# Computational Fun with Sturdy and Flimsy Numbers

**Trevor Clokie**
University of Waterloo, Canada
trevor.clokie@uwaterloo.ca

**Thomas F. Lidbetter**
University of Waterloo, Canada
finnlidbetter@gmail.com

**Antonio J. Molina Lovett**
University of Waterloo, Canada
antonio@amolina.ca

**Jeffrey Shallit**
University of Waterloo, Canada
shallit@uwaterloo.ca

**Leon Witzman**
University of Waterloo, Canada
lwitzman@uwaterloo.ca

## Abstract

Following Stolarsky, we say that a natural number $n$ is *flimsy* in base $b$ if some positive multiple of $n$ has smaller digit sum in base $b$ than $n$ does; otherwise it is *sturdy*. We develop algorithmic methods for the study of sturdy and flimsy numbers.

We provide some criteria for determining whether a number is sturdy. Focusing on the case of base $b = 2$, we study the computational problem of checking whether a given number is sturdy, giving several algorithms for the problem. We find two additional, previously unknown sturdy primes. We develop a method for determining which numbers with a fixed number of 0's in binary are flimsy. Finally, we develop a method that allows us to estimate the number of $k$-flimsy numbers with $n$ bits, and we provide explicit results for $k = 3$ and $k = 5$. Our results demonstrate the utility (and fun) of creating algorithms for number theory problems, based on methods of automata theory.

**2012 ACM Subject Classification** Theory of computation → Grammars and context-free languages; Theory of computation → Dynamic programming

**Keywords and phrases** sturdy number, flimsy number, context-free grammar, finite automaton, enumeration

**Digital Object Identifier** 10.4230/LIPIcs.FUN.2021.10

**Related Version** The full paper is available at `https://arxiv.org/abs/2002.02731`.

**Supplementary Material** Implementations of our algorithms can be found in the GitHub repository `https://github.com/FinnLidbetter/sturdy-numbers`.

**Acknowledgements** We would like to thank Kenneth Stolarsky and the referees for helpful comments.

## 1 Introduction

Let $s_b(n)$ denote the sum of the digits of $n$, when expressed in base $b$. Thus, for example, $s_2(9) = 2$. A number $n$ is said to be *$k$-flimsy in base $b$* if there exists a positive integer $k$ such that $s_b(kn) < s_b(n)$. Any such $k$, if one exists, is called a *flimsy witness* for $n$. If $n$ is $k$-flimsy for some $k$, it is said to be *flimsy*. If there is no such $k$, then $n$ is said to be *sturdy in base $b$*. For example, 7 is sturdy in base 2, while 13 is flimsy, because $s_2(13) = 3 > 2 = s_2(5 \cdot 13)$. Thus 5 is a flimsy witness for 13. In this paper we examine the computational aspects of sturdy and flimsy numbers.

Sturdy and flimsy numbers were introduced by Stolarsky in 1980 [28]. For other papers on the topic, see [25, 10, 8, 6].

Many of the sequences we discuss appear in the *On-Line Encyclopedia of Integer Sequences* [27]. For example, the base-2 sturdy numbers form sequence A125121 in the OEIS, while the base-2 sturdy primes form sequence A143027. The base-2 flimsy numbers form sequence A005360, while the base-2 flimsy primes form sequence A330696. The base-10 sturdy numbers form sequence A181862, while the base-10 sturdy primes form sequence A181863. Sequence A086342 gives the value of $\min_{k\geq 1} s_2(kn)$, while sequence A143069 gives $\operatorname{argmin}_{k\geq 1} s_2(kn) = \min\{k : s_2(kn) = \min_{k\geq 1} s_2(kn)\}$.

The goal of this paper is to examine the algorithmic aspects of sturdy and flimsy numbers. The outline of the paper is as follows. In Section 2, we prove some basic properties of digit sums of multiples. In Section 3, we give a criterion for determining if a number is flimsy, and use it to find two previously unknown sturdy primes.

Next, we turn to algorithms for sturdy and flimsy numbers. A priori it is not immediately clear that it is even decidable whether a given $n$ is flimsy or sturdy. Indeed, in a recent paper by Elsholtz [11], he asks, "How can one algorithmically find a "sparse" representation of a multiple of $p$?"

More precisely, there are four computational problems worthy of study:

1. Given a positive integer $n$, decide whether it is sturdy in base $b$.
2. Compute $\operatorname{swm}_b(n) := \min_{k\geq 1} s_b(kn)$. This is the *smallest weight of a multiple*; the smallest digit sum of a multiple of $n$; if $n$ is sturdy, then $\operatorname{swm}_b(n) = s_b(n)$.
3. Compute $\operatorname{msw}_b(n) := \operatorname{argmin}_{k\geq 1} s_b(kn)$. This is the *minimum sum witness*; the smallest $k$ such that $kn$ achieves its minimum digit sum; if $n$ is sturdy, then $\operatorname{msw}_b(n) = 1$.
4. Given that $n$ is flimsy, determine $\operatorname{mfw}_b(n) := \min\{k : s_b(kn) < s_b(n)\}$. This is the *minimal flimsy witness*.

A table of these functions is given in Table 1. Here the column labeled "char" is F if the number is flimsy and S if it is sturdy.

In Sections 4–7, we discuss algorithms to solve these problems. The fastest, based on automata theory, shows that we can check whether a number $n$ is sturdy in $O(n)$ time. Section 10 gives our computational results achieved with our algorithms.

In Section 11 we give an application of automata to help characterize the flimsy numbers with a fixed number of 0's.

Finally, in Section 12, we turn to estimating the number of $k$-flimsy numbers with $n$ bits. We use techniques from formal language theory to solve the problem.

## 2 Basic properties

In this section, we prove some of the basic properties of digit sums of multiples.

We start with some notation. For $n \geq 0$, we define $(n)_b$ to be the base-$b$ representation of $n$, starting with the most significant digit. If $x$ is a string, we define $[x]_b$ to be the integer that $x$ represents when interpreted in base $b$. If $b$ is fixed, we define $\overline{x}$ to be the base-$b$ complement of $x$, that is, the string where each digit $d$ in $x$ is replaced by $b-1-d$.

▶ **Theorem 1.** *Let $b \geq 2$ be an integer, and $t$ be a positive divisor of $b$. Then for all integers $n, r \geq 1$, there exists a positive integer $j$ such that $s_b(jn) = r$ if and only if there exists a positive integer $k$ such that $s_b(ktn) = r$.*

**Proof.** For one direction, take $j = kt$.

For the other direction, assume that there exists $j \geq 1$ such that $s_b(jn) = r$. Let $k = bj/t$. Then $s_b(ktn) = s_b(bjn) = s_b(jn) = r$. ◀

**Table 1** Table of sturdy and flimsy numbers.

| $n$ | char | swm($n$) | msw($n$) | mfw($n$) | $n$ | char | swm($n$) | msw($n$) | mfw($n$) |
|---|---|---|---|---|---|---|---|---|---|
| 3 | S | 2 | 1 | - | 5 | S | 2 | 1 | - |
| 7 | S | 3 | 1 | - | 9 | S | 2 | 1 | - |
| 11 | F | 2 | 3 | 3 | 13 | F | 2 | 5 | 5 |
| 15 | S | 4 | 1 | - | 17 | S | 2 | 1 | - |
| 19 | F | 2 | 27 | 27 | 21 | S | 3 | 1 | - |
| 23 | F | 3 | 3 | 3 | 25 | F | 2 | 41 | 41 |
| 27 | F | 2 | 19 | 3 | 29 | F | 2 | 565 | 5 |
| 31 | S | 5 | 1 | - | 33 | S | 2 | 1 | - |
| 35 | S | 3 | 1 | - | 37 | F | 2 | 7085 | 7085 |
| 39 | F | 3 | 7 | 7 | 41 | F | 2 | 25 | 25 |
| 43 | F | 2 | 3 | 3 | 45 | S | 4 | 1 | - |
| 47 | F | 3 | 11 | 3 | 49 | S | 3 | 1 | - |
| 51 | S | 4 | 1 | - | 53 | F | 2 | 1266205 | 5 |
| 55 | F | 3 | 7 | 3 | 57 | F | 2 | 9 | 9 |
| 59 | F | 2 | 9099507 | 3 | 61 | F | 2 | 17602325 | 5 |
| 63 | S | 6 | 1 | - | 65 | S | 2 | 1 | - |
| 67 | F | 2 | 128207979 | 128207979 | 69 | S | 3 | 1 | - |
| 71 | F | 3 | 119 | 119 | 73 | S | 3 | 1 | - |
| 75 | S | 4 | 1 | - | 77 | F | 3 | 5 | 5 |
| 79 | F | 3 | 13 | 7 | 81 | F | 2 | 1657009 | 1657009 |
| 83 | F | 2 | 26494256091 | 395 | 85 | S | 4 | 1 | - |
| 87 | F | 3 | 3 | 3 | 89 | S | 4 | 1 | - |
| 91 | F | 3 | 3 | 3 | 93 | S | 5 | 1 | - |
| 95 | F | 3 | 5519 | 3 | 97 | F | 2 | 172961 | 172961 |
| 99 | F | 2 | 331 | 11 | 101 | F | 2 | 11147523830125 | 365 |
| 103 | F | 3 | 5 | 5 | 105 | S | 4 | 1 | - |
| 107 | F | 2 | 84179432287299 | 3 | 109 | F | 2 | 2405 | 5 |
| 111 | F | 3 | 591 | 3 | 113 | F | 2 | 145 | 145 |
| 115 | F | 3 | 571 | 9 | 117 | F | 4 | 5 | 5 |
| 119 | F | 3 | 71 | 3 | 121 | F | 2 | 297758653049289 | 9 |
| 123 | F | 4 | 19 | 3 | 125 | F | 2 | 9007199254741 | 5 |
| 127 | S | 7 | 1 | - | 129 | S | 2 | 1 | - |

