MSc thesis

Master's Programme in Computer Science

Approximating the Permanent of a Matrix with Deep Rejection Sampling

Juha Harviainen

June 7, 2021

Faculty of Science

University of Helsinki

Supervisor(s)

Prof. Mikko Koivisto

Examiner(s)

Prof. Mikko Koivisto, Prof. Jyrki Kivinen

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Koulutusohjelma — Utbildningsprogram — Study programme
Faculty of Science	Master's Programme in Computer Science

Tekijä — Författare — Author

Juha Harviainen

Työn nimi — Arbetets titel — Title

Approximating the Permanent of a Matrix with Deep Rejection Sampling

Ohjaajat — Handledare — Supervisors

Prof. Mikko Koivisto

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc thesis	June 7, 2021	52 pages

Tiivistelmä — Referat — Abstract

Computing the permanent of a matrix is a famous #P-hard problem with a wide range of applications. The fastest known exact algorithms for the problem require an exponential number of operations, and all known fully polynomial randomized approximation schemes are rather complicated to implement and have impractical time complexities. The most promising recent advancements on approximating the permanent are based on rejection sampling and upper bounds for the permanent.

In this thesis, we improve the current state of the art by developing the deep rejection sampling method, which combines an exact algorithm with the rejection sampling method. The algorithm precomputes a dynamic programming table that tightens the initial upper bound used by the rejection sampling method. In a sense, the table is used to jump-start the sampling process.

We give a high probability upper bound for the time complexity of the deep rejection sampling method for random $(0, 1)$-matrices in which each entry is 1 with probability p. For matrices with $p < 1/5$, our high probability bound is stronger than in previous work. In addition to that, we empirically observe that our algorithm outperforms earlier rejection sampling methods by testing it with different parameters against other algorithms on multiple classes of matrices. The improvements in sampling times are especially notable in cases in which the ratios of the permanental upper bounds and the exact value of the permanent are huge.

ACM Computing Classification System (CCS)
Theory of computation → Design and analysis of algorithms → Approximation algorithms analysis

Avainsanat — Nyckelord — Keywords

permanent, approximation algorithms, rejection sampling

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Contents

1 Introduction

The computation of the permanent of a matrix is a famous example of a problem that is #P-hard (Valiant, 1979), implying that it is unlikely for a polynomial time algorithm for the problem to exist. The problem has a wide range of applications: For example, it has been used in communication theory (Smith et al., 2002), multi-target tracking (Kuck et al., 2019; Uhlmann, 2004), and for solving the dimer covering problem of physics (Beichl and Sullivan, 1999). Combinatorially, the permanent of an adjacency matrix of a bipartite graph is equal to the number of its perfect matchings, and for adjacency matrices of directed graphs it corresponds to the number of cycle covers in them (Ben-Dor and Halevi, 1993).

Some special cases of the permanent problem, like computing the permanent of an integer matrix modulo 2^k (Valiant, 1979) or an adjacency matrix of a planar graph (Kasteleyn, 1967), are solvable in polynomial time. On the other hand, even a simpler variant in which we wish to compute the permanent of a matrix whose entries are from $\{0, 1\}$ is a #P-complete problem (Valiant, 1979). In this thesis, we only consider matrices with nonnegative entries, which are a class of matrices for which the problem is #P-hard. In fact, it seems unlikely that the permanent could be approximated with an arbitrary precision in polynomial time for matrices with negative entries (Jerrum et al., 2004).

The best known exact algorithms run in $O(2^n n)$ time for $n \times n$ matrices (Glynn, 2010; Ryser, 1963). The first polynomial time algorithm for approximating the permanent in arbitrary precision was a Markov chain Monte Carlo algorithm developed by Jerrum et al. (2004) and later improved by Bezáková et al. (2008). However, they are rather complicated, and their time complexities are $O(n^{10} \log^2 n)$ and $O(n^7 \log^4 n)$, respectively, so they are impractical to implement. To our knowledge, such an algorithm for this problem has been implemented only once, and its constant factors proved to be too large to be a computationally feasible solution for it (Newman and Vardi, 2020, Sec. 3.5).

