usher in the Third Era, of FPT as polynomial-time extremal structure theory — a distinctive form of extremal combinatorics that natively suits the purposes of computing. The difference is basically that here we are permitted *polynomial-time pre-processing* and are invited to figure out how make the most of that for the parameter at hand. In order to do this in a way that beats the results of the Second Era of FPT design, we basically have to "beat" pure extremal combinatorialists at their own game (except that we cheat a little bit, because of our license to do *any kind of polynomial-time preprocessing*). Since those people are pretty good at what they do, this again could make us uneasy

concerning the expected scope of the effort.

Here we have managed, in quite a long abstract, to sketch (only!) something of how to carry out the program for the one case study parameterized problem of MAX LEAF, necessarily omitting a huge amount of detail that will be reported in full elsewhere. We have sketched and surveyed something of five inter-related practical payoffs of the parameter-specific structure theory that emerges from such an effort. This is intended to be the first of 4 or 5 case studies that will form the core material of a monograph: *When Hard Problems Must Be Solved: FPT Algorithm Design for Applied Scientists and Engineers* [FR08].

Acknowledgments. We would like to thank Pablo Moscato for stimulating discussions concerning "gradient local search" and the possible uses of reduction rules in genetic algorithms.

BIBLIOGRAPHY

[ACFLSS04] F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters and C. T. Symons. Kernelization algorithms for the vertex cover problem: theory and experiments. *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, New Orleans, January, 2004, ACM/SIAM, *Proc. Applied Mathematics 115*, L. Arge, G. Italiano and R. Sedgewick, eds.

[AF93] K. Abrahamson and M. Fellows, "Finite Automata, Bounded Treewidth and Wellquasiordering," In: *Graph Structure Theory*, American Mathematical Society, Contemporary Mathematics Series, vol. 147 (1993), 539–564.

[AFFFNRS04] J. Alber, H. Fan, M. Fellows, H. Fernau, R. Niedermeier, F. Rosamond and U. Stege. Refined search tree technique for dominating sets on planar graphs. *Journal of Computer and System Sciences* 2004.

[Alb02] J. Alber. *Exact Algorithms for NP-Hard Problems on Networks: Design, Analysis and Implementation*. Dissertation, Universität Tübingen, 2002.

[AN02] J. Alber and R. Niedermeier. "Improved Tree Decomposition Based Algorithms for Domination-Like Problems," *Proceedings of the 5th Latin American Theoretical IN-formatics (LATIN 2002)*, Springer-Verlag LNCS 2286 (2002), 613–627.

[BBW03] P. Bonsma, T. Brueggemann and G. Woeginger. A faster FPT algorithm for finding spanning trees with many leaves. *Proceedings of MFCS 2003*, Springer-Verlag, *Lecture Notes in Computer Science* 2747 (2003), 259–268.

[BEF96] H. Bodlaender, P.Evans and M. Fellows. Finite-state computability of annotations of strings and trees. *Proceedings of the 7th Symposium on Combinatorial Pattern Matching (CPM'96)*, Springer-Verlag, *Lecture Notes in Computer Science* 1075 (1996), 384–391.

[Bod93] H.L. Bodlaender. On linear time minor tests and depth-first search. *Journal of Algorithms* 14 (1993), 1–23.

[Cai01] Leizhen Cai, Parameterized complexity of vertex colouring. *Discrete Applied Mathematics* 127 (2003), 415–429.

[CDDFL00] K. Cattell, M. Dinneen, R. Downey, M. Fellows and M. Langston. On computing graph minor obstruction sets. *Theoretical Computer Science A* 233 (2000), 107–127.

[CFJ04] B. Chor, M. Fellows and D. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. *Proceedings of WG 2004*, Springer-Verlag, *Lecture Notes in Computer Science* 3353 (2004), 257–269.

[CKX05] J. Chen, I. Kanj and G. Xia. Simplicity is beauty: improved upper bounds for vertex cover. Manuscript communicated by email, April, 2005.

[DF99] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[DFLRS05] F. Dehne, M. Fellows, M. Langston, F. Rosamond and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. *Proceedings COCOON 2004*, Springer-Verlag, *Lecture Notes in Computer Science* (2005), to appear.

