THE COMBINATORIAL ALGORITHM FOR COMPUTING $\pi(x)$

by

Douglas B. Staple

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science

at

Dalhousie University
Halifax, Nova Scotia
August 2015

© Copyright by Douglas B. Staple, 2015

Table of Contents

List of Tables

List of Figures

Abstract

This thesis describes recent advances in the combinatorial method for computing $\pi(x)$, the number of primes $\leq x$. In particular, the memory usage has been reduced by a factor of $\log x$, and modifications for shared- and distributed-memory parallelism have been incorporated. The resulting method computes $\pi(x)$ with complexity $O(x^{2/3}\log^{-2}x)$ in time and $O(x^{1/3}\log^{2}x)$ in space. The algorithm has been implemented and used to compute $\pi(10^n)$ for $1 \leq n \leq 26$ and $\pi(2^m)$ for $1 \leq m \leq 86$. The mathematics presented here is consistent with and builds on that of previous authors.

List of Abbreviations and Symbols Used

$O(\ldots)$	Big O notation (computational complexity theory).
$\Omega(\ldots)$	Big Ω notation (computational complexity theory).
$\mu(n)$	The Möbius function.
$\mathbb{N}$	The set of natural numbers, where we follow the convention that 0 is not a natural number.
$\mathrm{Re}(s)$	The real part of a complex number s.
$\phi(x, a)$	The number of integers n, with $1 \leq n \leq x$, that are coprime to the first a primes.
$\phi_j(x, a)$	The number of integers n, with $1 \leq n \leq x$, that are coprime to the first a primes, and have exactly j prime factors, which are not necessarily distinct.
$\pi(x)$	The number of primes less-than or equal-to x.
$\mathbb{P}$	The set of primes.
$\mathbb{R}$	The set of real numbers.
$\theta(x)$	The Heaviside step function.
p_j	The j^{th} prime.
$p_{\min}(m)$	The smallest prime factor of a natural number m.
x	A natural number.

Acknowledgements

The author thanks Karl Dilcher for support, and for suggestions regarding the underlying algorithm, these calculations, and this thesis. Calculations were performed on the Guillimin, Briarée, and Colosse clusters from McGill University, Université de Montréal, and Laval Université, managed by Calcul Québec and Compute Canada. The operation of these supercomputers is funded by the Canada Foundation for Innovation (CFI), NanoQuébec, RMGA, and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT).

Chapter 1

Introduction

1.1 Historical overview

Algorithms used in exact calculations of $\pi(x)$ can be divided into roughly three categories. The simplest algorithms are based on identifying and counting each prime $p \leq x$, typically using some modification of the sieve of Eratosthenes. A naïve implementation of the sieve of Eratosthenes uses $O(x \log \log x)$ arithmetic operations[1] and $O(x)$ bits of memory. Modern variants based on bucket sieving reduce the memory usage to roughly $\pi(\sqrt{x})$ storage locations, each of width $\log_2 \pi(\sqrt{x})$ bits, while leaving the time complexity unchanged [1]. Given the prime number theorem, algorithms that enumerate the primes $p \leq x$ are limited to time complexity $\Omega(x/\log x)$.

The first published algorithm capable of computing $\pi(x)$ substantially faster than the sieve of Eratosthenes was a combinatorial algorithm due to E. Meissel [2]. Given that Meissel's method involved decisions based on human judgement, it is not clear what time complexity to attribute to it; despite this fact, authors usually estimate the time complexity of Meissel's original method as $\Omega(x^{1-\epsilon})$ for any $\epsilon > 0$ [3]. Meissel used his method in hand calculations of $\pi(10^8)$ and $\pi(10^9)$ in the late 1800s [4–6]; the method was substantially improved by multiple groups of authors, and used in record computations of $\pi(10^n)$ for $10 \leq n \leq 23$ between 1956 and 2007 [3, 7–12]. Meissel's method and its descendants are collectively known as "the" combinatorial algorithm for computing $\pi(x)$.

Analytic algorithms for computing $\pi(x)$ based on the Riemann zeta function were first presented by Lagarias and Odlyzko in the 1980s [13–15]. Despite the attractive complexity of $O(x^{1/2+\epsilon})$ in time and $O(x^{1/4+\epsilon})$ in space, for any $\epsilon > 0$, the implied constants were large, and no-one succeeded in developing a practical implementation of these methods until nearly 30 years later. The first record computation using an analytic method was $\pi(10^{24})$, under the assumption of the Riemann hypothesis,

[1] See Sect. 1.2 for the definition of an arithmetic operation used in this thesis.

by Franke, Kleinjung, Büthe, and Jost in 2010 [16]. This was followed by a 2012 computation of the same value by Platt without assuming the Riemann hypothesis [17]. Büthe *et al.* subsequently modified their algorithm to eliminate the assumption of the Riemann hypothesis, and presented the first computation of $\pi(10^{25})$ [16].

In Table 1.1 we provide a historical timeline of $\pi(x)$ calculations dating to Meissel. However, the problem of determining the number of primes up to some limit is directly tied to the history of the primes themselves, which dates to antiquity. We direct the interested reader to additional historical references on the topic [2, 18–21].

x	Date	Authors	Note	Ref.
10^8	1871	Meissel	Too large by 5	[4]
10^8	1882	Meissel		[5]
10^9	1884	Meissel	Too small by 56	[6]
10^9	1958	Lehmer		[7]
10^{10}	1958	Lehmer	Too large by 1	[7]
10^{10}	1972	Bohmann		[9]
10^{11}	1972	Bohmann		[9]
10^{12}	1972	Bohmann		[9]
10^{13}	1972	Bohmann	Too small by 941	[9]
10^{13}	1984	Lagarias, Miller, Odlyzko		[3]
10^{14}	1984	Lagarias, Miller, Odlyzko		[3]
10^{15}	1984	Lagarias, Miller, Odlyzko		[3]
10^{16}	1984	Lagarias, Miller, Odlyzko		[3]
10^{17}	1994	Deléglise and Rivat		[10]
10^{18}	1994	Deléglise and Rivat		[10]
10^{19}	1996	Deléglise		[22]
10^{20}	1996	Deléglise		[23]
10^{21}	2000	Gourdon		[24]
10^{22}	2000	Gourdon, Sebah, *et al.*	Part of "The $\pi(x)$ Project"	[25]
10^{23}	2001	Gourdon, Sebah, *et al.*	Failed double-check	[25]
10^{23}	2007	Oliveira e Silva		[26]
10^{24}	2010	Buethe, Franke, Jost, Kleinjung	Assumed the Riemann hypothesis	[16, 27]
10^{24}	2012	Platt		[17]
10^{25}	2013	Buethe, Franke, Jost, Kleinjung		[28]
10^{26}	2014	Staple	This work	[29]

Table 1.1: Historical timeline for calculations of $\pi(x)$ with $x = 10^n$.

1.2 Model of computation

When discussing space complexity, one must distinguish between bits of storage and storage locations, each of which grows as the problem size increases. It is commonplace to state that an algorithm has complexity $O(M)$ in space if it requires γM storage locations for some constant γ, each capable of storing a number with $\log_2 M$ bits [3, 10–12]. We use this convention here for consistency with other authors. Similarly, in a model of time complexity, one must specify which operations are considered to be performed in constant time. In this thesis, we count bitwise operations, addition, subtraction, multiplication, division, modulus, decisions (branches), and memory read and write operations of a single machine word.

1.3 Analytic methods

There are significant differences between the analytic methods due to different authors [13–17]. Here we provide a simple example to give the reader the flavour of such computations. The example presented here closely follows the one presented by Crandall and Pomerance [30].

We start with the Riemann zeta function:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \tag{1.3.1}$$

$$= \prod_{p \in \mathbb{P}} \sum_{k=1}^{\infty} \frac{1}{(p^s)^k} \tag{1.3.2}$$

$$= \prod_{p \in \mathbb{P}} \frac{1}{1 - \frac{1}{p^s}}, \tag{1.3.3}$$

where $\mathrm{Re}(s) > 1$, and where $p \in \mathbb{P}$ indicates that p ranges over the set of primes $\mathbb{P}$. This gives:

$$\log \zeta(s) = -\sum_{p \in \mathbb{P}} \log\left(1 - p^{-s}\right) \tag{1.3.4}$$

$$= \sum_{p \in \mathbb{P}} \sum_{m=1}^{\infty} \frac{1}{m p^{sm}}. \tag{1.3.5}$$

We next define

$$\pi^\star(x) = \sum_{\substack{p\in\mathbb{P}\\ m>0}} \frac{\theta(x-p^m)}{m} \tag{1.3.6}$$

$$= \begin{cases} \displaystyle\sum_{p^m<x} \frac{1}{m} & \text{if } x \text{ is not a prime power} \\ \displaystyle\left(\sum_{p^m<x} \frac{1}{m}\right) + \frac{1}{2k} & \text{if } x = p^k \text{ is a prime power.} \end{cases} \tag{1.3.7}$$

Supposing x not to be a prime power, (1.3.7) implies:

$$\pi^\star(x) = \pi(x) + \frac{1}{2}\pi(x^{1/2}) + \frac{1}{3}\pi(x^{1/3}) + \dots \tag{1.3.8}$$

Making use of the Perron forumla [30], we have:

$$\theta(x-n) = \frac{1}{2\pi i}\int_{\mathcal{C}} \left(\frac{x}{n}\right)^s \frac{ds}{s}, \tag{1.3.9}$$

where $\mathcal{C} = \{s : s = \sigma + it\}$ is a contour with fixed real part $\sigma > 0$ and imaginary part t ranging over $[0,\infty)$. Together, (1.3.6) and (1.3.9) give:

$$\pi^\star(x) = \sum_{\substack{p\in\mathbb{P}\\ m>0}} \frac{1}{m}\left(\frac{1}{2\pi i}\right)\int_{\mathcal{C}} \left(\frac{x}{p^m}\right)^s \frac{ds}{s} \tag{1.3.10}$$

$$= \frac{1}{2\pi i}\int_{\mathcal{C}} x^s \left(\sum_{\substack{p\in\mathbb{P}\\ m>0}} \frac{1}{mp^{sm}}\right)\frac{ds}{s} \tag{1.3.11}$$

$$= \frac{1}{2\pi i}\int_{\mathcal{C}} x^s \log\zeta(s)\frac{ds}{s}, \tag{1.3.12}$$

where we take $\sigma > 1$ to avoid the singularity in $\log\zeta(s)$ at $\sigma = 1$.

It follows from (1.3.12) and (1.3.8) that $\pi(x)$ can be computed from a contour integral to obtain $\pi^\star(x)$. The conversion from $\pi^\star(x)$ to $\pi(x)$ can be accomplished several different ways, for example by explicitly computing and subtracting the error terms from (1.3.8), or via an inversion formula [30]:

$$\pi(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n}\pi^\star\left(x^{1/n}\right), \tag{1.3.13}$$

where $\mu(n)$ is the Möbius function.

Difficulties in the application of (1.3.12) for explicit calculations of $\pi(x)$ are manifold. Firstly, in the contour integral (1.3.12), t ranges over $[0, \infty)$. In an explicit calculation, the integral must be approximated using a finite interval $[0, T)$. However, this approximation tends to converge slowly with respect to T, with the error terms varying erratically as they decay to zero [30]. As part of this, placing rigorous bounds on the error terms is nontrivial. Such bounds are necessary because exactly computing $\pi(x)$ requires proving that the computed approximate (real) value is within 0.5 of the true (integer) value for $\pi(x)$. Thirdly, fast, precise calculations of $\zeta(s)$ are themselves nontrivial. These issues are addressed differently by the different authors applying analytic methods to $\pi(x)$ calculations [13–17]. Typically, improvements focus on alternative integral representations that show superior convergence properties.

1.4 Structure of this thesis

In this thesis, we describe recent advances to the combinatorial algorithm for computing $\pi(x)$. First, in Chapter 2 we re-derive the combinatorial algorithm, including the advances presented by Olivera e Silva in [12]. We furthermore derive explicit formulae including the effects of blocks B_k; to our knowledge these formulae have not appeared elsewhere.

In Chapter 3, we show how the memory usage of the algorithm can be reduced by a factor of $\log x$: this memory reduction is one of the main results of this thesis. We note that this is not only a reduction in the memory complexity, but a substantial reduction in the actual memory usage for relevant values of x. Indeed, before the final step in the memory-complexity reduction was achieved, the author had already reduced the memory usage sufficiently to compute $\pi(10^{26})$, so the original announcement of $\pi(10^{26})$ claimed only a constant-factor reduction in the memory usage. As part of the memory reduction, we present an algorithm by which one can retrieve $\pi(y)$ for any $y < y_{\max}$ in (heuristically) $O(1)$ time, using only $O(\pi(y))$ storage locations, each of width $\log_2 \pi(y)$ bits. This algorithm was implemented, and found to be *faster* than a lookup table of all $\pi(y)$ values, due to cache effects associated with the increased memory usage of the larger lookup table.