We now show that in order to compute $\mathrm{swm}_b$, it suffices to consider only those arguments relatively prime to $b$.

▶ **Corollary 2.** *Write the prime factorization of $n$ as $\prod_{1\le i\le t} p_i^{e_i}$, and define $g = \prod_{p_i \mid b} p_i^{e_i}$. Then $\mathrm{swm}_b(n) = \mathrm{swm}_b(n/g)$, and $\gcd(b, n/g) = 1$.*

**Proof.** Let $p$ be any prime dividing both $b$ and $n$. From Theorem 1, we see that $\mathrm{swm}_b(n) = \mathrm{swm}_b(n/p)$. By repeatedly applying this observation, and replacing $n$ with $n/p$, we can remove from $n$ all primes dividing both $b$ and $n$, while maintaining the same value of $\mathrm{swm}_b$. At the end, the resulting $n/g$ is relatively prime to $b$. ◀

▶ **Theorem 3.** *There exists $j \ge 1$ such that $s_b(jn) = t$ if and only if there exist $t$* distinct *powers of $b$ that sum to a multiple of $n$.*

**Proof.** By Corollary 2, we may assume that $n$ is coprime with $b$.

In such cases, $b$ has finite order, say $\nu$, modulo $n$. Suppose $\sum_{i=0}^{\nu-1} c_i b^i \equiv 0 \pmod n$ where each $c_i \ge 0$ and $\sum_{i=0}^{\nu-1} c_i = t$. Then $\sum_{i=0}^{\nu-1} \sum_{j=0}^{c_i-1} b^{j\nu+i} \equiv 0 \pmod n$, and this sum consists of distinct powers of $b$. ◀

Empirical evidence suggests that if $b = 2$ and $\mathrm{swm}_b(n) = t$, then for all $i \geq 0$, some multiple of $n$ has digit sum $t + i$. However, the analogous result is false for $b = 3$. For example, $\mathrm{swm}_3(13) = 3$, but no multiple of 13 has digit sum 4. These observations are explained in the following theorem.

▶ **Theorem 4.** *Suppose $j, n$ are positive integers such that $s_b(jn) = t$. Then for all $r \geq 0$, there exists $k \geq 1$ such that $s_b(kn) = t + r(b-1)$.*

**Proof.** Assume $s_b(jn) = t$ for some $t \geq 1$. Then from Theorem 3 we know that $\sum_{i=1}^{t} b^{m_i} \equiv 0 \pmod{n}$ for some strictly increasing $m_i$ and (replacing $j$ by $bj$ if needed) we can assume $m_t \geq 1$. Now replace the high-order bit $b^{m_t}$ in this sum with the sum of $b$ terms $b^{\nu+m_t-1} + b^{2\nu+m_t-1} + \cdots + b^{(b-1)\nu+m_t-1}$, where $\nu$ is the order of $b$, modulo $n$. This has the effect removing 1 bit, while adding $b$ additional bits, and each of the $b$ new terms is congruent to $b^{m_t-1} \pmod{n}$. So we have found another multiple of $n$ with digit sum $t + b - 1$. We can repeat this transformation any number of times. ◀

▶ Remark 5. The result is optimal. Since $b-1$ divides the digit sum of any multiple of $b-1$, there is no $k \geq 1$ satisfying $b - 1 < s_b(k(b-1)) < 2(b-1)$.

## 3 Infinite classes of sturdy numbers

We first give a criterion for deciding whether a number is flimsy. This shows that Problem 1 on our list, determining whether a given positive integer is sturdy, is decidable.

▶ **Theorem 6.** *Let $n, b, j$ be positive integers, $b \geq 2$ such that $n$ divides $b^j - 1$. Then $n$ is flimsy in base $b$ if and only if $s_b(kn) < s_b(n)$ for some $k$ satisfying $1 \leq k \leq \frac{b^j-1}{n}$.*

**Proof.** One direction is easy: if $s_b(kn) < s_b(n)$ for some $k$, then $n$ is flimsy in base $b$.

For the other direction, suppose $n$ is flimsy, but $s_b(kn) \geq s_b(n)$ for all $k$ with $1 \leq k \leq \frac{b^j-1}{n}$. Let $k'$ be the smallest positive integer such that $s_b(k'n) < s_b(n)$. By assumption $k'n \geq b^j$, and so we can write $k'n = cb^j + d$ for uniquely-determined $c \geq 1$ and $0 \leq d < b^j$. Since $b^j \equiv 1 \pmod{n}$, it follows that $cb^j + d \equiv c + d \equiv 0 \pmod{n}$. Then $c + d = fn < cb^j + d = k'n$ for some integer $f$ with $1 \leq f < k'$. Thus $s_b(k'n) = s_b(cb^j + d) = s_b(c) + s_b(d) \geq s_b(c + d) = s_b(fn) \geq s_b(n)$, achieving the desired contradiction. ◀

▶ Remark 7. Since $j \leq \varphi(n)$, this together with Theorem 1 shows that sturdiness is reduced to a finite search. The result for $b = 10$ was observed by Phedotov [20].

We applied Theorem 6 to known prime factors of composite Mersenne numbers [29] and found

$$579126141132756490877 21 = \frac{2^{83} - 1}{167}$$

and

$$1035079443105516238671861923746823 4569 = \frac{2^{131} - 1}{263}$$

as previously unknown sturdy primes in base 2.

▶ **Corollary 8.** *If $b, j$ are positive integers, with $b \geq 2$, then $\frac{b^j-1}{m}$ is sturdy in base $b$ for every positive $m$ dividing $b - 1$.*

**Proof.** Let $k$ be an integer with $1 \leq k \leq m$. Then we have

$$s_b\left(k\frac{b^j-1}{m}\right) = s_b\left(\frac{k(b-1)}{m}\sum_{i=0}^{j-1} b^i\right) = kj\frac{b-1}{m} \geq j\frac{b-1}{m},$$

where we have used the fact that $k \leq m$. The result now follows by Theorem 6. ◀

▶ **Theorem 9.** *Let $n$ be sturdy in base $b \geq 2$, with $n$ dividing $b^j - 1$ for some $j \geq 1$. Fix a positive integer $r$, and define $m = n\left(\frac{b^{rj}-1}{b^j-1}\right)$. Then $m$ is sturdy in base $b$.*

**Proof.** Let $x = (n)_b$. Observe that $m = [(x\,0^{j-|x|})^{r-1}\,x]_b$, so $s_b(m) = rs_b(n)$. If $1 \leq k \leq \frac{b^j-1}{n}$, then $(km)_b$ consists of $r$ copies of $(kn)_b$ concatenated, separated by some number of 0's. So $s_b(km) = rs_b(kn) \geq rs_b(n) = s_b(m)$. The result now follows by Theorem 6. ◀

We can now get a generalization of a theorem of Stolarsky [28, Thm. 2.1].

▶ **Corollary 10.** *Let $e, k, r \geq 1$, and $b \geq 2$. Define $n = [(1^k\,0^{(e-1)k})^{r-1}\,1^k]_b = \left(\frac{b^k-1}{b-1}\right)\left(\frac{b^{rek}-1}{b^{ek}-1}\right)$. Then $n$ is sturdy in base $b$.*

**Proof.** Note that $\frac{b^k-1}{b-1}$ is sturdy in base $b$ by Corollary 8. Additionally, $b^k - 1$ divides $b^{ek} - 1$. The result now follows by Theorem 9. ◀

The next theorem gives another infinite class of sturdy numbers.