[DFS99] R. Downey, M. Fellows and U. Stege. Parameterized complexity: a framework for systematically confronting computational intractability. In: *Contemporary Trends in Discrete Mathematics* (R. Graham, J. Kratochvil, J. Nesetril and F. Roberts, eds.), Proceedings of the DIMACS-DIMATIA Workshop on the Future of Discrete Mathematics, Prague, 1997, *AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 49 (1999), 49–99.

[DFRS04] F. Dehne, M. Fellows, F. 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. *Proceedings of the First International Workshop on Parameterized and Exact Computation*, Springer-Verlag, *Lecture Notes in Computer Science* vol. 3162 (2004), 271–280.

[Dow03] R. G. Downey. Parameterized complexity for the skeptic. *Proc. 18th IEEE Annual Conf. on Computational Complexity* (2003), 147–169.

[Fe01] M. Fellows. Parameterized complexity: main ideas, connections to heuristics and research frontiers. *Proceedings ISAAC 2001*, Springer-Verlag, *Lecture Notes in Computer Science* 2223 (2001), 291–307.

[Fe02] M. Fellows. Parameterized complexity: the main ideas and connections to practical computing. In: *Experimental Algorithmics*, Springer-Verlag, *Lecture Notes in Computer Science* 2547 (2002), 51–77.

[Fe03] M. Fellows. Blow-ups, win/win's and crown rules: some new directions in FPT. *Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003)*, Springer-Verlag, *Lecture Notes in Computer Science* 2880 (2003), 1–12.

[Fer05] H. Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationschrift, University of Tübingen, 2005.

[FFG01] J. Flum, M. Frick and M. Grohe. Query evaluation via tree-decompositions. *Proc. ICDT*, Springer-Verlag, *Lecture Notes in Computer Science* 1973 (2001), 22–32.

[FHRST04] M. Fellows, P. Heggernes, F. Rosamond, C. Sloper and J.A. Telle. Finding k disjoint triangles in an arbitrary graph. *Proceedings WG 2004*, Springer-Verlag, *Lecture Notes in Computer Science* 3353 (2004), 235–244.

[FKW04] F. Fomin, D. Kratsch and G. Woeginger. Exact (exponential) algorithms for the dominating set problem. *Proceedings of WG 2004*, Springer-Verlag, *Lecture Notes in Computer Science* 3353 (2004), 245–256.

[FL89] M. R. Fellows and M. A. Langston, "An Analogue of the Myhill-Nerode Theorem and its Use in Computing Finite-Basis Characterizations," *Proceedings of the IEEE Symposium on the Foundations of Computer Science* (1989), 520–525.

[FL92] M. Fellows and M. Langston. On well-partial-order theory and its applications to combinatorial problems of VLSI design. *SIAM Journal on Discrete Mathematics* 5 (1992), 117–126.

[FMRS00] M. Fellows, C. McCartin, F. Rosamond and U. Stege. Coordinatized kernels and catalytic reductions: an improved FPT algorithm for max leaf spanning tree and other problems. *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FST-TCS 2000)*, Springer, *Lecture notes in Theoretical Computer Science* 1974 (2000), 240–251.

[FR08] M. Fellows and F. Rosamond. *When Hard Problems Must Be Solved: FPT Algorithm Design for Applied Scientists and Engineers*. Manuscript in preparation.

[GGHNW05] J. Guo, J. Gramm, F. Hueffner, R. Niedermeier, S. Wernicke. Improved fixed-parameter algorithms for two feedback set problems. *Proceedings of WADS 2005*, Springer-Verlag, Lecture Notes in Computer Science (2005), to appear.

[GM99] M. Grohe and J. Marino. Definability and descriptive complexity on databases with bounded treewidth. *Proceedings of the 7th International Conference on Database Theory*, Springer-Verlag, *Lecture Notes in Computer Science* 1540 (1999), 70–82.

[Gr01] M. Grohe. The parameterized complexity of database queries. *Proc. PODS 2001*, ACM Press (2001), 82–92.

[Gur89] Y. Gurevich, "The Challenger-Solver Game: Variations on the Theme of *P*=?*NP*," *Bulletin EATCS* 39 (1989), 112–121.

[Jac04] A. Jackson. As if summoned from the void: the life of Alexandre Grothendieck (parts I,II). *Notices of the American Mathematical Society* 51 (9,10), 2004.

