Toward computer-assisted discovery and automated proofs of cutting plane theorems

Matthias Köppe and Yuan Zhou

Dept. of Mathematics, University of California, Davis
`mkoeppe@math.ucdavis.edu, yzh@math.ucdavis.edu`

Abstract. Using a metaprogramming technique and semialgebraic computations, we provide computer-based proofs for old and new cutting-plane theorems in Gomory–Johnson's model of cut generating functions.

1 Introduction

Inspired by the spectacular breakthroughs of the polyhedral method for combinatorial optimization in the 1980s, generations of researchers have studied the facet structure of convex hulls to develop strong cutting planes. It is a showcase of the power of experimental mathematics: Small examples are generated, their convex hulls are computed (for example, using the popular tool PORTA [9]), conjectures are formed, theorems are proved. Some proofs feature brilliant new ideas; other proofs are routine. Once the theorems have been found and proved, separation algorithms for the cutting planes are implemented. Numerical tests are run, the strength-versus-speed trade-off is investigated, parameters are tuned, papers are written.

In this paper, we ask how much of this process can be automated: In particular, *can we use algorithms to discover and prove theorems about cutting planes?* This paper is part of a larger project in which we aim to automate more stages of this pipeline. We focus on general integer and mixed integer programming, rather than combinatorial optimization, and use the framework of cut-generating functions [10], specifically those of the classic single-row Gomory–Johnson model [14,15]. Cut-generating functions are an attractive framework for our study for several reasons. First, it is essentially dimensionless: Cuts obtained from cut-generating functions can be applied to problems of arbitrary dimension. Second, it may be a way to the mythical multi-row cuts, sometimes dubbed the "holy grail of integer programming," though the computational approaches so far have disappointed. Third, work on new cuts in the single-row Gomory–Johnson model has, with few exceptions, become a routine, but error-prone task that leads to proofs of enormous complexity; see for example [24,25]. Fourth, finding new cuts in the multi-row Gomory–Johnson model has a daunting complexity, and few attempts at a systematic study have been made. Fifth, working on the Gomory–Johnson model is timely because only recently, after decades of theoretical investigations, the first computational tools for cut-generating functions in this model became available in [3] and the software implementation [20].

Of course, automated theorem proving is not a new proposition. Probably the best known examples in the optimization community are the proof of the Four Color Theorem, by Appel–Haken [1], and more recently and most spectacularly the proof of the Kepler Conjecture by Hales [18] and again within Hales' Flyspeck project in [19]. In the domains of combinatorics, number theory, and plane geometry, Zeilberger with long-term collaborator Shalosh B. Ekhad have pioneered automated discovery and proof of theorems; see, for example [13]. Many sophisticated automated theorem provers, by names such are HOL light, Coq, Isabelle, Mizar, etc. are available nowadays; see [28] and the references within for an interesting overview.

Our approach is pragmatic. Our theorems and proofs come from a metaprogramming trick, applied to the practical software implementation [20] of computations with the Gomory–Johnson model; followed by computations with semialgebraic cell complexes. As such, all of our techniques are reasonably close to mathematical programming practice. The correctness of all of our proofs depends on the correctness of the underlying implementation. We make no claims that our proofs can be formalized in the sense of the above mentioned formal proof systems that break every theorem down to the axioms of mathematics; in fact, we make no attempt to even use an automated theorem proving system.

Our software is in an early, proof-of-concept stage of development. In this largely computational and experimental paper we report on the early successes of the software. We computationally verify the results on the `gj_forward_3_slope`[1] and `drlm_backward_3_slope` functions. We find a correction to a theorem by Chen [8] regarding the extremality conditions for his `chen_4_slope` family.[2] We find a correction to a result by Miller, Li and Richard [24] on the so-called $\mathrm{CPL}_3^{=}$ functions (`mlr_cpl3_...`). We discover several new parametric families, `kzh_3_slope_param_extreme_1` and `kzh_3_slope_param_extreme_2`, of extreme functions and corresponding theorems regarding their extremality, with automatic proofs.

These new theorems are entirely unremarkable; the plan for the future is to make up for it by sheer quantity.[3]

---

[1] A function name shown in typewriter font is the name of the constructor of this function in the Electronic Compendium, part of the SageMath program [20]. In an online copy of this paper, there are hyperlinks that lead to a search for this function in the GitHub repository.

[2] This is a new result, which should not be confused with our previous result in [22] regarding Chen's family of 3-slope functions (`chen_3_slope_not_extreme`).

[3] With this anticlimactic sentence, the introduction ends. We leave it to the reader to speculate about the possible computational implications of having a large library of families of cut-generating functions available. For example, could the diversity of the cuts offered by such a library, and their rich parametrization, become crucial to the success of a branch-and-cut algorithm auto-tuned by machine learning?

2 The Gomory–Johnson model

We restrict ourselves to the single-row (or, "one-dimensional") infinite group problem, which has attracted most of the attention in the past and for which the software [20] is available. It can be written as

$$
\sum_{r \in \mathbb{R}} r\, y(r) \equiv f \pmod 1,
$$

$$
y \colon \mathbb{R} \to \mathbb{Z}_+ \text{ is a function of finite support,}
$$

(1)

where f is a given element of $\mathbb{R} \setminus \mathbb{Z}$. We study the convex hull $R_f(\mathbb{R}, \mathbb{Z})$ of the set of all functions $y \colon \mathbb{R} \to \mathbb{Z}_+$ satisfying the constraints in (1). The elements of the convex hull are understood as functions $y \colon \mathbb{R} \to \mathbb{R}_+$.

After a normalization, valid inequalities for the convex set $R_f(\mathbb{R}, \mathbb{Z})$ can be described using so-called *valid functions* $\pi \colon \mathbb{R} \to \mathbb{R}$ via $\langle \pi, y \rangle := \sum_{r \in \mathbb{R}} \pi(r) y(r) \geq 1$. Valid functions π are cut-generating functions for pure integer programs. Take a row of the optimal simplex tableau of an integer program, corresponding to a basic variable x_i that currently takes a fractional value:

$$
x_i = -f_i + \sum_{j \in N} r_j x_j, \quad x_i \in \mathbb{Z}_+, \ \mathbf{x}_N \in \mathbb{Z}_+^N.
$$

Then a valid function π for $R_{f_i}(\mathbb{R}, \mathbb{Z})$ gives a valid inequality $\sum_{j \in N} \pi(r_j) x_j \geq 1$ for the integer program. (By a theorem of Johnson [21], this extends easily to the mixed integer case: A function ψ can be associated to π, so that they together form a *cut-generating function pair* (ψ, π), which gives the coefficients of the continuous and of the integer variables.)

In the finite-dimensional case, instead of merely valid inequalities, one is interested in stronger inequalities such as tight valid inequalities and facet-defining inequalities. These rôles are taken in our infinite-dimensional setting by *minimal functions* and *extreme functions*. Minimal functions are those valid functions that are pointwise minimal; extreme functions are those that are not a proper convex combination of other valid functions.

By a theorem of Gomory and Johnson [14], minimal functions for $R_f(\mathbb{R}, \mathbb{Z})$ are classified: They are the subadditive functions $\pi \colon \mathbb{R} \to \mathbb{R}_+$ that are periodic modulo 1 and satisfy the *symmetry condition* $\pi(x) + \pi(f - x) = 1$ for all $x \in \mathbb{R}$.

Obtaining a full classification of the *extreme* functions has proved to be elusive, however various authors have defined parametric families of extreme functions and provided extremality proofs for these families. These parametric families of extreme functions from the literature, as well as "sporadic" extreme functions, have been collected in an electronic compendium as a part of the software [20]; see [22].