In Chapter 4, we describe mechanisms by which the algorithm can be parallelized.

Multiple methods due to the author and others are presented for shared-memory parallelism. We also describe a previously unpublished algorithm for distributed-memory parallelism, loosely based on the idea presented by Gourdon in [11]. Next, in Chapter 5, we describe the numerical calculations undertaken as part of this work: the algorithms described here were implemented and used to compute $\pi(10^n)$ for $1 \leq n \leq 26$ and $\pi(2^m)$ for $1 \leq m \leq 86$. Finally, in Chapter 6, we summarize our results and suggest directions for future research.

Chapter 2

The combinatorial algorithm

In this chapter we present a simplified version of the combinatorial algorithm, based on the exposition by Tomás Oliveira e Silva [12]. First, in Sect. 2.1, we derive the central equation in the algorithm, (2.1.11), which separates the computation of $\pi(x)$ into computations of partial sieve functions $\phi(x, a)$ and $\phi_2(x, a)$. In Sections 2.2 and 2.3, we derive equations for $\phi_2(x, a)$ and $\phi(x, a)$ in terms of sums over other partial sieve values, $\phi(y, b)$, where $y \in [0, z)$, $c \leq b \leq a$, $z = x^{2/3}/\alpha$, $a = \pi(\alpha\sqrt[3]{x})$ and $\alpha \in \mathbb{R}$, $1 \leq \alpha < \pi(\sqrt[6]{x})$.

Notation in this chapter is consistent with that used in [12], and some important equations are written identically in this work as in [12]. This is done intentionally, to permit close comparisons between this work and [12]. However, the derivations and exposition presented here are consistent with, but not identical to, those given in [12]. In particular, in this work we give explicit formulae whenever possible, including formulae accounting for the division of the sieving interval $[0, z)$ into blocks B_k. As a result, the formulae presented here are very similar to the computer implementation written by the author. This close connection between derivation and implementation is intended to be a strength of the current work.

2.1 The central equation

Let $\phi(x, a)$ be the number of integers $n \leq x$ that are not divisible by the first a primes. Similarly, let $\phi_j(x, a)$ be the number of integers $n \leq x$ that are not divisible by the first a primes, and have exactly j prime factors, which are not necessarily distinct.

Then:

$$\phi(x,a) = \sum_{j=0}^{\infty} \phi_j(x,a), \tag{2.1.1}$$

$$\phi_0(x,a) = 1 \qquad \forall \quad x \geq 1, \tag{2.1.2}$$

$$\phi_1(x,a) = \pi(x) - a \quad \forall \quad 0 \leq a \leq \pi(x). \tag{2.1.3}$$

Together, (2.1.1)–(2.1.3) give:

$$\pi(x) = \phi(x,a) + a - 1 - \sum_{j=2}^{\infty} \phi_j(x,a) \quad \forall \quad x \geq 1, \quad 0 \leq a \leq \pi(x). \tag{2.1.4}$$

Suppose $\phi_j(x,a) \neq 0$. Then there exists an $n \leq x$ that has j prime factors, each larger than p_a. Then $n \geq p_{a+1}^j$, but $x \geq n$, so

$$x \geq p_{a+1}^j, \tag{2.1.5}$$

$$\ln(x) \geq j \ln(p_{a+1}), \tag{2.1.6}$$

$$\frac{\ln(x)}{\ln(p_{a+1})} \geq j, \tag{2.1.7}$$

$$j \leq \left\lfloor \frac{\ln(x)}{\ln(p_{a+1})} \right\rfloor, \tag{2.1.8}$$

where we have used the fact that $j \in \mathbb{N}$. This result permits us to refine (2.1.4):

$$\pi(x) = \phi(x,a) + a - 1 - \sum_{j=2}^{\left\lfloor \frac{\ln(x)}{\ln(p_{a+1})} \right\rfloor} \phi_j(x,a) \quad \forall \quad x \geq 1, \quad 0 \leq a \leq \pi(x). \tag{2.1.9}$$

(2.1.9) highlights the finite number of terms in the sum over $\phi_j(x,a)$, but is not convenient for direct computation. For explicit calculations, it's most useful to note $x < p_{a+1}^j$ if and only if $\pi(x^{\frac{1}{j}}) \leq a$. Thus, $\pi(x^{\frac{1}{j}}) \leq a$ implies $\phi_{j'}(x,a) = 0$ for all $j' \geq j$. In particular, we have:

$$\pi(x) = \phi(x,a) + a - 1 \qquad \forall \quad \pi(\sqrt{x}) \leq a \leq \pi(x), \quad x \geq 1, \tag{2.1.10}$$

$$\pi(x) = \phi(x,a) + a - 1 - \phi_2(x,a) \quad \forall \quad \pi(\sqrt[3]{x}) \leq a \leq \pi(x), \quad x \geq 1. \tag{2.1.11}$$

In this thesis we call (2.1.10) "Legendre's formula", because it is closely related to an inclusion-exclusion formula due to Legendre [12, 13]. (2.1.11) is central to the combinatorial algorithm.

2.2 Equation for $\phi_2(x,a)$ in terms of $\phi(y,b)$

From the definition of $\phi_j(x,a)$, we have:

$$\phi_2(x,a) = \sum_{b=a+1}^{\pi(x)} \sum_{d=b}^{\pi(x)} \left[p_b p_d \leq x\right], \tag{2.2.1}$$

where $[p_b p_d \leq x]$ indicates that the summand is 1 when the condition is met, and 0 otherwise, and where $0 \leq x$ and $0 \leq a$.

Note that $p_b p_d \leq x$ and $p_d \geq p_b$ implies $b \leq \pi(\sqrt{x})$. Meanwhile, $p_b p_d \leq x$ if and only if $d \leq \pi(x/p_b)$. Together with (2.2.1), these give:

$$\phi_2(x,a) = \sum_{b=a+1}^{\pi(\sqrt{x})} \sum_{d=b}^{\pi(\frac{x}{p_b})} 1. \tag{2.2.2}$$

Note as well that $b \leq \pi(\sqrt{x})$ implies $b \leq \pi(\frac{x}{p_b})$, so

$$\sum_{d=b}^{\pi\left(\frac{x}{p_b}\right)} 1 = \pi\left(\frac{x}{p_b}\right) - b + 1. \tag{2.2.3}$$

Together, (2.2.2) and (2.2.3) give:

$$\begin{aligned}
\phi_2(x,a) &= \sum_{b=a+1}^{\pi(\sqrt{x})} \left(\pi\left(\frac{x}{p_b}\right) - b + 1\right) && (2.2.4)\\
&= \sum_{b=a+1}^{\pi(\sqrt{x})} \pi\left(\frac{x}{p_b}\right) - \sum_{b=a+1}^{\pi(\sqrt{x})} (b-1) && (2.2.5)\\
&= \sum_{b=a+1}^{\pi(\sqrt{x})} \pi\left(\frac{x}{p_b}\right) - \sum_{b=a}^{\pi(\sqrt{x})-1} b && (2.2.6)\\
&= \sum_{b=a+1}^{\pi(\sqrt{x})} \pi\left(\frac{x}{p_b}\right) - \left(\sum_{b=0}^{\pi(\sqrt{x})-1} b - \sum_{b=0}^{a-1} b\right) && (2.2.7)\\
&= \binom{a}{2} - \binom{\pi(\sqrt{x})}{2} + \sum_{b=a+1}^{\pi(\sqrt{x})} \pi\left(\frac{x}{p_b}\right) \quad \forall \quad 0 \leq x, \quad 0 \leq a \leq \pi(\sqrt{x}). && (2.2.8)
\end{aligned}$$

At this point we would like to apply Legendre's formula, (2.1.10), to eliminate the reference to $\pi(y)$ with $y = x/p_b$ and $a+1 \leq b \leq \pi(\sqrt{x})$. To do this, we must first

establish the validity of Legendre's formula by proving $\pi(\sqrt{y}) \leq a \leq \pi(y)$ and $y \geq 1$, under the assumtion that $\pi(\sqrt[3]{x}) \leq a \leq \pi(\sqrt[2]{x})$. We start with:

$$b \leq \pi(\sqrt{x}), \tag{2.2.9}$$

$$p_b \leq \sqrt{x}, \tag{2.2.10}$$

$$\frac{x}{y} \leq \sqrt{x}, \tag{2.2.11}$$

$$y \geq \sqrt{x}, \tag{2.2.12}$$

$$\pi(y) \geq \pi(\sqrt{x}). \tag{2.2.13}$$

Since $\pi(\sqrt{x}) \geq a + 1$ in the sum limits, we have:

$$\pi(\sqrt{x}) \geq a + 1, \tag{2.2.14}$$

$$\pi(\sqrt{x}) > a, \tag{2.2.15}$$

which, when combined with (2.2.13), gives:

$$\pi(y) > a. \tag{2.2.16}$$

Thus we clearly have the condition $\pi(y) \geq a$ demanded by Legendre's formula. The upper limit on $\pi(y)$ is established similarly:

$$b \geq a + 1, \tag{2.2.17}$$

$$p_b \geq p_{a+1}, \tag{2.2.18}$$

$$y \leq \frac{x}{p_{a+1}}. \tag{2.2.19}$$

Suppose now that $a \geq \pi(\sqrt[3]{x})$. This implies:

$$p_{a+1} > x^{1/3}, \tag{2.2.20}$$

$$y < x^{2/3}, \tag{2.2.21}$$

$$\sqrt{y} < x^{1/3}, \tag{2.2.22}$$

$$\pi(\sqrt{y}) \leq \pi(\sqrt[3]{x}), \tag{2.2.23}$$

$$\pi(\sqrt{y}) \leq a. \tag{2.2.24}$$

Finally, before applying Legendre's forumla on $\pi(y)$, we need to establish $y \geq 1$,

which is given if $x \geq 1$, since:

$$b \leq \pi(\sqrt{x}), \tag{2.2.25}$$

$$p_b \leq \sqrt{x}, \tag{2.2.26}$$

$$\frac{x}{p_b} \geq \sqrt{x}, \tag{2.2.27}$$

$$\frac{x}{p_b} \geq 1. \tag{2.2.28}$$

Together (2.2.16), (2.2.24), and (2.2.28) establish the validity of Legendre's formula, so:

$$\sum_{b=a+1}^{\pi(\sqrt{x})} \pi\left(\frac{x}{p_b}\right) = \sum_{b=a+1}^{\pi(\sqrt{x})} \left[\phi\left(\frac{x}{p_b}, a\right) + a - 1\right] \quad \forall \quad a \geq \pi(\sqrt[3]{x}), \quad x \geq 1, \tag{2.2.29}$$

$$\phi_2(x,a) = \binom{a}{2} - \binom{\pi(\sqrt{x})}{2} + \sum_{b=a+1}^{\pi(\sqrt{x})} \left[\phi\left(\frac{x}{p_b}, a\right) + a - 1\right] \tag{2.2.30}$$

$$= \binom{a}{2} - \binom{\pi(\sqrt{x})}{2} + \left[\pi(\sqrt{x}) - a\right](a-1) + \sum_{b=a+1}^{\pi(\sqrt{x})} \phi\left(\frac{x}{p_b}, a\right) \tag{2.2.31}$$

$$= -\binom{\pi(\sqrt{x}) - a + 1}{2} + \sum_{b=a+1}^{\pi(\sqrt{x})} \phi\left(\frac{x}{p_b}, a\right), \tag{2.2.32}$$

where $\pi(\sqrt[3]{x}) \leq a \leq \pi(x^{1/2})$ and $x \geq 1$.

2.3 Equation for $\phi(x,a)$ in terms of $\phi(y,b)$

2.3.1 Recurrence relation for $\phi(x,a)$

The calculation of $\phi(x,a)$ is based on repeated application of the recurrence relation

$$\phi(y,b) = \phi(y,b-1) - \phi\left(\frac{y}{p_b}, b-1\right), \tag{2.3.1}$$

starting with $y = x$ and $b = a$, and terminating when $b = c$ and $y \geq z$, or when $y < z$ for some fixed c and $z = x^{2/3}/\alpha$. In the first case, when $b = c$, $\phi(y,b)$ can be rapidly computed using a lookup table of $\phi(y',c)$ for all $y' < p_c\#$, where

$$p_c\# = \prod_{b=1}^{c} p_b, \tag{2.3.2}$$

denotes the primorial of p_c. Values of $\phi(y,c)$ for $y \geq p_c\#$ can be computed using the periodicity of $\phi(y,c)$:

$$\phi(y,c) = \phi(y',c) \quad \forall \quad y \equiv y' \pmod{p_c\#}, \tag{2.3.3}$$

$$\phi(y,c) = \phi\left(y - p_c\# \left\lfloor \frac{y}{p_c\#} \right\rfloor, c\right). \tag{2.3.4}$$

Leaves falling into the second case, $y < z$, can be computed by sieving the interval $[0,z)$.

