BIMONOTONE ENUMERATION

MICHAEL EISERMANN

ABSTRACT. Solutions of a diophantine equation $f(a,b) = g(c,d)$, with a, b, c, d in some finite range, can be efficiently enumerated by sorting the values of f and g in ascending order and searching for collisions. This article considers functions $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ that are bimonotone in the sense that $f(a,b) \leq f(a',b')$ whenever $a \leq a'$ and $b \leq b'$. A two-variable polynomial with non-negative coefficients is a typical example. The problem is to efficiently enumerate all pairs (a,b) such that the values $f(a,b)$ appear in increasing order. We present an algorithm that is memory-efficient and highly parallelizable. In order to enumerate the first n values of f, the algorithm only builds up a priority queue of length at most $\sqrt{2n}+1$. In terms of bit-complexity this ensures that the algorithm takes time $O(n \log^2 n)$ and requires memory $O(\sqrt{n} \log n)$, which considerably improves on the memory bound $\Theta(n \log n)$ provided by a naïve approach, and extends the semimonotone enumeration algorithm previously considered by R.L. Ekl and D.J. Bernstein.

1. INTRODUCTION AND STATEMENT OF RESULTS

1.1. **Motivation.** Given polynomial functions $f, g\colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$, how can we efficiently enumerate solutions of the equation $f(a,b) = g(c,d)$? One standard way to do this is to sort the sets $F = \{\, (f(a,b), a, b) \mid 1 \leq a, b \leq N \,\}$ and $G = \{\, (g(c,d), c, d) \mid 1 \leq c, d \leq N \,\}$ into ascending order with respect to the first coordinate and to look for collisions. As stated, this requires storing all elements before sorting, which consumes memory $\Theta(n \log n)$, where $n = N^2$ is the number of values to enumerate, and time between $\Omega(n \log n)$ and $O(n \log^2 n)$.

The present article develops a less memory consuming algorithm under the hypothesis that f and g are *bimonotone*, that is, monotone in each variable. This is sufficiently often the case to be of interest, for example, when f and g are given by polynomials with non-negative coefficients. Given a bimonotone function f, Algorithm 4, discussed below, produces a stream $x_1, x_2, x_3, \ldots$ enumerating all parameters $x_i = (a_i, b_i)$ in the domain of f such that $f(x_1) \leqslant f(x_2) \leqslant f(x_3) \leqslant \ldots$. Having at hand such sorted enumerations for f and g, one can easily enumerate solutions of the equation $f(x) = g(y)$: start with $i = 1$ and $j = 1$; whenever $f(x_i) < g(y_j)$, increment i; whenever $f(x_i) > g(y_j)$, increment j. If eventually $f(x_i) = g(y_j)$, then output the solution (x_i, y_j) and continue searching.

1.2. **Main result.** The idea of sorted enumeration was applied by D.J. Bernstein [1] with great success to equations of the special form $p(a) + q(b) = r(c) + s(d)$. We generalize his approach to arbitrary bimonotone functions. The main result can be stated as follows:

Theorem 1. *Suppose that $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ is bimonotone and proper in the sense that for every $z \in \mathbb{Z}$ only finitely many pairs (a, b) satisfy $f(a, b) \leq z$. Then Algorithm 4 stated below produces a stream enumerating all pairs $(a, b) \in \mathbb{N} \times \mathbb{N}$ such that the values $f(a, b)$ appear in increasing order. While enumerating the first n values, the algorithm only builds up a priority queue of length $m \leq \sqrt{2n} + 1$. If f is a polynomial, this ensures that the algorithm takes time $O(n \log^2 n)$ and requires memory $O(\sqrt{n} \log n)$.*

The precise bound $m \leq \sqrt{2n} + 1$ is free of hidden constants and thus uniformly valid for all bimonotone functions f. The less explicit bounds of time $O(n \log^2 n)$ and memory $O(\sqrt{n} \log n)$ concern the bit-complexity and the hidden constants necessarily depend on f. We shall assume throughout that f behaves polynomially, see §2.3.

To place this result into perspective, notice that the time requirement $O(n \log^2 n)$ is nearly optimal: enumerating n elements obviously needs n iterations, and one $\log n$ factor is due to their increasing size. On the other hand, the standard approach would require memory $\Theta(n \log n)$ to store *all* values before outputting them. Here the stream approach can achieve considerable savings and reduce memory to $O(\sqrt{n} \log n)$.

Example 2. Consider $f(a, b) = p(a) + q(b)$ where p and q are non-decreasing polynomial functions of degree $\alpha = \deg p$ and $\beta = \deg q$, respectively. Assuming $1 \leq \alpha \leq \beta$, Algorithm 4 builds up a priority queue of length $m \in \Theta(n^\varepsilon)$ with $\varepsilon = \frac{\alpha}{\alpha + \beta} \in [0, \frac{1}{2}]$.

This illustrates that in the uniform bound $m \in O(n^{1/2})$, stated in the theorem for all bimonotone functions, the exponent $\frac{1}{2}$ cannot be improved. Notwithstanding, the algorithm performs better on certain subclasses of bimonotone functions, where $\varepsilon < \frac{1}{2}$.

Remark 3. The predecessor of our algorithm is semimonotone enumeration, recalled in §3. It was devised in [2, 1] for polynomials of the form $f(a, b) = p(a) + q(b)$, where it provides the desired memory bound $O(\sqrt{n} \log n)$. In the more general setting of bimonotone functions, however, we show that it only guarantees the memory bound $O(n \log n)$ and in general the exponent 1 cannot be improved. See §5 for a detailed discussion.

As an additional benefit, our algorithm turns out to be highly parallelizable:

Remark 4. Algorithm 4 can be adapted to enumerate only those pairs $(a, b) \in \mathbb{N} \times \mathbb{N}$ for which the values $f(a, b)$ lie in a given interval $[z_1, z_2]$. Time and memory requirements are essentially the same as before; only initialization induces some additional cost and can usually be neglected. This means that searching solutions $f(a, b) = g(c, d)$ can be split up into disjoint intervals and thus parallelized on independent machines (see §6).

1.3. **How this article is organized.** Section 2 introduces the necessary nota-
tion and recalls the generic algorithm of sorted enumeration for an arbitrary map
$f: X \to Z$, where X is a finite set. Section 3 discusses a refined algorithm,
essentially due to R.L. Ekl [2] and D.J. Bernstein [1], under the hypothesis that
$f: A \times B \to Z$ is semimonotone, that is, monotone in the first variable. Section
4 develops a sorted enumeration algorithm for bimonotone functions, and Section
5 analyses the asymptotic complexity. Section 6 highlights the intrinsically par-
allel structure of such a search problem. Section 7 generalizes our algorithms to
functions $f: X \to Z$ restricted to suitable domains $X \subset A \times B$ that are of of
practical interest. Finally, Section 8 briefly indicates applications to diophantine
enumeration problems, such as the taxicab problem.

2. Sorted enumeration for arbitrary functions

Before discussing more sophisticated versions, let us first describe the general
problem of sorted enumeration and recall its generic solution.

2.1. **The generic problem.** Throughout this article we consider an ordered set
$(Z, \leqslant)$. By *order* we always mean a reflexive, transitive relation that is complete
and antisymmetric, i.e. each pair $z \neq z'$ in Z satisfies either $z \leqslant z'$ or $z' \leqslant z$.
Without completeness we may have neither $z \leqslant z'$ nor $z' \leqslant z$, in which case we
speak of a *partial* order. Without antisymmetry we may have both $z \leqslant z'$ and
$z' \leqslant z$, in which case we speak of a *preorder*.

We assume that X is a finite or countably infinite set. An *enumeration* of X
is a stream $x_1, x_2, x_3, \ldots$ in which each element of X occurs exactly once. Such
an enumeration is *monotone* or *sorted* with respect to $f: X \to Z$ if it satisfies
$f(x_1) \leqslant f(x_2) \leqslant f(x_3) \leqslant \ldots$. Whenever the function f is understood from the
context, we will simply speak of a *sorted enumeration* of X.

Remark 5. The map $f: X \to Z$ can be used to pull back the order $\leqslant$ from Z
to the initially unordered set X. More explicitly, we define $x \preccurlyeq x'$ if and only if
$f(x) \leqslant f(x')$. A sorted enumeration of X is thus a stream in which all elements of
X appear in increasing order with respect to the preorder $\preccurlyeq$.

2.2. **The generic algorithm.** In the general setting, where X is finite and f has no
further structure, there is essentially only one way to produce a sorted enumeration:

Algorithm 1   Sorted enumeration for an arbitrary function
Requires: a function $f: X \to Z$ from a finite set X to an ordered set Z
Output: an enumeration of X, monotone with respect to f
1: Generate a list L of all pairs $(f(x), x)$ with $x \in X$.
2: Sort the list L according to the first coordinate $f(x)$.
3: Output the arguments x as sorted in the list L.

Algorithm 1 is obviously correct. Given a set X of size n, generating and reading
the list L takes n iterations, while sorting requires $O(n \log n)$ operations. Not much
optimization can be expected concerning these time requirements, since enumer-
ation (sorted or not) takes at least n iterations. Memory requirements, however,
may be far from optimal, and the more specialized algorithms discussed below will
mainly be concerned with minimizing the use of temporary memory.

2.3. **Time and memory requirements.** Throughout this article we use standard asymptotic notation, as in [3, §9]. It is customary to consider the cost for storing and handling elements x and $f(x)$ to be constant. This is no longer realistic when the size $n = |X|$ grows without bound. As a typical example, consider a polynomial function $f \colon \mathbb{N} \to \mathbb{Z}$ restricted to $X = \{1, \ldots, n\}$. If each element $x \in X$ is stored in binary form, the maximal memory required is $\Theta(\log n)$. Likewise, the maximal time to calculate, copy, and compare values $f(x)$ is $\Theta(\log n)$, neglecting factors of order $\log \log n$ or less. Most elements require nearly maximum cost, so we shall only consider the worst case.

In general, we say that f *behaves polynomially* if the bit-complexity per element is $\Theta(\log n)$, as above. In this case we arrive at the following more realistic account:

Proposition 6. *In order to enumerate a set X of size n, the generic Algorithm 1 builds up a list of size $m = n$, and thus requires time $O(n \log^2 n)$ and memory of size $\Theta(n \log n)$.*   □

3. Sorted enumeration for semimonotone functions

In this section we consider a semimonotone function $f \colon A \times B \to Z$. By this we mean that $(A, \leqslant)$ is an ordered set and $a \leqslant a'$ implies $f(a, b) \leqslant f(a', b)$ for all $b \in B$. This is the same as saying that f is monotone with respect to the partial order $(a, b) \leq (a', b')$ defined by the condition $a \leqslant a'$ and $b = b'$.