We refer the interested reader to the recent surveys [11,4,5] for a more detailed exposition.

3 Examples of cutting-plane theorems in the Gomory–Johnson model

To illustrate what cutting-plane theorems in the Gomory–Johnson model look like, we give three examples of such theorems, paraphrased for precision from the literature where they were stated. As it turns out, the last theorem is incorrect.
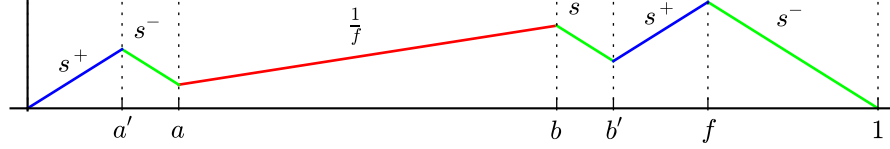

Fig. 1. `gj_forward_3_slope`

Theorem 3.1 (reworded from Gomory–Johnson [16, Theorem 8]). *Let $f \in (0,1)$ and $\lambda_1, \lambda_2 \in \mathbb{R}$. Define the periodic, piecewise linear* `gj_forward_3_slope` *function $\pi \colon \mathbb{R}/\mathbb{Z} \to \mathbb{R}$ as follows. The function π satisfies $\pi(0) = \pi(1) = 0$; it has 6 pieces between 0 and 1 with breakpoints at $0, a', a, b, b', f$ and 1, where $a = \frac{\lambda_1 f}{2}$, $a' = a + \frac{\lambda_2(f-1)}{2}$, $b = f - a$ and $b' = f - a'$. The slope values of π on these pieces are $s^+, s^-, \frac{1}{f}, s^-, s^+$ and s^-, repectively, where $s^+ = \frac{\lambda_1 + \lambda_2}{\lambda_1 f + \lambda_2(f-1)}$ and $s^- = \frac{1}{f-1}$. If the parameters λ_1 and λ_2 satisfy that (i) $0 \le \lambda_1 \le \frac{1}{2}$, (ii) $0 \le \lambda_2 \le 1$ and (iii) $0 < \lambda_1 f + \lambda_2(f-1)$, then the function π is an extreme function for $R_f(\mathbb{R}/\mathbb{Z})$.*
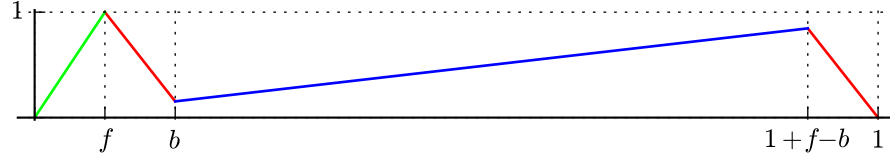

Fig. 2. `drlm_backward_3_slope`

Theorem 3.2 (Dey–Richard–Li–Miller [12]; in this form, for the real case, in [22, Theorem 4.1]). *Let f and b be real numbers such that $0 < f < b \le \frac{1+f}{4}$. The periodic, piecewise linear* `drlm_backward_3_slope` *function $\pi \colon \mathbb{R}/\mathbb{Z} \to \mathbb{R}$ defined as follows is an extreme function for $R_f(\mathbb{R}/\mathbb{Z})$:*

$$\pi(x) = \begin{cases} \frac{x}{f} & \text{if } 0 \le x \le f \\ 1 + \frac{(1+f-b)(x-f)}{(1+f)(f-b)} & \text{if } f \le x \le b \\ \frac{x}{1+f} & \text{if } b \le x \le 1+f-b \\ \frac{(1+f-b)(x-1)}{(1+f)(f-b)} & \text{if } 1+f-b \le x \le 1 \end{cases}$$
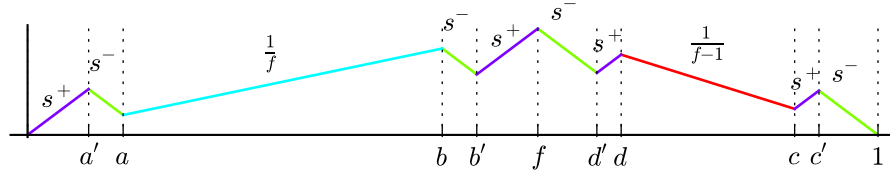
Fig. 3. chen_4_slope

Theorem 3.3 (reworded from Chen [8, Theorem 2.2.1]). *Let $f \in (0, 1)$, $s^+ > 0, s^- < 0$ and $\lambda_1, \lambda_2 \in \mathbb{R}$. Define the periodic, piecewise linear* chen_4_slope *function $\pi \colon \mathbb{R}/\mathbb{Z} \to \mathbb{R}$ as follows. The function π satisfies $\pi(0) = \pi(1) = 0$; it has 10 pieces between 0 and 1 with breakpoints at $0, a', a, b, b', f, d', d, c, c', 1$, where*

$$a' = \frac{\lambda_1(1 - s^- f)}{2(s^+ - s^-)}, \quad a = \frac{\lambda_1 f}{2}, \quad c = 1 - \frac{\lambda_2(1 - f)}{2}, \quad c' = 1 - \frac{\lambda_2(1 - s^+(1 - f))}{2(s^+ - s^-)}$$

and $b = f - a$, $b' = f - a'$, $d = 1 + f - c$, $d' = 1 + f - c'$. The slope values of π on these pieces are $s^+, s^-, \frac{1}{f}, s^-, s^+, s^-, s^+, \frac{1}{f-1}, s^+$ and s^-, repectively. If the parameters $f, \lambda_1, \lambda_2, s^+$ and s^- satisfy that

$$f \geq \frac{1}{2}, \quad s^+ \geq \frac{1}{f}, \quad s^- \leq \frac{1}{f - 1}, \quad 0 \leq \lambda_1 < \min\{\frac{1}{2}, \frac{s^+ - s^-}{s^+(1 - s^- f)}\}, \quad and$$

$$f - \frac{1}{s^+} < \lambda_2 < \min\left\{\frac{1}{2}, \frac{s^+ - s^-}{s^-(s^+(f - 1) - 1)}\right\},$$

then π is an extreme function for $R_f(\mathbb{R}, \mathbb{Z})$.

Observation 3.4   *(i) These theorems are about families of periodic, continuous piecewise linear functions $\pi \colon \mathbb{R} \to \mathbb{R}$ that depend on a finite number of real parameters in a way that breakpoints and slope values can be written as rational functions of the parameters.*

 (ii) There are natural conditions on the parameters to make the function even constructible; for example, in Theorem 3.2, if $f < b$ is violated, then the function is not well-defined. These conditions are inequalities of rational functions of the parameters. Hence the set of parameter tuples such that the construction describes a function is a semialgebraic set.

(iii) There are additional conditions on the parameters that ensure that the function is an extreme function. Again, all of these conditions are inequalities of rational functions of the parameters. Hence the set of parameter tuples such that the construction gives an extreme function is a semialgebraic set.

Remark 3.5. Some families of extreme functions in the literature are defined in more general ways. Some use parameters that are integers (for example, drlm_2_slope_limit has integer parameters that control the number of pieces of the function). Others use non-algebraic operations such as the floor/ceiling/fractional

part operations to define the breakpoints and slope values of the function (for example, `dg_2_step_mir`). Another family, `bhk_irrational`, requires an arithmetic condition, the $\mathbb{Q}$-linear independence of certain parameters, for extremality. These families are beyond the scope of this paper.