The most promising recent advancements on approximating the permanent are based on rejection sampling. In rejection sampling, samples are drawn from some distribution and then either accepted or rejected based on some method to ensure that the samples come from some other desired distribution. Huber (2006) was the first to apply rejection sampling on approximating the permanent. However, their analysis and algorithm works

only for matrices with entries from $\{0, 1\}$. This was improved by Huber and Law (2008), who relaxed the bound so that it works for all matrices with entries in the real interval $[0, 1]$. They also made the matrix nearly doubly stochastic to speed up the rejection sampling process and to prove its worst case time complexity. Recently, Kuck et al. (2019) introduced the adaptive partitioning method, whose purpose is to enable using tighter upper bounds in the hope that it makes the sampling succeed more likely.

This thesis focuses on improving the rejection sampling methods. We introduce the deep rejection sampling method, which precomputes a dynamic programming table to make the initial upper bound of the rejection sampling tighter and to speed up the sampling process. We give a detailed description on how the algorithms can be implemented and also the first analysis of how to implement the adaptive partitioning method work in $O(n^3)$ time per trial given certain assumptions.

We will prove an upper bound for the time complexity of approximating permanents of random matrices when the bound of Huber and Law (2008) is used. The entries of matrices considered here are independently Bernoulli distributed with a parameter p. The expected upper bound will then be $O(n^{3/2+c/p})$ with high probability, in which c is any number greater than $1/2$. This improves a previously known result $O(n^{1/p-1}\omega)$ by Fürer and Kasiviswanathan (2004) when $p < 1/5$, where $\omega = \omega(n)$ is any function that tends to infinity as n tends to infinity.

We show empirically that the deep sampling method outperforms all previous rejection sampling methods by testing the algorithms with different parameters on multiple classes of matrices. Especially on harder classes of matrices, in which the upper bounds differ significantly from the actual value of the permanent, our method can approximate permanents of much larger matrices than other known methods. To our understanding, this is also the first time the effects of preprocessing done by Huber and Law (2008) have been tested empirically.

The present work is based on the author's work in professor Mikko Koivisto's research group and an unpublished research paper (Harviainen et al., 2021). The author has programmed the implementations and computed the empirical results as well as worked on some of the proofs. Our implementations will be made available in a GitHub repository (https://github.com/Kalakuh/permanent) once the paper has been accepted to a journal or a conference.

Chapter 2 explains the necessary preliminaries for understanding the thesis. In Chapter 3, we discuss rejection sampling methods, their efficient implementations for permanents,

and optional preprocessing for the matrices. In Chapter 4 we introduce our deep rejection sampling method and discuss its time complexity for approximating the permanent. In Chapter 5 we analyze the time complexity of the algorithm for random matrices whose entries are independently Bernoulli distributed with a parameter p. In Chapter 6 we compare our deep rejection sampler with different parameters against other approximation algorithms for the permanent by showing empirical results from testing the methods on different classes of matrices. Finally, in Chapter 7 we conclude the thesis and discuss possible future work on the subject.

2 Preliminaries

In this chapter, we will cover the preliminaries required for understanding the rest of the thesis. We start by defining the basic concepts and discussing the connection between permanents and perfect bipartite matching. Then, we go through some complexity theory and give an overview of approximation algorithms for permanents. Finally, we examine one generic approximation scheme that turns out to be useful in approximating the permanent.

2.1 Permanents

We denote the set of integers $\{1, 2, \ldots, n\}$ by $[n]$ and the set of all permutations of n elements by S_n. Then, the *permanent* of an $n \times n$ matrix $A = (a_{ij})$ is defined as

$$\operatorname{per} A = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{\sigma(i),i}.$$

Despite its similarity to the definition of the determinant of a matrix that can be computed in polynomial time, it is generally believed that there does not exist a polynomial time algorithm for computing the permanent.