The proof of (2.3.1) is definitional. Consider the natural numbers $n \leq y$ coprime to the first $b-1$ primes. There are $\phi(y,b-1)$ such n. These numbers n can be separated into two categories: (i) those also coprime to p_b, (ii) those divisible by p_b. Given that these n are assumed to be coprime to the first $b-1$ primes, the numbers in category (i) are in fact coprime to the first b primes. Thus, there are $\phi(y,b)$ numbers in category (i). The numbers in category (ii) can be written in the form $n = p_b m$, where $m \leq y/p_b$ and m is coprime to the first $b-1$ primes. There are $\phi(y/p_b, b-1)$ such choices for m, so there are $\phi(y/p_b, b-1)$ numbers in category (ii). Given that there are $\phi(y,b-1)$ numbers separated in these two categories, we have:

$$\phi(y,b-1) = \phi(y,b) + \phi\left(\frac{y}{p_b}, b-1\right), \tag{2.3.5}$$

which is rearranged to give (2.3.1).

2.3.2 Ordinary and special leaves (S_0 and S)

Repeated application of (2.3.1) generates terms of the form $\mu(n)\phi(y,b)$ for various n and b, where $y = x/z$ and $\mu(n)$ is the Möbius function. These can be separated into two categories, corresponding to the two truncation rules. We call terms generated by the truncation rule $b = c$ and $y \geq z$ "ordinary leaves", and those generated by $y < z$ "special leaves". We denote the sum over the ordinary leaves by S_0, and the sum over the special leaves as S, such that

$$\phi(x,a) = S_0 + S. \tag{2.3.6}$$

Terms contributing to S_0 all have $b = c$ by definition. The requirement that $y \geq z$ limits the possible denominators n to those with $n \leq \alpha \sqrt[3]{x}$. Together, these give:

$$S_0 = \sum_{\substack{1 \leq n \leq \alpha \sqrt[3]{x} \\ p_{\min}(n) > p_c}} \mu(n)\phi\left(\frac{x}{n}, c\right), \tag{2.3.7}$$

where $p_{\min}(n)$ denotes the smallest prime factor of n. Given that all the terms in S_0 have $b = c$, they can be rapidly computed using (2.3.4). The computationally difficult part of $\phi(x, a)$ is the contribution of the special leaves:

$$S = -\sum_{c<b+1<a} \sum_{\substack{m\leq\alpha\sqrt[3]{x}<mp_{b+1} \\ p_{\min}(m)>p_{b+1}}} \mu(m)\phi\left(\frac{x}{mp_{b+1}}, b\right). \tag{2.3.8}$$

2.3.3 Special leaves (S_1 and S_2)

For efficient application of the combinatorial algorithm, it is critical to reduce the number of terms in the sum S. Indeed, the truncation rules in Sect. 2.3 were chosen with this in mind [12]. Further reducing the number of leaves involves subdividing the types of leaves considered. We first subdivide S depending on whether $b+1 \leq a^\star$ or $b+1 > a^\star$, where $a^\star = \pi(\sqrt{\alpha}\sqrt[6]{x})$:

$$S = S_1 + S_2, \tag{2.3.9}$$

$$S_1 = -\sum_{c<b+1\leq a^\star} \sum_{\substack{m\leq\alpha\sqrt[3]{x}<mp_{b+1} \\ p_{\min}(m)>p_{b+1}}} \mu(m)\phi\left(\frac{x}{mp_{b+1}}, b\right), \tag{2.3.10}$$

$$S_2 = -\sum_{a^\star<b+1<a} \sum_{\substack{m\leq\alpha\sqrt[3]{x}<mp_{b+1} \\ p_{\min}(m)>p_{b+1}}} \mu(m)\phi\left(\frac{x}{mp_{b+1}}, b\right). \tag{2.3.11}$$

We wish to rewrite the limits on S_1 and S_2 in order to make them more amenable to computation. For S_1, we simply note

$$\alpha\sqrt[3]{x} < mp_{b+1} \quad \Leftrightarrow \quad \frac{\alpha\sqrt[3]{x}}{p_{b+1}} < m, \tag{2.3.12}$$

such that

$$m \leq \alpha\sqrt[3]{x} < mp_{b+1} \quad \Leftrightarrow \quad \frac{\alpha\sqrt[3]{x}}{p_{b+1}} < m \leq \alpha\sqrt[3]{x}, \tag{2.3.13}$$

which gives:

$$S_1 = -\sum_{c<b+1\leq a^\star} \sum_{\substack{\frac{\alpha\sqrt[3]{x}}{p_{b+1}}<m\leq\alpha\sqrt[3]{x} \\ p_{\min}(m)>p_{b+1}}} \mu(m)\phi\left(\frac{x}{mp_{b+1}}, b\right). \tag{2.3.14}$$

Rewriting the limits on S_2 is more complicated, but comes with a larger payoff.

Firstly, suppose $p_{b+1}^2 \leq \alpha \sqrt[3]{x}$. This would imply:

$$p_{b+1} \leq \sqrt{\alpha} \sqrt[6]{x}, \tag{2.3.15}$$

$$p_{b+1} \leq p_{a^\star}, \tag{2.3.16}$$

$$b+1 \leq a^\star, \tag{2.3.17}$$

which is never the case in the summand of S_2. Thus, we can assume $\alpha \sqrt[3]{x} < p_{b+1}^2$. Clearly $p_{b+1} < m$, since $p_{\min}(m) > p_{b+1}$. Together, these give $\alpha \sqrt[3]{x} < m p_{b+1}$. Thus, $\alpha \sqrt[3]{x} < m p_{b+1}$ is implied by the other conditions in S_2, and can be eliminated:

$$S_2 = - \sum_{a^\star < b+1 < a} \sum_{\substack{m \leq \alpha \sqrt[3]{x} \\ p_{\min}(m) > p_{b+1}}} \mu(m) \phi \left(\frac{x}{m p_{b+1}}, b \right). \tag{2.3.18}$$

Now, given that $\alpha \sqrt[3]{x} < p_{b+1}^2$, and given $m \leq \alpha \sqrt[3]{x}$, we have:

$$m < p_{b+1}^2, \tag{2.3.19}$$

$$p_{b+1} > \sqrt{m}, \tag{2.3.20}$$

$$p_{\min}(m) > \sqrt{m}, \tag{2.3.21}$$

where we have used the fact that $p_{\min}(m) > p_{b+1}$. (2.3.21) implies that m is prime, so we can rewrite the limits on S_2 to consider only prime values of m, and take $p_{\min}(m) = m$:

$$S_2 = - \sum_{a^\star < b+1 < a} \sum_{\substack{m \text{ prime} \\ p_{b+1} < m \leq \alpha \sqrt[3]{x}}} \mu(m) \phi \left(\frac{x}{m p_{b+1}}, b \right). \tag{2.3.22}$$

Letting $m = p_d$ and noting $\mu(m) = -1$, we have:

$$S_2 = \sum_{a^\star < b+1 < a} \sum_{b+1 < d \leq a} \phi \left(\frac{x}{p_{b+1} p_d}, b \right). \tag{2.3.23}$$

2.3.4 Trivial special leaves (S_{2t})

The process of subdividing and refining the calculation of ϕ continues by separating S_2 into three categories, the so-called 'trivial', 'easy', and 'hard' leaves, represented by S_{2t}, S_{2e}, and S_{2h}. We first define the trivial leaves. Suppose

$$\frac{x}{p_{b+1} p_d} < p_{b+1}. \tag{2.3.24}$$

This would imply

$$\phi\left(\frac{x}{p_{b+1}p_d}, b\right) = 1. \tag{2.3.25}$$

(2.3.24) is equivalent to:

$$\frac{x}{p_{b+1}^2} < p_d, \tag{2.3.26}$$

$$p_d > \frac{x}{p_{b+1}^2}, \tag{2.3.27}$$

$$d > \pi\left(\frac{x}{p_{b+1}^2}\right). \tag{2.3.28}$$

This motivates the definition of the trivial leaves:

$$S_{2t} = \sum_{a^\star < b+1 < a} \sum_{\substack{b+1 < d \le a \\ \pi\left(\frac{x}{p_{b+1}^2}\right) < d}} \phi\left(\frac{x}{p_{b+1}p_d}, b\right) \tag{2.3.29}$$

$$= \sum_{a^\star < b+1 < a} \sum_{\substack{b+1 < d \le a \\ \pi\left(\frac{x}{p_{b+1}^2}\right) < d}} 1 \tag{2.3.30}$$

$$= \sum_{a^\star < b+1 < a} S_{2tb} \tag{2.3.31}$$

$$= \sum_{a^\star \le b < a-1} S_{2tb}, \tag{2.3.32}$$

where

$$S_{2tb} = \begin{cases} a - \max\left\{b+1, \pi\left(\frac{x}{p_{b+1}^2}\right)\right\} & \text{when } a > \max\left\{b+1, \pi\left(\frac{x}{p_{b+1}^2}\right)\right\}, \\ 0 & \text{otherwise,} \end{cases} \tag{2.3.33}$$

and where we have used the fact that $p_{b+1}p_d \le x$, i.e., $p_{a-1}p_a \le x$, which is ensured by $p_a \le \sqrt{x}$.

We wish to rewrite the somewhat cumbersome condition from (2.3.33). Suppose that

$$\frac{x}{p_{b+1}^2} \ge p_a. \tag{2.3.34}$$

This would imply

$$\pi\left(\frac{x}{p_{b+1}^2}\right) \ge a, \tag{2.3.35}$$

and subsequently

$$S_{2tb} = 0. \tag{2.3.36}$$

Consider next the possibility

$$\frac{x}{p_{b+1}^2} < p_a. \tag{2.3.37}$$

In this situation we have:

$$\pi\left(\frac{x}{p_{b+1}^2}\right) < a, \tag{2.3.38}$$

which implies

$$a > \max\left\{b+1, \pi\left(\frac{x}{p_{b+1}^2}\right)\right\}. \tag{2.3.39}$$

Thus, we can rewrite (2.3.33) as

$$S_{2tb} = \begin{cases} a - \max\left\{b+1, \pi\left(\frac{x}{p_{b+1}^2}\right)\right\} & \text{when } \frac{x}{p_{b+1}^2} < p_a, \\ 0 & \text{otherwise.} \end{cases} \tag{2.3.40}$$

Note that the condition in (2.3.40) means that $\pi(x/p_{b+1}^2)$ only needs to be evaluated in cases where $x/p_{b+1}^2 < \alpha\sqrt[3]{x}$, which can be rapidly performed using a lookup table or the methods to be developed in Sect. 3.1. Thus, the total work involved in the computation of S_{2t} is of order a operations, which is a negligible contribution to the overall computation of $\pi(x)$.