3.1. **The idea of semimonotone enumeration.** We will first assume that A and B are finite sets. This entails that $(A, \leqslant)$ is isotonic to an interval $\{1, \ldots, l\}$ of integers. The minimal and maximal element of A is denoted by $a_{\min}$ and $a_{\max}$, respectively, and the successor function is denoted by $a \mapsto \sigma a$. Of course $a_{\max}$ cannot have a successor in A, so by convention we set $\sigma a_{\max} = +\infty$.

We equip $X = A \times B$ with the partial order $\leq$ as defined above. Given a subset $X_i \subset X$, we denote by $M_i = \mathrm{Min}(X_i)$ the set of its minimal elements. Conversely, M_i defines its upper set $M_i^+ = \{\, x \in X \mid m \leq x \text{ for some } m \in M_i \,\}$. Figure 1 shows a subset X_i (indicated by crosses) together with its set of minima M_i (circled crosses). In this example X_i is *saturated* in the sense that $X_i = M_i^+$.
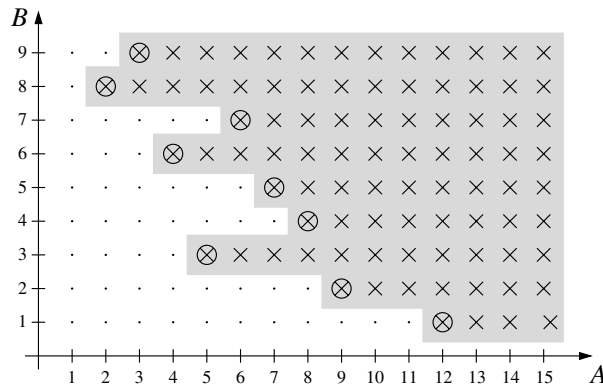

FIGURE 1. A subset of $A \times B$ and its minima with respect to $\leq$

Since f is monotone with respect to $\leq$, the minimum of $f(X_i)$ is attained on M_i. It thus suffices to find $x_i \in M_i$ realizing $f(x_i) = \min f(M_i)$. We can then output x_i

and continue with the set $X_{i+1} = X_i \smallsetminus \{x_i\}$. Notice that X_{i+1} is again saturated and M_{i+1} can be easily constructed from M_i. This is the idea of Algorithm 2 below.

3.2. **Suitable data structures.** The following algorithm has been independently developed by R.L. Ekl [2] and D.J. Bernstein [1], and formalizes the above approach: instead of handling the entire set X_i, it operates on two smaller sets, $M = \text{Min}(X_i)$ and $F = f(M)$. In order to efficiently find $x_i \in M$ realizing $f(x_i) = \min f(M)$, we store the set of images $f(M)$ in a priority queue F. Recall that a priority queue F for elements of $(Z, \leqslant)$ provides the following elementary operations:

- Inserting an element $z \in Z$ into F ("push").
- Reading and removing a minimal element of F ("pop").

Priority queues are typically implemented using a heap or a binary tree; in either case the elementary operations need $O(\log m)$ steps, where m is the number of elements in the priority queue. For a general presentation see Knuth [7, §5.2.3].

3.3. **The semimonotone enumeration algorithm.** Instead of $f \colon A \times B \to Z$ it is more convenient to work with the map $f^* \colon A \times B \to Z \times A \times B$ defined by $f^*(a, b) = (f(a, b), a, b)$. In our formulation of Algorithm 2 we thus use a priority queue F for elements in $Z \times A \times B$, sorted by the first coordinate.

Algorithm 2 Sorted enumeration for a semimonotone function

Requires: a semimonotone function $f \colon A \times B \to Z$
Output: an enumeration of $A \times B$, monotone with respect to f

1: Start with an empty priority queue F, then insert $f^*(a_{\min}, b)$ for all $b \in B$.
2: **while** F is non-empty **do**
3:   Remove a minimal element $f^*(a, b)$ from F and output (a, b).
4:   **if** $\sigma a < +\infty$ **then** insert $f^*(\sigma a, b)$ into F **end if**
5: **end while**

Remark 7. All algorithms presented here can be regarded as templates, to be instantiated for the given map $f \colon A \times B \to Z$. Alternatively, one could consider them as taking the sets A and B and the map f as input data. In this case, of course, we do not pass the entire sets A and B as parameters, nor the map f, say as some subset of $A \times B \times Z$: for finite sets this would be as inefficient as Algorithm 1; for infinite sets it is simply impossible.

Instead, it suffices to call some function that calculates $f(a, b)$ for any given pair of parameters $a \in A$ and $b \in B$. To represent the sets A and B, all we need is the usual *iterator* concept, providing a pointer to the first (and possibly the last) element of the set and a method for incrementing, denoted by $a \mapsto \sigma a$ above. (Algorithms 5 and 7 also decrement, denoted by $b \mapsto \pi b$.)

One can easily add suitable specifications when passing to concrete implementations. For the present general exposition, however, we shall maintain the slightly coarser description, trying to strike a balance between general concepts and implementation details.

Algorithm 2 is obviously correct. The point is, as motivated above, that it usually uses less memory than the generic Algorithm 1.

Proposition 8. *In order to enumerate a set $X = A \times B$ of size n, Algorithm 2 only builds up a priority queue of size $m = |B|$. Let $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ be a semimonotone map that behaves polynomially. Applied to subsets $A = \{1, \ldots, l\}$ and $B = \{1, \ldots, m\}$ with $m \geq 2$, the algorithm thus takes time $O(n \log n \log m)$ and requires memory $\Theta(m \log n)$.*

Proof. The algorithm needs memory to hold m elements $f^*(a, b)$ in the priority queue F. Since most elements need memory of size $\Theta(\log n)$, we arrive at a total memory cost of $\Theta(m \log n)$. During each one of the n iterations, the most time consuming operation is updating the priority queue F which requires time $O(\log n \log m)$. Here m is the size of the queue and $\log n$ is the typical size of its elements. $\qquad\square$

Remark 9. Notice that in the degenerate case $|B| = 1$, Algorithm 2 simply enumerates A in increasing order, which takes time $O(n \log n)$ and memory $\Theta(\log n)$. In the opposite extreme $|A| = 1$, it sorts B with respect to f via heap-sort. We thus fall back on the generic Algorithm 1, which takes time $O(m \log^2 m)$ and space $\Theta(m \log m)$.

3.4. **Enumerating infinite sets.** Sorted enumeration can be generalized from finite to infinite sets. First of all, in order to be amenable to enumeration, A must be either finite or isotonic to the natural numbers. Moreover, we have to require that $f \colon A \times B \to Z$ be a *proper* map in the sense that for every $z \in \mathrm{Im}(f)$ only finitely many pairs (a, b) satisfy $f(a, b) \leq z$. (This condition actually implies that A is finite or isotonic to $\mathbb{N}$.) Of course, we also have to assume that comparisons and all other operations are computable; as before their cost will be assumed to be of order $O(\log n)$.

Proposition 10. *Suppose that A is an infinite ordered set and B is a finite set of size m. Let $f \colon A \times B \to Z$ be a proper semimonotone map. Then Algorithm 2 produces a stream enumerating all pairs $(a, b) \in A \times B$ such that the values $f(a, b)$ appear in increasing order. Producing the first n values takes time $O(n \log n \log m)$ and requires memory $\Theta(m \log n)$.*

Proof. For every $z \in \mathrm{Im}(f)$ the set $\{(a, b) \in A \times B \mid f(a, b) \leq z\}$ is finite, and thus contained in some finite product $[a_{\min}, a_1] \times B$. Hence Algorithm 4 correctly enumerates all parameters (a, b) with $f(a, b) \leq z$ as in the finite case. Since this is true for all z, the enumeration exhausts $A \times B$. Bit-complexity behaves as in Proposition 8. $\qquad\square$

We wish to adapt semimonotone enumeration to the case where both A and B are infinite. Algorithm 2 is certainly not suited for this task, because the initialization will get stuck in an infinite loop. As a necessary restriction we require that $f \colon A \times B \to Z$ be proper, and as before we assume that f is monotone with respect to A. For every $z \in \mathrm{Im}(f)$, we can thus enumerate the finite set $\{f \leqslant z\} := \{(a, b) \in A \times B \mid f(a, b) \leqslant z\}$ by applying Algorithm 2 to the relevant finite set $B(z) = \mathrm{pr}_2\{f \leqslant z\} = \{b \in B \mid f(a_{\min}, b) \leq z\}$.

In order to formulate an explicit algorithm, we assume that the set B is ordered and that $f_2 \colon B \to Z$, $f_2(b) = f(a_{\min}, b)$ is non-decreasing. This is strictly weaker than demanding f to be bimonotone, because we require monotonicity in b only on the axis $\{a_{\min}\} \times B$. This technical condition ensures that we can easily construct the relevant finite set $B(z)$. In fact, the monotonicity of f_2 is not at all restrictive,

because we can *choose* the order on B, for example by pulling back the order on Z via f_2 to a preorder on B, and then refining to an order by arbitrating collisions. In other words, the order on B is just a convenient way to encode some preparatory analysis of the proper map $f_2 \colon B \to Z$.

This idea is formalized in Algorithm 3, which is a slight variation of Algorithm 2. The only difference is that it automatically adapts the relevant interval $B(z) = [b_{\min}, b_{\max}[$ according to the level z attained during the enumeration.

Algorithm 3 Sorted enumeration for a semimonotone function

Requires: a proper semimonotone function $f \colon A \times B \to Z$
Output: an enumeration of $A \times B$, monotone with respect to f.

1: Initialize the priority queue $F \leftarrow \langle\, f^*(a_{\min}, b_{\min})\, \rangle$ and set $b_{\max} \leftarrow b_{\min}$.
2: **while** F is non-empty **do**
3:    Remove a minimal element $f^*(a, b)$ from F and output (a, b).
4:    **if** $\sigma a < +\infty$ **then** insert $f^*(\sigma a, b)$ into F **end if**
5:    **if** $b = b_{\max}$ **then**
6:       Set $b_{\max} \leftarrow \sigma b_{\max}$.
7:       **if** $b_{\max} < +\infty$ **then** insert $f^*(a_{\min}, b_{\max})$ into F **end if**
8:    **end if**
9: **end while**

Here we have formulated Algorithm 3 so that it applies to finite and infinite sets alike. If A or B is infinite, then $\sigma a < +\infty$ or $b_{\max} < +\infty$, respectively, is always true and the corresponding test can be omitted.