4 Semialgebraic cell structure of extremality proofs

The minimality of a given periodic piecewise linear function can be easily tested algorithmically; see, for example, [4, Theorem 3.11]. Basu, Hildebrand, and Köppe [3] gave the first algorithmic tests for extremality for a given function π whose breakpoints are rational with a common denominator q. The simplest of these tests uses their finite-oversampling theorem (see [5, Theorem 8.6] for its strongest form). Extremality of the function π is equivalent to the extremality of its restriction to the refined grid $\frac{1}{3q}\mathbb{Z}/\mathbb{Z}$ for the finite master group problem. Thus it can be tested by finite-dimensional linear algebra.

The proof of the finite-oversampling theorem in [3] (see also [5, section 7.1] for a more high-level exposition) provides another algorithm, based on the computation of "affine-imposing" ("covered") intervals and the construction of "equivariant" perturbation functions. This algorithm in [3] is also tied to the use of the grid $\frac{1}{q}\mathbb{Z}/\mathbb{Z}$; but it has since been generalized in the practical implementation [20] to give a completely *grid-free algorithm*, which is suitable also for rational breakpoints with huge denominators and for irrational breakpoints.[4]

Observation 4.1 *On close inspection of this grid-free algorithm, we see that it only uses algebraic operations, comparisons, and branches (*if-then-else *statements and loops), and then returns either* True *(to indicate extremality) or* False *(non-extremality).*

Enter *parametric analysis* of the algorithm, that is, we wish to run the algorithm for a function from a parametric family and observe how the run of the algorithm and its answer changes, depending on the parameters. It is then a simple observation that for any algorithm of the type described in Observation 4.1, the set of parameters where the algorithm returns *True* must be a union of sets described by equations and inequalities of rational functions in the parameters. If the number of operations (and thus the number of branches) that the algorithm takes is bounded finitely, then it will be a finite union of "cells", each corresponding to a particular outcome of comparisons that led to branches, and each described by finitely many equations and inequalities of rational functions in the parameters. Thus it will be a semialgebraic set.

Within each of the cells, we get the "same" proof of extremality. A complete proof of extremality for a parametric family is merely a collection of cells, with one proof for each of them. This is what we compute as we describe below.

---

[4] The finiteness proof of the algorithm, however, does depend on the rationality of the data. In this paper we shall ignore the case of functions with non-covered intervals and irrational breakpoints, such as the `bhk_irrational` family, which necessitates testing the $\mathbb{Q}$-linear independence of certain parameters.

5 Computing one proof cell by metaprogramming

Now we describe how we compute one cell of the proof. We assume that we are given a tuple of *concrete* parameter values; we will compute a semialgebraic description of a cell of parameter tuples for which the algorithm takes the same branches.

It is well known that modern programming languages and systems provide facilities known as "operator overloading" or "virtual methods" that allow us to conveniently write "generic" programs that can be run on objects of various types. For example, the program [20], which is written in the SageMath system [26], by default works with (arbitrary-precision) rational numbers; but when parameters are irrational algebraic numbers, it constructs a suitable number field, embedded into the real numbers, and makes exact computations with the elements of this number field.[5]

We make use of the same facilities for a metaprogramming technique that transforms the program [20] for testing extremality for a function corresponding to a given parameter tuple into a program that computes a description of the cell that contains the given parameter tuple. No code changes are necessary.

We define a class of elements[6] that support the algebraic operations and comparisons that our algorithm uses, essentially the operations of an ordered

---

[5] The details of this are not relevant to the present paper, but one reader of a version of this paper was intrigued by this sentence, so we elaborate. These number fields are algebraic field extensions (of some degree d) of the field $\mathbb{Q}$ of rational numbers, in much the same way that the field $\mathbb{C}$ of complex numbers is an algebraic field extension (of degree $d = 2$) of the field $\mathbb{R}$ of real numbers. Elements of the field are represented as a coordinate vector of dimension d over the base field, and all arithmetic computations are done by manipulating these vectors. The computational overhead, compared to arithmetic over $\mathbb{Q}$, is negligible for quadratic field extensions thanks to a specialized implementation in SageMath, and within a factor of 10 at least for field extensions of degree $d \leq 60$, as tested by `%timeit extremality_test(copy(h))` after the initial setup `h=gmic(2^(1/d)/2)`. The computation time for the construction of the field, part of the initial setup, does grow more rapidly with the degree, from milliseconds for small d to seconds for $d = 60$.

The number fields can be considered either abstractly or as embedded subfields of an enclosing field. When we say that the number fields are embedded into the enclosing field of real numbers, this means in particular that they inherit the linear order from the real numbers. To decide whether $a < b$, one computes sufficiently many digits of both numbers using a rigorous version of Newton's method; this is guaranteed to terminate because the $a = b$ test can be decided by just comparing the coordinate vectors. The program [20] includes a function `nice_field_values` that provides convenient access to the standard facilities of SageMath that construct such an embedded number field.

[6] In the category-based object system of SageMath, which extends Python's standard inheritance-based object system, these elements are instances of the class `ParametricRealFieldElement`. Their `parent`, representing the field in which the element lies, is an instance of the class `ParametricRealField`.

field. Each element stores (1) a symbolic expression[7] of the parameters in the problem, for example $x + y$ and (2) a concrete value, which is the evaluation of this expression on the given parameter tuple, for example 13. In the following, we denote elements in the form $x + y \mid_{=13}$. Every algebraic operation $(+, -, *, \ldots)$ on the elements of the class is performed both on the symbolic expressions and on the concrete values. For example, if one multiplies the element $x \mid_{=7}$ and another element $x + y \mid_{=13}$, one gets the element $x^2 + xy \mid_{=91}$.

When a comparison $(<, \leq, =, \ldots)$ takes place on elements of the class, their concrete values are compared to compute the Boolean return value of the comparison. For example, the comparison $x^2 + xy \mid_{=91} > 42$ evaluates to *True*. But we now have a constraint on the parameters x and y: The inequality $x^2 + xy > 42$ needs to hold so that our answer *True* is correct. We record this constraint.[8]

After a run of the algorithm, we have a description of the parameter region for which all the comparisons would give the same truth values as they did for the concrete parameter tuple, and hence the algorithm would take the same branches. This description is typically a very long and highly redundant list of inequalities of rational functions in the parameters.

It is crucial to simplify the description. "In theory", manipulation of inequalities describing semialgebraic sets is a solved problem of effective real algebraic geometry. Normal forms such as Cylindrical Algebraic Decomposition (CAD) [6, Chapters 5, 11] are available in various implementations, such as in the standalone QEPCAD B [7] or those integrated into CAS software such as Maple and Mathematica, underlying these systems' 'solve' and 'assume' facilities. In computational practice, we however observed that these systems are extremely sensitive to the number of inequalities, rendering them unsuitable for our purposes; see [27, section 5] for a study with Maple. We therefore roll our own implementation.

1. Transform inequalities and equations of rational functions into those of polynomials by multiplying by denominators, and bring them in the normal form $p(x) < 0$ or $p(x) = 0$. In the case of inequalities, this creates the extra constraint that the denominator takes the same sign as it does on the test point. So this transformation may break cells into smaller cells.
2. Factor the polynomials $p(x)$ and record the distinct factors as equations and inequalities. In the case of inequalities, this potentially breaks cells into smaller cells. We can ignore the factors with even exponents in inequalities.
3. Reformulation–linearization: Expand the polynomial factors in the standard monomial basis and replace each monomial by a new variable. This gives a linear system of inequalities and equations and thus a not-necessarily-closed polyhedron in an extended variable space. We use this polyhedron to represent our cell. Indeed, its intersection with the algebraic variety of monomial relations is in linear bijection with the semialgebraic cell.

---

[7] Since all expressions are, in fact, rational functions, we use exact seminumerical computations in the quotient field of a multivariate polynomial ring, instead of the slower and less robust general symbolic computation facility.

[8] This information is recorded in the `parent` of the elements.