▶ **Theorem 11.** *Fix $b \geq 2$. Let $n$ be a positive integer, and $x$ be the base-$b$ representation of $n$. Then every integer with base-$b$ representation of the form $x\,(b-1)^i\,\overline{x}$, where $i \geq 0$ and $\overline{x}$ is the base-$b$ complement of $x$, is sturdy in base $b$.*

**Proof.** Suppose $y = x\,(b-1)^i\,\overline{x}$ for some $i \geq 0$. Then $[y]_b + n = nb^{|x|+i} + b^{|x|+i} - 1$. Then $[y]_b = (n+1)(b^{|x|+i}-1)$. Observe that $s_b(b^{|x|+i}-1) = (|x|+i)(b-1) = s_b([y]_b)$. Furthermore, $b^{|x|+i} - 1$ is sturdy in base $b$ by Corollary 8. Then for every positive integer $k$ we have $s_b(k[y]_b) = s_b(k(n+1)(b^{|x|+i}-1)) \geq s_b(b^{|x|+i}-1) = s_b([y]_b)$. ◀

▶ **Corollary 12.** *Let $b \geq 2$ be an integer, and $m$ be a positive integer dividing $b-1$. Then $\frac{(b^n-1)^2}{m}$ is sturdy in base $b$ for all $n \geq 1$.*

**Proof.** Suppose $m$ divides $b-1$. Then we have

$$\begin{aligned}
\frac{(b^n-1)^2}{m} &= \frac{b^n-1}{m}b^n - \frac{b^n-1}{m} \\
&= \frac{b^n-1}{m}b^n - b^n + b^n - \frac{b^n-1}{m} \\
&= \left(\frac{b^n-1}{m} - 1\right)b^n + b^n - \frac{b^n-1}{m} \\
&= \left(\frac{b^n-1}{m} - 1\right)b^n + (b^n-1) - \left(\frac{b^n-1}{m} - 1\right),
\end{aligned}$$

which has base-$b$ representation $x\overline{x}$ where $x = \left(\frac{b^n-1}{m} - 1\right)_b$. Then by Theorem 11, $\frac{(b^n-1)^2}{m}$ is sturdy in base $b$. ◀

In the rest of this paper we are almost exclusively concerned with the case $b = 2$, and so from now on we omit the subscripts on the functions msw, swm, mfw, and use the terms *flimsy* or *sturdy* without further elaboration. In this case $s_2(n)$ equals the number of 1's in the binary representation of $n$, also known as the *Hamming weight* of $n$.

## 4 Algorithms when $\mathrm{swm}(n)$ is small

As we will see in Section 5, for general $n$ we can determine whether $n$ is sturdy in $O(n)$ time. We call this a linear-time algorithm.[1] Therefore, it is of interest to see when this can be improved.

If $\mathrm{swm}(n)$ is small, this fact can be verified efficiently in some cases. This is particularly relevant in the case where $n$ is prime because, according to a recent result of Elsholtz [11], almost all primes $p$ have $\mathrm{swm}(p) \leq 7$. Furthermore, we know from results of Hasse [14] and Odoni [18] that a positive proportion of all primes satisfy $\mathrm{swm}(p) = 2$; asymptotically, this fraction is $17/24$. For general $n$, however, the situation is different: the set of $n$ for which $\mathrm{swm}(n) = 2$ has density 0; see the results of Moree in [21, Appendix B].

### 4.1 The case $\mathrm{swm}(n) = 2$

If $\mathrm{swm}(n) = 2$, then $n \cdot \mathrm{msw}(n) = 2^k + 1$ for some integer $k \geq 1$. Hence $n \mid 2^k + 1$, and so $-1$ belongs to the subgroup generated by 2 (mod $n$). We can decide if $-1$ belongs to the subgroup generated by 2 (mod $n$) by using an algorithm for the discrete logarithm problem. For example, the baby-step giant-step algorithm can be used to find $k$ such that $2^k \equiv -1 \pmod{n}$, if such a $k$ exists, with time complexity $O(\sqrt{n} \log n)$ [26]. If the factorization of $n$ is known, this running time can be substantially improved.

### 4.2 The case $\mathrm{swm}(n) = 3$

If $\mathrm{swm}(n) = 3$, then $n \cdot \mathrm{msw}(n) = 2^k + 2^\ell + 1$ for some integers $k > \ell \geq 1$. It follows that $2^k + 2^\ell \equiv -1 \pmod{n}$, which means that we are dealing with a 2-SUM problem. This can be solved in $O(n \log n)$ time using sorting and binary search. (Briefly, compute a table of powers of 2, mod $n$; sort them in ascending order, and then for each power $2^k$ use binary search to see if there is an $\ell$ such that $2^\ell \equiv -1 - 2^k \pmod{n}$.) Although this does not beat our $O(n)$ algorithm given below asymptotically, in many cases it will run more quickly because of the simplicity of the operations. This is particularly true if the subgroup generated by 2 (mod $n$) is small.

## 5 A dynamic programming algorithm

In this section we show how to check whether $n$ is sturdy using dynamic programming.

By Corollary 2, we can restrict our attention to the case where $n$ is odd. In this case, the powers of two $P_n = \{2^i : i \geq 0\}$ form a cyclic subgroup of $(\mathbb{Z}/(n))^*$, the multiplicative group of integers relatively prime to $n$. Define $\nu = \mathrm{ord}_2\, n = |P_n|$, the order of 2 in the group $(\mathbb{Z}/(n))^*$. Hence, to find a positive multiple of $n$ whose binary expansion contains exactly $k$ 1's, it suffices to find an appropriate linear combination of $k$ elements of $P_n$ (counted with repetition) that sums to 0 (mod $n$). More precisely, we need to find non-negative integers $a_1, a_2, \ldots, a_i$ and distinct elements $e_1, e_2, \ldots, e_i \in P_n$ such that

$$\begin{aligned} a_1 e_1 + \cdots + a_i e_i &\equiv 0 \pmod{n} \\ a_1 + a_2 + \cdots + a_i &= k, \end{aligned}$$

[1] Strictly speaking, the usage "linear-time" in the context of algorithms on integers would usually mean an algorithm that runs in $O(\log n)$ time. But since no algorithm is this efficient, we stray from the common usage for brevity.

for integers $k \geq 1$. This is the kind of problem that dynamic programming is well-suited for. To restrict the amount of work required in a dynamic programming algorithm for this we make use of the following lemma.

▶ **Lemma 13.** *For an integer base $b \geq 2$ let $P_{b,n} = \{b^i \bmod n : i \in \mathbb{N}\}$ and suppose that $e_1, e_2, \ldots, e_m$ are the distinct elements of $P_{b,n}$. If there exist non-negative integers $a_1, a_2, \ldots, a_m$ such that $a_1e_1+a_2e_2+\cdots+a_me_m \equiv 0 \pmod n$ and $a_1+\cdots+a_m = k$, then there exist non-negative integers $c_1, c_2, \ldots, c_m < b$ and $l \leq k$ such that $c_1e_1 + c_2e_2 + \cdots + c_me_m \equiv 0 \pmod n$ and $c_1 + \cdots + c_m = l$.*

**Proof.** Suppose we have non-negative integers $a_1, a_2, \ldots, a_m$ such that $a_1e_1 + a_2e_2 + \cdots + a_me_m \equiv 0 \pmod n$ and $a_1 + \cdots + a_m = k$. If we have $a_1, a_2, \ldots, a_m < b$ then we are done. So suppose that there is some $i$ such that $a_i \geq b$. Let $j$ be the integer such that $be_i \equiv e_j \pmod n$. Then we can take

$$\left(\sum_{r=1, r\neq i, r\neq j}^{m} a_re_r\right) + (a_i - b)e_i + (a_j + 1)e_j \equiv 0 \pmod n,$$

giving

$$\left(\sum_{r=1, r\neq i, r\neq j}^{m} a_r\right) + (a_i - b) + (a_j + 1) = a_1 + \cdots + a_m - b + 1 = k - b + 1 < k.$$

Setting $a_i := a_i - b$ and $a_j := a_j + 1$ and $k := k - b + 1$, we can repeat this argument until $a_1, \ldots, a_m < b$. ◀

Let us start with determining whether $n$ is sturdy. It suffices to solve the problem of the previous paragraph for $1 \leq k < s_2(n)$. The idea is that we will fill in the entries of a 3-dimensional boolean array $x$ with the following meaning: the entry $x[i, j, r]$ is `true` if and only if the integer $j$ has a representation as a sum of $i \geq 1$ powers of 2, using as summands only the first $r$ elements of the set $P_n$ without repetition. We fill in the array $x$ in increasing order of $r$.