Proposition 11. *Suppose that both A and B are infinite ordered sets and that $f \colon A \times B \to Z$ is a proper semimonotone function. We also assume that $b \mapsto f(a_{\min}, b)$ is non-decreasing. Then Algorithm 2 provides a sorted enumeration of $A \times B$. While enumerating the first n values, it builds up a priority queue of length $m \le n + 1$. This ensures that the algorithm takes time $O(n \log^2 n)$ and memory $O(n \log n)$.* □

Semimonotone functions are tailor-made for applications where we have monotonicity in a but not necessarily in b. They are halfway towards bimonotone functions, which are more restrictive but support much better algorithms. These will be discussed next.

4. Sorted enumeration for bimonotone functions

In this section we finally turn to bimonotone functions $f \colon A \times B \to Z$. By this we mean that both $(A, \leqslant)$ and $(B, \leqslant)$ are ordered sets, and that $a \le a'$ and $b \le b'$ implies $f(a, b) \le f(a', b')$. This is the same as saying that f is monotone with respect to the partial order $(a, b) \leqslant (a', b')$ defined by $a \leqslant a'$ and $b \leqslant b'$.

4.1. **The idea of bimonotone enumeration.** We will first assume that both sets A and B are finite. Given a subset $X_i \subset X$ we denote by $M_i = \mathrm{Min}(X_i)$ the set of its minimal elements with respect to $\leqslant$. Conversely, M_i defines its upper set $M_i^{\#} = \{\, x \in X \mid m \leqslant x \text{ for some } m \in M_i \,\}$. Figure 2 shows a subset X_i (indicated by crosses) together with its set of minima M_i (circled crosses). In this example X_i is *saturated* in the sense that $X_i = M_i^{\#}$.

FIGURE 2. A subset of $A \times B$ and its minima with respect to $\leqslant\!\!\leqslant$

Since f is monotone with respect to $\leqslant\!\!\leqslant$, the minimum of $f(X_i)$ is attained on M_i. It thus suffices to find $x_i \in M_i$ realizing $f(x_i) = \min f(M_i)$. We can then output x_i and continue with the set $X_{i+1} = X_i \smallsetminus \{x_i\}$, which is again saturated. Moreover, it is possible to construct M_{i+1} directly from M_i, without having to construct X_i or X_{i+1}. (See Algorithm 4 below.) Thus, instead of searching the entire set X_i, we only need to keep track of M_i, the set of minimal elements.

4.2. **Suitable data structures.** According to the previous remark, the bimonotone enumeration algorithm will operate on two sets: $M = \mathrm{Min}(X_i)$ and $F = f(M)$. The set M can profitably be implemented as a list $\langle\, (a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m) \,\rangle$ with $a_i \in A$ and $b_i \in B$. During the algorithm, the list M will always be ordered in the sense that $a_1 < a_2 < \cdots < a_m$ and $b_1 > b_2 > \cdots > b_m$, as already indicated in Figure 2. We call (a_k, b_k) the *predecessor* of (a_{k+1}, b_{k+1}) in M, and conversely (a_{k+1}, b_{k+1}) the *successor* of (a_k, b_k) in M. By convention the predecessor of (a_1, b_1) is $(-\infty, +\infty)$, and the successor of (a_m, b_m) is $(+\infty, -\infty)$.

Given an element (a, b) in the list M, the required operations are:

- Finding the successor or predecessor of (a, b) in M.
- Inserting an element into M right after (a, b).
- Removing (a, b) from M.

The cost of these operations can be assumed to be $O(\log n)$, which is the typical cost for storing and handling one of the elements of the set $X = A \times B$ of size n. In particular, the cost is independent of the size $m = |M|$. For details on bidirectional lists see Knuth [6, §2.2.5], or any other textbook on algorithms and data structures.

As before, the set $F = f(M)$ will be implemented as a priority queue containing the values $f(a, b)$ for all (a, b) in M. It is recommendable to store $f(a, b)$ together with a pointer to the element (a, b) in the list M. This allows us to extract (a, b), and, moreover, we can directly address (a, b) in M without searching the list. For notational convenience we will not explicitly mention this pointer in the sequel.

4.3. **The bimonotone enumeration algorithm.** Having suitable data structures at our disposal, it is an easy matter to formalize bimonotone enumeration (Algorithm 4).

The only subtlety of this algorithm is updating the list M. We want to remove (a, b), of which we know that it is a minimal element of X_i. The set of elements

Algorithm 4 Sorted enumeration for a bimonotone function

Requires: a bimonotone function $f\colon A \times B \to Z$

Output: an enumeration of $A \times B$, monotone with respect to f

---

1: Initialize $M \leftarrow \langle\, (a_{\min}, b_{\min})\, \rangle$ and $F \leftarrow \langle\, f(a_{\min}, b_{\min})\, \rangle$.

2: **while** F is non-empty **do**

3:    Remove a minimal element $f(a,b)$ from F and output (a,b).

4:    Let (a^*, b_*) be the successor of (a,b) in the list M.

5:    **if** $\sigma a < a^*$ **then**

6:      Insert $(\sigma a, b)$ into M right after (a,b) and insert $f(\sigma a, b)$ into F.

7:    **end if**

8:    Let (a_*, b^*) be the predecessor of (a,b) in the list M.

9:    **if** $\sigma b < b^*$ **then**

10:     Insert $(a, \sigma b)$ into M right after (a,b) and insert $f(a, \sigma b)$ into F.

11:    **end if**

12:    Remove (a,b) from the list M.

13: **end while**

---

strictly greater than (a,b) is given by

$$\{(a,b)\}^{\#} \smallsetminus \{(a,b)\} = \{\, (a, \sigma b), (\sigma a, b)\, \}^{\#}.$$

Hence, removing (a,b) creates at most two new minima, $(a, \sigma b)$ and $(\sigma a, b)$. It is easy to check whether they are actually minimal for $X_i \smallsetminus \{(a,b)\}$: since our list M of minima is ordered, it suffices to compare $(a, \sigma b)$ to the predecessor (a_*, b^*), and $(\sigma a, b)$ to the successor (a^*, b_*).

To illustrate the different possibilities, we consider Figure 2 again. The following table indicates, for each possible minimum (a,b), how the list M has to be modified in order to obtain a new ordered list of minima satisfying $M^{\#} = X_i \smallsetminus \{(a,b)\}$:

(a,b)	(a^*, b_*)	$(\sigma a, b)$	insert?	(a_*, b^*)	$(a, \sigma b)$	insert?
$(3,9)$	$(5,8)$	$(4,9)$	yes	$(-\infty, +\infty)$	$(3, +\infty)$	no
$(5,8)$	$(7,7)$	$(6,8)$	yes	$(3,9)$	$(5,9)$	no
$(7,7)$	$(8,5)$	$(8,7)$	no	$(5,8)$	$(7,8)$	no
$(8,5)$	$(9,2)$	$(9,5)$	no	$(7,7)$	$(8,6)$	yes
$(9,2)$	$(12,1)$	$(10,2)$	yes	$(8,5)$	$(9,3)$	yes
$(12,1)$	$(+\infty, -\infty)$	$(13,1)$	yes	$(9,2)$	$(12,2)$	no

Lemma 12. *Algorithm 4 is correct: if A and B are finite ordered sets and $f\colon A \times B \to Z$ is a bimonotone map, then Algorithm 4 produces a stream enumerating all pairs $(a,b) \in A \times B$ such that the values $f(a,b)$ appear in increasing order.*

Proof. At the beginning of the i-th iteration of the algorithm we denote M by M_i, and F by F_i, and the set of remaining parameters by $X_i := M_i^{\#}$. The initialization states that $M_1 = \langle\, (a_{\min}, b_{\min})\, \rangle$ and $F_1 = \langle\, f(a_{\min}, b_{\min})\, \rangle$, so $X_1 = A \times B$.

By induction we can assume that the set X_i is of size $n - i + 1$ and saturated, with $M_i = \mathrm{Min}(X_i)$ and $F_i = f(M_i)$. Furthermore, we can assume that the list representing M_i is ordered in the sense that $\langle\, (a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)\, \rangle$ satisfies $a_1 < a_2 < \cdots < a_m$ and $b_1 > b_2 > \cdots > b_m$. It is straightforward to verify that the i-th iteration of our algorithm ensures the following assertions:

- The output x_i satisfies $f(x_i) = \min F_i = \min f(M_i) = \min f(X_i)$.
- The set $X_{i+1} = X_i \smallsetminus \{x_i\}$ is saturated and of size $n - i$.
- We have $M_{i+1} = \text{Min}(X_{i+1})$ and $F_{i+1} = f(M_{i+1})$.
- The new list representing the set M_{i+1} is again ordered.

The algorithm stops after n iterations when it reaches $X_{n+1} = \emptyset$, hence $M_{n+1} = \emptyset$ and $F_{n+1} = \emptyset$. We conclude that the output sequence $x_1, x_2, \ldots, x_n$ is an enumeration of $A \times B$ satisfying $f(x_1) \leqslant f(x_2) \leqslant \ldots \leqslant f(x_n)$. $\qquad\square$

Since the sorted enumeration algorithm outputs one element x_i at each iteration, the loop is repeated exactly $n = |A| \cdot |B|$ times. At the i-th iteration, the algorithm occupies memory of size $m_i = |M_i|$ to store the list M_i and the priority queue F_i. Let $m = \max m_i$ be the maximum during the entire execution.