4. All of this is implemented in an incremental way. We use the excellent Parma Polyhedra Library [2] via its SageMath interface written in Cython. PPL is based on the double description method and supports not-necessarily-closed polyhedra. It also efficiently supports adding inequalities dynamically and injecting a polyhedron into a higher space. The latter becomes necessary when a new monomial appears in some constraint. The PPL also has a fast implementation path for discarding redundant inequalities.

In our preliminary implementation, we forgo opportunities for strengthening this extended reformulation by McCormick inequalities, bounds propagation etc., which would allow for further simplification. We remark that all of these polyhedral techniques ultimately should be regarded as a preprocessing of input for proper real-algebraic computation. They are not strong enough on their own to provide "minimal descriptions" for semialgebraic cells. In a future version of our software, we will combine our preprocessing technique with the CAD implementation in Mathematica.

6 Computing the cell complex using wall-crossing BFS

Define the graph of the cell complex by introducing a node for each cell and an edge if a cell is obtained from another cell by flipping one inequality. We compute the cell complex by doing a breadth-first search (BFS) in this graph. This is a well-known method for the case of the cells of arrangements of hyperplanes; see [17, chapter 24] and the references within. The nonlinear case poses challenges due to degeneracy and possible singularities, which we have not completely resolved. Our preliminary implementation uses a heuristic numerical method to construct a point in the interior of a neighbor cell, which will be used as the next concrete parameter tuple for re-running the algorithm described in section 5. This may fail, and so we have no guarantees that the entire parameter space is covered by cells when the breadth-first search terminates. This is the weakest part of our current implementation.

7 Automated proofs and corrections of old theorems

Using our implementation, we verified Theorems 3.1 and 3.2, as well as other theorems regarding classical extreme functions from the literature. Figure 4 shows the visualizations of the corresponding cell complexes; for additional illustrations see Appendix A.1. Using our implementation we also investigated Theorem 3.3 regarding `chen_4_slope` and discovered that it is incorrect. (See Appendix A.2 for an illustration.) For example, the function with parameters $f = 7/10$, $s^+ = 2$, $s^- = -4$, $\lambda_1 = 1/100$, $\lambda_2 = 49/100$ satisfies the hypotheses of the theorem; however, it is not subadditive and thus not an extreme function. On the other hand, the stated hypotheses are also not necessary for extremality. For example, the function with parameters $f = 7/10$, $s^+ = 2$, $s^- = -4$, $\lambda_1 = 1/10$, $\lambda_2 = 1/10$
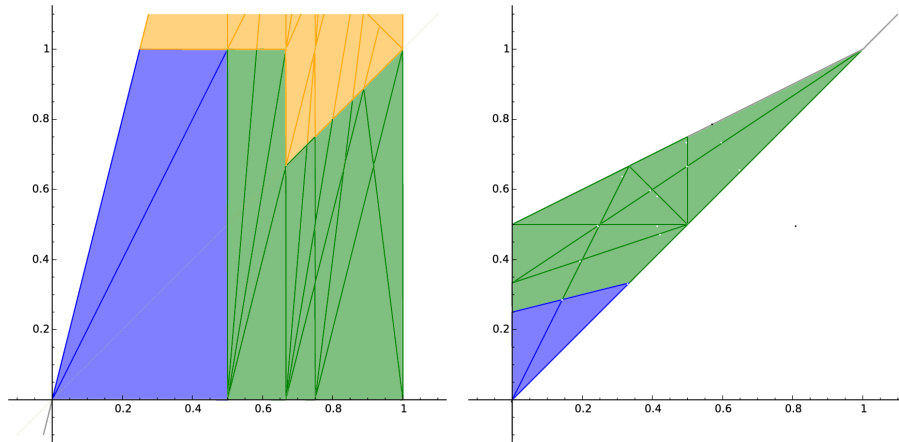
Fig. 4. The cell complexes of two parametric families of functions. *Left*, `gj_forward_3_slope`, showing the plane of parameters (λ_1, λ_2) for fixed $f = 4/5$. *Right*, `drlm_backward_3_slope`, showing the parameters $(f, bkpt)$. Cells are color-coded: 'not constructible' (*white*), 'constructible, but not minimal' (*yellow*), 'minimal, but not extreme' (*green*) or 'extreme' (*blue*).

does not satisfy the hypotheses, however it is extreme. We omit a statement of corrected hypotheses that we found using our code.

We also investigated another family of functions, the so-called $\mathrm{CPL}_3^{=}$ functions, introduced by the systematic study by Miller, Li, and Richard [24]. Their method can be regarded as a predecessor of our method, albeit one that led to an error-prone manual case analysis (and human-generated proofs). Though our general method can be applied directly, we developed a specialized version of our code that follows Miller, Li, and Richard's method to allow a direct comparison. This revealed mistakes in [24], as we report in Appendix A.3.

8   Computer-assisted discovery of new theorems

In [23], the authors conducted a systematic computer-based search for extreme functions on the grids $\frac{1}{q}\mathbb{Z}$ for values of q up to 30. This resulted in a large catalog of extreme functions that are "sporadic" in the sense that they do not belong to any parametric family described in the literature.

Our goal is to automatically embed these functions into parametric families and to automatically prove theorems about their extremality. In this section, we report on cases that have been done successfully with our preliminary implementation; the process is not completely automatic yet.

We picked an interesting-looking 3-slope extreme function found by our computer-based search on the grid $\frac{1}{q}\mathbb{Z}$. We then introduced parameters f, a, b, v to describe a preliminary parametric family that we denote by `param_3_slope_1`. In the concrete function that we started from, these parameters take the values $\frac{6}{19}, \frac{1}{19}, \frac{5}{19}, \frac{8}{15}$; see Figure 5 (left). So a denotes the length of the first interval
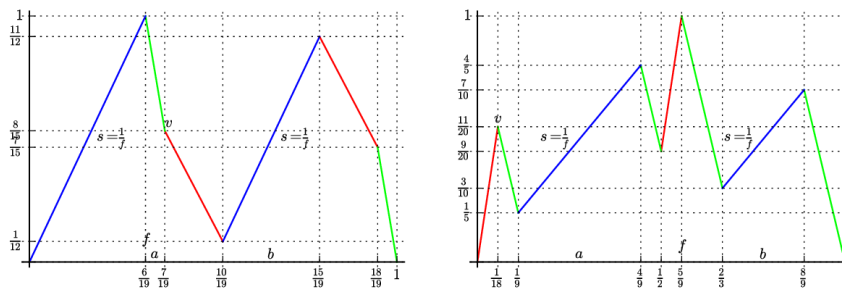
Fig. 5. Two new parametric families of extreme functions. *Left*, `kzh_3_slope_param_extreme_1`. *Right*, `kzh_3_slope_param_extreme_2`.

right to f, b denotes the length of interval centered at $(1+f)/2$ and $v = \pi(f+a)$. By this choice of parameters, the function automatically satisfies the equations corresponding to the symmetry conditions. Next we run the parametric version of the minimality test algorithm. It computes *True*, and as a side-effect computes a description of the cell in which the minimality test is the same.

```
sage: K.<f,a,b,v>=ParametricRealField([6/19,1/19,5/19,8/15])
sage: h = param_3_slope_1(f,a,b,v)
sage: minimality_test(h)
True
sage: K._eq_factor
{-f^2*v + 3*f*b*v + f^2 + f*a - 3*f*b - 3*a*b - f*v + b}
```