For initialization, we set all elements of $x$ to `false`, except that we set $x[0, 0, r]$ to `true` for $0 \leq r \leq \nu$.

To solve the remaining three problems, we need to record more information than just the ability to represent $j$ as a sum of powers of 2. The integer array $y[i, j, r]$ is used to record the smallest integer congruent to $j \pmod n$ that is the sum of exactly $i$ powers of 2 (without repetition), using only the first $r$ elements of the set $P_n$.

In the pseudocode that follows, the scope of loops is indicated by the indentation.

```
minrep(n)   { assumes n odd and at least 3 }

sumd := sumdig(2,n);   {sum of base-2 digits of n}

{make a table of powers of 2}
b := 1;
a := 0;
repeat
   b := (2*b) mod n;
   a := a+1;
```

```
until
   b = 1;
ord2 := a;   { the order of 2 mod n }
power2 := array[0..ord2-1] of integer;
for m := 0 to ord2-1 do
   power2[m] := b;
   b := (2*b) mod n;

{ the intent is that x[i,j,r] = true, if j (mod n) has a representation
as a sum of exactly i powers of 2, using only the first r elements of
power2 (without repetition), and false otherwise.
y[i,j,r] = smallest integer congruent to j (mod n)
representable by the sum of exactly i powers of 2,
using only the first r elements of power2 (without repetition) }

x := array[0..sumd-1, 0..n-1, 0..ord2] of boolean;
y := array[0..sumd-1, 0..n-1, 0..ord2] of integer;

{ initialize }

for r := 0 to ord2 do
     for i := 1 to sumd-1 do
         for j := 0 to n-1 do
            x[i,j,r] := false;
            y[i,j,r] := infinity;
    x[0,0,r] := true;
    y[0,0,r] := 0;

{ fill in table }

for r := 1 to ord2 do   {consider summand 2^{r-1} mod p}
   for j := 0 to n-1 do      { check position j }
       for i := 1 to sumd-1 do     {fill in level i of the array}
            x[i,j,r] := x[i,j,r-1];
            y[i,j,r] := y[i,j,r-1];
            {check if we can use 2^{r-1} }
            if x[i-1, (j-power2[r-1]) mod n, r-1] then
                x[i,j,r] := true;
                y[i,j,r] := min(y[i,j,r],
                y[i-1, (j-power2[r-1]) mod n, r-1] + 2^{r-1});

sturdy := true;
for i := 2 to sumd-1 do
    sturdy := sturdy and x[i,0,ord2];

if (sturdy) then
    print("swm(n) = ",sumd);
    print("msw(n) = ",1);
```

```
else
    i := 1;
    while (not x[i,0,ord2]) do i := i+1;
    print("swm(n) = ",i);
    print("msw(n) = ",y[i,0,ord2]/n);
    mfw := infinity;
    while (i < sumd) do
        mfw := min(mfw, y[i,0,ord2]);
        i := i+1;
    print("mfw(n) = ",mfw);

end;
```

Our dynamic programming algorithm has three nested loops, which gives a running time of $O(\nu \cdot n \cdot s_2(n))$. Since $\nu = \mathrm{ord}_n\, 2$ could be as large as $n-1$, and $s_2(n)$ could be as big as $\log_2 n$, this gives a worst-case running time of $O(n^2 \log n)$, where we are measuring the run time in terms of RAM operations on integers of size about $n$. This means that this algorithm will only be feasible for integers smaller than about $10^7$.

## 6 An algorithm based on finite automata

In this section we provide a different, much faster algorithm for checking sturdiness, based on finite automata.

The idea is simple. It is easy to create a deterministic finite automaton (DFA) accepting the binary representations of the positive integers divisible by $n$. Such an automaton has $n$ states [1] and exactly one final state. Next, we can easily construct a DFA $A_t$ accepting those strings starting with a 1 and having at most $t$ ones. Using the standard "direct product" construction [15, pp. 59–60], we can construct a DFA $M_t$ of $(t+2)n$ states for the intersection of these two languages; it has exactly $t+1$ final states $f_0, f_1, \ldots, f_t$ corresponding to positive integers divisible by $n$ with $0, 1, \ldots, t$ 1's respectively. Then some multiple of $n$ has at most $t$ 1's iff $M_t$ accepts at least one string. We can test this condition (and even find the lexicographically least string accepted) using breadth-first search to decide if some $f_i$ for $0 \le i \le t$ is reachable from the start state of $M_t$, in linear time in the size of $M$, so in $O((t+2)n)$ time.

By choosing $t = s_2(n) - 1$ we can determine if $n$ is sturdy in $O(n \log n)$ steps. Similarly, by allowing the breadth-first search to run to completion and keeping track of the least string in radix order used to reach each state, we can recover $\mathrm{swm}(n)$, $\mathrm{msw}(n)$, and $\mathrm{mfw}(n)$ by examining each of the final states for whether or not they were visited in the search and looking at the least string in radix order used in each case. More precisely, the value of $\mathrm{swm}(n)$ is the least integer $i$ such that final state $f_i$ in $M_t$ is reached in the breadth-first search, or $s_2(n)$ if no final state is reached. The value of $\mathrm{msw}(n)$ is the least string in radix order used to reach $f_{\mathrm{swm}(n)}$ interpreted as an integer and divided by $n$, or 1 if $n$ is sturdy. The value of $\mathrm{mfw}(n)$, if it is defined, is the least string in radix order among all such strings used to reach a final state, interpreted as an integer and divided by $n$. To avoid needing to store the representation of large integers, we instead store the exponents of the current power of 2 and a pointer to the previous power. From this linked list we can reconstruct the appropriate number.

▶ **Theorem 14.** *We can decide whether $n$ is sturdy $O(n \log n)$ steps. In the same time bound we can compute* $\mathrm{swm}(n)$ *and* $\mathrm{msw}(n)$*. If $n$ is flimsy, we can compute* $\mathrm{mfw}(n)$ *in the same time bound.*

This algorithm is practical for $n$ up to about $10^{10}$. The main constraint is likely to be space and not time.

## 7 Improving the automaton-based algorithm

With a small modification to this idea of using a breadth-first search on the graph defined by automaton $M$, we can make further improvements to the time complexity. Consider the deterministic finite automaton $M_n$ accepting the binary representations of the positive integers divisible by $n$. We then define a directed graph $G_n$ with vertices given by the states of $M_n$ and directed, weighted edges given by the transitions of $M_n$ where transitions on the symbol 0 are given an edge weight of 0 in $G_n$ and transitions on the symbol 1 are given an edge weight of 1 in $G_n$. We augment $G_n$ with one additional vertex, $v_s$, with a single outgoing edge of weight 1 to the vertex corresponding to the state reached when $M_n$ reads any input of the form $0^*1$. If $v_f$ is the vertex corresponding to the accepting state in $M_n$, then there is a path from $v_s$ to $v_f$ of weight $k$ if and only if there is a non-zero multiple of $n$ with Hamming weight $k$. The shortest path problem on a graph $G = (V, E)$ with edge weights in $\{0, 1\}$ can be solved in time $O(|V| + |E|)$ using a variation of the breadth-first search algorithm. In place of the queue used in a standard breadth-first search, we use a double ended queue. We process a node by traversing incident edges of weight 0 and pushing the nodes reached to the front of the queue if they have not been processed already. Edges of weight 1 are also traversed, but the nodes reached are pushed to the back of the queue provided that they have not been processed already. After a node has been processed, the next node at the front of the queue is dequeued and processed if it has not been processed already, otherwise it is just discarded. The depth of the search can be tracked as in a standard breadth-first search. Thus we achieve the following improvement.

▶ **Theorem 15.** *We can test if $n$ is sturdy in $O(n)$ steps.*

From this approach we are still able to construct an example of a multiple of $n$ achieving the minimum Hamming weight over all multiples of $n$. It is simply a matter of maintaining the path used in the breadth-first search algorithm finding the shortest path from $v_s$ to $v_f$ in $G_n$. However, there is no guarantee that this is the least multiple of $n$ with this property. To find the least multiple we can use the linear-time algorithm to first determine the minimum Hamming weight. For minimum Hamming weight $k$, we take the direct product of automaton $M_n$ accepting the base-2 representations of all multiples of $n$ and the automaton accepting all binary strings with exactly $k$ 1's. A breadth-first search on this product automaton gives the least non-zero multiple of $n$ with Hamming weight $k$. This second breadth-first search has worst case time complexity $O(n \log n)$, giving overall complexity $O(n \log n)$ for finding the least non-zero multiple of $n$ having the minimum Hamming weight over all non-zero multiples of $n$.