Lemma 13. *In order to enumerate a set $X = A \times B$ of size n, Algorithm 4 only builds up a priority queue of length $m \leq \min\{|A|, |B|\}$, which entails, in particular, $m \leq \sqrt{n}$.*

Proof. During the enumeration algorithm the list representing M is always strictly increasing in a and strictly decreasing in b. In particular, the projections $M \to A$ and $M \to B$ are both injective. The required memory m is thus bounded by $\min\{|A|, |B|\}$. $\qquad\square$

Proposition 14. *Let $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ be a bimonotone map that behaves polynomially. Applied to subsets $A = \{1, \ldots, l\}$ and $B = \{1, \ldots, m\}$ with $m \geq 2$, Algorithm 4 takes time $O(n \log^2 n)$ and requires memory $O(\sqrt{n} \log n)$.*

Proof. The loop is repeated n times. The most time consuming operation is updating the priority queue F_i to F_{i+1} which requires time $O(\log n \log m_i)$, where m_i is the size of the queue F_i and its elements are typically of size $\Theta(\log n)$. The total cost is time $O(n \log n \log m)$ and memory $\Theta(m \log n)$. With $m \leq \sqrt{n}$ we obtain the stated bounds. $\qquad\square$

Example 15. The following extreme cases illustrate Algorithm 4 and the possible behaviour of the memory bound m. We consider $f \colon A \times B \to \mathbb{N}$ where $A = \{1, \ldots, k\}$ and $B = \{1, \ldots, l\}$ are two intervals of integers, with $k \geq l \geq 2$ say.

The best case occurs for $f(a, b) = la + b$, where Algorithm 4 enumerates $A \times B$ in lexicographic order. During the i-th iteration of the algorithm the set of minima M_i contains only 1 or 2 elements, so that $m = 2$, independent of the sizes $|A|$ and $|B|$.

The worst case occurs for $f(a, b) = a + b$. Having enumerated all elements x_i with $f(x_i) \leq l$, the list M contains exactly l elements, namely $(1, l), (2, l - 1), \ldots, (l, 1)$. Thus the upper bound $m = \min\{|A|, |B|\}$ is actually attained.

4.4. **Enumerating infinite sets.** As with semimonotone enumeration, bimonotone enumeration can be generalized from finite to infinite sets. The interesting point is that now both sets A and B can be infinite, and the algorithm applies without change.

Theorem 16. *Suppose that A and B are ordered sets and that $f \colon A \times B \to Z$ is a proper bimonotone map. Then Algorithm 4 produces a stream enumerating all pairs $(a, b) \in A \times B$ such that the values $f(a, b)$ appear in increasing order. In order to enumerate the first n values, the algorithm builds up a priority queue of length*

at most $\sqrt{2n} + 1$. If f behaves polynomially, the algorithm takes time $O(n \log^2 n)$ and requires memory $O(\sqrt{n} \log n)$.

Proof. For every $z \in \mathrm{Im}(f)$ the set $\{ (a,b) \in A \times B \mid f(a,b) \leq z \}$ is finite, it is thus contained in some finite product $[a_{\min}, a_1] \times [b_{\min}, b_1]$. Hence Algorithm 4 correctly enumerates all parameters (a,b) with $f(a,b) \leq z$ as in the finite case. Since this is true for all z, the enumeration exhausts $A \times B$.

Let us suppose that, after n outputs, the list M holds m pairs (a_i, b_i) ordered such that $a_1 < a_2 < \cdots < a_m$ and $b_1 > b_2 > \cdots > b_m$. This obviously implies $n \geq \frac{1}{2} m(m-1)$, whence $m \leq \sqrt{2n} + 1$. The time needed for n outputs is thus $O(n \log n \log \sqrt{n}) = O(n \log^2 n)$, while the required memory is $O(\sqrt{n} \log n)$. $\qquad\square$

Example 17. Again the worst case occurs for the map $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ with $f(a,b) = a + b$, where $m \sim \sqrt{2n}$. The best case occurs for $f(a,b) = \max\{a,b\}$, where $m \leq 4$.

This example shows that for bimonotone enumeration the memory bound $m \in O(\sqrt{n})$ is best possible: there exist bimonotone functions f for which Algorithm 4 actually requires temporary memory $m \sim \sqrt{2n}$. Notwithstanding, the algorithm performs significantly better on certain subclasses of bimonotone functions:

Proposition 18 (separate variables). *Consider $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ with $f(a,b) = p(a) + q(b)$, where p and q are non-decreasing polynomial functions of degree $\alpha = \deg p$ and $\beta = \deg q$, respectively. Assuming $1 \leq \alpha \leq \beta$, the bimonotone enumeration algorithm requires memory $m \in \Theta(n^\varepsilon)$ with exponent $\varepsilon = \frac{\alpha}{\alpha+\beta}$.*

For example, sorted enumeration of $f(a,b) = a^3 + b^7$ requires memory $m \in \Theta(n^{3/10})$, while the a priori bound of Theorem 16 only tells us $m \in O(n^{1/2})$.

Proof. Let $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be defined by $f(a,b) = a^\alpha + b^\beta$ with $1 \leq \alpha \leq \beta$. The general case $f(a,b) = p(a) + q(b)$ works essentially the same, but the notation is more cumbersome.

Suppose that the n-th output x_n has attained the level $f(x_n) = z$, and the list M holds m parameters $(a_1, b_1), \ldots, (a_m, b_m)$. Then we have $a_1 = 0$ and $f(a_1, b_1) = b_1^\beta \geq z$. On the other hand, $(a_1, b_1 - 1)$ has already been output, which means $(b_1 - 1)^\beta \leq z$. We conclude that $\sqrt[\beta]{z} \leq b_1 \leq \sqrt[\beta]{z} + 1$, whence $b_1 \sim \sqrt[\beta]{z}$. Analogously, $\sqrt[\alpha]{z} \leq a_m \leq \sqrt[\alpha]{z} + 1$, whence $a_m \sim \sqrt[\alpha]{z}$. This situation is depicted in Figure 3. In the sequel we set $a := a_m$ and $b := b_1$.
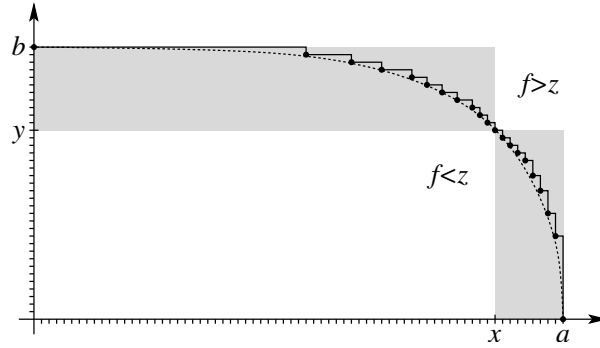

FIGURE 3. Estimating the size of the set $\mathrm{Min}_{\leqslant}\{a^\alpha + b^\beta \geq z\}$

The upper bound $n \leq (a+1)(b+1)$ is clear. Since f is convex, we also have the lower bound $n > \frac{1}{2}ab$. To see this, apply Pick's theorem to count integer points in the triangle $\Delta = [(0,0),(a,0),(0,b)]$. Both inequalities together imply that $n \in \Theta(z^{(\alpha+\beta)/\alpha\beta})$, or equivalently, $z \in \Theta(n^{\alpha\beta/(\alpha+\beta)})$.

The upper bound $m \leq b+1$ is clear, and it remains to establish a lower bound. We will assume $\alpha < \beta$. (The symmetric case $\alpha = \beta$ is easier and will be examined more closely in Example 27 below.) There exists a unique point $(x,y) \in \mathbb{R}^2_+$ on the contour whose normal vector points in the direction $(1,1)$: this is the solution of $x^\alpha + y^\beta = z$ and $\alpha x^{\alpha-1} = \beta y^{\beta-1}$. It is easy to see that $m \geq b - \lceil y \rceil$ and $m \sim (a-x) + (b-y)$ as indicated in Figure 3. We have $x = cy^{(\beta-1)/(\alpha-1)}$ with $c = \sqrt[\alpha-1]{\beta/\alpha}$, whence $c^\alpha y^{\alpha(\beta-1)/(\alpha-1)} + y^\beta = z$. Since $\alpha(\beta-1)/(\alpha-1) > \beta$ we deduce that $y \in o(\sqrt[\beta]{z})$. The bounds $b - \lceil y \rceil \leq m \leq b + 1$ thus entail $m \sim \sqrt[\beta]{z}$, whence $m \in \Theta(n^{\alpha/(\alpha+\beta)})$. $\square$

We remark that in the above examples semimonotone enumeration achieves the same asymptotic bounds. This warrants a more detailed analysis, which we endeavour next.

5. Asymptotic complexity

We are now ready to address the crucial question: is bimonotone enumeration (Algorithm 4) better than semimonotone enumeration (Algorithm 3)? We shall compare the size m of the priority queue built up during the algorithm. The test class consists of all proper bimonotone functions $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$, which is where both algorithms apply. First of all, the following observation is worth emphasizing:

Remark 19. Bimonotone enumeration is at least as good as semimonotone enumeration. More explicitly, both algorithms have to trace the contour of the finite set

$$\{f \leqslant z\} := \{(a,b) \in A \times B \mid f(a,b) \leqslant z\}$$

and construct the set of minima of the complement, $\mathrm{Min}\{f > z\}$. To this end semimonotone enumeration uses the partial order $(a,b) \leq (a',b')$ defined by $a \leqslant a'$ and $b = b'$. (Here we are ordering with respect to a for fixed b; since f is bimonotone, we could also order with respect to b for fixed a, whichever is advantageous.) Bimonotone enumeration uses the partial order $(a,b) \leqslant (a',b')$ defined by $a \leqslant a'$ and $b \leqslant b'$. This entails the inclusion

$$\mathrm{Min}_{\leqslant}\{f > z\} \subset \mathrm{Min}_{\leq}\{f > z\}.$$

This means that the priority queue for bimonotone enumeration is a subset of the queue for semimonotone enumeration, and consequently the required memory is less or equal.

At this point we should clarify a possible ambiguity. Both Algorithms 3 and 4 have to choose *one* minimal element of the priority queue. In order to disambiguate multiple minima, we choose the one with minimal B-coordinate. This ensures that it belongs to both $\mathrm{Min}_{\leqslant}\{f > z\}$ and $\mathrm{Min}_{\leq}\{f > z\}$, and the inclusion propagates inductively.

Whether the bimonotone algorithm can achieve a significant improvement depends on the function f. Let us begin with a trivial example where no savings are possible:

Example 20 (linear contour). Consider $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defined by $f(a, b) = (a+b)^\gamma$ with $\gamma \geq 1$. In this case $\mathrm{Min}_{\leqslant}\{f > z\} = \mathrm{Min}_{\leq}\{f > z\}$ is given by the line $a + b = 1 + \lfloor \sqrt[\gamma]{z} \rfloor$.