The *weight of a permutation $\sigma \in S_n$ in A* is the product $\prod_{i=1}^{n} a_{\sigma(i),i}$. This definition allows us to interpret the permanent as the sum of the weights of all permutations of n elements. For matrices with entries from $\{0, 1\}$, the $(0, 1)$-*matrices*, the permanent is equal to the number of ways for choosing a single 1 entry from each column such that all of them are from distinct rows.

2.2 Perfect Bipartite Matchings

Bipartite graphs and matrices are closely related to each other: Consider an $n \times n$ matrix $A = (a_{ij})$, and define a bipartite graph $G = (U, V, E)$, in which $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$ are independent sets of vertices and E is a set of edges such that $\{u_i, v_j\} \in E$ if and only if $a_{ij} > 0$. Let m denote the number of edges in the graph. Define also a function $w \colon E \to \mathbb{R}$ such that $w(\{u_i, v_j\}) = a_{ij}$. This is called the *weight of an edge* $\{u_i, v_j\}$. Then, the matrix is the weighted adjacency matrix of the bipartite graph, and

thus each nonnegative matrix corresponds to a unique weighted bipartite graph, and vice versa.

A *perfect matching* in G is a set of edges of size n such that no two edges have a common endpoint. Therefore, each perfect matching corresponds to some permutation $\sigma \in S_n$: All vertices u_i need to be matched with some vertex v_j, and they all need to be distinct. Hence, we can define $\sigma(i) \coloneqq j$, after which every u_i is matched with $v_{\sigma(i)}$. The *weight of a perfect matching* $\{e_1, e_2, \ldots, e_n\}$ is equal to $\prod_{i=1}^{n} w(e_i)$. By these definitions, it is well known that the permanent of a matrix is equal to the sum of the weights of the perfect matchings in its corresponding bipartite graph. An immediate consequence of this is that computing the sum of those weights is as hard as computing the permanent (Valiant, 1979).

Although computing the sum of the weights of the perfect bipartite matchings is a hard problem, there are many algorithms for finding a single perfect matching in polynomial time. The fastest currently known algorithm for finding one such matching has time complexity $\tilde{O}\left(m + n^{1.5}\right)$ (van den Brand et al., 2020), where the $\tilde{O}$ means the O-notation in which the logarithmic factors are ignored. However, the $O\left(m\sqrt{n}\right)$ time Hopcroft–Karp algorithm (Hopcroft and Karp, 1973) is enough for most purposes. In Chapter 3, we will use bipartite matchings to preprocess the matrices so that they have desirable properties.

2.3 Complexity Theory

Complexity theory gives us tools for discussing how hard solving or approximating some problem roughly is. The complexity classes #P, #P-*hard*, and #P-*complete* are related to counting problems, in which we want to count the number of solutions. The terms were first defined by Valiant (1979) as follows:

Definition 1. A problem Π is in #P if it is equivalent to counting the number of acceptable solutions computable by a nondeterministic polynomial time Turing machine.

Definition 2. A problem Π is called #P-hard if every problem in #P can be solved by a deterministic polynomial time Turing machine that has access to an oracle for the problem Π.

Definition 3. A problem Π is called #P-complete if it is in #P and it is #P-hard.

The term *oracle* mentioned in the definitions refers to a machine that solves a certain problem in $O(1)$ time.

Computing the permanent of a $(0,1)$-matrix is a classic example of a #P-complete problem (Valiant, 1979), and thus the problem is #P-hard for nonnegative matrices. It is generally assumed that no #P-complete problem is solvable in polynomial time. Sometimes, a single solution can be found in polynomial time, but counting all of them is a #P-complete problem, as is the case with the perfect bipartite matching problem (Valiant, 1979).

2.4 Approximation Algorithms

As no polynomial time exact algorithm is expected to exist for the problem of computing the permanent, a natural next step is to look for approximation algorithms for the problem. To discuss them, we will first define some terminology:

Definition 4. A randomized algorithm is an (ϵ, δ)-*approximation algorithm* if it computes a value that is within a factor of $1+\epsilon$ of the optimal value with a probability at least $1-\delta$ for any problem instance and numbers $\epsilon, \delta > 0$.