In particular, the above line shows that it has discovered one nonlinear equation that holds in the cell corresponding to the minimality proof of the concrete function. We use this equation, quadratic in f and multilinear in the other parameters, to eliminate one of the parameters.[9] This gives our parametric family `kzh_3_slope_param_extreme_1`, which depends only on parameters f, a, b. See Appendix A.4 to see how this family is defined in the code; the definition of the parametric family is the only input to our algorithm. Cells with respect to this family will be full-dimensional; this helps to satisfy a current implementation restriction of our software. Indeed, re-running the algorithm yields the following simplified description of the cell in which the concrete parameter tuple lies.

```
3*f + 4*a - b - 1 < 0                     -a < 0
-f^2 - f*a + 3*f*b + 3*a*b - b < 0        -f + b < 0
f*a - 3*a*b - f + b < 0                   -f - 3*b + 1 < 0
-f^2*a + 3*f*a*b - 3*a*b - f + b < 0
```

We then compute the cell complex by BFS as described in section 6; see Appendix A.5 for an illustration. By inspection, we observe that the collection of the cells for which the function is extreme happens to be a convex polytope (this is not guaranteed). We discard the inequalities that appear twice and thus

---

[9] We plan to automate this in a future version of our software.

describe inner walls of the complex. By inspection, we discard nonlinear inequalities that are redundant. We obtain a description of the union of the cells for which the function is extreme as a convex polytope. We obtain the following:

Theorem 8.1. *Let $f \in (0,1)$ and $a, b \in \mathbb{R}$ such that*

$$0 \le a, \quad 0 \le b \le f \text{ and } 3f + 4a - b - 1 \le 0.$$

The periodic, piecewise linear `kzh_3_slope_param_extreme_1` *function* $\pi \colon \mathbb{R}/\mathbb{Z} \to \mathbb{R}$ *defined as follows is extreme. The function π has breakpoints at*

$$0, f, f+a, \frac{1+f-b}{2}, \frac{1+f+b}{2}, 1-a, 1.$$

The values at breakpoints are given by $\pi(0) = \pi(1) = 0$, $\pi(f+a) = 1 - \pi(1-a) = v$ and $\pi(\frac{1+f-b}{2}) = 1 - \pi(\frac{1+f+b}{2}) = \frac{f-b}{2f}$, where $v = \frac{f^2 + fa - 3fb - 3ab + b}{f^2 + f - 3bf}$.

A similar process leads to a theorem about the family `kzh_3_slope_param_extreme_2` shown in Figure 5 (right). We omit the statement of the theorem. Appendix A.6 shows a visualization of the cell complex.

References

1. K. Appel and W. Haken, *Every planar map is four colorable. Part I: Discharging*, Illinois J. Math. **21** (1977), no. 3, 429–490, available from `http://projecteuclid.org/euclid.ijm/1256049011`.
2. R. Bagnara, P. M. Hill, and E. Zaffanella, *The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems*, Science of Computer Programming **72** (2008), no. 1–2, 3–21, Special Issue on Second issue of experimental software and toolkits (EST), `doi:http://dx.doi.org/10.1016/j.scico.2007.08.001`.
3. A. Basu, R. Hildebrand, and M. Köppe, *Equivariant perturbation in Gomory and Johnson's infinite group problem. I. The one-dimensional case*, Mathematics of Operations Research **40** (2014), no. 1, 105–129, `doi:10.1287/moor.2014.0660`.
4. _____, *Light on the infinite group relaxation I: foundations and taxonomy*, 4OR (2015), 1–40, `doi:10.1007/s10288-015-0292-9`.
5. _____, *Light on the infinite group relaxation II: sufficient conditions for extremality, sequences, and algorithms*, 4OR (2015), 1–25, `doi:10.1007/s10288-015-0293-8`.
6. S. Basu, R. Pollack, and M.-F. Roy, *Algorithms in real algebraic geometry*, second ed., Algorithms and Computation in Mathematics, vol. 10, Springer-Verlag, Berlin, 2006, MR 2248869 (2007b:14125), ISBN 978-3-540-33098-1; 3-540-33098-4.
7. C. W. Brown, *QEPCAD B: A program for computing with semi-algebraic sets using CADs*, SIGSAM Bull. **37** (2003), no. 4, 97–108, `doi:10.1145/968708.968710`.
8. K. Chen, *Topics in group methods for integer programming*, Ph.D. thesis, Georgia Institute of Technology, June 2011.
9. T. Christof and A. Löbel, *Porta: Polyhedron representation transformation algorithm*, `http://comopt.ifi.uni-heidelberg.de/software/PORTA/`.

10. M. Conforti, G. Cornuéjols, A. Daniilidis, C. Lemaréchal, and J. Malick, *Cut-generating functions and S-free sets*, Mathematics of Operations Research **40** (2013), no. 2, 253–275, `http://dx.doi.org/10.1287/moor.2014.0670`.

11. M. Conforti, G. Cornuéjols, and G. Zambelli, *Corner polyhedra and intersection cuts*, Surveys in Operations Research and Management Science **16** (2011), 105–120.

12. S. S. Dey, J.-P. P. Richard, Y. Li, and L. A. Miller, *On the extreme inequalities of infinite group problems*, Mathematical Programming **121** (2010), no. 1, 145–170, `doi:10.1007/s10107-008-0229-6`.

13. S. B. Ekhad, XIV and D. Zeilberger, *Plane geometry: An elementary textbook*, 2050, available from `http://www.math.rutgers.edu/~zeilberg/GT.html`.

14. R. E. Gomory and E. L. Johnson, *Some continuous functions related to corner polyhedra, I*, Mathematical Programming **3** (1972), 23–85, `doi:10.1007/BF01584976`.

15. _____, *Some continuous functions related to corner polyhedra, II*, Mathematical Programming **3** (1972), 359–389, `doi:10.1007/BF01585008`.

16. _____, *T-space and cutting planes*, Mathematical Programming **96** (2003), 341–375, `doi:10.1007/s10107-003-0389-3`.

17. J. E. Goodman and J. O'Rourke (eds.), *Handbook of discrete and computational geometry*, CRC Press, Inc., Boca Raton, FL, USA, 1997, ISBN 0-8493-8524-5.

18. T. C. Hales, *A proof of the Kepler conjecture*, Ann. of Math. (2) **162** (2005), no. 3, 1065–1185, `doi:10.4007/annals.2005.162.1065`, MR 2179728 (2006g:52029).

19. T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, et al., *A formal proof of the Kepler conjecture*, arXiv preprint arXiv:1501.02155 (2015).

20. C. Y. Hong, M. Köppe, and Y. Zhou, *Sage program for computation and experimentation with the 1-dimensional Gomory–Johnson infinite group problem*, 2014, available from `https://github.com/mkoeppe/infinite-group-relaxation-code`.

21. E. L. Johnson, *On the group problem for mixed integer programming*, Mathematical Programming Study **2** (1974), 137–179.

22. M. Köppe and Y. Zhou, *An electronic compendium of extreme functions for the Gomory–Johnson infinite group problem*, Operations Research Letters **43** (2015), no. 4, 438–444, `doi:10.1016/j.orl.2015.06.004`.

23. _____, *New computer-based search strategies for extreme functions of the Gomory–Johnson infinite group problem*, eprint arXiv:1506.00017 [math.OC], 2015.

24. L. A. Miller, Y. Li, and J.-P. P. Richard, *New inequalities for finite and infinite group problems from approximate lifting*, Naval Research Logistics (NRL) **55** (2008), no. 2, 172–191, `doi:10.1002/nav.20275`.

25. J.-P. P. Richard, Y. Li, and L. A. Miller, *Valid inequalities for MIPs and group polyhedra from approximate liftings*, Mathematical Programming **118** (2009), no. 2, 253–277, `doi:10.1007/s10107-007-0190-9`.

26. W. A. Stein et al., *Sage Mathematics Software (Version 6.7)*, The Sage Development Team, 2015, `http://www.sagemath.org`.