In general, however, the inclusion $\mathrm{Min}_{\leqslant}\{f > z\} \subset \mathrm{Min}_{\leq}\{f > z\}$ is strict. Generally speaking, bimonotone enumeration adapts better to the contour and achieves savings whenever the contour deviates from being a straight line. We now quantify this observation.

5.1. **Polynomial functions.** Consider $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Q}$ defined by a polynomial $f(a, b) = \sum_{i,j} c_{ij} a^i b^j$ with rational coefficients $c_{ij} \geq 0$. This condition ensures that f is bimonotone.

Let $f_1(a) = f(a, 0)$ and $f_2(b) = f(0, b)$ be the induced polynomial functions on the axes, and set $\alpha := \deg f_1$ and $\beta := \deg f_2$. We assume that $\alpha, \beta \geq 1$, which ensures that f is proper. Without loss of generality, we can also assume that $\alpha \leq \beta$.

Let $\gamma := \deg f = \max\{i + j \mid c_{ij} \neq 0\}$ be the total degree of f. We have $\alpha \leq \beta \leq \gamma$.

We denote by $n := \sharp\{f \leqslant z\}$ the number of values of f up to some level z, and by $m := \sharp \mathrm{Min}\{f > z\}$ the length of the priority queue at level z.

Proposition 21. *Semimonotone enumeration of the set $\{f \leqslant z\}$ requires memory $m \in \Theta(\sqrt[\beta]{z})$ whereas bimonotone enumeration requires memory $m \in O(\sqrt[\gamma]{z})$.*

Whenever $\beta < \gamma$, bimonotone enumeration is thus significantly better than semimonotone enumeration. As an example, for $f(a, b) = a^4 + a^3 b^4 + b^5$ semimonotone enumeration requires memory $m \in \Theta(\sqrt[5]{z})$, while bimonotone enumeration requires only $m \in O(\sqrt[7]{z})$.

Proof. Assuming $\beta \geq \alpha$, it is advantageous to sort by a in the semimonotone enumeration algorithm. In this case we see that $m = 2 + b$ where b satisfies $f_2(b) \leq z < f_2(b + 1)$. We have $f_2(b) \sim cb^\beta$ with some leading coefficient $c > 0$, whence $m \sim \sqrt[\beta]{z/c}$.
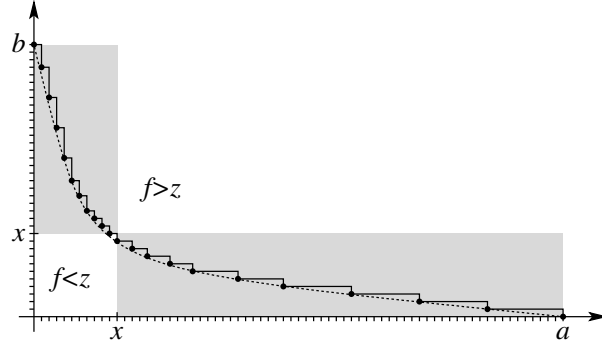

FIGURE 4. Estimating the size of the set $\mathrm{Min}_{\leqslant}\{f \geq z\}$

Evaluating f on the diagonal, we find $f(x, x) = d_0 + d_1 x + \cdots + d_\gamma x^\gamma$ with non-negative coefficients $d_k \geq 0$ and $d_\gamma > 0$. For $x = 1 + \lfloor \sqrt[\gamma]{z/d_\gamma} \rfloor$ we obtain $f(x, x) > z$. This is illustrated in Figure 4, where the dotted line corresponds to $f = z$, and black dots represent the elements of $\mathrm{Min}_{\leqslant}\{f \geq z\}$. We conclude that $m \leq 1 + 2x$, whence $m \in O(\sqrt[\gamma]{z})$. $\qquad\square$

5.2. **Asymptotic bounds.** In order to express the required memory m in terms of the number n of enumerated values, we wish to relate n and z. For $z \to \infty$ we can replace counting points $(a, b) \in \mathbb{N}^2$ satisfying $f(a, b) \leq z$ by the Lebesgue measure of the set $\{(x, y) \in \mathbb{R}_+^2 \mid f(x, y) \leq z\}$.

Proposition 22. *Let $f \colon \mathbb{R}_+^2 \to \mathbb{R}_+$ be a polynomial function given by*

$$f(x, y) = \sum_{(i,j) \in K} c_{ij} x^i y^j \qquad \text{with } c_{ij} > 0 \text{ for all indices } (i, j) \in K.$$

With f we associate the convex polygon $D = \{(u, v) \in \mathbb{R}^2 \mid iu + jv \leq 1 \text{ for all } (i, j) \in K\}$. Suppose that f is proper in the sense that for all $z \in \mathbb{R}_+$ the set

$$\{f \leq z\} = \{(x, y) \in \mathbb{R}_+^2 \mid f(x, y) \leq z\}$$

is compact. Then its Lebesgue measure satisfies $\lambda(\{f \leq z\}) \in \Theta(z^\delta \log(z)^d)$ where

$$\delta := \max\{u + v \mid (u, v) \in D\}$$

and d is the dimension of the set where this maximum is attained: either $d = 0$ for a vertex, or $d = 1$ for a segment. (See Figure 5 below for examples.)

The proof is a nice application of the so-called "tropical" approach. The idea is to identify $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$ and $\hat{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$ via the natural logarithm $\log \colon \mathbb{R}_+ \to \hat{\mathbb{R}}$, and to formally replace the semiring $(\mathbb{R}_+, +, \cdot)$ by the semiring $(\hat{\mathbb{R}}, \max, +)$. Of course, we have $\log(x \cdot y) = \log x + \log y$ but for $\log(x + y)$ we only obtain an inequality,

$$\max(\log x, \log y) \leq \log(x + y) \leq \log 2 + \max(\log x, \log y).$$

This means that $\log \colon \mathbb{R}_+ \to \hat{\mathbb{R}}$ is a *quasi-isomorphism*, i.e., its failure to be an isomorphism is bounded by some constant. For asymptotic arguments this is usually sufficient.

Proof of Proposition 22. As a logarithmic analogue of f we define

$$\hat{f} \colon \hat{\mathbb{R}}^2 \to \hat{\mathbb{R}}, \quad \hat{f}(\hat{x}, \hat{y}) := \max_{(i,j) \in K} (i\hat{x} + j\hat{y}).$$

We can choose a constant $c \in \mathbb{R}_+$ such that $c_{ij} \geq e^{-c}$ and $(\sharp K) \cdot c_{ij} \leq e^{+c}$ for all $(i, j) \in K$. A small calculation then shows that

$$\left| \log f(x, y) - \hat{f}(\log x, \log y) \right| \leq c.$$

The measure of the set $\{f \leq z\}$ equals the integral over the associated indicator function $[f \leq z]$ and we can apply the change of variables $\hat{x} = \log x$, $\hat{y} = \log y$, $\hat{z} = \log z$:

$$F(z) := \int_{\mathbb{R}_+^2} [f(x, y) \leq z] \, dx \, dy = \int_{\mathbb{R}^2} [\log f(\hat{x}, \hat{y}) \leq \hat{z}] \exp(\hat{x} + \hat{y}) \, d\hat{x} \, d\hat{y}.$$

It is easier to calculate this integral with $\hat{f}$ instead of f, so let us do this first. Since $\hat{f}$ is homogeneous, we perform another change of variables $\hat{x} = u\hat{z}$ and $\hat{y} = v\hat{z}$ to obtain:

$$\hat{F}(z) := \int_{\mathbb{R}^2} \left[\hat{f}(\hat{x}, \hat{y}) \leq \hat{z} \right] \exp(\hat{x} + \hat{y}) \, d\hat{x} \, d\hat{y} = \hat{z}^2 \int_{\mathbb{R}^2} \left[\hat{f}(u, v) \leq 1 \right] \exp(u\hat{z} + v\hat{z}) \, du \, dv.$$

We are now integrating over the convex polygon $D := \{(u, v) \in \mathbb{R}^2 \mid \hat{f}(u, v) \leq 1\}$. The asymptotic behaviour of $\log \hat{F}(z)$ is easy to understand: for $z \to \infty$ we obtain

$$\frac{\log \hat{F}(z)}{\log z} = \log \left[\hat{z}^{2/\hat{z}} \left(\int_D \exp(u + v)^{\hat{z}} \, du \, dv \right)^{1/\hat{z}} \right] \to \delta.$$

For $z \to \infty$ the first factor $\hat{z}^{2/\hat{z}} \to 1$ plays no rôle. The remaining factor is the $\hat{z}$-norm $\| \exp(u + v) \|_{\hat{z}}$ and tends to the sup-norm $\| \exp(u + v) \|_\infty = \exp(\delta)$ for $\hat{z} \to \infty$.

This shows that $\log \hat{F}(z) \sim \delta \log z$, but does not yet suffice to imply $\hat{F}(z) \sim z^\delta$ for $z \to \infty$. We thus have a closer look at the quotient

$$\frac{\hat{F}(z)}{z^\delta} = \log(z)^2 \int_D z^{(u+v-\delta)} \, du \, dv.$$

We change variables $u = \frac{\delta - t}{2} - s$ and $v = \frac{\delta - t}{2} + s$ so that $u + v = \delta - t$ and $du \, dv = ds \, dt$:

$$\int_D z^{(u+v-\delta)} \, du \, dv = \int_0^\infty \ell(t) z^{-t} dt$$

where $\ell(t)$ is the length of the segment $\{(u, v) \in D \mid u + v = \delta - t\}$. Since D is a polygon, there exist $a_0, a_1 \geq 0$ and $T > 0$ such that $\ell(t) = a_0 + a_1 t$ for all $t \in [0, T]$. We thus find

$$\int_0^\infty \ell(t) z^{-t} dt \sim a_0 \log(z)^{-1} + a_1 \log(z)^{-2} \qquad \text{for } z \to \infty.$$

Notice that $a_0 = 0$ if and only if the maximum $u + v = \delta$ is attained in a single vertex. We thus obtain $\hat{F}(z) \in \Theta(z^\delta \log(z)^d)$ where δ is the maximum of $u + v$ on D and d is the dimension of the maximising set. Since $\hat{F}(e^{-c} z) \leq F(z) \leq \hat{F}(e^{+c} z)$, we conclude that $F(z) \in \Theta(z^\delta \log(z)^d)$. $\qquad \square$

Remark 23. It is clear that the proposition and its proof generalize to proper polynomial functions $f \colon \mathbb{R}_+^n \to \mathbb{R}_+$ with non-negative coefficients, in any number n of variables. We have concentrated on $n = 2$, which is the case of interest to us here.