Definition 5. An algorithm is a *fully polynomial time randomized approximation scheme* (*FPRAS*) if it is an (ϵ, δ)-approximation algorithm for any given $\epsilon, \delta > 0$ and its running time is polynomial in n and $1/\epsilon$.

The first FPRAS for estimating the permanent of a nonnegative $n \times n$ matrix was developed by Jerrum et al. (2004). Their method uses Markov chain Monte Carlo with a state space of all perfect and near-perfect matchings, that is, matchings with $n-1$ edges. This approach was later improved by Bezáková et al. (2008), reducing the time complexity from $O(n^{10} \log^2 n)$ to $O(n^7 \log^4 n)$.

One way of computing an (ϵ, δ)-approximation of the permanent are *Godsil–Gutman type estimators* (1981). The original estimators by Godsil and Gutman create a new matrix by taking the square root of each entry in the matrix and multiplying them independently by a random element from $\{-1, 1\}$. They then compute the square of the determinant of the modified matrix to obtain an estimate whose expected value is the permanent of the original matrix. Later, Karmarkar et al. (1993) introduced a new improved estimator in which each element is multiplied by a random 3rd root of unity and proved the time complexity of getting an (ϵ, δ)-approximation of the permanent with both estimators to be exponential in n. Barvinok (1999) used a somewhat similar approach with quaternions, and later Chien et al. (2003) generalized Godsil–Gutman type estimators for all second Clifford algebras.

Chien et al. (2003) also showed that there would exist an FPRAS based on Godsil–Gutman type estimators if one were able to compute the determinant of matrices with elements from arbitrary second Clifford algebras in polynomial time. Unfortunately, only the first two second Clifford algebras are commutative, so no efficient way for computing this determinant in the general case is known. For more research on estimators based on noncommutative algebras, see, for example, the works of Arvind and Srinivasan (2010), Chien et al. (2011), and Moore and Russell (2012).

Even if an FPRAS is unobtainable from Godsil–Gutman type estimators, they are still quite efficient most of the time for approximating the permanent. A result conjectured by Karmarkar et al. (1993) and proven by Frieze and Jerrum (1995) shows that for random matrices $O(n)$ samples tend to be enough. However, they also argue that Godsil–Gutman type estimators are not as trustworthy as other estimators in a sense that there is no known way to find out how hard an instance is. To always guarantee getting an (ϵ, δ)-approximation, we need take a number of samples equal to the worst case scenario, which is roughly $3^{n/2}$ for the estimator based on reals, $2^{n/2}$ for complex numbers, and $(3/2)^{n/2}$ for quaternions.

To approximate the permanent of larger matrices for which the Godsil–Gutman type estimators are not feasible, importance sampling can be used. The basic idea of the importance sampling is to draw samples from some distribution and then use the probability distribution and the weights of the samples for the estimation. This approach for permanents has been investigated by, for example, Kuznetsov (1996), Beichl and Sullivan (1999), Smith and Dawkins (2001), and Chen and Liu (2007). Although importance sampling estimators can have a smaller variance than Godsil–Gutman type estimators, giving good guarantees about their performance seems to be hard because the magnitude of variance is hard to approximate.

Rejection samplers are easier to analyze than the previous estimators. This is because, in a sense, we simply draw Bernoulli distributed random variables from a specific distribution. The variables are then used to approximate the permanent using known bounds for sums of Bernoulli distributed variables. We will cover the rejection samplers in more detail in the next chapter.

Approximation algorithms that estimate the permanent in polynomial time within a factor ϵ for almost all $(0, 1)$-matrices have been given by, for example, Rasmussen (1994) and Fürer and Kasiviswanathan (2004). The Rasmussen estimator produces an approximation in $O(n^3\omega)$ time, where $\omega = \omega(n)$ is any function that tends to infinity as n tends to infinity.

The estimator of Fürer and Kasiviswanathan (2004) does the same for matrices that are Bernoulli distributed with parameter p, yielding time complexities $O(n^2\omega)$ for $p > 1/3$, $O(n^2 \log^2 n \cdot \omega)$ for $p = 1/3$, and $O(n^{1/p-1}\omega)$ for $p < 1/3$.