27. M. Sugiyama, *Cut-generating functions for integer linear programming*, Bachelor thesis, University of California, Davis, June 2015, available from `https://www.math.ucdavis.edu/files/1514/4469/2452/Masumi_Sugiyama_ugrad_thesis.pdf`.

28. F. Wiedijk, *The seventeen provers of the world*, available from `http://www.cs.ru.nl/~freek/comparison/comparison.pdf`.

A   Appendix

## A.1   Additional illustrations for `gj_forward_3_slope`

In Figure 4 in section 7 we showed the slice of the cell complex for fixed parameter $f = 4/5$. In Figure 6 below, we show two other views on the parameter space.
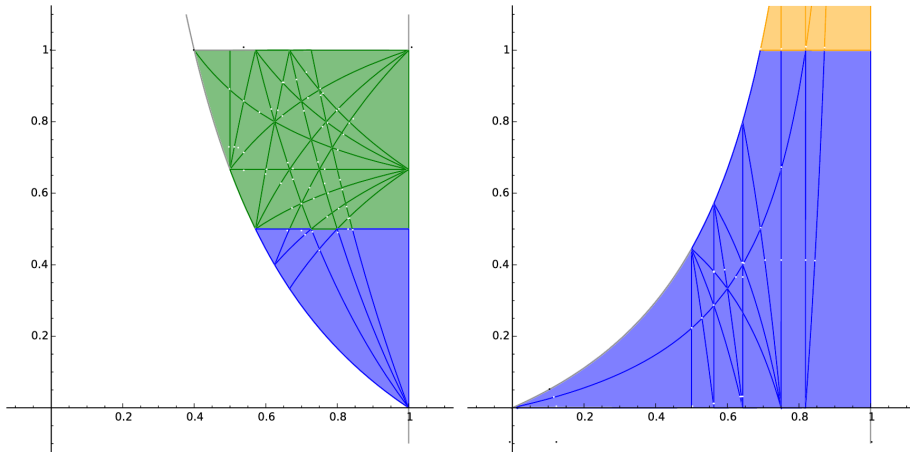
Fig. 6. The cell complex of the parametric family `gj_forward_3_slope`. *Left*, showing the plane of parameters (f, λ_1) for fixed $\lambda_2 = 2/3$; *Right*, showing the plane of parameters (f, λ_2) for fixed $\lambda_1 = 4/9$. Cells are color-coded: 'not constructible' (*white*), 'constructible, but not minimal' (*yellow*), 'minimal, but not extreme' (*green*) or 'extreme' (*blue*).

A.2   Cell complex for `chen_4_slope`

Figure 7 illustrates our remarks in section 7 regarding the extremality conditions in Chen's theorem (Theorem 3.3) about his `chen_4_slope` family. The parameter space is 5-dimensional; the figures show a 2-dimensional slice corresponding to varying parameters λ_1 and λ_2 and fixed parameters f, s^+, s^-.
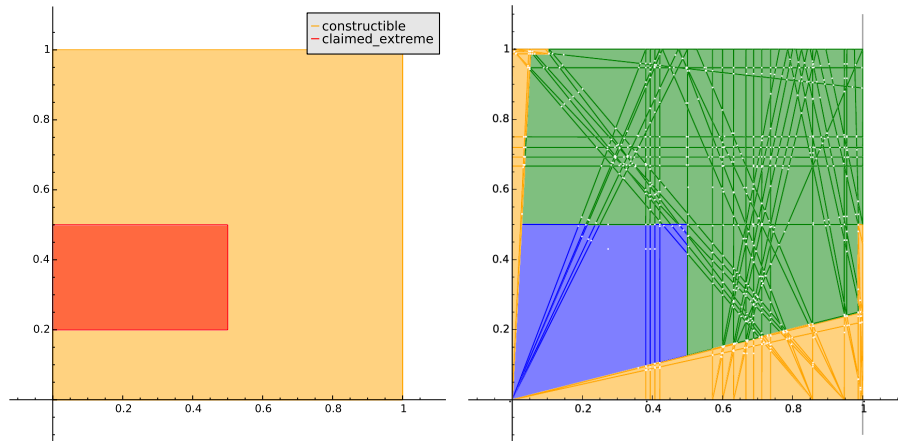
Fig. 7. The extreme region of `chen_4_slope` claimed in the literature is incorrect. Parameters (λ_1, λ_2); fixed $f = 7/10$, $s^+ = 2$, $s^- = -4$. *Left*, the incorrect hypotheses of Theorem 3.3 (*orange*) within the region of constructibility of the function (*yellow*). *Right*, the cell complex computed by our implementation. Cells are color-coded: 'constructible, but not minimal' (*yellow*), 'minimal, but not extreme' (*green*) or 'extreme' (*blue*).

A.3   Miller, Li, and Richard's $\mathrm{CPL}_3^{=}$-extreme functions

Miller, Li and Richard [24] derived extreme functions using an approximate lift-
ing scheme proposed in [25]. The authors studied a family of continuous piece-
wise linear functions ϕ, called $\mathrm{CPL}_3^{=}$-lifting functions, that have 4 parameters
$r_0, z_1, \theta_1, \theta_2$. ($r_0$ is our f.) Given $(r_0, z_1) \in \mathbb{R}^2$ such that $r_0 + 4z_1 \leq 1$, the pa-
rameters (θ_1, θ_2) that define a superadditive $\mathrm{CPL}_3^{=}$-lifting function ϕ belong to a
certain polytope $P\Theta_3^{=}$. They investigated the $\mathrm{CPL}_3^{=}$-group functions π obtained
by converting the $\mathrm{CPL}_3^{=}$-lifting functions ϕ that correspond to extreme points
of the polytope $P\Theta_3^{=}$. These functions π are potentially strong valid inequalities
for the infinite group problem, but they are not always extreme. As a result of a
manual inspection which required extensive case analysis, the authors summa-
rized in [24, Table 5] the conditions on the parameters $r_0, z_1, \theta_1, \theta_2$ for which the
corresponding $\mathrm{CPL}_3^{=}$-group functions π are extreme for the infinite group prob-
lem. These extreme functions π belong to 14 parametric families of two- and
three-slope continuous piecewise linear functions, among which some had not
been discovered before. The objective of this section is to verify and reproduce
the results stated in [24] by an automated discovery process.

In order to get the description of the polytope $P\Theta_3^{=}$, one writes out all the
superadditivity constraints $\phi(x) + \phi(y) \leq \phi(z)$ where $z = x + y$, (x, y) or (x, z)
is a pair of breakpoints of the $\mathrm{CPL}_3^{=}$-lifting function ϕ. (Or equivalently all the
subadditivity constraints on the breakpoints of the $\mathrm{CPL}_3^{=}$-group functions π.)
For example, at the breakpoint $r_0 + z_1$, ϕ takes value θ_1, so the superadditivity
constraint $\phi(r_0+z_1)+\phi(r_0+z_1) \leq \phi(2r_0+2z_1)$ reduces to $2\theta_1 \leq \phi(2r_0+2z_1)$. This
inequality is satisfied by any $(\theta_1, \theta_2) \in P\Theta_3^{=}$. To determine the value $\phi(2r_0+2z_1)$
in terms of θ_1 and θ_2, one needs to know which linear piece of ϕ the value $2r_0+2z_1$
falls into. So the form of an inequality that defines $P\Theta_3^{=}$ depends on how the
breakpoints of ϕ are arranged. We would first compute the cells of (r_0, z_1) where
the arrangement of breakpoints stays the same. That is, if for ϕ corresponding
to a given (r_0^*, z_1^*), the sum (or difference) of its i-th and j-th breakpoints is
contained in its k-th linear piece, then same holds for any (r_0, z_1) inside the
cell which contains (r_0^*, z_1^*). In consequence, the inequalities defining $P\Theta_3^{=}$ take
a fixed form for any (r_0, z_1) inside a cell, but differ from cell to cell. Figure 8
illustrates these cells that are obtained by running the following code:

```
sage: regions = regions_r0_z1_from_arrangement_of_bkpts()
sage: g = plot_cpl_components(regions)
sage: g.show(xmin=0, xmax=1, ymin=0, ymax=1/4)
```

We observe that the result agrees with [24, Table 1].