Example 24. For $f(x, y) = x^4 + y^5$ the set $\{\hat{f} \leq 1\}$ is depicted in Figure 5 on the left. Here we obtain $\delta = \frac{9}{20}$ and $d = 0$ because the maximum is attained in a single vertex.

The figure in the middle shows $\{\hat{f} \leq 1\}$ for $f(a, b) = a^4 + a^3 b^3 + b^5$. Here $\delta = \frac{1}{3}$ and $d = 1$, because the maximum is attained on a segment, so that $n \in \Theta(z^{1/3} \log z)$.
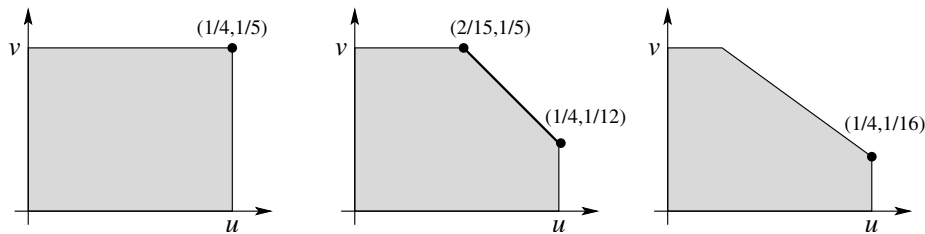

FIGURE 5. Maximizing $u + v$ under the constraint $\hat{f}(u, v) \leq 1$

The figure on the right shows $\{\hat{f} \leq 1\}$ for $f(a,b) = a^4 + a^3b^4 + b^5$. Here we find $\delta = \frac{5}{16}$, which means that $z \in \Theta(n^{16/5})$. According to Proposition 21, semimonotone enumeration requires memory $m \in \Theta(z^{1/5})$, whence $m \in \Theta(n^{16/25})$.

Notice, in particular, that $\frac{16}{25} > \frac{1}{2}$. This illustrates that, unlike bimonotone enumeration, semimonotone enumeration cannot guarantee the memory bound $m \in O(\sqrt{n})$.

Corollary 25. *The semimonotone enumeration algorithm guarantees the memory bound $m \leq n + 1$, and as a uniform bound the exponent 1 is best possible: enumerating the values of $f(a,b) = a^\alpha + a^\alpha b^\beta + b^\beta$ with $\alpha < \beta$ requires memory $m \in \Theta(n^{\alpha/\beta})$.* $\qquad\square$

Proof. According to Proposition 22 the number of enumerated values up to level z is $n \in \Theta(z^\delta)$ with $\delta = 1/\alpha$, and thus $z \in \Theta(n^\alpha)$. According to Proposition 21 the required memory is $m \in \Theta(z^{1/\beta})$. We conclude that $m \in \Theta(n^{\alpha/\beta})$. $\qquad\square$

5.3. **Constant factors.** Proposition 21 exhibits many polynomial functions where bimonotone enumeration is clearly worth the effort. Depending on the envisaged application and the given function f, a finer analysis and a more modest conclusion may be necessary:

Example 26. Consider polynomials of the form $f(a,b) = p(a) + q(b)$, for which semimonotone enumeration was initially devised [2, 1]. We obtain $n \in \Theta(z^\delta)$ with $\delta = \frac{1}{\alpha} + \frac{1}{\beta}$, as already remarked in the proof of Proposition 18. Assuming $\alpha \leq \beta$, bimonotone and semimonotone enumeration both require memory $m \in \Theta(n^\varepsilon)$ with $\varepsilon = \frac{\alpha}{\alpha+\beta}$.

Even if memory requirements are of the same order of magnitude, we can usually expect to gain a constant factor with the bimonotone algorithm:

Example 27. Reconsider $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ defined by $f(a,b) = a^\gamma + b^\gamma$ with $\gamma > 1$. In this case semimonotone enumeration requires memory $m \sim \sqrt[\gamma]{z}$, whereas bimonotone enumeration requires memory $m \sim c_\gamma \cdot \sqrt[\gamma]{z}$ with a factor $c_\gamma = 2(1 - \sqrt[\gamma]{1/2}) < 1$.

Proof. For $a = b = \lfloor \sqrt[\gamma]{z} \rfloor$ we have $f_1(a) = f_2(b) \leq z$ and $f_1(a+1) = f_2(b+1) > z$, and semimonotone enumeration requires memory $\sharp \mathrm{Min}_\leq \{f > z\} = 2 + b \sim \sqrt[\gamma]{z}$.

Choosing $x = y = \lfloor \sqrt[\gamma]{z/2} \rfloor$ we find $f(x,y) \leq z$ and $f(x+1, y+1) > z$. As indicated in Figure 3, we have $m \sim (a - x) + (b - y) \sim c_\gamma \cdot \sqrt[\gamma]{z}$. $\qquad\square$

Though less impressive, for practical applications even a constant factor may be a welcome improvement: reducing memory consumption means that we can scale to considerably larger problems before running out of RAM. In our example we have $c_2 \approx 0.59$, $c_3 \approx 0.41$, $c_4 \approx 0.32$, $c_5 \approx 0.26$, and $c_\gamma \to 0$ for $\gamma \to \infty$.

6. Parallelization

Let us reconsider the application of sorted enumeration to a diophantine equation $f(a,b) = g(c,d)$, where $f, g \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$ are proper bimonotone functions. Suppose we are looking for solutions $x = (a,b)$, $y = (c,d)$ with values in some large interval $z_{\min} \leq f(x) = g(y) < z_{\max}$. This problem can be split into s independent subproblems, namely searching solutions with $z_{k-1} \leq f(x) = g(y) < z_k$, where $z_{\min} = z_0 < z_1 < z_2 < \cdots < z_s = z_{\max}$ is a subdivision of our search interval. This allows us to distribute the search on several computers in parallel.

6.1. **The initialization algorithm.** To put the parallelization idea into practice, Algorithm 5, stated below, initializes the enumeration stream to begin at level z. Graphically speaking, it traces the contour of $X = \{f \geq z\}$ in order to determine the set of minimal elements $M = \operatorname{Min} X$. From M we can then immediately build up the priority queue $F = f(M)$.

As usual we require that $f \colon A \times B \to Z$ be a proper bimonotone map. For simplicity we first assume that both A and B are infinite. (We will treat the general case in the next paragraph.) As before the successor function is denoted by $a \mapsto \sigma a$ and $b \mapsto \sigma b$, respectively. We also use the predecessor function, denoted by $b \mapsto \pi b$.

Algorithm 5   Constructing the set of minima on $X = A \times B$

Requires: a proper bimonotone function $f \colon A \times B \to Z$
Input: a level $z \in Z$
Output: the list of minima
              $\operatorname{Min}\{\, x \in X \mid f(x) \geqslant z \,\} = \langle\, (a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)\,\rangle$
Ensures: $a_1 < a_2 < \cdots < a_m$ and $b_1 > b_2 > \cdots > b_m$

1: Initialize $M \leftarrow \emptyset$ and $a \leftarrow a_{\min}$, $b \leftarrow b_{\min}$
2: **while** $f(a, b) < z$ **do** $b \leftarrow \sigma b$ **end while**
3: Insert (a, b) at the end of the list M; continue with $a \leftarrow \sigma a$, $b \leftarrow \pi b$
4: **while** $b \geqslant b_{\min}$ **do**
5:    **while** $f(a, b) < z$ **do** $a \leftarrow \sigma a$ **end while**
6:    **while** $b > b_{\min}$ and $f(a, \pi b) \geq z$ **do** $b \leftarrow \pi b$ **end while**
7:    Insert (a, b) at the end of the list M; continue with $a \leftarrow \sigma a$, $b \leftarrow \pi b$
8: **end while**
9: **return** M

The reader is invited to apply Algorithm 5 to the example given in Figure 2, in order to see how it traces the contour of $X = \{f \geq z\}$. By the way, the method applies to any set $X \subset A \times B$ that is saturated and has finite complement. We shall give a detailed proof in the more general situation of Algorithm 7 below.

Remark 28. The loop in line 2 determines $b \leftarrow \min\{\, b \in B \mid f(a_{\min}, b) \geqslant z \,\}$. This could, of course, be improved by replacing the linear search with a binary search, provided that b can easily be incremented and decremented by integer values. The same holds for the loops in lines 5 and 6. This optimization is straightforward to implement whenever the application requires it.

Remark 29. Let $M = \langle\, (a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)\,\rangle$ be the list of minima. Then building a priority queue from M requires time $O(m \log n)$. Moreover, let $k = \#[a_{\min}, a_m]$ and $l = \#[b_{\min}, b_1]$, with $k \geq l$, say. Then $n \geq k \geq l \geq m$. Constructing the list M itself requires time $O(k \log n)$ using linear search, and $O(m \log^2 n)$ using binary search. We cannot expect to do much better, because constructing a list of length m requires at least m iterations.

6.2. **Applications.** Having initialized M and F, we can apply the bimonotone enumeration algorithm to produce a sorted enumeration $x_1, x_2, \ldots$ of the set $\{f \geq z_{k-1}\}$. Applying the same method to g, we can produce a sorted enumeration $y_1, y_2, \ldots$ of $\{g \geq z_{k-1}\}$. We can thus search for solutions $f(x) = g(y)$ starting at level z_{k-1} and ending at level z_k.

Expected speed-up. Concerning time requirements, initialization entails a reasonably small overhead, so we can expect an amortized speed-up by a factor s. For each $k = 1, \ldots, s$, computer number k manages its own priority queues of length $O(\sqrt{n})$. in order to produce enumeration streams for f and g, with values ranging from z_{k-1} to z_k. As before, advancing from position n to position $n+1$ takes time $O(\log^2 n)$.

Robustness. The initialization procedure is already very useful on a single computer, since it can make implementations much more robust: it is possible to continue searching, without much loss, after a shut-down or a power failure. This is particularly important when carrying out a long-term search.