2.3.5 Easy and hard special leaves (S_{2e} and S_{2h})

We have

$$S_2 = S_{2t} + \sum_{a^\star < b+1 < a} \sum_{\substack{b+1 < d \le a \\ d \le \pi\left(\frac{x}{p_{b+1}^2}\right)}} \phi\left(\frac{x}{p_{b+1}p_d}, b\right). \tag{2.3.41}$$

$$= S_{2t} + \sum_{a^\star \le b < a-1} \sum_{\substack{b+2 \le d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1}} \phi\left(\frac{x}{p_{b+1}p_d}, b\right). \tag{2.3.42}$$

The limits in (2.3.41) give:

$$b+1 < \pi\left(\frac{x}{p_{b+1}^2}\right), \tag{2.3.43}$$

$$p_{b+1} < \frac{x}{p_{b+1}^2}, \tag{2.3.44}$$

$$p_{b+1} < \sqrt[3]{x}, \tag{2.3.45}$$

$$b+1 \leq \pi(\sqrt[3]{x}), \tag{2.3.46}$$

$$b < \pi(\sqrt[3]{x}), \tag{2.3.47}$$

so we can write:

$$S_2 = S_{2t} + \sum_{a^\star \leq b < \pi(\sqrt[3]{x})} \sum_{\substack{b+2 \leq d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1}} \phi\left(\frac{x}{p_{b+1}p_d}, b\right). \tag{2.3.48}$$

Suppose now that $x/p_{b+1}^3 < p_d$. Then:

$$\frac{x}{p_{b+1}p_d} = \left(\frac{x}{p_{b+1}^3}\right)\left(\frac{p_{b+1}^2}{p_d}\right) \tag{2.3.49}$$

$$< p_d\left(\frac{p_{b+1}^2}{p_d}\right) \tag{2.3.50}$$

$$< p_{b+1}^2, \tag{2.3.51}$$

such that

$$\sqrt{\frac{x}{p_{b+1}p_d}} < p_{b+1}, \tag{2.3.52}$$

$$\pi\left(\sqrt{\frac{x}{p_{b+1}p_d}}\right) < b+1, \tag{2.3.53}$$

$$\pi\left(\sqrt{\frac{x}{p_{b+1}p_d}}\right) \leq b, \tag{2.3.54}$$

which establishes the validity of Legendre's formula, such that:

$$\phi\left(\frac{x}{p_{b+1}p_d}, b\right) = \pi\left(\frac{x}{p_{b+1}p_d}\right) - b + 1 \qquad \left(\frac{x}{p_{b+1}^3} < p_d\right). \tag{2.3.55}$$

In order to use (2.3.55) in explicit calculations of the summand in S_2, Olivera e Sliva made use of a precomputed table of values of $\pi(y)$ for $y \leq \alpha\sqrt[3]{x}$ [12]. Later, in Sect. 3.1, we will see that it is possible to make do with a smaller table, storing

only the primes p_b for $b \leq a$, rather than the values $\pi(y)$ themselves. In either case, we restrict the direct application of (2.3.55) to situations where $x/p_{b+1}p_d \leq \alpha\sqrt[3]{x}$. In order to convert this condition to an explicit bound on d, we note that $a^\star \leq b$ implies:

$$\pi(\sqrt{\alpha}\sqrt[6]{x}) \leq b, \tag{2.3.56}$$

$$\sqrt{\alpha}\sqrt[6]{x} \leq p_b, \tag{2.3.57}$$

$$\alpha\sqrt[3]{x} \leq p_b^2, \tag{2.3.58}$$

$$\alpha\sqrt[3]{x} < p_{b+1}^2, \tag{2.3.59}$$

$$\frac{x}{p_{b+1}\alpha\sqrt[3]{x}} > \frac{x}{p_{b+1}^3}. \tag{2.3.60}$$

Suppose now that

$$\frac{x}{p_{b+1}p_d} \leq \alpha\sqrt[3]{x} \tag{2.3.61}$$

$$p_d \geq \frac{x}{p_{b+1}\alpha\sqrt[3]{x}}. \tag{2.3.62}$$

Combined with (2.3.60), this gives

$$p_d > \frac{x}{p_{b+1}^3}, \tag{2.3.63}$$

satisfying the condition on (2.3.55). (2.3.61) is itself implied by the condition

$$\pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right) \leq d \tag{2.3.64}$$

allowing us to rewrite (2.3.55) as:

$$\phi\left(\frac{x}{p_{b+1}p_d}, b\right) = \pi\left(\frac{x}{p_{b+1}p_d}\right) - b + 1 \qquad \left(\pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right) \leq d\right). \tag{2.3.65}$$

Finally, we write:

$$S_2 = S_{2t} + S_{2e} + S_{2h}, \tag{2.3.66}$$

where

$$S_{2h} = \sum_{a^\star \le b < \pi(\sqrt[3]{x})} \sum_{\substack{b+2 \le d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1 \\ d < \pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right)+1}} \phi\left(\frac{x}{p_{b+1}p_d}, b\right), \tag{2.3.67}$$

$$S_{2e} = \sum_{a^\star \le b < \pi(\sqrt[3]{x})} \sum_{\substack{b+2 \le d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1 \\ \pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right)+1 \le d}} \phi\left(\frac{x}{p_{b+1}p_d}, b\right), \tag{2.3.68}$$

$$= \sum_{a^\star \le b < \pi(\sqrt[3]{x})} \sum_{\substack{b+2 \le d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1 \\ \pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right)+1 \le d}} \left[\pi\left(\frac{x}{p_{b+1}p_d}\right) - b + 1\right]. \tag{2.3.69}$$

2.3.6 Clustered and sparse easy leaves (S_{2ec} and S_{2es})

The contribution of the easy special leaves, S_{2e}, can be further subdivided, according to whether or not consecutive values of d tend to yield the same value for the summand in (2.3.55): the transition for this tendency occurs at $p_d = \sqrt{x/p_{b+1}}$. We define clustered and sparse easy leaves as contributing S_{2ec} and S_{2es}, respectively, with

$$S_{2ec} = \sum_{a^\star \le b < \pi(\sqrt[3]{x})} \sum_{\substack{b+2 \le d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1 \\ \pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right)+1 \le d \\ \pi\left(\sqrt{\frac{x}{p_{b+1}}}\right)+1 \le d}} \left[\pi\left(\frac{x}{p_{b+1}p_d}\right) - b + 1\right] \tag{2.3.70}$$

$$S_{2es} = \sum_{a^\star \le b < \pi(\sqrt[3]{x})} \sum_{\substack{b+2 \le d < a+1 \\ d < \pi\left(\frac{x}{p_{b+1}^2}\right)+1 \\ \pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right)+1 \le d \\ d < \pi\left(\sqrt{\frac{x}{p_{b+1}}}\right)+1}} \left[\pi\left(\frac{x}{p_{b+1}p_d}\right) - b + 1\right] \tag{2.3.71}$$

Rather than applying (2.3.70) directly, we can compute the number of consecutive d-values for which the summand holds a given value, calculate the summand once, and multiply by the number of repetitions.

For a given value of b, the summand in (2.3.70) changes values whenever $x/(p_{b+1}p_d)$ is prime. Let

$$l = \pi\left(\frac{x}{p_{b+1}p_d}\right) - b + 1. \tag{2.3.72}$$

Then

$$l + b - 1 = \pi\left(\frac{x}{p_{b+1}p_d}\right), \tag{2.3.73}$$

such that

$$p_{l+b-1} \leq \frac{x}{p_{b+1}p_d} < p_{l+b}. \tag{2.3.74}$$

Clearly

$$\frac{x}{p_{b+1}p_d} < p_{l+b} \Leftrightarrow \frac{x}{p_{b+1}p_{l+b}} < p_d, \tag{2.3.75}$$

and

$$p_{l+b-1} \leq \frac{x}{p_{b+1}p_d} \Leftrightarrow p_d \leq \frac{x}{p_{b+1}p_{l+b-1}}, \tag{2.3.76}$$

so we have

$$\frac{x}{p_{b+1}p_{b+1}} < p_d \leq \frac{x}{p_{b+1}p_{b+l-1}}, \tag{2.3.77}$$

$$\pi\left(\frac{x}{p_{b+1}p_{b+l}}\right) < d \leq \pi\left(\frac{x}{p_{b+1}p_{b+l-1}}\right), \tag{2.3.78}$$

$$\pi\left(\frac{x}{p_{b+1}p_{b+l}}\right) + 1 \leq d < \pi\left(\frac{x}{p_{b+1}p_{b+l-1}}\right) + 1. \tag{2.3.79}$$

It is relatively straightforward to implement an iteration scheme that uses the above formulae to compute S_{2ec} more rapidly than a direct application of (2.3.70). Unfortunately, such an iteration scheme is not easy to represent as an equation with an explicit sum. The procedure is represented as pseudocode in Fig. 2.1, and described here in text. We start by writing explicit limits for d in S_{2ec} from (2.3.70):

$$d_{\min,S_{2ec}} = \max\left\{b+2, \pi\left(\frac{x}{p_{b+1}\alpha\sqrt[3]{x}}\right) + 1, \pi\left(\sqrt{\frac{x}{p_{b+1}}}\right) + 1\right\}, \tag{2.3.80}$$

$$d_{\max,S_{2ec}} = \min\left\{a+1, \pi\left(\frac{x}{p_{b+1}^2}\right) + 1\right\}. \tag{2.3.81}$$

The sum is then rewritten as follows:

$$S_{2ec} = \sum_{a^\star \leq b < \pi(\sqrt[3]{x})} \sum_{d \in \mathcal{D}} (d' - d)l, \tag{2.3.82}$$

where l is computed using (2.3.72), and

$$d' = \min\left\{\pi\left(\frac{x}{p_{b+1}p_{b+l-1}}\right)+1, d_{\max,S_{2ec}}\right\}. \tag{2.3.83}$$

The set $\mathcal{D}$ of d-values is defined recursively via $d_{\min,S_{2ec}} \in \mathcal{D}$ and for all $d \in \mathcal{D}, d' \in \mathcal{D}$. This recursive sequence terminates when $d = d'$, which occurs when $d = d_{\max,S_{2ec}}$. From a practical point of view, the iteration procedure involves initializing d using $d_{\min,S_{2ec}}$, and iterating through subsequent d-values using (2.3.83).

2.4 Introducing blocks B_k

There is a one-to-one correspondence between the terms in (2.2.32) and (2.3.67), and terms ultimately summed in a machine implementation of the Meissel-Lehmer method. The individual terms, $\phi(y, b)$ for each y and b, are calculated by sieving the interval $[0, z)$, where $z = x^{2/3}/\alpha$ is an upper bound for the possible y-values. This procedure will be briefly described in Sect. 2.5. However, in order to make use of the sieve, the interval $[0, z)$ must first be subdivided into blocks B_k of width Δ. A typical choice for Δ is $\Delta = \lfloor \alpha \sqrt[3]{x} \rfloor$. We will see in Sect. 3.3 that it is possible to use smaller values of Δ, resulting in a lower memory requirement for the algorithm. In this section we rewrite (2.2.32) and (2.3.67) in terms of these blocks.

2.4.1 Defining B_k and computing $k_{\max}$

Let $B_k = [(k-1)\Delta, k\Delta)$, with $k \in [1, k_{\max}]$, where $\Delta, k, k_{\max} \in \mathbb{N}$, and where $k_{\max}$ is defined by:

$$\lfloor z \rfloor \in B_{k_{\max}} = [(k_{\max}-1)\Delta, k_{\max}\Delta). \tag{2.4.1}$$

Then

$$\lfloor z \rfloor \geq (k_{\max}-1)\Delta \tag{2.4.2}$$

$$\frac{\lfloor z \rfloor}{\Delta} \geq k_{\max}-1 \tag{2.4.3}$$

$$\left\lfloor \frac{\lfloor z \rfloor}{\Delta} \right\rfloor \geq k_{\max}-1 \tag{2.4.4}$$

$$\left\lfloor \frac{z}{\Delta} \right\rfloor \geq k_{\max}-1 \tag{2.4.5}$$

$$\left\lfloor \frac{z}{\Delta} \right\rfloor + 1 \geq k_{\max} \tag{2.4.6}$$

where (2.4.4) follows because $k_{\max} \in \mathbb{N}$. Similarly, we have:

$$\lfloor z \rfloor < k_{\max}\Delta, \tag{2.4.7}$$

$$\left\lfloor \frac{z}{\Delta} \right\rfloor < k_{\max}, \tag{2.4.8}$$

$$\left\lfloor \frac{z}{\Delta} \right\rfloor + 1 \leq k_{\max}. \tag{2.4.9}$$

Together, (2.4.6) and (2.4.9) give:

$$k_{\max} = \left\lfloor \frac{z}{\Delta} \right\rfloor + 1. \tag{2.4.10}$$

2.4.2 Subdividing $\phi_2(x, a)$

Suppose now that

$$\sum_{b=a+1}^{\pi(\sqrt{x})} \phi\left(\frac{x}{p_b}, a\right) = \sum_{k=1}^{k_{\max}} \sum_{b=b_{\min}}^{b_{\max}} \phi\left(\frac{x}{p_b}, a\right), \tag{2.4.11}$$

where $x/p_b \in B_k$ and $b_{\min}, b_{\max} \in \mathbb{N}$. We wish to compute $b_{\min}$ and $b_{\max}$ as a function of k. Consider first the case $k = 1$. Then:

$$(k-1)\Delta \leq \frac{x}{p_b} \tag{2.4.12}$$

is equivalent to

$$0 \leq \frac{x}{p_b}, \tag{2.4.13}$$

which is satisfied for all $x \in \mathbb{N}$. Suppose next that $k > 1$. This gives:

$$(k-1)\Delta \leq \frac{x}{p_b}, \tag{2.4.14}$$

$$p_b \leq \frac{x}{(k-1)\Delta}, \tag{2.4.15}$$

$$b \leq \pi\left(\frac{x}{(k-1)\Delta}\right). \tag{2.4.16}$$

On the lower-limit, irrespective of the value of k, we have:

$$\frac{x}{p_b} < k\Delta, \tag{2.4.17}$$

$$\frac{x}{k\Delta} < p_b, \tag{2.4.18}$$

$$\left\lfloor \frac{x}{k\Delta} \right\rfloor < p_b, \tag{2.4.19}$$

$$\left\lfloor \frac{x}{k\Delta} \right\rfloor + 1 \leq p_b, \tag{2.4.20}$$

and

$$\pi\left(\frac{x}{k\Delta}\right) < b, \tag{2.4.21}$$

$$\pi\left(\frac{x}{k\Delta}\right) + 1 \leq b. \tag{2.4.22}$$

b is furthermore bounded by the limits in the original sum, (2.2.32), so we have:

$$b_{\min} \leq b \leq b_{\max}, \tag{2.4.23}$$

$$b_{\min} = \max\left\{a+1, \pi\left(\frac{x}{k\Delta}\right)+1\right\} \tag{2.4.24}$$

$$b_{\max} = \begin{cases} \pi(\sqrt{x}) & (k=1) \\ \min\left\{\pi(\sqrt{x}), \pi\left(\dfrac{x}{(k-1)\Delta}\right)\right\} & (k>1) \end{cases} \tag{2.4.25}$$