2.5 Gamma Bernoulli Approximation Scheme

Assume that we are interested in approximating some value p with $0 \leq p \leq 1$, but we can only draw Bernoulli random variables with parameter p without knowing the parameter. As we do not know the value of p, it is nontrivial to estimate how many samples we require to achieve the desired accuracy. For this purpose, we use Huber's (2017) Gamma Bernoulli approximation scheme (GBAS), which returns an unbiased estimator $\hat{p}$ of the parameter p.

The algorithm requires a parameter k to be computed before running it. This parameter describes how many successful samples are needed for the desired accuracy. Huber (2017) shows that $p/\hat{p} \sim \text{Gamma}(k, k-1)$ and notes that

$$\Pr\left(|\hat{p}/p - 1| > \epsilon\right) = \Pr\left(p/\hat{p} < (1+\epsilon)^{-1} \text{ or } p/\hat{p} > (1-\epsilon)^{-1}\right).$$

Given that we are able to compute the cumulative distribution function of the gamma distribution, these two properties together provide us a straightforward way for finding a value k that is guaranteed to give an (ϵ, δ)-approximation of p simply by increasing it until the probability of failure is low enough. In the case that the cumulative distribution function is not available, Huber gives an upper bound

$$k \geq 2\epsilon^{-2}(1 - (4/3)\epsilon)^{-1}\ln(2\delta^{-1})$$

for the required value of k. Algorithm 1 describes in pseudocode how $\hat{p}$ is computed for a given k.

Algorithm 1: GBAS

Input : An integer k

Output: An estimate $\hat{p}$

$s \leftarrow 0$, $r \leftarrow 0$;

while $s \neq k$ **do**

  $X \leftarrow \text{Bernoulli}(p)$, $A \leftarrow \text{Exp}(1)$;

  $s \leftarrow s + X$, $r \leftarrow r + A$;

$\hat{p} \leftarrow (k-1)/r$;

return $\hat{p}$;

3 Rejection Sampling

Rejection sampling is a method for estimating some value by repeatedly drawing samples from a distribution and either rejecting or accepting them. The rejected and accepted samples then contribute to the estimate in some way. In this chapter, we will start by discussing how rejection sampling works and how it is applied in approximating the permanent. Then, we give detailed implementations of the rejection samplers of Huber and Law (2008) and Kuck et al. (2019), and, finally, we will examine certain preprocessing techniques for matrices that are sometimes helpful in rejection sampling.

3.1 Nesting Partition Trees

Partition trees provide a mathematical way for formalizing how the rejection sampling algorithms for the permanent make decisions. The algorithms of Huber (2006) and Huber and Law (2008) use the same simple fixed partition tree, so the concept is not necessary there, but for the adaptive partitioning (AdaPart) method of Kuck et al. (2019) it simplifies the discussion.

Consider a rooted tree T and some arbitrary nonempty set Ω. The tree T is called a partition tree if each node of the tree has a nonempty subset of Ω associated with it such that

1. the subset associated with the root of T is Ω,
2. all sets associated with leaf nodes are singletons, and
3. for any internal node and its associated subset Ω', the subsets associated with its children are a partition of Ω'.

An example of such a tree for the permutations of $\{1, 2, 3\}$ can be found in Figure 3.1.

Furthermore, assume each element $\sigma \in \Omega$ is associated with a weight $w(\sigma)$. We say that a partition tree T of Ω is *nesting* with respect to some function $U\colon \mathcal{P}(\Omega) \to \mathbb{R}$ if $U(\{\sigma\}) = w(\sigma)$ for all $\sigma \in \Omega$ and for all internal nodes v and their children $v_1, v_2, \ldots, v_\ell$