## 8 Another breadth-first search approach

We can take advantage of Lemma 13 to evaluate sturdiness and compute swm and msw using a breadth-first search on a different graph structure. As before, to test the sturdiness of an integer $n \geq 3$, we construct an $(n + 1)$-vertex graph with $n$ of the vertices representing the

distinct residue classes modulo $n$, which we will refer to as $[0], [1], \ldots, [n-1]$, and one special vertex, $v_0$, corresponding to the number 0. The graph contains a directed edge from vertex $[x]$ to vertex $[y]$ if and only if $x + 2^j \equiv y \pmod{n}$ for some integer $j \geq 0$. Similarly, there is an edge from $v_0$ to $[y]$ if and only if $2^j \equiv y \pmod{n}$. Hence each vertex has out-degree $\nu = \mathrm{ord}_n\, 2$. The idea of this construction is to treat traversing an edge from $[x]$ to $[x + 2^j]$ as choosing to use the $j$th power of 2 as a summand in a summation to a value congruent to 0 modulo $n$. Thus, to compute $\mathrm{swm}(n)$ we are looking for the length of the shortest path from $v_0$ to $[0]$ and this can be found via a breadth-first search. Furthermore, by keeping track of the smallest sum required to reach each state, we can also recover $\mathrm{msw}(n)$ from such a breadth-first search. Rather than running the breadth-first search to completion, we can terminate as soon as we reach a depth equal to $s_2(n)$, as we will know by then whether or not $n$ is sturdy.

With this approach we can take advantage of the structure of the graph to speed up testing for sturdiness. If during the breadth first search we visit a node $[x]$ such that $[n - x]$ has already been visited, then since the length of the shortest path from $[x]$ to $[0]$ is equal to the length of the shortest path from $v_0$ to $[n - x]$ either we will know that $n$ is not sturdy, or that it is not necessary to continue searching from $[x]$. This greatly improves the efficiency of the testing for sturdiness.

The complexity of this approach, for evaluating sturdiness, and computing $\mathrm{swm}(n)$ and $\mathrm{msw}(n)$ is $O(n^2)$ since we are performing a breadth-first search on a graph with $n+1$ vertices each with $\nu = O(n)$ outgoing edges. In practice this approach seems to perform much better than our naive $O(n^2)$ upper bound would suggest, especially in testing for sturdiness, due to the early exit conditions.

## 9 Running time comparison

To demonstrate how these algorithms behave in practice, we compiled timing information for each of the approaches and each of the four functions of interest for consecutive integers starting from 1. Each of the algorithms are implemented as described above. However, before applying each algorithm the baby-step giant-step algorithm, as in Section 4.1, is used to exit faster in those cases where $\mathrm{swm}(n) = 2$. Running times for the mfw function with the `order_deg_bfs` algorithm are not given because there does not seem to be a natural approach for using this idea to evaluate mfw. The computations producing the given running times were performed on macOS Catalina version 10.15.2 on a 2.3 GHz Intel Core i5 processor. Implementations of our algorithms can be found in the GitHub repository

`https://github.com/FinnLidbetter/sturdy-numbers`.

**Table 2** Running time in milliseconds for each of the algorithms to evaluate the functions for all values of $n$ (odd and even) between 1 and 2000 inclusive.

| Algorithm | `is_sturdy` | swm | msw | mfw |
|---|---|---|---|---|
| `dp` | 22836 | 2667063 | 5675228 | 5167556 |
| `aut` | 1042 | 1050 | 1473 | 1430 |
| `order_deg_bfs` | 322 | 1646 | 4339 | — |
| `bfs01` | 224 | 226 | 650 | 1416 |

In Tables 2 and 3, the algorithmic approaches are named according to the commands used in the program-runner in the GitHub repository. Here, the `dp` algorithm refers to the dynamic programming approach described in Section 5, the `aut` algorithm refers to the

automaton-based approach described in Section 6, the `bfs01` algorithm refers to the improved automaton-based approach described in Section 7, and the `order_deg_bfs` algorithm refers to the alternative breadth-first search approach described in Section 8.

**Table 3** Running time in milliseconds for the algorithms to evaluate the functions for all values of $n$ (odd and even) between 1 and 10000 inclusive. The dynamic programming algorithm was not included because it was not feasible to evaluate the functions for all integers between 1 and 10000 with this approach.

| Algorithm | `is_sturdy` | swm | msw | mfw |
|---|---|---|---|---|
| `aut` | 31950 | 31990 | 42229 | 41747 |
| `order_deg_bfs` | 7378 | 164543 | 439794 | — |
| `bfs01` | 5209 | 5207 | 15515 | 41761 |

## 10 Computational results

Sequence A143027 in the OEIS [27] gives a list of the first few sturdy primes, namely,

$$2, 3, 5, 7, 17, 31, 73, 89, 127, 257, 1801, 2089, 8191, 65537, 131071,$$
$$178481, 262657, 524287, 2099863,$$

and mentions 616318177 as an additional sturdy prime, although it was not known if this was the next sturdy prime to occur in the sequence. Using our methods, we checked all primes $p < 2^{32}$. We confirmed the results in the OEIS and found that 616318177 and 2147483647 are the only remaining sturdy primes in that range.

We also computed frequency counts for the values of swm for odd $n > 1$, not just primes, and they are given in Table 4.

## 11 Numbers with few 0's

We can also use finite automata to determine when numbers with few 0's are flimsy. More precisely, for each pair of integers $j, k$ we can build a DFA $M_2(j, k)$ accepting those $(n)_2$ for which $(n)_2$ has $j$ 0's and $(kn)_2$ has more than $j + t$ 0's, where $t = |(kn)_2| - |(n)_2|$. Such an $n$ is guaranteed to be flimsy. We can determine $t$ by reading the input $n$, least significant digit first, and computing $(kn)_2$ on the fly, keeping track of the carries.

Let $j$ be a fixed natural number. By choosing an appropriate set of flimsy witnesses $k$ (which can be guessed empirically), we can determine all flimsy numbers having exactly $j$ 0's in their binary representation. We do this by computing the DFA's $M_2(j, k)$ and unioning them together to get a final automaton $M'_j$. We expect there to be a finite set of "sporadic" sturdy exceptions, and (according to Theorem 11) an infinite set of sturdy exceptions consisting of those numbers with binary representation of the form $s1^i\overline{s}$, where $s$ begins with 1 and ends with 0. This expectation can then be verified by considering the language accepted by $M'_j$; the finite set of sturdy exceptions can be tested using our algorithms previously discussed. The multipliers we used in constructing $M'_j$ are the odd numbers $\leq 2^{j+1} + 1$.

**Table 4** Counts of $\mathrm{swm}(n)$ for odd $n > 1$.

| $\mathrm{swm}(n)$ | $n < 2^{20}$ | $2^{20} < n < 2^{21}$ | $2^{21} < n < 2^{22}$ | $2^{22} < n < 2^{23}$ |
|---|---|---|---|---|
| 2 | 115931 | 107650 | 208333 | 403823 |
| 3 | 286681 | 294938 | 596522 | 1205753 |
| 4 | 83895 | 83958 | 168138 | 336448 |
| 5 | 19287 | 19242 | 38566 | 77071 |
| 6 | 9903 | 9892 | 19812 | 39635 |
| 7 | 4246 | 4265 | 8510 | 17023 |
| 8 | 2274 | 2269 | 4548 | 9104 |
| 9 | 1027 | 1030 | 2058 | 4119 |
| 10 | 529 | 527 | 1059 | 2118 |
| 11 | 256 | 257 | 514 | 1024 |
| 12 | 130 | 131 | 260 | 521 |
| 13 | 64 | 64 | 128 | 256 |
| 14 | 32 | 33 | 64 | 129 |
| 15 | 16 | 16 | 32 | 64 |
| 16 | 8 | 8 | 16 | 32 |
| 17 | 4 | 4 | 8 | 16 |
| 18 | 2 | 2 | 4 | 8 |
| 19 | 1 | 1 | 2 | 4 |
| 20 | 1 | 0 | 1 | 2 |
| 21 | 0 | 1 | 0 | 1 |
| 22 | 0 | 0 | 1 | 0 |
| 23 | 0 | 0 | 0 | 1 |

We also computed counts of odd sturdy numbers up to $10^i$ for $i = 1, 2, 3, 4, 5, 6$, and they are given below:

**Table 5** Counts of sturdy numbers.

| $i$ | Number of odd sturdy numbers $< 10^i$ |
|---|---|
| 1 | 5 |
| 2 | 22 |
| 3 | 81 |
| 4 | 292 |
| 5 | 995 |
| 6 | 3438 |

With these ideas we can prove the following theorem.