In practice, it is more computationally convenient to represent the limits in terms of p_b:

$$p_{b_{\min}} \leq p_b \leq p_{b_{\max}}, \tag{2.4.26}$$

$$p_{b_{\min}} = \max\left\{p_{a+1}, \left\lfloor\frac{x}{k\Delta}\right\rfloor + 1\right\}, \tag{2.4.27}$$

$$p_{b_{\max}} = \begin{cases} \lfloor\sqrt{x}\rfloor & (k=1) \\ \min\left\{\lfloor\sqrt{x}\rfloor, \left\lfloor\dfrac{x}{(k-1)\Delta}\right\rfloor\right\} & (k>1) \end{cases} \tag{2.4.28}$$

Finally, combining (2.2.32) and (2.4.11), we have:

$$\phi_2(x,a) = -\binom{\pi(\sqrt{x})-a+1}{2} + \sum_{k=1}^{k_{\max}} \sum_{b=b_{\min}}^{b_{\max}} \phi\left(\frac{x}{p_b}, a\right), \tag{2.4.29}$$

where $k_{\max}$, $b_{\min}$, and $b_{\max}$ are defined by (2.4.10), (2.4.24), and (2.4.25), respectively, and where $y = x/p_b$ is guaranteed to fall within $B_k = [(k-1)\Delta, k\Delta)$.

The summand $\phi\left(x/p_b, a\right)$ in (2.4.29) is evaluated by sieving the interval $[0, z)$, see Sect. 2.5. The primes p_b appearing in (2.4.29) range up to $\lfloor\sqrt{x}\rfloor$. These primes are identified using a secondary sieve, namely an ordinary sieve of Eratosthenes to identify the primes p_b in the interval $[p_{b_{\max}}, p_{b_{\min}}]$. To see that this is reasonable, it is helpful to prove that $p_{b_{\max}} - p_{b_{\min}} < \Delta$, which ensures that this secondary sieve has a smaller memory requirement than the main sieve in the combinatorial algorithm. This proof is straightforward but somewhat cumbersome; we therefore delegate the proof to Appendix A.

2.4.3 Subdividing S_1 and S_{2h}

As shown in Sections 2.3.2, 2.3.4, and 2.3.5, S_0, S_{2t}, and S_{2e} can be computed without directly computing $\phi(y,b)$ by sieving. Therefore, only S_1 and S_{2h} need to be considered when distributing the calculation of $\phi(x,a)$ across blocks B_k. These can be distributed in a similar fashion as $\phi_2(y,b)$. In particular, we find:

$$S_1 = -\sum_{k=1}^{k_{\max}} \sum_{c \le b < a^\star} \sum_{\substack{m_{\min} \le m < m_{\max} \\ p_{\min}(m) > p_{b+1}}} \mu(m) \phi\left(\frac{x}{m p_{b+1}}, b\right), \tag{2.4.30}$$

$$S_{2h} = \sum_{k=1}^{k_{\max}} \sum_{a^\star \le b < \pi(\sqrt[3]{x})} \sum_{d_{\min} \le d < d_{\max}} \phi\left(\frac{x}{p_{b+1} p_d}, b\right), \tag{2.4.31}$$

where

$$m_{\min} = \max\left\{ \left\lfloor \frac{x}{k \Delta p_{b+1}} \right\rfloor + 1, \left\lfloor \frac{\alpha \sqrt[3]{x}}{p_{b+1}} \right\rfloor + 1 \right\}, \tag{2.4.32}$$

$$m_{\max} = \left\lfloor \alpha \sqrt[3]{x} \right\rfloor + 1 \qquad (k = 1), \tag{2.4.33}$$

$$= \min\left(\left\lfloor \alpha \sqrt[3]{x} \right\rfloor + 1, \left\lfloor \frac{x}{(k-1)\Delta p_{b+1}} \right\rfloor + 1 \right) \quad (k \ne 1), \tag{2.4.34}$$

$$d_{\min} = \max\left\{ \pi\left(\frac{x}{k \Delta p_{b+1}}\right) + 1, b + 2 \right\} \tag{2.4.35}$$

$$d_{\max} = \min\left[a+1, \pi\left(\frac{x}{p_{b+1}^2}\right) + 1, \pi\left(\frac{x}{p_{b+1} \alpha \sqrt[3]{x}}\right) + 1 \right] \quad (k = 1), \tag{2.4.36}$$

$$= \min\left[a+1, \pi\left(\frac{x}{p_{b+1}^2}\right) + 1, \pi\left(\frac{x}{p_{b+1} \alpha \sqrt[3]{x}}\right) + 1, \pi\left(\frac{x}{(k-1)\Delta p_{b+1}}\right) + 1 \right] \qquad (k \ne 1). \tag{2.4.37}$$

2.5 Sieve machinery

The "main sieve" in the combinatorial algorithm provides a mechanism by which $\phi(y,b)$ can be rapidly computed for any $y < z$ and $b \le a$. This is performed block-by-block and prime-by-prime: at any given time, one has fast access to $\phi(y,b)$ for a single value of b, for any $y \in B_k$. Starting with $k = 1$ and $b = c$, one adds the contributions to the sums S_1 and S_{2h} that make use of the current value of b and lay in the given block B_k. One then moves to the next value of b, and sieves the interval B_k with p_b. Once $b = a$ is reached, the contribution to ϕ_2 for the given k is added.

Finally, the sieve is re-initialized with $b = c$ and $k \to k + 1$, until the entire interval $[0, z)$ has been sieved, and all contributions added to S_1, S_{2h}, and ϕ_2.

Unlike the explicit formulae for ϕ_2 and ϕ given in this chapter, and the algorithmic improvements to follow in Chapters 3 and 4, we made use of the sieve methods as described in [12] with relatively little modification. Thus, for brevity, we refer the interested reader to [12] for additional details.

```
S2ec = 0;
d = d_min_S2ec;
while (d < d_max_S2ec) {
    l = pi[x / (p[b+1] * p[d])] - b + 1;
    dPrime = min(pi[x / (p[b+1] * p[b+l-1])] + 1, d_max_S2ec);
    S2ec += l * (dPrime - d);
    d = dPrime;
}
```

Figure 2.1: Pseudocode for the computation of S_{2ec}. The pseudocode style used in this thesis is valid C-code, with variable and function definitions omitted. Here `p[b]` represents p_b, and `pi[y]` represents $\pi(y)$. In an actual implementation, there are more cost-effective ways to store $\pi(y)$ than a simple array, see Sect. 3.1.

Chapter 3

Reducing space complexity

Three data structures dominate the memory usage in the combinatorial algorithm [3, 7, 10–12]: a table of $\pi(y)$ for $y \leq y_{\max}$, a table of the smallest prime factor $p_{\min}(y)$, also for $y \leq y_{\max}$, and a set of $\Delta = 2^L$ sieve counters, where a typical choice for $L \in \mathbb{N}$ is $L = \lfloor \log_2 y_{\max} \rfloor$ [10]. Each of these three data structures limits the space complexity of the algorithm to $O(y_{\max})$. The choice $y_{\max} = \alpha x^{1/3}$ with $\alpha = \beta \log^3 x$ for some $\beta \in \mathbb{R}$ is used in the most recent versions of the algorithm [10, 12] to achieve the time complexity $O(x^{2/3} \log^{-2} x)$, simultaneously setting the space complexity at $O(x^{1/3} \log^3 x)$. The next largest data structure is a table of primes p_b for $b \leq \pi(y_{\max})$, which has size $O(y_{\max} / \log y_{\max}) = O(x^{1/3} \log^2 x)$. Thus, to decrease the memory usage of the algorithm by a factor of $\log x$, we must either reduce each of the limiting data structures by a factor of $\log y_{\max}$ or more, or else eliminate them entirely.

We note that not all expositions of the algorithm are limited by all three of the above data structures. For example, Oliveira e Silva was aware that significantly smaller sieve counters can be used than implied by $L = \lfloor \log_2 y_{\max} \rfloor$, although he does advocate storing $\pi(y)$ and $p_{\min}(y)$ for $y \leq y_{\max}$ [12]. This is to be contrasted with Deléglise and Rivat, who use $y_{\max}$ sieve counters, and store $\pi(y)$ for $y \leq y_{\max}$, but manage to eliminate $p_{\min}(y)$ from their final formulae [10].

3.1 Retrieving $\pi(y)$ for $y \leq y_{\max}$ in $O(1)$ time using $O(y_{\max} / \log y_{\max})$ space

The values $\pi(y)$ are used in many places in the algorithm [10, 12]. The authors of past studies advocate the use of a table of values for this purpose, which requires $O(y_{\max})$ storage locations. The implied constant is 1 in the simplest implementation, where a single storage location is used to store a single value of $\pi(y)$. This constant can be reduced somewhat by only storing $\pi(y)$ for those y coprime to the first c primes, for some $c \in \mathbb{N}$. Such a mechanism is commonly referred to as a "wheel" [30]. However, a wheel cannot be used to reduce the space complexity of the algorithm, as the table

used to store the wheel itself grows rapidly, namely with the primorial of c. From a practical point of view, even with a wheel the table $\pi(y)$ becomes prohibitively large, and had to be eliminated to permit the computation of $\pi(10^{26})$.

Given the prime number theorem, it turns out that it is possible to retrieve $\pi(y)$ for any $y \leq y_{\max}$ in constant expected time, using only $O(\pi(y_{\max})) = O(x^{1/3}\log^2 x)$ precomputed values. The method is represented using pseudocode in Figures 3.1 and 3.2, and described here. The trick is to only store $\pi(\widetilde{y})$ for values $\widetilde{y}$ that are multiples of $\lfloor \log_2 y_{\max} \rfloor$. We also make use of a table of all the primes p_b for $b \leq \pi(y_{\max})$: such a table also requires $\pi(y_{\max})$ storage locations, and is anyway required elsewhere in the combinatorial method [10, 12]. The method for determining $\pi(y)$ for a specific value of y is then as follows: firstly, we look up the value $\pi(\widetilde{y})$ at the closest value $\widetilde{y} \leq y$. We then iterate through the array of primes p_b, starting at $b = \pi(\widetilde{y}) + 1$, checking whether $p_b > y$ at each value of b. If $p_b > y$, we return $\pi(y) = b - 1$; if $p_b \leq y$, then we move on to $b + 1$, repeating the process.

The surprising thing is the rapid speed with which this algorithm converges: from the prime number theorem, we expect on average one prime in the range $(\widetilde{y}, y]$, because $y - \widetilde{y} < \lfloor \log_2 y_{\max} \rfloor$. Thus, the most likely situation is that $\pi(\widetilde{y}) = \pi(y)$, i.e., the initial guess for $\pi(y)$ is in fact the correct value, and the algorithm terminates after a single iteration. In practice, in the combinatorial algorithm we retrieve $\pi(y)$ for many values of y, such that the average performance is indeed the relevant quantity. Even in the worst case, it is impossible for this algorithm to require more than $\lfloor \log_2 y_{\max} \rfloor$ iterations, which is $O(\log x)$, because this would contradict the assumption that $\widetilde{y}$ was the closest value $\widetilde{y} \leq y$ in the table $\pi(\widetilde{y})$.

3.2 Iterating over the squarefree m coprime to the first m primes

Determining $p_{\min}(y)$ for any $y \leq y_{\max}$ means finding the smallest prime factor of any such value of y on demand. $p_{\min}(y)$ is accessed sufficiently often that trivial algorithms such as trial factoring are too slow for this purpose.