7. Enumerating bimonotone domains

Suppose we want to enumerate the values of a *symmetric* bimonotone function $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$, that is, $f(a,b) = f(b,a)$ for all $a, b \in \mathbb{N}$. It is often desirable to enumerate only pairs (a,b) with $a \geqslant b$. In other words, we wish to *restrict* f to the domain $X = \{\, (a,b) \in \mathbb{N} \times \mathbb{N} \mid a \geqslant b \,\}$ and enumerate only the values of $f \colon X \to \mathbb{Z}$. Figure 6 shows a possible configuration during bimonotone enumeration.
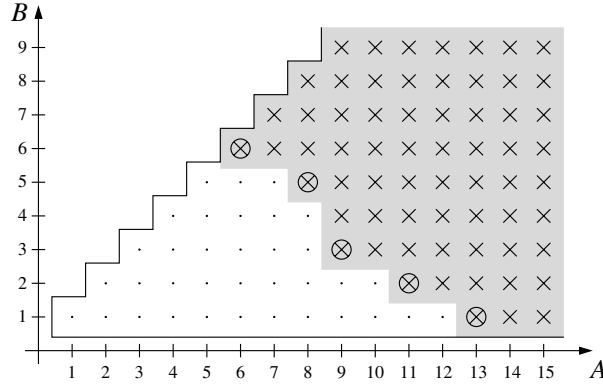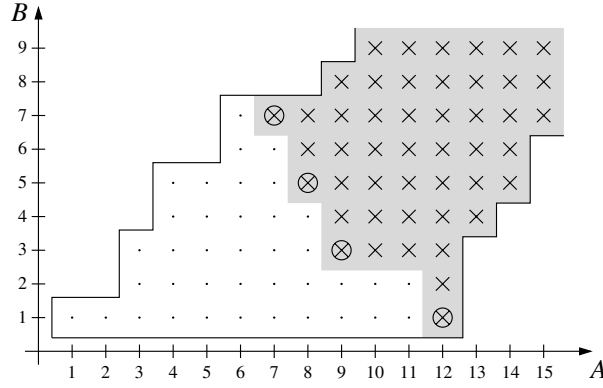

FIGURE 6. Enumerating the domain $X = \{\, (a,b) \in \mathbb{N} \times \mathbb{N} \mid a \geqslant b \,\}$

It is straightforward to adapt bimonotone enumeration (Algorithm 4) and initialization (Algorithm 5) to such a domain X. Still other restrictions are possible, for example $X = \{\, (a,b) \in \mathbb{N} \times \mathbb{N} \mid a \leq b \text{ and } b \leq 2a \,\}$ or more complicated cases such as $X = \{\, (a,b) \in \mathbb{N} \times \mathbb{N} \mid a \leq b \text{ and } b^2 \leq 1 + 10a \,\}$. This raises the question as to what are "reasonable" domains $X \subset A \times B$ to which Algorithms 4 and 5 can be efficiently applied.

7.1. **Bimonotone domains.** As usual we assume that A and B are isotonic to finite intervals or to the natural numbers. Figure 7 shows a domain $X \subset A \times B$ which will turn out to be well suited to bimonotone enumeration. Graphically speaking, it is bounded by the graphs of two non-decreasing functions $\alpha \colon A \to B$ and $\beta \colon B \to A$. We will show that this condition suffices to adapt our algorithms to work on the domain X rather than the entire product $A \times B$.

We say that $X \subset A \times B$ is *bounded* by functions $\alpha \colon A \to B$ and $\beta \colon B \to A$ if

$$X = \{\, (a,b) \in A \times B \mid a \geqslant \beta(b) \text{ and } b \geqslant \alpha(a) \,\}.$$

FIGURE 7. Enumerating a bimonotone domain $X \subset A \times B$

For example, the domain X of Figure 7 is bounded by $\alpha(1) = \cdots = \alpha(12) = 1$, $\alpha(13) = 4$, $\alpha(14) = 5$, $\alpha(15) = 7$, and $\beta(1) = 1$, $\beta(2) = \beta(3) = 3$, ..., $\beta(9) = 10$.

Definition 30. We say that a domain $X \subset A \times B$ is *bimonotone* if it is bounded by two functions $\alpha \colon A \to B$ and $\beta \colon B \to A$ such that:

   (1) The functions α and β are non-decreasing, that is,
       $a \leqslant a'$ implies $\alpha(a) \leqslant \alpha(a')$, and $b \leqslant b'$ implies $\beta(b) \leqslant \beta(b')$.
   (2) We have $\beta(\alpha(a)) \leqslant a$ for all $a \in A$, with equality only for $a = a_{\min}$,
       and $\alpha(\beta(b)) \leqslant b$ for all $b \in B$, with equality only for $b = b_{\min}$.

Condition (2) ensures that $(a, \alpha(a)) \in X$ for each $a \in A$, and $(\beta(b), b) \in X$ for each $b \in B$. In particular, α and β are determined by X via

$$\alpha(a) = \min\{\, b \in B \mid (a, b) \in X \,\},$$
$$\beta(b) = \min\{\, a \in A \mid (a, b) \in X \,\}.$$

Moreover, $(a_{\min}, b_{\min})$ is the smallest element of X. If both A and B are finite, then $(a_{\max}, b_{\max})$ is the greatest element of X.

The definition of X via bounding functions is easy to formulate and well suited to implementation. It can also be reformulated in more geometric terms:

Proposition 31. *A domain $X \subset A \times B$ is bimonotone if and only if it satisfies* $\mathrm{pr}_1 X = A$ *and* $\mathrm{pr}_2 X = B$ *and the following two properties:*

   (1′) *If (a_1, b_1) and (a_2, b_2) in X satisfy $a_1 \leqslant a_2$ and $b_2 \leqslant b_1$, then X contains the entire rectangle $\{\, (a, b) \in A \times B \mid a_1 \leqslant a \leqslant a_2 \text{ and } b_2 \leqslant b \leqslant b_1 \,\}$.*
   (2′) *If (a_1, b_1) and (a_2, b_2) in X satisfy $a_1 \leqslant a_2$ and $b_1 \leqslant b_2$, then we can go from (a_1, b_2) to (a_2, b_2) within X by repeatedly incrementing a and b.* □

The proof is not difficult and will be omitted.

7.2. **Bimonotone enumeration.** We are now in position to generalize our enumeration algorithm to a bimonotone domain. As before, Algorithm 6 processes a bidirectional list M and a priority queue F.

Algorithm 6   Sorted enumeration of a bimonotone domain

Requires: a bimonotone domain $X \subset A \times B$ and a proper bimonotone function
$\qquad f \colon X \to Z$

Output: an enumeration of X, monotone with respect to f

1: Initialize $M \leftarrow \langle\, (a_{\min}, b_{\min})\, \rangle$ and $F \leftarrow \langle\, f(a_{\min}, b_{\min})\, \rangle$.
2: **while** F is non-empty **do**
3:    Remove a minimal element $f(a, b)$ from F and output (a, b).
4:    **if** (a, b) is the last element of the list M **then**
5:       **if** $(\sigma a, b) \in X$ **then** insert $(\sigma a, b)$ into M and $f(\sigma a, b)$ into F **end if**
6:    **else**
7:       Let (a^*, b_*) be the successor of (a, b) in the list M.
8:       **if** $\sigma a < a^*$ **then** insert $(\sigma a, b)$ into M and $f(\sigma a, b)$ into F **end if**
9:    **end if**
10:    **if** (a, b) is the first element of the list M **then**
11:       **if** $(a, \sigma b) \in X$ **then** insert $(a, \sigma b)$ into M and $f(a, \sigma b)$ into F **end if**
12:    **else**
13:       Let (a_*, b^*) be the predecessor of (a, b) in the list M.
14:       **if** $\sigma b < b^*$ **then** insert $(a, \sigma b)$ into M and $f(a, \sigma b)$ into F **end if**
15:    **end if**
16:    Remove (a, b) from the list M.
17: **end while**

Proposition 32. *Algorithm* 6 *is correct.*

Proof. The proof is essentially the same as for Algorithm 4. There are, however, some modifications when updating the list M and the priority queue F:

- If the current minimum (a, b) is somewhere in the middle of the list M, then the previous arguments apply without change, because we still have

$$\{(a, b)\}^\# \smallsetminus \{(a, b)\} = \{\, (a, \sigma b), (\sigma a, b)\, \}^\#.$$

- If (a, b) is at the end of the list, then possibly $(\sigma a, b) \notin X$: in this case we have $\{(a, b)\}^\# \smallsetminus \{(a, b)\} = \{(a, \sigma b)\}^\#$, so we discard $(\sigma a, b)$.
- If (a, b) is at the beginning of the list, then possibly $(a, \sigma b) \notin X$: in this case we have $\{(a, b)\}^\# \smallsetminus \{(a, b)\} = \{(\sigma a, b)\}^\#$, so we discard $(a, \sigma b)$.
- If ever $M = \langle\, (a, b)\, \rangle$ and neither $(a, \sigma b)$ nor $(\sigma a, b)$ is in X, then (a, b) is the greatest element of X and the algorithm terminates correctly.

Since f is proper, every element $(a, b) \in X$ will eventually be enumerated. $\square$

7.3. **Initialization on a bimonotone domain.** As for the unrestricted case $X = A \times B$, we want to formulate an initialization algorithm for a proper bimonotone function $f \colon X \to Z$ defined on some bimonotone domain $X \subset A \times B$. The idea is essentially the same: Algorithm 7 traces the contour of $X(z) = \{\, x \in X \mid f(x) \geqslant z\, \}$ to construct the list $M = \operatorname{Min} X(z)$ of its minimal elements.

Remark 33. The loops in lines 3, 7, and 9 implement linear searches. This can be improved by a binary search whenever the application requires such optimization.