We then study these cells one by one. Assume that (r_0, z_1) is restricted to
a cell where the arrangement of breakpoints is combinatorially the same. The
superadditivity constraints on pairs of breakpoints of ϕ give a set of $N = 52$
linear inequalities that define $P\Theta_3^{=}$. This description includes many redundant
inequalities. To save time on the case analysis later, a simplified description of
$P\Theta_3^{=}$ which consists of $N = 9$ linear inequalities is obtained in [24, section 3.1].

This step is not necessary in our automated study, because the code can afford $N = 52$.

The value of (θ_1, θ_2) in $(r_0, z_1, \theta_1, \theta_2)$ that gives an extreme function π must be an extreme point (θ_1, θ_2) of the polytope $P\Theta_3^=$. To obtain the extreme point (θ_1, θ_2), we pick 2 inequalities to be tight in the description of $P\Theta_3^=$. Suppose that the i-th and the j-th inequalities of $P\Theta_3^=$ are tight, for $1 \leq i < j \leq N$. One can solve for (θ_1, θ_2) symbolically in terms of (r_0, z_1). Then, we plug in this solution (θ_1, θ_2) in π to get a $CPL_3^=$-group function π with parameters r_0 and z_1. Run the BFS (always restricted to a (r_0, z_1) cell with same arrangement of breakpoints) to find regions where π is extreme (*blue*); is minimal but not extreme (*green*); is not minimal (*yellow*) (not minimal implies some subadditivity constraint of π is violated, so the solution (θ_1, θ_2) does not satisfy all other inequalities of $P\Theta_3^=$, and thus (θ_1, θ_2) is not an extreme point of the polytope at this (r_0, z_1).) Repeat this process for all $1 \leq i < j \leq N$, and for all arrangement cells. For each different expression of (θ_1, θ_2) that has an extreme (*blue*) region, plot the results on a diagram. See Figure 9. These diagrams are obtained by running the following code:
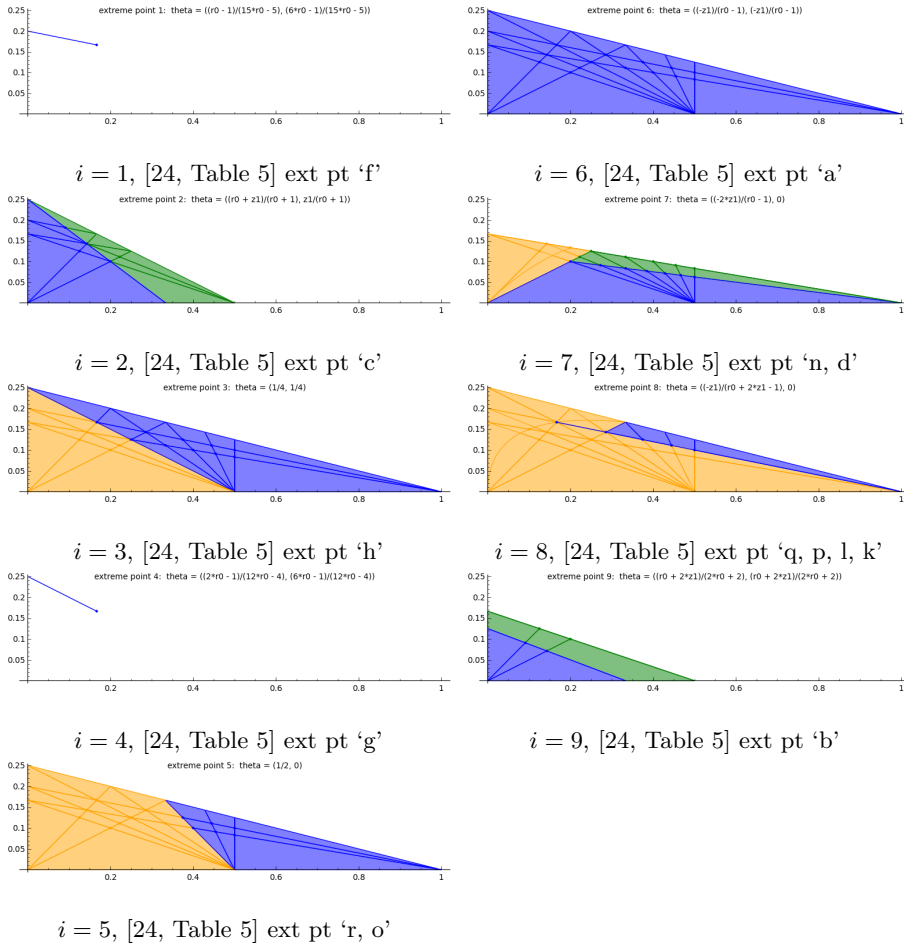
```
# 87 regions in Figure 8.
# For each region, find theta solutions, then subdivide into
# smaller components by running bfs with parametric field.
sage: regions = cpl_regions_with_thetas_and_components()
# Gather the blue components that correspond to the same
# expression of theta together.
sage: thetas_and_regions = cpl_thetas_and_regions_extreme(regions)
# Get diagrams "cpl_ext_theta_i" that show only blue regions.
sage: save_cpl_extreme_theta_regions(thetas_and_regions)
# Get diagrams "cpl_theta_i", that show a bit more.
sage: thetas_and_components = {}
sage: for theta in thetas_and_regions.keys():
....:     components = cpl_regions_fix_theta(regions, theta)
....:     thetas_and_components[theta]=components
sage: save_cpl_extreme_theta_regions(thetas_and_components)
```

We compare the diagrams (and their inequality descriptions) with the conditions for extremality stated in [24, Table 5]. We verify that the code is successful in finding all extreme regions claimed in the literature.[10] Furthermore, we observe that for the case 'k' the condition for extremality $r_0 \leq 2z_1$ and $r_0 + 5z_1 = 1$ stated in [24, Table 5] is incorrect; it should be $r_0 \geq z_1$ and $r_0 + 5z_1 = 1$. We remark that in the course of adding these functions (`mlr_cpl3_`...) to the Electronic Compendium [20], Sugiyama [27] had already found another, unrelated mistake: [24, Table 3] shows incorrect slope values s_3 of case 'l' and s_4 of case 'p'.

---

[10] Again we note that some heuristics and manual inspection were necessary, in particular to deal with some redundant nonlinear inequalities, which our current implementation is not able to remove; see the discussion at the end of section 4.
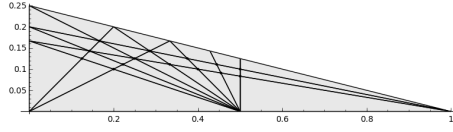
Fig. 8. The plane of the parameters (r_0, z_1), which determine the location of breakpoints, has 30 two-dimensional cells derived from the arrangement of breakpoints of the CPL function. Together with the lower-dimensional cells that arise as intersections, there are a total of 87 cells.