The author of [12] actually advocated storing the values $p_{\min}(y)\mu(y)$ for $y \leq y_{\max}$, where $\mu(y)$ is the Möbius function, rather than storing $p_{\min}(y)$ in isolation. However, whether $p_{\min}(y)$ and $\mu(y)$ are stored separately or as a product is immaterial for the current analysis. The values $p_{\min}(y)$ require an array of $y_{\max}$ storage locations, each

```
// We map 2^compressionFactor y-values to the same piTilde value.
// piTilde[y>>compressionFactor] then gives a lower bound for pi(y).
compressionFactor = log2log2yMax;
piTilde[yMax >> compressionFactor] = a + 1;
for (b = a; b >= 1; b--) {
    for (y = p[b]; y < p[b + 1]; y++) {
        piTilde[y >> compressionFactor] = b;
    }
}
```

Figure 3.1: Pseudocode describing the initialization of an array storing $\pi(\widetilde{y})$ for values $\widetilde{y}$ that are multiples of $\lfloor \log_2 y_{\max} \rfloor$. This initialization is to be performed only once, after which the procedure described in Fig. 3.2 can be repeatedly applied to retrieve $\pi(y)$ for various $y \leq y_{\max}$. For the purposes of this pseudocode, we assume $\pi(y_{\max}) = a + 1$. We also assume the variable `log2log2yMax` to be initialized with $\lfloor \log_2 \log_2 y_{\max} \rfloor$.

```
// This empty loop finds the greatest piY for which p[piY] <= y.
for (piY = piTilde[y >> compressionFactor]; p[piY + 1] <= y; piY++) {
}
```

Figure 3.2: Pseudocode describing how an array storing $\pi(\widetilde{y})$ for values $\widetilde{y}$ that are multiples of $\lfloor \log_2 y_{\max} \rfloor$ can be used to quickly retrieve $\pi(y)$ for $y \leq y_{\max}$.

of width at least $\log_2 y_{\max}$: 2-bits per y-entry are sufficient to store $\mu(y)$, which is negligible in comparison.

As was the case with the array $\pi(y)$, a wheel can be used to compress the array $p_{\min}(y)$. Indeed, the calculation for $\pi(10^{26})$ was performed using a wheel to compress $p_{\min}(y)$ and $\mu(y)$, see Sect. 5.1. However, even with a wheel the array $p_{\min}(y)$ eventually becomes prohibitively large, and would have precluded a possible future computation of $\pi(10^{27})$. Luckily, it turns out that the data structure $p_{\min}(y)$ can be completely eliminated, and $\mu(y)$ along with it. In order to do this, we investigate the purpose of storing $p_{\min}(y)$ [12].

The only situation where the array $p_{\min}(y)$ is used is in the computation of S_1, see (2.3.10). In order to compute S_1, one must iterate over all squarefree values of $m \in \mathbb{N}$ with $m_{\min} \leq m < m_{\max}$ having $p_{\min}(m) > p_{b+1}$ for varying values of b, $m_{\min}$, and $m_{\max}$. The author of [12] does this by iterating over all $m_{\min} \leq m < m_{\max}$, and explicitly checking the condition $p_{\min}(m) > p_{b+1}$ using an array lookup for $p_{\min}(m)$. $\mu(m)$ is used for exactly the same values of m. Note that we use different indices y and m to refer to the indices in the array $p_{\min}(y)$ as compared to the particular values $p_{\min}(m)$ occurring in (2.3.10). The array $p_{\min}(y)$ may be used to look up the values $p_{\min}(m)$, because $m < m_{\max}$ with $m_{\max} \leq y_{\max} + 1$, so $m \leq y_{\max}$.

In order to eliminate the array $p_{\min}(y)$, we require an iteration scheme over the squarefree numbers $m \in [m_{\min}, m_{\max})$ coprime to the first $b+1$ primes. Although somewhat cumbersome, it is straightforward to construct such an iteration scheme using a variable number of nested loops. Firstly, we loop over the primes

$$m = p_{b_1}, \tag{3.2.1}$$

where b_1 assumes the values:

$$\max\{b+2, \pi(m_{\min}-1)+1\} \leq b_1 < \pi(m_{\max}-1)+1. \tag{3.2.2}$$

Here, (3.2.2) ensures both $p_{\min}(m) > p_{b+1}$ and $m \in [m_{\min}, m_{\max})$. Next, we loop over the biprime numbers

$$m = p_{b_1} p_{b_2}, \tag{3.2.3}$$

where b_1 ranges from $b+2$ until the product $p_{b_1} p_{b_1+1}$ exceeds $m_{\max}$, and b_2 ranges from $b_1 + 1$ until the product $p_{b_1} p_{b_2}$ exceeds $m_{\max}$. For each such biprime value

of m, we explicitly check the condition $m_{\min} \leq m$. We subsequently loop over all numbers m that are the product of three distinct primes p_{b_1}, p_{b_2}, and p_{b_3}, each having $b+1 < b_1 < b_2 < b_3$ and $p_{b_1}p_{b_2}p_{b_3} < m_{\max}$, using similar break conditions as with biprime numbers above. This process is repeated until the largest possible number of factors for m has been exceeded, which occurs when $p_{b+1}p_{b+2}p_{b+3}\ldots p_{b+n} \geq m_{\max}$, where n is the number of prime factors. For example, $p_1p_2\ldots p_{16} = 2{\cdot}3\ldots 53 > 2^{64}$, so if $m_{\max}$ is 64 bits or smaller, then $n < 16$. Furthermore, each value of $m = p_{b_1}p_{b_2}\ldots p_{b_n}$ is squarefree by construction, so $\mu(m) = (-1)^n$ for each m.

3.3 Reducing the size of the sieve counters

Reducing the size of the sieve counters is easy in comparison to $\pi(y)$ and $p_{\min}(y)$. Firstly, we note that one can simply reduce the number of counters, without negative effects on the runtime [12]. By definition, the width of the sieving intervals in the combinatorial algorithm for computing $\pi(x)$ is equal to the number of sieve counters, which we have denoted $\Delta = 2^L$. Given that the upper limit of the sieve is $x/y_{\max}$, there are a total of $x/(2^L y_{\max})$ intervals. Supposing that the overhead per sieving interval is proportional to the number of sieving primes, $\pi(y_{\max})$, the total overhead associated with subdividing the sieving intervals is proportional to $x/(2^L \log x)$ by the prime number theorem. If the overall time complexity is to be kept at $O(x^{2/3}\log^{-2}x)$, then this implies $2^L > \gamma x^{1/3}\log x$, for some constant $\gamma \in \mathbb{R}$. Choosing this minimal value of L results in sieve counters a factor of $\log x$ smaller than needed to achieve our target space complexity of $O(x^{1/3}\log^2 x)$. This is consistent with numerical experiments, where we find that the optimal value of 2^L to minimize the runtime is substantially smaller than $y_{\max}$.

Chapter 4

Parallelizing the combinatorial algorithm

4.1 Shared-memory architectures

There are several practical approaches for parallelizing the algorithm on a shared-memory architecture. Firstly, there is one important part of the algorithm, namely the computation of S_{2e} from Sect. 2.3.5, which is "embarrassingly parallel". Note that "embarrassingly parallel" indicates an algorithm where iterations are completely independent from one another, and can be performed in isolation. Such algorithms are typically easy to implement and scale well on parallel architectures. The so-called easy leaves follow this pattern as they do not depend on the main sieve, do not need to be interleaved with other parts of the algorithm, and can be computed completely in isolation of one another.

The difficult part of the parallelism is the main sieve, where the partial sieve function $\phi(m, b)$ is made available for each $m \leq x/y_{\max}$ and each prime $b \leq \pi(y_{\max})$. The values of $\phi(m, b)$ for smaller values of m and b are needed in order to compute $\phi(m, b)$ for larger m and b, which precludes the embarrassingly parallel computation of $\phi(m, b)$. The approach taken by the current author is to exploit the fact that the sieving is already broken into blocks of length 2^L. Specifically, one sieves each of N subsequent blocks in parallel, working not with $\phi(m, b)$, but with $\phi(m, b) - \phi(m_{\min}, b)$, where $m_{\min}$ is the beginning of the sieving interval under consideration. Each time a value $\phi(m, b)$ needs to be added to a running sum without knowledge of $\phi(m_{\min}, b)$, this discrepancy is recorded in a tally. Once each thread is done sieving the interval $[m_{\min}, m_{\min} + 2^L)$, the values $\phi(m_{\min} + 2^L, b) - \phi(m_{\min}, b)$ can be used to compute each $\phi(m_{\min}, b)$, starting at the smallest value of $m_{\min}$, and the discrepancies represented by the tallies can be resolved.

An algorithm that relies on the above idea has several drawbacks. Firstly, separate sieve counters are needed for each thread, which multiplies the memory usage of the sieve counters by a factor of N. Secondly, the tallies needed to keep track of the

discrepancies between $\phi(m, b)$ and $\phi(m, b) - \phi(m_{\min}, b)$ require a similar amount of memory as the sieve counters. Finally, synchronization is required after each thread sieves a single block, which carries unnecessary overhead. Nonetheless, this approach was found to be efficient enough for the purposes of the author.

After completing the bulk of the current project, the author was made aware of the work of Kim Walisch [31]. Walisch employs an adaptive algorithm for shared-memory parallelism, where blocks are scheduled dynamically depending on the runtime of previous blocks. Such an approach is certainly more efficient than synchronizing each iteration, which is important if a large shared-memory machine is to be used.

Another potentially attractive approach for shared-memory parallelism, in terms of both time and space, would be to combine adaptive scheduling with the distributed-memory parallelism algorithm that will be explained in the next section. By leveraging a distributed-memory algorithm even on a shared-memory architecture, the dependence between subsequent iterations would be broken, completely eliminating the need for communication between threads. Any constant arrays, such as the table of primes p_b for $b \leq \pi(y_{\max})$, could still be shared between the threads to save space on a single shared-memory node.

4.2 Distributed-memory architectures

Distributing the computation of $\pi(x)$ between multiple compute nodes was necessary for the author to compute $\pi(10^{26})$. The principal issue with distributing the computation is that the simplest algorithms described in Sect. 4.1 rely on rapid exchange of information between compute nodes. Although it is in principle possible to efficiently distribute such a calculation, the greatest degree of parallelism can only be achieved if internode communication can be minimized or eliminated.

Fortunately, it is possible to parallelize the combinatorial algorithm for computing $\pi(x)$ in a way that requires no interprocess communication whatsoever, with the exception of summing the contribution to $\pi(x)$ for each job after the fact. This is highly efficient for the machine, but requires use of a supporting algorithm to break the interdependence of the jobs.

The following algorithm for distributed-memory parallelism is loosely based on an unpublished idea of X. Gourdon [11]. Specifically, the issue is that the sums in the

main part of the combinatorial algorithm depend on the partial sieve function $\phi(m, b)$, which represents the count of numbers up to m that are coprime to the first b primes. Sieving an interval $[m_{\min}, m_{\min}+2^L)$ only reveals the values $\phi(m, b)-\phi(m_{\min}, b)$. Thus, determining $\phi(m, b)$ requires storing $\phi(m_{\min}, b)$, updating it after sieving each block, and using the updated value while sieving the next block to obtain any values $\phi(m, b)$ of interest. This approach works fine if the sieve is started at $m_{\min} = 0$, because the recursive dependence terminates with $\phi(0, b) = 0$. If the sieve is to be started somewhere in the middle because, for example, earlier blocks are being simultaneously sieved on some other computer, then we need a method to independently compute $\phi(m_{\min}, b)$.

What is needed is an algorithm that can compute $\phi(m, b)$ for a given value of $m = m_{\min}$ and every $c \leq b \leq \pi(y_{\max})$. An idea for how to do this was given in [11], namely to repeatedly apply the recurrence

$$\phi(m, b) = \phi(m, b-1) - \phi(m/p_b, b-1). \tag{4.2.1}$$

Here c is the size of the wheel being used in the sieve, so $\phi(m, c)$ is accessible for any $m \in \mathbb{N}$ in $O(1)$ time [12]. Given $\phi(m, c)$, the idea is to compute $\phi(m/p_c, c)$ to obtain $\phi(m, c+1)$. This can be done using the same implementation intended for the overall computation of $\pi(x)$, which is able to compute $\phi(x, a)$ for varying values of x and a. The process is then repeated, to obtain $\phi(m, c+2)$, $\phi(m, c+3)$ and onwards up to $\phi(m, \pi(y_{\max}))$.

The difficulty with the above idea is the amount of time needed to perform this process; it would not affect the overall computational complexity of computing $\pi(x)$, but a simple interpretation of this idea was too slow to be used for the computation of $\pi(10^{26})$. The general idea, however, is sound, and modifications can be made to substantially decrease the cost.