Proposition 34. *Algorithm* 7 *is correct.*

Algorithm 7   Constructing the set of minima on a bimonotone domain

---

Requires: a bimonotone domain $X \subset A \times B$ and a proper bimonotone function
$f \colon X \to Z$
Input: a level $z \in Z$
Output: the list of minima
$\mathrm{Min}\{\, x \in X \mid f(x) \geqslant z \,\} = \langle\, (a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m) \,\rangle$
Ensures: $a_1 < a_2 < \cdots < a_m$ and $b_1 > b_2 > \cdots > b_m$

1: Initialize $M \leftarrow \emptyset$ and $a \leftarrow a_{\min}$, $b \leftarrow b_{\min}$
2: **while** $(a, b) \in X$ and $f(a, b) < z$ **do**
3:     **while** $f(a, b) < z$ and $(a, \sigma b) \in X$ **do** $b \leftarrow \sigma b$ **end while**
4:     **if** $f(a, b) < z$ **then** $a \leftarrow \sigma a$ **end if**
5: **end while**
6: **while** $(a, b) \in X$ **do**
7:     **while** $(a, \pi b) \in X$ and $f(a, \pi b) \geqslant z$ **do** $b \leftarrow \pi b$ **end while**
8:     Insert (a, b) at the end of the list M; continue with $a \leftarrow \sigma a$, $b \leftarrow \pi b$
9:     **while** $(a, b) \in X$ and $f(a, b) < z$ **do** $a \leftarrow \sigma a$ **end while**
10: **end while**
11: **return** M

Proof. The first loop (lines 1–5) finds an element $(a, b) \in X(z)$ with minimal a. Beginning with $a \leftarrow a_{\min}$ and $b \leftarrow b_{\min}$, we repeatedly increment b in order to arrive at $f(a, b) \geqslant z$. If this is not possible within X, then the candidate a is eliminated, and we continue with $a \leftarrow \sigma a$. If we never run out of the domain X, then we finally end up with $f(a, b) \geqslant z$, because a or b increase and f is proper.

The only obstacle occurs when $f(a, b) < z$ but neither $(a, \sigma b)$ nor $(\sigma a, b)$ are in X. But in this case we have reached the greatest element of X, hence $f(x) < z$ for all $x \in X$. Thus $X(z) = \emptyset$ and we correctly return the empty list $M = \emptyset$. In any case, the first loop terminates with either $(a, b) \notin X$ or $f(a, b) \geqslant z$, as desired.

When arriving at line 7 we know that $(a, b) \in X(z) \smallsetminus M^{\#}$, and a is minimal with this property. The loop in line 7 minimizes b, so we know that (a, b) is a minimal element of $X(z)$. We thus add (a, b) to our list M and continue with $a \leftarrow \sigma a$ and $b \leftarrow \pi b$. We then repeatedly increment a in order to arrive at $f(a, b) \geqslant z$. If this is not possible in X, then $X(z) = M^{\#}$ by the rectangle condition (1'), so we have found all minimal elements of $X(z)$. Otherwise, we obtain $(a, b) \in X(z) \smallsetminus M^{\#}$, and a is minimal with this property. We can thus reiterate by looping back to line 7.

During each iteration, a is strictly increasing while b is strictly decreasing. We conclude that the second loop terminates and produces the list M of minima, as desired, ordered in the sense that $a_1 < a_2 < \cdots < a_m$ and $b_1 > b_2 > \cdots > b_m$. $\qquad\square$

8. APPLICATIONS TO DIOPHANTINE ENUMERATION

Algorithms 6 and 7 for bimonotone enumeration have been implemented as a class template in C++. This seems to be a good compromise between general applicability, ease of use, and high performance. The source files are available on the author's homepage:

<p align="center"><code>http://www-fourier.ujf-grenoble.fr/~eiserm/software</code></p>

As an illustration of sorted enumeration, let us mention searching multiple values of a polynomial function $f \colon \mathbb{N} \times \mathbb{N} \to \mathbb{Z}$, $f(a, b) = \sum_{i,j} c_{ij} a^i b^j$ with non-negative

coefficients $c_{ij} \in \mathbb{N}$. The cited implementation has been successfully tested to reproduce some known results taken from Richard Guy's *Unsolved problems in number theory* [4].

8.1. **The quest for the sixth taxicab number.** As an illustrative example we briefly sketch the taxicab problem. The kth taxicab number, denoted by $\mathrm{taxicab}(k)$, is the least positive integer that can be expressed as a sum of two positive cubes in k distinct ways, up to order of summands. That is, it is the smallest k-fold value of $f(a,b) = a^3 + b^3$ defined on $X = \{\, (a,b) \in \mathbb{N} \times \mathbb{N} \mid 1 \le a \le b \,\}$.

G. H. Hardy and E. M. Wright [5, Thm. 412] proved that, for every $k \ge 1$, there exist such k-fold values. This guarantees the existence of a least k-fold value, that is, the kth taxicab number. Unfortunately, the construction given in the proof is of no help in finding the *least* k-fold value. Apart from (variants of) exhaustive search, no such method is known today. The first taxicab number is trivially

$$\mathrm{taxicab}(1) = 2 = 1^3 + 1^3.$$

The next taxicab numbers are:

$$\mathrm{taxicab}(2) = 1729 = 1^3 + 12^3 = 9^3 + 10^3,$$

(re)discovered by Ramanujan according to Hardy's famous anecdote, but previously published by Bernard Frénicle de Bessy in 1657,

$$\mathrm{taxicab}(3) = 87\,539\,319$$
$$= 167^3 + 436^3 = 228^3 + 423^3 = 255^3 + 414^3,$$

discovered by John Leech [8] in 1957,

$$\mathrm{taxicab}(4) = 6\,963\,472\,309\,248 = 2421^3 + 19083^3 = 5436^3 + 18948^3$$
$$= 10200^3 + 18072^3 = 13322^3 + 16630^3,$$

discovered by E. Rosenstiel, J.A. Dardis, and C.R. Rosenstiel [9] in 1991,

$$\mathrm{taxicab}(5) = 48\,988\,659\,276\,962\,496 = 38787^3 + 365757^3 = 107839^3 + 362753^3$$
$$= 205292^3 + 342952^3 = 221424^3 + 336588^3 = 231518^3 + 331954^3,$$

discovered independently by D.W. Wilson [10] in 1997 and shortly afterwards by D.J. Bernstein [1] in 1998. Finally, the smallest known 6-fold value is

$$T = 24\,153\,319\,581\,254\,312\,065\,344$$
$$= 28906206^3 + 582162^3 = 28894803^3 + 3064173^3 = 28657487^3 + 8519281^3$$
$$= 27093208^3 + 16218068^3 = 26590452^3 + 17492496^3 = 26224366^3 + 18289922^3,$$

found by R.L. Rathbun in 2002. Is this actually the sixth taxicab number, or is there a smaller solution?

8.2. **Feasibility of an exhaustive search.** In order to verify that T is indeed the smallest 6-fold value, there are exactly $n = 369\,039\,037\,733\,393 < 4 \cdot 10^{14}$ pairs $(a,b) \in \mathbb{N} \times \mathbb{N}$ to be checked with $a^3 + b^3 \le T$ and $a \le b$. Such counting results can easily be obtained from Algorithm 7 tracing the contour of the set $X = \{f \le z\}$: as a by-product, the initialization can be used to determine the sizes $n = \sharp\{f \le z\}$ and $m = \sharp \mathrm{Min}\{f > z\}$.

Memory requirements are, fortunately, no problem. In the worst case we would have to check all n parameters, which would finally build up a priority queue of

size $m = 5\,963\,352 < 6 \cdot 10^6$. Notice that each entry requires 32 bytes: 12 bytes for the value $f(a, b)$, 4 bytes for a and 4 bytes for b, plus 4 bytes for each of the three pointers. In the worst case the priority queue thus requires 180 megabytes of memory, which fits nicely in a PC with 256 megabytes RAM. Such memory requirements seem acceptable; on today's PCs such a task can reasonably be run in the background.

Time requirements, however, are on the edge of being feasible. Updating a priority queue of $2 \cdot 10^6$ entries, say, takes about 4000 CPU cycles. On a PC running at 2GHz, we can expect to process about $500\,000$ steps per second, that is, around $4 \cdot 10^{10}$ steps per day. This is not too far away from $4 \cdot 10^{14}$, but on a single computer the search would still require about $10\,000$ days, roughly 25 years. On 25 computers, however, we would be done within a year, possibly earlier.

Partial results. Up to June 2005, I have run the search on a few available PCs at the Institut Fourier, but the use of parallelization has still been rather limited (to a dozen PCs). As a result I obtained the lower bound $\mathrm{taxicab}(6) > 5 \cdot 10^{20}$ by sorted enumeration of the $2.8 \cdot 10^{13}$ smallest values of $f(a, b) = a^3 + b^3$. (At a speed of $500\,000$ values per second this takes about 650 days on a single computer.) This leaves us with the inequality

$$5 \cdot 10^{20} < \mathrm{taxicab}(6) \le T \approx 2.42 \cdot 10^{22}.$$

It will now be a matter of sufficient hardware and patience to find the exact answer.

Acknowledgement

References

1. D. J. Bernstein, *Enumerating solutions to $p(a) + q(b) = r(c) + s(d)$*, Math. Comp. **70** (2001), no. 233, 389–394. MR2001f:11203
2. R. L. Ekl, *Equal sums of four seventh powers*, Math. Comp. **65** (1996), no. 216, 1755–1756. MR97a:11050
3. R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete mathematics*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1989. MR1001562 (91f:00001)
4. R. K. Guy, *Unsolved problems in number theory*, third ed., Problem Books in Mathematics, Springer-Verlag, New York, 2004. MR2076335
5. G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, third ed., The Clarendon Press, Oxford University Press, New York, 1954. MR0067125 (16,673c)
6. D. E. Knuth, *The art of computer programming, volume 1: fundamental algorithms*, second ed., Addison-Wesley Publishing Co., Reading, Massachusetts, 1969. MR0286317 (44:3530)
7. _____, *The art of computer programming, volume 3: sorting and searching*, second ed., Addison-Wesley Publishing Co., Reading, Massachusetts, 1998. MR0445948 (56:4281)
8. J. Leech, *Some solutions of Diophantine equations*, Proc. Cambridge Philos. Soc. **53** (1957), 778–780. MR0090602 (19,837f)
9. E. Rosenstiel, J. A. Dardis, and C. R. Rosenstiel, *The four least solutions in distinct positive integers of the Diophantine equation $s = x^3 + y^3 = z^3 + w^3 = u^3 + v^3 = m^3 + n^3$*, Bull. Inst. Math. Appl. **27** (1991), no. 7, 155–157. MR1125858 (92i:11134)
10. D. W. Wilson, *The fifth taxicab number is $48\,988\,659\,276\,962\,496$*, J. Integer Seq. **2** (1999), Article 99.1.9, HTML document (electronic). MR1722364 (2000i:11195)

Institut Fourier, Université Grenoble I, France
E-mail address: Michael.Eisermann@ujf-grenoble.fr
URL: http://www-fourier.ujf-grenoble.fr/~eiserm