▶ **Theorem 16.**

**(a)** *Every integer with no* 0*'s is sturdy.*

**(b)** *Every odd integer with one* 0 *is flimsy, with the exception of* $5 = [101]_2$*, and is proven flimsy by multiplier* 3 *or* 5*.*

**(c)** *Every odd integer with two* 0*'s is flimsy, with the exception of* 51 *and numbers of the form* $101^i01$*,* $i \geq 0$*, which are all sturdy.*

**(d)** *Every odd integer with three* 0*'s is flimsy, with the exception of* $17, 85, 89, 455$ *and numbers of the form* $1001^i011$ *or* $1101^i001$*,* $i \geq 0$*, which are all sturdy.*

**(e)** *Every odd integer with four* 0*'s is flimsy, with the exception of* $33, 69, 73, 153, 3855$*, and numbers of the form* $10001^i0111$*,* $11001^i0011$*,* $10101^i0101$*,* $11101^i0001$*,* $i \geq 0$*, which are all sturdy.*

**(f)** *Every odd integer with five* 0*'s is flimsy, with the exception of* $65, 133, 161, 267, 275, 1365, 31775$*, and numbers specified by Theorem 11.*

**(g)** *Every odd integer with six* 0*'s is flimsy, with the exception of* $129, 259, 261, 273, 385, 525, 549, 561, 585, 645, 657, 705, 771, 777, 801, 1729, 1801, 2275, 3185, 11565, 13107, 258111$*, and numbers specified by Theorem 11.*

**(h)** *Every odd integer with seven* 0*'s is flimsy, with the exception of* $257, 515, 517, 529, 1035, 1065, 1105, 1155, 1157, 1185, 1285, 1545, 1665, 2077, 2201, 2325, 2449, 2573, 2697, 2821, 2945, 19065, 19275, 21845, 26985, 95325, 2080895$*, and numbers specified by Theorem 11.*

**(i)** *Every odd integer with eight* 0*'s is flimsy, with the exception of* $513, 1027, 1029, 1057, 1281, 2055, 2085, 2089, 2097, 2115, 2145, 2193, 2313, 2337, 2563, 2565, 2625, 3075, 3105, 3585, 4123, 4185, 4371, 4389, 4433, 4619, 4675, 4681, 4867, 4929, 5187, 6169, 6417, 6665, 6913, 8253, 8505, 8525, 8645, 8757, 9009, 9261, 9513, 9765, 10017, 10269, 10465, 10521, 10773, 11025, 11277, 11529, 11781, 12033, 12483, 13505, 14497, 18631, 25623, 34695, 39321, 42405, 50115, 57825, 158875, 222425, 774333, 16711935$*, and numbers specified by Theorem 11.*

**(j)** *Every odd integer with nine* 0*'s is flimsy, with the exception of* $1025, 2051, 2057, 2065, 2177, 3073, 4131, 4165, 4233, 4361, 4369, 4417, 4641, 5129, 5185, 6273, 8215, 8277, 8339, 8401, 8711, 8773, 8835, 8897, 10261, 10385, 10757, 10881, 12307, 12369, 12803, 12865, 14353, 14849, 16443, 16569, 16835, 16947, 17073, 17451, 17577, 17745, 17955, 18081, 18459, 18585, 18963, 19089, 19467, 19593, 19971, 20097, 24605, 24633, 25025, 25137, 25641, 26145, 26649, 26691, 27153, 27657, 28161, 28679, 32893, 33401, 33909, 34417, 34925, 35433, 35941, 36449, 36957, 37465, 37973, 38481, 38989, 39497, 40005, 40513, 41021, 41529, 41769, 42037, 42545, 43053, 43561, 44069, 44577, 45085, 45593, 46101, 46609, 47117, 47625, 48133, 48641, 178481, 285975, 349525, 413075, 476625, 1290555, 1806777, 1864135, 6242685, 133956095$*, and numbers specified by Theorem 11.*

Based on this theorem, we make the following conjecture.

▶ **Conjecture 17.** Every number with $j$ 0's is flimsy, with exceptions of the form $s1^i\overline{s}$, $i \geq 0$, where $|s| = j$ and $s$ begins with 1 and ends with 0, and only finitely many additional exceptions.

## 12 The $k$-flimsy numbers via formal language theory

In this section we describe a new approach, based on formal language theory, for understanding the distribution of the $k$-flimsy numbers. Recall these are the numbers

$$F_k = \{n \geq 1 \ : \ s_2(kn) < s_2(n)\}.$$

The majority of the results in this section are about the case $k = 3$, although in principle our technique can be applied to any odd $k$.

Kátai [16] studied the difference $s_2(3n) - s_2(n)$, and proved that this quantity is essentially normally distributed, in a certain sense. Stolarsky [28] conjectured that the natural density of the $k$-flimsy numbers is $1/2$ for all odd $k$. His conjecture was later proved by W. M. Schmidt [25] and J. Schmid [24]. All these results use rather sophisticated tools of number theory and probability.

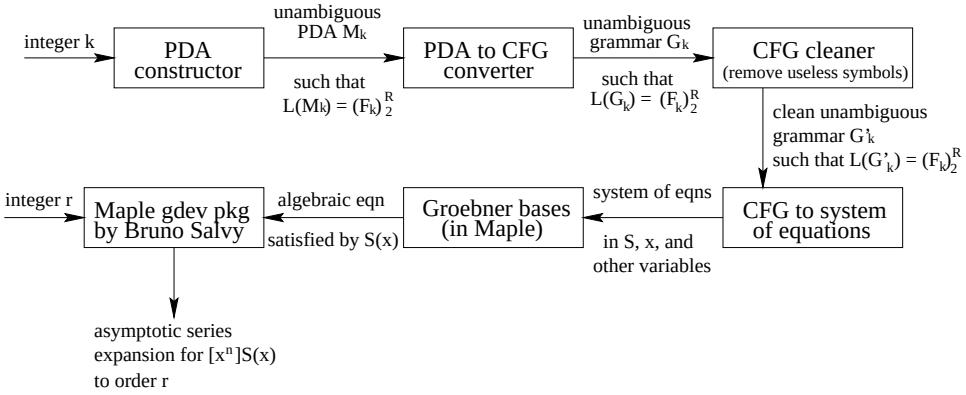
In contrast, in this section we obtain rather detailed results on the distribution of 3-flimsy numbers through a (more or less) purely mechanical approach based on formal language theory. The main result of this section is the following:

▶ **Theorem 18.** *The number of 3-flimsy numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left( \frac{1}{4} - cN^{-1/2} + O(N^{-3/2}) \right), \tag{1}$$

*where* $c = \frac{7\sqrt{6}}{24\sqrt{\pi}} \doteq 0.4030765$.

Our method starts with a pushdown automaton (PDA) recognizing the $k$-flimsy numbers, and by a series of steps, it is converted into an asymptotic series expansion for the number of $k$-flimsy numbers with $N$ bits. Previously, the basic approach has been used for a wide variety of combinatorial enumerations; see, for example, [4, 5, 2, 3]. We have implemented all the steps, and the flow of control is explained in the diagram below.




We now explain briefly what each box in the diagram does, with more detailed explanation to follow. For all undefined terms, see any textbook on automata theory or formal languages, such as [15].

First, given an odd integer $k \geq 3$, we build an unambiguous pushdown automaton (PDA) $M_k$ that recognizes the base-2 representation of elements of $F_k$; more precisely, $M_k$ recognizes the language $(F_k)_2^R$. The length-$N$ strings in $(F_k)_2^R$ are in 1-1 correspondence with the flimsy numbers in the half-open interval $[2^{N-1}, 2^N)$, so our goal is to estimate the cardinality of $(F_k)_2^R \cap \{0,1\}^N$ as precisely as possible.

Second, we convert $M_k$ to an unambiguous context-free grammar $G_k$ generating $(F_k)_2^R$.

We simplify this context-free grammar by deleting useless symbols (those symbols that do not participate in the derivation of any terminal string, or are not reachable from the start variable), obtaining a new CFG $G'_k$.

Third, we convert $G'_k$ to a system of equations in the variables of $G'_k$. These variables represent formal power series, with the property that the number of length-$N$ strings generated by a variable $A$ is given by $[x^N]A(x)$, the coefficient of $x^N$ in the power series $A(x)$.

Fourth, using Gröbner bases, we solve this system of equations, obtaining an algebraic equation satisfied by the formal power series $S(x)$, where $S$ is the start variable of the grammar $G'_k$.