$i = 1$, [24, Table 5] ext pt 'f'

$i = 6$, [24, Table 5] ext pt 'a'

$i = 2$, [24, Table 5] ext pt 'c'

$i = 7$, [24, Table 5] ext pt 'n, d'

$i = 3$, [24, Table 5] ext pt 'h'

$i = 8$, [24, Table 5] ext pt 'q, p, l, k'

$i = 4$, [24, Table 5] ext pt 'g'

$i = 9$, [24, Table 5] ext pt 'b'

$i = 5$, [24, Table 5] ext pt 'r, o'

Fig. 9. The plane of the parameters (r_0, z_1), which determine the location of breakpoints, with cells corresponding to the automatic proof. Cells are color-coded: 'not constructible' (*white*), 'constructible, but not minimal' (*yellow*), 'minimal, but not extreme' (*green*) or 'extreme' (*blue*)

A.4 Definition of the new functions

The following Sage code defines the new families of functions.

```
def kzh_3_slope_param_extreme_1(f=6/19, a=1/19, b=5/19, field=None, conditioncheck=True):
    if not bool(0 < f < f+a < (1+f-b)/2 < (1+f+b)/2 < 1-a < 1):
        raise ValueError, "Bad parameters. Unable to construct the function."
    if conditioncheck:
        if not bool(0 <= a and  0 <= b <= f and 3*f+4*a-b-1 <= 0):
            logging.info("Conditions for extremality are NOT satisfied.")
        else:
            logging.info("Conditions for extremality are satisfied.")
    v = (f*f+f*a-3*f*b-3*a*b+b)/(f*f+f-3*f*b)
    bkpts = [0, f, f+a, (1+f-b)/2, (1+f+b)/2, 1-a, 1]
    values = [0, 1, v, (f-b)/2/f, (f+b)/2/f, 1-v, 0]
    return piecewise_function_from_breakpoints_and_values(bkpts, values, field=field)

def kzh_3_slope_param_extreme_2(f=5/9, a=3/9, b=2/9, field=None, conditioncheck=True):
    if not bool(0 < a < f < 1 and 0 < b < f):
        raise ValueError, "Bad parameters. Unable to construct the function."
    if conditioncheck:
        if not bool(b <= a and f <= a + b and f <= (1+a-b)/2):
            logging.info("Conditions for extremality are NOT satisfied.")
        else:
            logging.info("Conditions for extremality are satisfied.")
    v = (f*(f-a+b-2)-a*b+2*a)/(f+b-1)/f/4;
    bkpts = [0, (f-a)/4, (f-a)/2, (f+a)/2, f-(f-a)/4, f, \
             (1+f-b)/2, (1+f+b)/2, 1]
    values = [0, v, (f-a)/f/2, (f+a)/f/2, 1-v, 1, (f-b)/2, (f+b)/f/2, 0]
    return piecewise_function_from_breakpoints_and_values(bkpts, values, field=field)
```

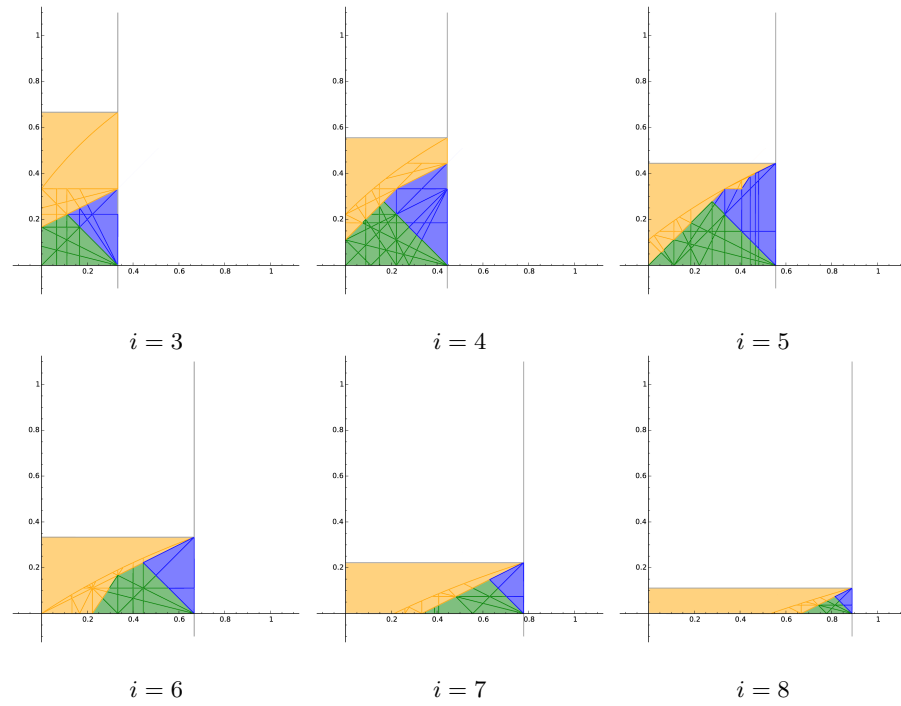### A.5   Cell complex for `kzh_3_slope_param_extreme_1`

Figure 10 illustrates the 3-dimensional cell complex for the new family `kzh_3_slope_param_extreme_1`, introduced in section 8, by showing slices for various fixed values of the parameter f.
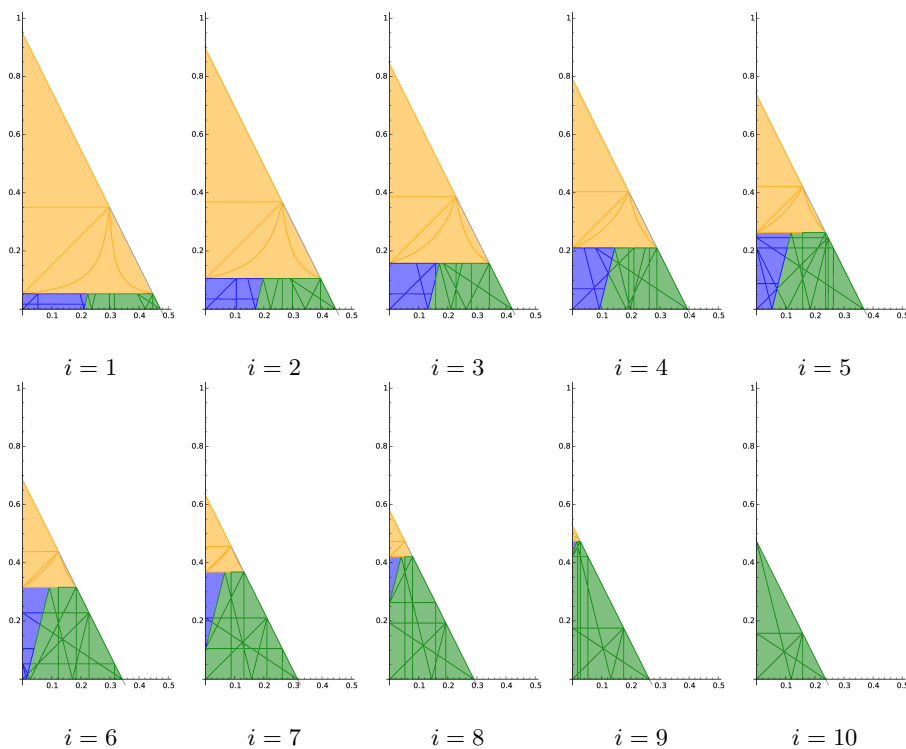
Fig. 10. Slices of the cell complex for the family `kzh_3_slope_param_extreme_1` for $f = i/19$

A.6   Cell complex for `kzh_3_slope_param_extreme_2`

Figure 10 illustrates the 3-dimensional cell complex for the new family `kzh_3_slope_param_extreme_2`, introduced in section 8, by showing slices for various fixed values of the parameter f.

Fig. 11. Slices of the cell complex for the family `kzh_3_slope_param_extreme_2` for $f = i/9$