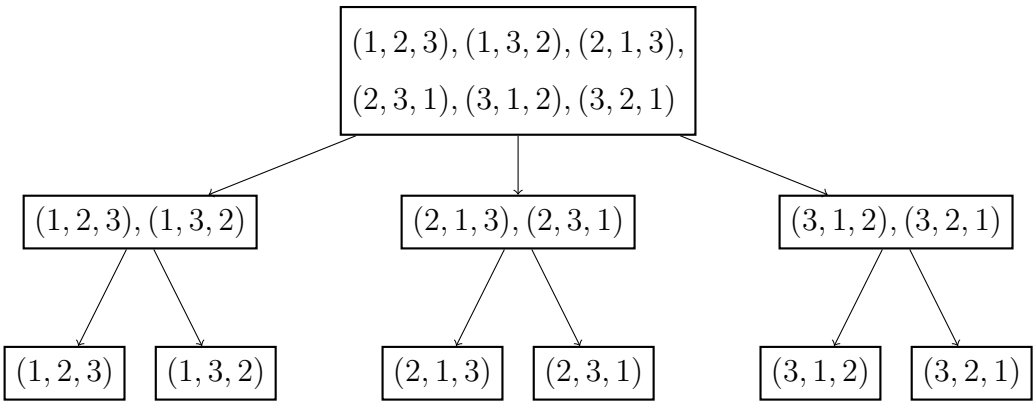

Figure 3.1: An example of a partition tree for the permutations of $\{1, 2, 3\}$

with associated sets Ω' and $\Omega'_1, \Omega'_2, \ldots, \Omega'_\ell$, respectively, it holds that

$$U(\Omega') \geq \sum_{i=1}^{\ell} U(\Omega'_i).$$

It is worth noting that any upper bound U has at least one nesting partition tree because

$$U(\Omega) \geq \sum_{\sigma \in \Omega} U(\{\sigma\}).$$

3.2 Rejection Sampling

The basic idea of *rejection sampling* is to repeatedly take samples from a distribution and either *accept* or *reject* them. Then, the ratio of accepted samples and the total number of samples is used to estimate some value, which is the permanent in our case. For the permanent problem, the method was first used by Huber (2006) and later by Huber and Law (2008) and Kuck et al. (2019).

Assume that we have a partition tree T of Ω that is nesting with respect to a bound U and that we are in some internal node v of the tree. Denote the subset of Ω associated with v by Ω' and the subsets associated with the children $v_1, v_2, \ldots v_\ell$ by $\Omega'_1, \Omega'_2, \ldots, \Omega'_\ell$, respectively. We call the number

$$U(\Omega') - \sum_{i=1}^{\ell} U(\Omega'_i)$$

the *slack* of v. The slack is always nonnegative because the partition tree is nesting.

We can use the values $U(\Omega'_i)$ to create a probability distribution for moving from v to one of its children: We move from v to v_i with probability

$$\frac{U(\Omega'_i)}{U(\Omega')}$$

and choose the slack with probability

$$\frac{U(\Omega') - \sum_{i=1}^{\ell} U(\Omega'_i)}{U(\Omega')}.$$

If the slack is chosen, we reject the sample. If we start from the root and never reject the sample, we will eventually get to some leaf node v with an associated subset Ω'. By the definition of the partition tree, Ω' is a singleton $\{\sigma\}$ for some $\sigma \in \Omega$. This σ will be our accepted sample. Let $\Omega_1, \Omega_2, \ldots, \Omega_\ell$ be the subsets associated with the nodes on the path from the root of the partition tree to v. Then, the probability of sampling σ is equal to

$$\frac{U(\Omega_2)}{U(\Omega_1)} \cdot \frac{U(\Omega_3)}{U(\Omega_2)} \cdot \ldots \cdot \frac{U(\Omega_\ell)}{U(\Omega_{\ell-1})} = \frac{U(\Omega_\ell)}{U(\Omega_1)} = \frac{w(\sigma)}{U(\Omega)}.$$

Therefore, we get an accepted sample with probability

$$\frac{\sum_{\sigma \in \Omega} w(\sigma)}{U(\Omega)}.$$

If R is the ratio of the number of accepted samples and the total number of samples taken by a rejection sampling algorithm, then

$$\mathbf{E}[R] \cdot U(\Omega) = \sum_{\sigma \in \Omega} w(\sigma).$$

This is written as pseudocode in Algorithm 2. By integrating Algorithm 2 with GBAS such that for an accepted sample we return 1 and for a rejected sample 0, we can compute an (ϵ, δ)-approximation of $\mathbf{E}[R]$.