The approach taken here is a multifaceted one, where varying methods are used to compute $\phi(m/p_b, b)$ depending on the values of b. Again, $\phi(m, c)$ is available in $O(1)$ time for any $m \in \mathbb{N}$ using the sieving wheel. The wheel can also be used to compute $\phi(m, c+1)$ in $O(1)$ time via $\phi(m, c)$ and $\phi(m/p_c, c)$. The difficult cases occur for $c+2 \leq b \leq \pi(\sqrt{m})$. We first check whether $p_{b-1}^2 \leq m/p_b$. If this is the case, then we directly apply (4.2.1), using the combinatorial algorithm to compute $\phi(m/p_b, b-1)$. If, on the other hand, $p_{b-1}^2 > m/p_b$, then Legendre's formula applies,

such that $\phi(m/p_b, b-1) = \pi(m/p_b)-b+2$. We next check whether $m/p_b < y_{\max}$. If this is the case, then we can use the method described in Sect. 3.1 to retrieve $\pi(m/p_b)$ in $O(1)$ time. If $m/p_b \geq y_{\max}$ then Legendre's formula still applies, but we must compute $\pi(m/p_b)$ by some other method, e.g., using a second application of Legendre's formula or the combinatorial algorithm. For the remaining values $\pi(\sqrt{m}) < b \leq \pi(y_{\max})$, determining $\phi(m, b)$ is trivial given $\phi(m, b-1)$. Specifically, if $m < y_{\max}$ then $\phi(m, b) = \phi(m, b-1) - 1$ for $\pi(\sqrt{m}) + 1 \leq b \leq \pi(m)$ and $\phi(m, b) = 1$ for $\pi(m) + 1 \leq b \leq \pi(y_{\max})$. If $m \geq y_{\max}$, then $\phi(m, b) = \phi(m, b-1) - 1$ for all $\pi(\sqrt{m}) + 1 \leq b \leq \pi(y_{\max})$.

Chapter 5

Numerical calculations

5.1 Implementation details

The description in [12] was used as a starting point for the implementation, with the enhancements of Chapters 3–4 gradually incorporated. The implementation was written in the C99 programming language, with significant effort devoted to ensuring the correctness of the program. Fast unit tests were run on a development machine for every committed version of the code, with more extensive unit tests frequently run on the target cluster. All code was demanded to compile without warning using the GCC 4.9.1 compiler with the default warning level, and to pass static analysis with the Clang Static Analyzer. Precisions of finite-width data types were artificially reduced to intentionally break the program and identify failure modes. Unit tests were written covering wide ranges of parameter values, including edge-cases chosen specifically with the intention of breaking the program. In general, all code was written and checked as strictly as the author was capable at the time of writing.

In Table 5.1 we show resources usage for computing $\pi(10^n)$ using two different versions of the author's implementation of the combinatorial algorithm. In this table, time is measured in "node seconds", i.e., it is the sum of the actual time spent on all compute nodes for that calculation. Similarly, memory usage is memory per node. Here a "compute node" was an IBM iDataplex dx360 M4, having a total of 16 CPU cores (2 × Intel Xeon E5-2670 eight-core 2.60 GHz CPUs) with either 64 or 128 GB RAM (8 GB PC3-12800 ECC RDIMM modules) depending on the requirements of the calculation. Thus, 2.98×10^7 node s for computing $\pi(10^{26})$ corresponds to roughly 15.1 CPU core-years.

Both versions of the software in Table 5.1 implemented the algorithm as described in Chapters 2–4, with the exception that the first version of the software, 2014.10.19, was missing the advancement presented in Sect. 3.2. Version 2014.10.19 was used in the record computations of $\pi(10^{26})$ and $\pi(2^m)$ for $81 \leq m \leq 86$. Note that $\pi(10^{24})$,

	Time [node s]	Memory [bytes]	Time [node s]	Memory [bytes]
x	Version 2014.10.19		Version 2015.01.30	
10^{15}	1.48×10^{0}	3.81×10^{7}	1.18×10^{0}	1.95×10^{7}
10^{16}	6.07×10^{0}	4.31×10^{7}	5.27×10^{0}	2.16×10^{7}
10^{17}	2.68×10^{1}	5.78×10^{7}	2.59×10^{1}	2.71×10^{7}
10^{18}	1.31×10^{2}	1.69×10^{8}	1.08×10^{2}	1.01×10^{8}
10^{19}	5.83×10^{2}	3.44×10^{8}	6.07×10^{2}	1.74×10^{8}
10^{20}	2.89×10^{3}	1.73×10^{9}	2.56×10^{3}	1.27×10^{9}
10^{21}	1.20×10^{4}	3.22×10^{9}	1.04×10^{4}	1.92×10^{9}
10^{22}	5.06×10^{4}	5.81×10^{9}	4.68×10^{4}	2.98×10^{9}
10^{23}	2.27×10^{5}	1.16×10^{10}	2.17×10^{5}	5.23×10^{9}
10^{24}	1.07×10^{6}	2.41×10^{10}	–	1.00×10^{10}
10^{25}	5.25×10^{6}	5.16×10^{10}	–	2.01×10^{10}
10^{26}	2.98×10^{7}	1.12×10^{11}	–	4.16×10^{10}

Table 5.1: Resource usage for computing $\pi(x)$ with $x = 10^n$.

$\pi(10^{25})$, and $\pi(10^{26})$ were not recalculated using version 2015.01.30, because these calculations are computationally expensive. Thus, the runtimes for these values are absent in Table 5.1. The values for the memory usage can be accurately determined without actually performing the calculations, and are thus present in the table.

As can be seen from Table 5.1, eliminating the table of values of $p_{\min}(y)$ substantially reduced the memory usage of the software. The implementation was also slightly faster. This slight improvement in speed was due to the fact that, with the improvement of Sect. 3.2, only the values $m_{\min} \leq m < m_{\max}$ satisfying $p_{\min}(m) > p_{b+1}$ are iterated over in the calculation of S_1 using version 2015.01.30. This is to be compared with version 2014.10.19, where all values $m_{\min} \leq m < m_{\max}$ are iterated over, and the condition $p_{\min}(m) > p_{b+1}$ checked for each m.

5.2 Values of $\pi(x)$ for $x = 10^n$ with $1 \leq n \leq 26$ and $x = 2^m$ with $1 \leq m \leq 86$

The combinatorial algorithm was implemented and used to compute $\pi(10^n)$ for $1 \leq n \leq 26$ and $\pi(2^m)$ for $1 \leq m \leq 86$, see Tables 5.2 and 5.3. The values $\pi(10^n)$ for $1 \leq n \leq 25$ and $\pi(2^m)$ for $1 \leq m \leq 80$ were checked and found to be consistent with the work of previous authors [12, 16]. We note that the values $\pi(2^m)$ for $m = 77, 78, 79, 80$ were previously computed under the assumption of the Riemann hypothesis [16], and

were apparently never verified unconditionally until this study. The values $\pi(10^{26})$ and $\pi(2^m)$ for $81 \leq m \leq 86$ were first reported in this study. These new values were checked in three ways. First, each new value was computed twice, using separate clusters and differing numerical parameters (α, c, and L). Second, the values were checked against the logarithmic integral to ensure the results were reasonable. Third, at the suggestion of Robert Gerbicz [32], the parities of the new values of $\pi(x)$ were checked and found to be consistent with those computed by Lifchitz using a yet-unpublished algorithm [33].

x	$\pi(x)$	$\mathrm{li}(x) - \pi(x)$
10^1	4	2.166
10^2	25	5.126
10^3	168	9.610
10^4	1229	17.137
10^5	9592	37.809
10^6	78498	129.549
10^7	664579	339.405
10^8	5761455	754.375
10^9	50847534	1700.957
10^{10}	455052511	3103.587
10^{11}	4118054813	11587.622
10^{12}	37607912018	38262.805
10^{13}	346065536839	108971.050
10^{14}	3204941750802	314889.954
10^{15}	29844570422669	1052618.581
10^{16}	279238341033925	3214631.793
10^{17}	2623557157654233	7956588.778
10^{18}	24739954287740860	21949555.022
10^{19}	234057667276344607	99877775.223
10^{20}	2220819602560918840	222744643.548
10^{21}	21127269486018731928	597394254.333
10^{22}	201467286689315906290	1932355208.151
10^{23}	1925320391606803968923	7250186215.780
10^{24}	18435599767349200867866	17146907278.151
10^{25}	176846309399143769411680	55160980939.379
10^{26}	1699246750872437141327603	155891678120.791

Table 5.2: Values of $\pi(x)$ for $x = 10^n$.

x	$\pi(x)$	x	$\pi(x)$	x	$\pi(x)$
2^1	1	2^{30}	54400028	2^{59}	14458792895301660
2^2	2	2^{31}	105097565	2^{60}	28423094496953330
2^3	4	2^{32}	203280221	2^{61}	55890484045084135
2^4	6	2^{33}	393615806	2^{62}	109932807585469973
2^5	11	2^{34}	762939111	2^{63}	216289611853439384
2^6	18	2^{35}	1480206279	2^{64}	425656284035217743
2^7	31	2^{36}	2874398515	2^{65}	837903145466607212
2^8	54	2^{37}	5586502348	2^{66}	1649819700464785589
2^9	97	2^{38}	10866266172	2^{67}	3249254387052557215
2^{10}	172	2^{39}	21151907950	2^{68}	6400771597544937806
2^{11}	309	2^{40}	41203088796	2^{69}	12611864618760352880
2^{12}	564	2^{41}	80316571436	2^{70}	24855455363362685793
2^{13}	1028	2^{42}	156661034233	2^{71}	48995571600129458363
2^{14}	1900	2^{43}	305761713237	2^{72}	96601075195075186855
2^{15}	3512	2^{44}	597116381732	2^{73}	190499823401327905601
2^{16}	6542	2^{45}	1166746786182	2^{74}	375744164937699609596
2^{17}	12251	2^{46}	2280998753949	2^{75}	741263521140740113483
2^{18}	23000	2^{47}	4461632979717	2^{76}	1462626667154509638735
2^{19}	43390	2^{48}	8731188863470	2^{77}	2886507381056867953916
2^{20}	82025	2^{49}	17094432576778	2^{78}	5697549648954257752872
2^{21}	155611	2^{50}	33483379603407	2^{79}	11248065615133675809379
2^{22}	295947	2^{51}	65612899915304	2^{80}	22209558889635384205844
2^{23}	564163	2^{52}	128625503610475	2^{81}	43860397052947409356492
2^{24}	1077871	2^{53}	252252704148404	2^{82}	86631124695994360074872
2^{25}	2063689	2^{54}	494890204904784	2^{83}	171136408646923240987028
2^{26}	3957809	2^{55}	971269945245201	2^{84}	338124238545210097236684
2^{27}	7603553	2^{56}	1906879381028850	2^{85}	668150111666935905701562
2^{28}	14630843	2^{57}	3745011184713964	2^{86}	1320486952377516565496055
2^{29}	28192750	2^{58}	7357400267843990		

Table 5.3: Values of $\pi(x)$ for $x = 2^m$.

Chapter 6

Conclusions

6.1 Future work

Two additional approaches for further reducing the size of the sieve counters are apparent to the author. Firstly, it should be possible to substantially reduce the amount of overhead per interval using a variant of the bucket sieve algorithm developed by Oliveira e Silva [1]. The basic idea of bucket sieving is to not sieve every interval by every sieving prime, but rather to allocate each sieving prime to a "bucket" that indicates the next interval in which a multiple of the prime appears. Buckets are then sequentially processed, one bucket per interval, with each sieving prime encountered being moved to a later bucket. In this fashion, the only primes that are encountered in each sieving interval are the ones for which multiples actually appear in that interval. This permits significantly smaller sieving intervals to be used, effectively eliminating the width of the sieving interval as a contributor to memory usage. Such an approach may even permit the entire sieve table to be stored in the processor's data cache, providing greatly enhanced performance as compared to main memory [1].

The other potential approach for further reducing the memory usage of the sieve counters involves more efficiently packing the values. The sieve counters suggested by Oliveira e Silva, and used by the present author, have a fractal-like structure [12]. For a complete description of the workings and necessity of the sieve counters, we direct the reader to [12]. What matters for us is that the counters are each initialized with a number 2^ℓ, for some $0 \leq \ell \leq L$, and then decremented from that initial value. This implies that the largest counters need to be stored using integer data types with at least L bits. Thus, if a common binary representation is used for each of the 2^L sieve counters, then the total storage requirement is $L2^L$ bits. With the sieve counters indexed using a single variable as in [12], one can probably not avoid using a common binary representation for each of the 2^L counters. We note, however, that it is possible to pack the values much more efficiently, resulting in an average of 2 bits

per counter, such that the total requirement is only 2^{L+1} bits.

6.2 Summary

Recent advances in the combinatorial algorithm for computing $\pi(x)$ were presented together with numerical results. Specifically, memory usage has been reduced by a factor of $\log x$, and algorithms for shared- and distributed-memory parallelism have been developed. The resulting algorithm computes $\pi(x)$ using $O(x^{2/3}\log^{-2}x)$ arithmetic operations and $O(x^{1/3}\log^2 x)$ memory locations, each of width proportional to $\log x$. An algorithm for shared memory parallelism appeared previously in the literature [3], but not for the most recent versions of the algorithm [10, 12]; the basic idea necessary for distributed memory parallelism appeared in an unpublished manuscript [11]. The memory reduction presented here appears to be new. Previously reported values [12, 16] of $\pi(10^n)$ for $1 \le n \le 25$ and $\pi(2^m)$ for $1 \le m \le 80$ were verified; the values $\pi(10^{26})$ and $\pi(2^m)$ for $81 \le m \le 86$ were computed and checked in several ways.