Finally, using Bruno Salvy's `gdev` package, written in Maple, we can determine the asymptotic behavior of $[x^N]S(x)$ using the saddle-point method (as discussed by, e.g., Flajolet and Sedgewick [12]). In principle, we can obtain as many terms as we wish of the asymptotic expansion.

Theorem 18 now follows by performing each of these steps. The first four steps are done with original code written by the first author in Python, and the last two steps are done with Maple. The code for each step is available at

`https://git.uwaterloo.ca/Flimsy/CFLpy`.

We now give more complete details of some of the steps.

## 12.1 Constructing the PDA $M_k$

The general idea is as follows: we create a PDA accepting the base-2 representation of $k$-flimsy numbers $n$. We use the stack of the PDA to record the absolute value of $s_2(n) - s_2(kn)$, and we use the state to record both the carry needed when multiplying input by $k$, and the sign of $s_2(n) - s_2(kn)$. We accept the input if the carry is 0, the sign of $s_2(n) - s_2(kn)$ is positive, and the stack has at least one counter.

Our PDA is assumed to begin its computation with a special symbol, `Z`, on top of the stack, and if the input is accepted, to end its computation when the stack becomes empty.

The sketch above is not quite enough because of two technical issues. First, (a) in some cases this approach requires reading extra leading zeroes (which, because we are representing numbers starting with the least significant digit first, would be at the end of the input), in order to guarantee that the carry for $s_2(kn)$ was taken into account and (b) we must have that the leading bit of the input is 1, to avoid incorrectly counting smaller numbers as having $n$ bits.

To handle both these issues, we slightly modify the construction in several ways. First, if the state has a minus sign, then the stack holds $|y|_1 - |x|_1$ `X`'s, where $x$ is the input seen so far and $y$ is the $|x|$ least significant bits of $k(x)_2^R$. On the other hand, if the state has a positive sign, then the stack holds $|x|_1 - |y|_1 - 1$ `X`'s.

Second, to simulate the needed leading zeroes required to handle the carry, without actually reading them, we use a special series of $\log_2 k$ states to pop `X`'s from the stack.

Finally, we have a special state used to empty the stack when acceptance is detected. The total number of states is therefore at most $2k + \log_2 k$. The resulting PDA $M_3$ is depicted in Figure 1.

One important property of our construction is that our PDA $M_k$ is *unambiguous*. By this we mean that every accepted word has exactly one accepting computational path.

## 12.2 Converting the PDA to a CFG

We can convert $M_k$ to an equivalent context-free grammar $G_k$ using a standard technique called the "triple construction" [15, pp. 115–119]. This gives us a grammar $G_k$ with $O(k^2)$ variables and $O(k^3)$ productions.

Now we use the fact, proved in [13, Thm. 5.4.3, p. 151], that performing the triple construction on an unambiguous PDA gives us an unambiguous grammar.

## 12.3 Cleaning the CFG

We can remove useless symbols from our grammar $G_k$ by removing all variables that do not derive a terminal string, then removing all productions containing these removed variables, and then removing all variables and terminals that are not reachable from the start variable. This is a standard procedure, and is described in greater detail in [15, pp. 88–90].

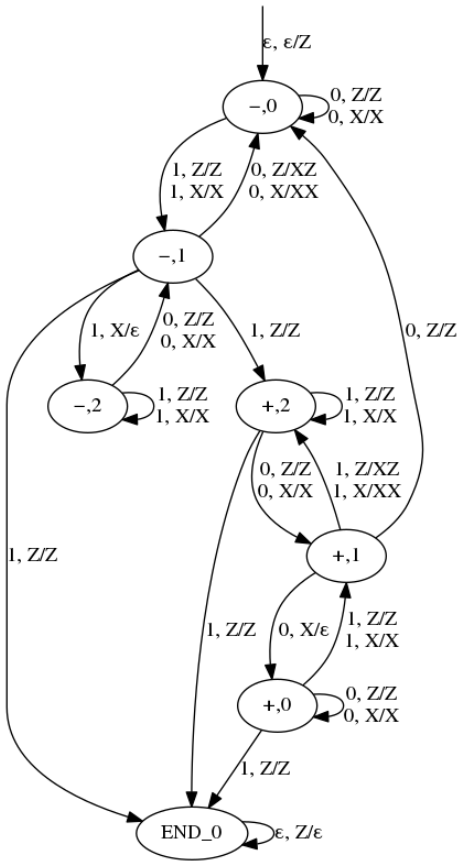



**Figure 1** PDA $M_3$.

Once this is complete, it may be found that there is a variable $X$ that has only one production $X \to \alpha$. If $X$ is not the start variable, then it can be deleted from the set of variables, and all instances of $X$ in production rules can be replaced with $\alpha$.

For example, when we convert our PDA $M_3$, we get an unambiguous grammar $G_3$; cleaning $G_3$ using this procedure gives us the following grammar $G_3'$:

$S \to 1F \mid 0S$
$A \to 1E \mid 0A$
$B \to 1G \mid 0B$
$C \to 1H \mid 1 \mid 0C$
$D \to 1I \mid 0D$
$E \to 1 \mid 0AJ$
$F \to 1N \mid 0AK$
$G \to 1LB \mid 0$
$H \to 1M \mid 1LC \mid 1$
$I \to 1M \mid 1LD \mid 1 \mid 0S$
$J \to 1J \mid 0E$
$K \to 1K \mid 0F$
$L \to 1L \mid 0G$
$M \to 1M \mid 1 \mid 0H$
$N \to 1N \mid 0I$

## 12.4 Converting the CFG to a system of equations

This transformation was discussed in [9]. It suffices to replace, in each set of productions $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_i$ of a grammar $G$, each terminal symbol by the indeterminate $x$, each $\mid$ symbol by a plus sign, and the $\to$ with an equals sign. For a proof of correctness, see [17, 19].

Performing this transformation on $G'_3$ gives us the following system of equations:

$$\begin{aligned}
S &= xF + xS & A &= xE + xA \\
B &= xG + xB & C &= xH + x + xC \\
D &= xI + xD & E &= x + xAJ \\
F &= xN + xAK & G &= xLB + x \\
H &= xM + xLC + x & I &= xM + xLD + x + xS \\
J &= xJ + xE & K &= xK + xF \\
L &= xL + xG & M &= xM + x + xH \\
N &= xN + xI
\end{aligned}$$

## 12.5 Solving the system

We can now solve the resulting system of equations for $S$, obtaining an algebraic equation for which $S$ is the root. The main tool is Groebner bases, for which a helpful package already exists in `Maple`.

Using the `Maple` code below, we find the following quadratic equation for $S$ in the case $k = 3$.

$$x(2x-1)^2(x+1)(2x^2-x+1)S(x)^2+(2x-1)(x-1)^2(x+1)(2x^2-x+1)S(x)+x^4(x^2-x+1) = 0.$$

To use this code, you will first need to download the `algolib` package from `http://algo.inria.fr/libraries/`.

```
eqs := [-S + x*V_F + x*S,
-V_A + x*V_E + x*V_A,
-V_B + x*V_G + x*V_B,
-V_C + x*V_H + x + x*V_C,
-V_D + x*V_I + x*V_D,
-V_E + x + x*V_A*V_J,
-V_F + x*V_N + x*V_A*V_K,
-V_G + x*V_L*V_B + x,
-V_H + x*V_M + x*V_L*V_C + x,
-V_I + x*V_M + x*V_L*V_D + x + x*S,
-V_J + x*V_J + x*V_E,
-V_K + x*V_K + x*V_F,
-V_L + x*V_L + x*V_G,
-V_M + x*V_M + x + x*V_H,
-V_N + x*V_N + x*V_I]:
Groebner[Basis](eqs, lexdeg([V_A, V_B, V_C, V_D, V_E, V_F, V_G, V_H,
V_I, V_J, V_K, V_L, V_M, V_N], [S]));
algeq := %[1]:
map(series, [solve(algeq, S)], x);
f := solve(algeq,S);
```

```
ps := f[1]:
assume(x, positive):
series(ps, x, 40);
libname:="<insert current directory path>",libname:
combine(equivalent(ps,x,n,5));
```

Solving this quadratic for $S$ gives

$$S(x) = \frac{-(x-1)^2(x+1)(2x^2-x+1) + \sqrt{-(x-1)(2x-1)(2x^2-x+1)(x^3+x^2-x+1)^2}}{2x(2x-1)(x+1)(2x^2-x+1)}.$$

Since the grammar $G_3'$ is unambiguous, the formal power series $S(x)$ is the census generating function for the set $(F_3)_2^R$. In particular, this means that $[x^N]S(x) = |F_3 \cap [2^{N-1}, 2^N)|$, or in other words, the coefficient of $x^N$ in $S(x)$ is the number $k$-flimsy numbers in $[2^{N-1}, 2^N)$.