In the context of the permanent of an $n \times n$ matrix A, Ω is the set of permutations of positive weight in A and $U = U(A)$ is some permanental upper bound. Then, the sum $\sum_{\sigma \in \Omega} w(\sigma)$ is equal to the permanent of A. Therefore, by repeatedly generating permutations step by step, we can try sampling a permutation of positive weight in A.

3.3 Permanental Upper Bounds

Permanental upper and lower bounds have been researched extensively, as they provide a straightforward way of getting a somewhat precise approximation of the magnitude of the permanent. They also play an integral role in rejection sampling for the permanent problem.

Algorithm 2: Rejection sampling

Input : The root r of a nesting partition tree and an upper bound U

Output: An accepted sample σ or *slack*

Let Ω' be the set associated with r;

while $|\Omega'| > 1$ **do**

  Let $v_1, v_2, \ldots, v_\ell$ be the children of r;

  Let Ω'_k be the set associated with v_k for all $k \in [\ell]$;

  $S \leftarrow \sum_{k=1}^{\ell} U(\Omega'_k)$;

  Let X be v_k with probability $U(\Omega'_k)/U(\Omega')$ and *slack* with probability $(U(\Omega') - S)/U(\Omega')$;

  if $X = slack$ **then**

    return *slack*;

  $r \leftarrow X$;

  Let Ω' be the set associated with r;

if $|\Omega'| = 0$ **then**

  return *slack*;

Let σ be the only element in Ω';

return σ;

Consider a function U that gives an upper bound for the permanent of any nonnegative matrix $A = (a_{ij})$ of size $n \times n$. We call such a bound a *permanental upper bound*. In other words, it gives an upper bound for the sum of the weights of all the permutations $\sigma \in S_n$ with a positive weight in A. The ratio $U(A)/\operatorname{per} A$ is called the *fit ratio of A with respect to U*.

With a slight abuse of terminology, for a matrix A with the set of permutations of positive weight Ω, we call a set of matrices $A_1, A_2, \ldots, A_\ell$ such that

1. for every $A_k = (a'_{ij})$ it holds that $A_k \neq A$ and either $a'_{ij} = 0$ or $a'_{ij} = a_{ij}$ for all $i, j \in [n]$,
2. any permutation $\sigma \in \Omega$ has a positive weight in exactly one matrix A_k, and

a *partition of A*. If also $\sum_{k=1}^{n} U(A_k) \leq U(A)$, then we say that this partition of A is nesting with respect to U.

If every matrix A has a nesting partition, then we have in fact found a way to construct a nesting partition tree for the permutations of positive weight in any A: Denote the sets of

permutations of positive weight in A and in its nesting partition $A_1, A_2, \ldots, A_\ell$ by Ω and $\Omega_1, \Omega_2, \ldots, \Omega_\ell$, respectively. After that we can construct a bound U' with $U'(\Omega) = U(A)$ and $U'(\Omega_k) = U(A_k)$ and let the partition of Ω be $\{\Omega_1, \Omega_2, \ldots, \Omega_\ell\}$. We say that a nesting partition tree constructed using a permanental upper bound U is a *partition tree with respect to (a permanental upper bound) U*.

Let $f(A, i, j)$ denote the matrix where all entries in the i-th row and in the j-th column have been replaced with zeros in the matrix $A = (a_{ij})$ with the exception that a_{ij} remains the same. The permanent of a matrix has the property called *self-reducibility*, which in this case means that

$$\operatorname{per} A = \sum_{i=1}^{n} a_{ij} \operatorname{per} f(A, i, j).$$

Note that $U(f(A, i, j))$ gives an upper bound for the sum of the weights of all $\sigma \in S_n$ in A with $\sigma(j) = i$. Let Ω be the set of permutations of positive weight in A. If we choose a column j from the matrix and create matrices $f(A, 1, j), f(A, 2, j), \ldots, f(A, n, j)$, then each permutation $\sigma \in \Omega$ with $\sigma(j) = i$ has a positive weight only in $f(A, i, j)$. This suggests that creating matrices $f(A, 1, j), f(A, 2, j), \ldots, f(A, n, j)$ could be a viable way of partitioning A. We call these matrices the *jth column partition of A*.