We are now in the interesting situation where two different types of algorithms, combinatorial and analytic, are closely matched for practical calculations of $\pi(x)$. If nothing else, this situation gives unprecedented confidence in any numerical results computed consistently using both types of methods, which is currently the case with $\pi(10^n)$ for $1 \le n \le 25$ and $\pi(2^m)$ for $1 \le m \le 80$.

Appendix A

Proof that $p_{b_{\max}} - p_{b_{\min}} < \Delta$ in (2.4.29)

Let $\widetilde{\Delta} = p_{b_{\max}} - p_{b_{\min}}$. We aim to prove $\widetilde{\Delta} < \Delta$. Suppose first that $k = 1$. Then:

$$p_{b_{\max}} = \left\lfloor \sqrt{x} \right\rfloor, \tag{A.0.1}$$

$$p_{b_{\min}} = \max\left\{ p_{a+1}, \left\lfloor \frac{x}{\Delta} \right\rfloor + 1 \right\}, \tag{A.0.2}$$

$$p_{b_{\min}} \geq \left\lfloor \frac{x}{\Delta} \right\rfloor + 1, \tag{A.0.3}$$

$$\widetilde{\Delta} \leq \left\lfloor \sqrt{x} \right\rfloor - \left\lfloor \frac{x}{\Delta} \right\rfloor - 1. \tag{A.0.4}$$

Recall:

$$\lfloor m \rfloor - \lfloor n \rfloor - 1 < m - n \quad \forall \quad m, n \in \mathbb{R}. \tag{A.0.5}$$

Combined with (A.0.4), this gives:

$$\widetilde{\Delta} < \sqrt{x} - \frac{x}{\Delta}, \tag{A.0.6}$$

$$\Delta - \widetilde{\Delta} > \Delta - \sqrt{x} + \frac{x}{\Delta}. \tag{A.0.7}$$

Note that

$$\left(\Delta - \sqrt{x}\right)^2 + \Delta\sqrt{x} > 0 \quad \forall \quad \Delta, x > 0, \tag{A.0.8}$$

so

$$\Delta^2 - \Delta\sqrt{x} + x > 0, \tag{A.0.9}$$

$$\Delta - \sqrt{x} + \frac{x}{\Delta} > 0, \tag{A.0.10}$$

$$\Delta - \widetilde{\Delta} > 0, \tag{A.0.11}$$

$$\Delta > \widetilde{\Delta} \qquad (k = 1). \tag{A.0.12}$$

Suppose now that $k \neq 1$. Then:

$$\widetilde{\Delta} = p_{b_{\max}} - p_{b_{\min}} \tag{A.0.13}$$

$$\leq \left\lfloor \frac{x}{(k-1)\Delta} \right\rfloor - \left\lfloor \frac{x}{k\Delta} \right\rfloor - 1. \tag{A.0.14}$$

Note that for all $m, n \in \mathbb{R}$:

$$\lfloor m \rfloor \leq m, \tag{A.0.15}$$

$$n - 1 < \lfloor n \rfloor, \tag{A.0.16}$$

$$-\lfloor n \rfloor < -n + 1, \tag{A.0.17}$$

$$\lfloor m \rfloor - \lfloor n \rfloor < m - n + 1, \tag{A.0.18}$$

$$\lfloor m \rfloor - \lfloor n \rfloor - 1 < m - n. \tag{A.0.19}$$

This gives:

$$\widetilde{\Delta} < \frac{x}{(k-1)\Delta} - \frac{x}{k\Delta} \tag{A.0.20}$$

$$< \frac{x}{\Delta}\left(\frac{1}{1-k} - \frac{1}{k}\right) \tag{A.0.21}$$

$$< \frac{x}{\Delta k(k-1)}. \tag{A.0.22}$$

We claim:

$$\frac{x}{\Delta k(k-1)} \leq \Delta. \tag{A.0.23}$$

We will prove this claim by contradiction. First, note that if $p_{b_{\max}} < p_{b_{\min}}$, then $\widetilde{\Delta} < 0 < \Delta$, which would prove the entire claim of this appendix, so we are free to take $p_{b_{\max}} \geq p_{b_{\min}}$. This gives:

$$\lfloor \sqrt{x} \rfloor \geq \left\lfloor \frac{x}{k\Delta} \right\rfloor + 1, \tag{A.0.24}$$

$$\sqrt{x} \geq \left\lfloor \frac{x}{k\Delta} \right\rfloor + 1, \tag{A.0.25}$$

but

$$\left\lfloor \frac{x}{k\Delta} \right\rfloor > \frac{x}{k\Delta} - 1, \tag{A.0.26}$$

so we have:

$$\sqrt{x} > \frac{x}{k\Delta}, \tag{A.0.27}$$

$$k\Delta > \sqrt{x}. \tag{A.0.28}$$

We now proceed with the proof of (A.0.23). Suppose

$$\frac{x}{\Delta k(k-1)} > \Delta. \tag{A.0.29}$$

Then

$$\frac{x}{\Delta(k-1)} > k\Delta, \tag{A.0.30}$$

$$\frac{x}{\Delta(k-1)} > \sqrt{x}, \tag{A.0.31}$$

$$\frac{\sqrt{x}}{\Delta} > k-1, \tag{A.0.32}$$

$$k < \frac{\sqrt{x}}{\Delta} + 1. \tag{A.0.33}$$

Meanwhile

$$\widetilde{\Delta} \leq \lfloor \sqrt{x} \rfloor - \left\lfloor \frac{x}{\Delta k} \right\rfloor - 1, \tag{A.0.34}$$

$$\widetilde{\Delta} < \sqrt{x} - \frac{x}{\Delta k}, \tag{A.0.35}$$

but

$$k < \frac{\sqrt{x}}{\Delta} + 1, \tag{A.0.36}$$

so

$$\widetilde{\Delta} < \sqrt{x} - \frac{x}{\Delta\left(\frac{\sqrt{x}}{\Delta}+1\right)}, \tag{A.0.37}$$

$$\widetilde{\Delta} < \sqrt{x} - \frac{x}{\sqrt{x}+\Delta}, \tag{A.0.38}$$

$$\widetilde{\Delta} < \frac{\sqrt{x}(\sqrt{x}-\Delta)-x}{\sqrt{x}+\Delta}, \tag{A.0.39}$$

$$\widetilde{\Delta} < \frac{\sqrt{x}\Delta}{\sqrt{x}+\Delta}, \tag{A.0.40}$$

$$\widetilde{\Delta} < \frac{\Delta}{1+\frac{\Delta}{\sqrt{x}}}, \tag{A.0.41}$$

$$\widetilde{\Delta} < \Delta, \tag{A.0.42}$$

which proves the claim for $k \neq 1$. Combined with (A.0.12), we have $\widetilde{\Delta} < \Delta$ for all k.

Bibliography

[1] T. Oliveira e Silva, S. Herzog, and S. Pardi. Empirical verification of the even Goldbach conjecture and computation of prime gaps up to $4 \cdot 10^{18}$. *Math. Comp.*, 83(288):2033–2060, 2014.

[2] E. Meissel. Ueber die Bestimmung der Primzahlenmenge innerhalb gegebener Grenzen. *Math. Ann.*, 2(4):636–642, 1870.

[3] J. C. Lagarias, V. S. Miller, and A. M. Odlyzko. Computing $\pi(x)$: The Meissel-Lehmer method. *Math. Comp.*, 44(170):537–560, 1985.

[4] E. Meissel. Berechnung der Menge von Primzahlen, welche innerhalb der ersten Hundert Millionen natürlicher Zahlen vorkommen. *Math. Ann.*, 3(4):523–525, 1871.

[5] E. Meissel. Ueber Primzahlmengen. *Math. Ann.*, 21(2):304, 1883.

[6] E. Meissel. Berechnung der Menge von Primzahlen, welche innerhalb der ersten Milliarde natürlicher Zahlen vorkommen. *Math. Ann.*, 25(2):251–257, 1885.

[7] D. H. Lehmer. On the exact number of primes less than a given limit. *Illinois J. Math.*, 3(3):381–388, 1959.

[8] D. C. Mapes. Fast method for computing the number of primes less than a given limit. *Math. Comp.*, 17(82):179–185, 1963.

[9] J. Bohman. On the number of primes less than a given limit. *BIT Numer. Math.*, 12(4):576–577, 1972.

[10] M. Deleglise and J. Rivat. Computing $\pi(x)$: The Meissel, Lehmer, Lagarias, Miller, Odlyzko method. *Math. Comp.*, 65:235–245, 1996.

[11] X. Gourdon. Computation of $\pi(x)$: Improvements to the Meissel, Lehmer, Lagarias, Miller, Odlyzko, Deléglise and Rivat method, 2001. Preprint.

[12] T. Oliveira e Silva. Computing $\pi(x)$: The combinatorial method. *Revista do DETUA*, 4(6):759–768, 2006.

[13] J. C. Lagarias and A. M. Odlyzko. New algorithms for computing $\pi(x)$. *Lect. Notes Math.*, 1052:176–193, 1984.

[14] J. C. Lagarias and A. M. Odlyzko. Computing $\pi(x)$: An analytic method. *J. Algorithms*, 8(2):173–191, 1987.

[15] W. F. Galway. *Analytic Computation of the Prime-Counting Function*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[16] J. Franke, T. Kleinjung, J. Büthe, and A. Jost. A practical analytic method for calculating $\pi(x)$, 2015. To appear in Math. Comp.

[17] D. J. Platt. Computing $\pi(x)$ analytically. *Math. Comp.*, 84(293):1521–1535, 2015.

[18] G. G. Stokes, W. Thomson, J. Glaisher, and J. W. L. Glaisher. Report of the committee, consisting of Professor Cayley, Professor G. G. Stokes, Sir William Thomson, Mr. James Glaisher, and Mr. J. W. L. Glaisher, on mathematical tables. In *Report of the Fifty-Third Meeting of the British Association for the Advancement of Science*, pages 118–126. John Murray, London, 1884.

[19] D. H. Lehmer. *Guide to Tables in the Theory of Numbers*. National Research Council, National Academy of Sciences, Washington, D. C., 1941.

[20] A. Brauer. On the exact number of primes below a given limit. *Amer. Math. Monthly*, 53(9):521–523, 1946.

[21] J. Peetre. Outline of a scientific biography of Ernst Meissel (1826–1895). *Hist. Math.*, 22(2):154–178, 1995.

[22] M. Deleglise. New values of pi(x), 1996. Email to Chris Caldwell. `http://primes.utm.edu/notes/md.html`; accessed Jul. 27, 2015.

[23] M. Deleglise. New values of pi(x), 1996. Email to Chris Caldwell. `http://primes.utm.edu/notes/md6-96.html`; accessed Jul. 27, 2015.

[24] X. Gourdon and P. Sebah. Counting the number of primes, 2001. Preprint. `http://numbers.computation.free.fr/Constants/Primes/countingPrimes.ps`; accessed Jan. 27, 2015.

[25] Multiple contributors. The pi(x) project, 2001. Website. `http://numbers.computation.free.fr/Constants/Primes/Pix/pixproject.html`; accessed Jul. 27, 2015.

[26] T. Oliveira e Silva. Tables of values of $pi(x)$ and of $pi2(x)$. Website. `http://sweet.ua.pt/tos/primes.html`; accessed Feb. 25, 2015.

[27] J. Büthe, J. Franke, A. Jost, and T. Kleinjung. Email from Jens Franke [Thu 7/29/2010 2:47 PM], 2010. `http://primes.utm.edu/notes/pi(10^24).html`; accessed Jan. 27, 2015.

[28] J. Büthe, J. Franke, A. Jost, and T. Kleinjung. Analytic computation of the prime-counting function. `http://www.math.uni-bonn.de/people/jbuethe/topics/AnalyticPiX.html`; accessed Jan. 27, 2015.

[29] D. B. Staple. The combinatorial algorithm for computing $\pi(x)$, 2015. Preprint. `http://arxiv.org` (arXiv:1503.01839 [math.NT]); accessed Jul. 27, 2015.

[30] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer, New York, second edition, 2005. ISBN 0-387-25282-7.

[31] K. Walisch. Fast C++ library for counting primes. Website. `https://github.com/kimwalisch/primecount`; accessed Feb. 22, 2015.

[32] R. Gerbicz. Private communication, 2014.

[33] H. Lifchitz. Quick computation of the parity of $\pi(x)$, 2001. Preprint. `http://www.primenumbers.net/Henri/us/parpius.pdf`; accessed Feb. 24, 2015.