## 12.6 Asymptotic expansion of the coefficients of the power series

Finally, we use Flajolet-Sedgewick-style asymptotic analysis [12, §VII. 7.1] to determine an asymptotic formula for the $N$'th coefficient of the power series expansion for $S(x)$. Conveniently, there is a `Maple` package `algolib`, written by Bruno Salvy [23], to accomplish this. When we run this on our formula for $S(x)$, we get our desired result.

This completes our discussion of the proof of Theorem 18.

▶ Remark 19. We could easily determine more terms in the asymptotic expansion, if we wanted, using the same ideas. For example, we can find that the number of 3-flimsy numbers in the interval $[2^{N-1}, 2^N)$ is

$$2^N \left( \frac{1}{4} - \frac{\sqrt{6}}{\sqrt{\pi}} \left( \frac{7}{24} N^{-1/2} + \frac{13}{72} N^{-3/2} - \frac{17}{64} N^{-5/2} + \frac{3365}{13824} N^{-7/2} + \cdots \right) \right).$$

▶ **Corollary 20.** *The number of 3-flimsy numbers $< 2^N$ is $2^{N-1} - O(2^N N^{-1/2})$.*

**Proof.** For any real number $a > 0$ we have

$$\begin{aligned} 2^N N^{-a} \le \sum_{1 \le n \le N} 2^n n^{-a} &\le \sum_{1 \le n \le N/2} 2^n n^{-a} + \sum_{N/2 < n \le N} 2^n n^{-a} \\ &\le \sum_{1 \le n \le N/2} 2^n + (N/2)^{-a} \sum_{N/2 < n \le N} 2^n \\ &\le 2^{N/2+1} + (N/2)^{-a} 2^{N+1}. \end{aligned}$$

Summing (1) and applying the inequalities above gives the desired result. ◀

▶ **Theorem 21.** *The number of 5-flimsy numbers in the interval $[2^{N-1}, 2^N)$ is*

$$2^N \left( \frac{1}{4} - cN^{-1/2} + O(N^{-3/2}) \right), \tag{2}$$

*where $c = \frac{3\sqrt{5}}{8\sqrt{\pi}} \doteq 0.473087348$.*

**Proof.** This is determined using the same method as the proof for Theorem 18. The details will appear in the full paper. ◀

We can also use the same ideas to compute the distribution of flimsy numbers in other bases. As an example we proved

▶ **Theorem 22.** *The number of integers in the range* $[3^{N-1}, 3^N)$ *that are* 2*-flimsy in base* 3 *is*

$$3^N \left( \frac{1}{3} + \frac{\sqrt{3}}{\sqrt{\pi}} \left( -\frac{1}{3} N^{-1/2} + \frac{1}{48} N^{-3/2} - \frac{13}{1536} N^{-5/2} - \frac{65}{24576} N^{-7/2} + O(N^{-9/2}) \right) \right).$$

**Proof.** As before. We omit the details. ◀

## 13 The $k$-equal numbers via formal language theory

Another quantity of interest is the number of $n$ for which $s_2(n) = s_2(kn)$. We call such $n$ *$k$-equal*. By generalizing the approach used in Section 12, we can compute how many integers $n \in [2^{N-1}, 2^N)$ are $k$-equal.

In particular, we modify PDA $M_k$ by changing the transitions to the `END` states. Whereas $M_k$ transitions to `END` when reading a 1 if following that 1 with sufficiently many zeros would reach the state $(+, 0)$, instead we want such an input to reach the state $(-, 0)$ with no counters on the stack. With this approach we can prove

▶ **Theorem 23.** *The number of* 3*-equal numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left( cN^{-1/2} + O(N^{-3/2}) \right), \tag{3}$$

*where* $c = \frac{\sqrt{6}}{4\sqrt{\pi}} \doteq 0.345494149$.

▶ **Theorem 24.** *The number of* 5*-equal numbers in the interval* $[2^{N-1}, 2^N)$ *is*

$$2^N \left( cN^{-1/2} + O(N^{-3/2}) \right), \tag{4}$$

*where* $c = \frac{\sqrt{5}}{4\sqrt{\pi}} \doteq 0.315391565$.

The details will appear in the final paper.

## 14 Conclusions and open problems

We have shown that techniques from automata theory can be used to solve problems in number theory. For other fun along these lines, see [7, 22].

It would be interesting to understand the distribution of values of $\mathrm{msw}(n)$ and $\mathrm{mfw}(n)$ for $n$ flimsy. We leave this as an open problem.

## References


1. B. Alexeev. Minimal DFAs for testing divisibility. *J. Comput. System Sci.*, 69:235–243, 2004.
2. A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns: enumerative aspects. In S. T. Klein et al., editors, *LATA 2018*, volume 10792 of *Lecture Notes in Computer Science*, pages 195–206. Springer-Verlag, 2018.
3. A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns, the vectorial kernel method, and generating functions for pushdown automata. *Algorithmica*, 82:386–428, 2020.
4. C. Banderier and M. Drmota. Coefficients of algebraic functions: formulae and asymptotics. In *FPSAC 2013*, volume AS of *DMTCS Proc.*, pages 1065–1076. DMTCS, 2013.
5. C. Banderier and M. Drmota. Formulae and asymptotics for coefficients of algebraic functions. *Combin. Prob. Comput.*, 24:1–53, 2015.

6 B. Bašić. The existence of $n$-flimsy numbers in a given base. *Ramanujan J.*, 43:359–369, 2017.
7 J. Bell, K. Hare, and J. Shallit. When is an automatic set an additive basis? *Proc. Amer. Math. Soc. Ser. B*, 5:50–63, 2018.
8 L. H. Y. Chen, H.-K. Hwang, and V. Zacharovas. Distribution of the sum-of-digits function of random integers: a survey. *Prob. Surveys*, 11:177–236, 2014.
9 N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North Holland, Amsterdam, 1963.
10 C. Dartyge, F. Luca, and P. Stănică. On digit sums of multiples of an integer. *J. Number Theory*, 129:2820–2830, 2009.
11 C. Elsholtz. Almost all primes have a multiple of small Hamming weight. *Bull. Austral. Math. Soc.*, 94:224–235, 2016.
12 P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
13 M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.
14 H. Hasse. Über die Dichte der Primzahlen $p$, für die einevorgegebene ganz rationale Zahl $a \neq 0$ von gerader bzw. ungerader Ordnung mod $p$ ist. *Math. Annalen*, 166:19–23, 1966.
15 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
16 I. Kátai. Change of the sum of digits by multiplication. *Acta Sci. Math. (Szeged)*, 39:319–328, 1977.
17 W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Springer-Verlag, 1986.
18 R. W. K. Odoni. A conjecture of Krishnamurthy on decimal periods and some allied problems. *J. Number Theory*, 13:303–319, 1981.
19 A. Panholzer. Gröbner bases and the defining polynomial of a context-free grammar generating function. *J. Automata, Languages, and Combinatorics*, 10:79–97, 2005.
20 Pavel V. Phedotov. Sum of digits of a multiple of a given number (in Russian), 2002. Available at `http://digitsum.narod.ru/Index.htm`.
21 V. Pless, P. Solé, and Z. Qian. Cyclic self-dual $Z_4$-codes. *Finite Fields Appl.*, 3:48–69, 1997.
22 A. Rajasekaran, J. Shallit, and T. Smith. Additive number theory via automata theory. *Theor. Comput. Sys.*, 64:542–567, 2020.
23 B. Salvy. gdev package of algolib version 17.0. Available at `http://algo.inria.fr/libraries/`, 2013.
24 J. Schmid. The joint distribution of the binary digits of integer multiples. *Acta Arith.*, 43:391–415, 1984.
25 W. M. Schmidt. The joint distributions of the digits of certain integer $s$-tuples. In P. Erdős, editor, *Studies in Pure Mathematics to the Memory of Paul Turán*, pages 605–622. Birkhäuser, 1983.
26 D. Shanks. Class number, a theory of factorization and genera. In *Proc. Sympos. Pure Math.*, volume 20, pages 415–440, 1969.
27 N. J. A. Sloane et al. The on-line encyclopedia of integer sequences. Available at `https://oeis.org`, 2019.
28 K. B. Stolarsky. Integers whose multiples have anomalous digital frequencies. *Acta Arith.*, 38:117–128, 1980/81.
29 S. S. Wagstaff et al. The Cunningham project. Available at `https://homes.cerias.purdue.edu/~ssw/cun/index.html`, 2019.