[KW91] D.J. Kleitman and D.B. West. Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics* 4 (1991), 99–106.

[LR98] H.-I. Lu and R. Ravi. Approximating maximum leaf spanning trees in almost linear time. *Journal of Algorithms* 29 (1998), 132–141.

[Ma04] D. Marx. Chordal deletion is fixed-parameter tractable. Manuscript, 2004.

[Mehl84] K. Mehlhorn. *Graph Algorithms and NP-Completeness*, Springer, 1984.

[MP94] S. Mahajan and J. G. Peters, "Regularity and Locality in k-Terminal Graphs," *Discrete Applied Mathematics* 54 (1994), 229–250.

[Nie02] R. Niedermeier. *Invitation to fixed-parameter algorithms*, Habilitationschrift, University of Tubingen, 2002. (Electronic file available from R. Niedermeier.)

[Nie04] R. Niedermeier. Ubiquitous parameterization — invitation to fixed-parameter algorithms. In: *Mathematical Foundations of Computer Science MFCS 2004*, Springer-Verlag, *Lecture Notes in Computer Science* 3153 (2004), 84–103.

[Nie05] R. Niedermeier. *Invitation to Fixed Parameter Algorithms.* Oxford University Press, forthcoming, 2005.

[Pr05] E. Prieto-Rodriguez. Systematic kernelization in FPT algorithm design. Ph.D. dissertation, School of Electrical Engineering and Computer Science, University of Newcastle, NSW, Australia, 2005.

[PS04] E. Prieto and C. Sloper. Looking at the stars. *Proceedings of the First International Workshop on Parameterized and Exact Computation*, Springer-Verlag, *Lecture Notes in Computer Science* vol. 3162 (2004), 138–149.

[Ra97] V. Raman, "Parameterized Complexity," in: *Proceedings of the 7th National Seminar on Theoretical Computer Science*, Chennai, India (1997), 1–18.

[Rob86] J.M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms* 7 (1986), 425–440.

[RS85] N. Robertson and P. Seymour. Graph minors: a survey. In: J. Anderson, ed., *Surveys in Combinatorics*, Cambridge University Press (1985), 153–171.

[RSV04] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters* 32 (2004), 299–301.

[S-Ob98] R. Solis-Oba. 2-approximation algorithm for finding a spanning tree with the maximum number of leaves. *Proceedings of the 6th Annual European Symposium on Algorithms (ESA'98)*, Springer, *Lecture Notes in Computer Science* 1461 (1998), 441–452.

[Slo04] Christian Sloper, University of Bergen, private communication, 2004.

[TP93] J.A. Telle and A. Proskurowski. "Practical Algorithms on Partial k-Trees with an Application to Domination-Like Problems." *Proceedings WADS'93 – The Third Workshop on Algorithms and Data Structures*, Springer-Verlag LNCS 709 (1993), 610–621.

[Wei98] K. Weihe, "Covering Trains by Stations, or the Power of Data Reduction," *Proc. ALEX'98* (1998), 1–8.

[Wei00] K. Weihe, "On the Differences Between 'Practical' and 'Applied' (invited paper)," *Proc. WAE 2000*, Springer-Verlag, *Lecture Notes in Computer Science* 1982 (2001), 1–10.

[Woe03] G. J. Woeginger. Exact algorithms for NP-hard problems: a survey. *Proceedings of 5th International Workshop on Combinatorial Optimization-Eureka, You Shrink! Papers dedicated to Jack Edmonds*, M. Junger, G. Reinelt, and G. Rinaldi (Festschrift Eds.) Springer-Verlag, *Lecture Notes in Computer Science* 2570 (2003), 184-207.

Vladimir Estivill-Castro
School of Computing and Information Technology
Griffith University
Brisbane QLD 4111
Australia
V.Estivill-Castro@cit.gu.edu.au

Michael R. Fellows
The Retreat for the Arts and Sciences
Newcastle NSW 2300
Australia
mfellows@cs.newcastle.edu.au

Michael A. Langston
Department of Computer Science
University of Tennessee
Knoxville, TN 37996-3450
USA
langston@cs.utk.edu

Frances A. Rosamond
The Retreat for the Arts and Sciences
Newcastle NSW 2300
Australia
fran@cs.newcastle.edu.au