The simplest permanental upper bound for nonnegative matrices is the so-called *trivial bound*, which is equal to the product of all row sums of a matrix. Unsurprisingly, the trivial bound is really weak and mostly interesting only as an example. However, like other permanental upper bounds, it can be *sharpened* to make it significantly tighter in some cases (Soules, 2003). Unfortunately, sharpening bounds tends to be time expensive and it is hard to guarantee that there are efficiently discoverable partition trees with respect to sharpened bounds.

One well-known bound for $(0, 1)$-matrices is the Minc–Brègman bound conjectured by Minc (1963) and first proven by Brègman (1973). Alternative proofs have been later given by Schrijver (1978) and Radhakrishnan (1997). We will denote this bound by U^{M}, and it is defined as follows:

$$U^{\mathrm{M}}(A) = \prod_{i=1}^{n} (r_i!)^{1/r_i},$$

where r_i is the sum of the elements on the i-th row. It is worth noting that the bound is tight when the rows and columns of a matrix can be permuted such that it becomes a block diagonal matrix.

One of the tightest currently known permanental bound for nonnegative matrices that is also efficiently computable was developed by Soules (2005). It is defined by letting a^*_{ij}

be the jth largest entry on the ith row of A and defining $\gamma(0) = 0$, $\gamma(j) = (j!)^{1/j}$, and $\delta(j) = \gamma(j) - \gamma(j-1)$. Then, for any nonnegative $n \times n$ matrix A, it holds that

$$\operatorname{per} A \leq \prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^{*} \delta(j).$$

We denote this *Soules bound* by $U^{\mathrm{S}}(A)$. For $(0,1)$-matrices, the Soules bound reduces to the Minc–Brègman bound.

Kuck et al. (2019) observed that for most nonnegative matrices A there is at least one column partition that is nesting when the Soules bound is used. However, this is not always the case: Consider, for example, the matrix

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

whose upper bound is $4\sqrt{6}$. Whichever column j we choose, the sum of the upper bounds of the matrices in the partition is equal to $2\sqrt{2}\sqrt[3]{6} + \sqrt{2}\sqrt[3]{36} \approx 9.809$, which is greater than $4\sqrt{6} \approx 9.798$. A similar result follows if we fill the main diagonal of any $4n \times 4n$ matrix with such 4×4 matrices and let the other entries be zeros. This also shows that the Minc–Brègman does not always nest.

Huber (2006) developed a relaxation of the Brègman–Minc bound for $(0,1)$-matrices by using a recursive formula in which $g(0) = 0$, $g(1) = \mathrm{e}$ and

$$g(r+1) = g(r) + 1 + \frac{1}{2g(r)} + \frac{0.6}{g(r)^2}.$$

Then,

$$U^{\mathrm{H}}(A) = \prod_{i=1}^{n} \frac{g(r_i)}{\mathrm{e}}$$

is an upper bound for the permanent of a matrix A with row sums $r_1, r_2, \ldots, r_n$. For all r, it holds that $\gamma(r) \leq g(r)$. The motivation for this relaxed bound is that all column partitions of A are nesting with respect to U^{H}.

Later, Huber and Law (2008) found a similar bound for matrices A with entries in $[0,1]$: Let

$$h(r) = \begin{cases} r + (1/2)\ln(r) + e - 1, & r \geq 1 \\ 1 + (e-1)r, & r \in (0,1) \end{cases}.$$

Then,

$$U^{\mathrm{HL}}(A) = \prod_{i=1}^{n} \frac{h(r_i)}{\mathrm{e}}$$

is an upper bound for A. It is worth noticing that the U^{HL} bound is not as tight as U^{H}, but it works for all nonnegative matrices because they can be scaled to contain only entries in $[0, 1]$.

For more permanental upper bounds, see for example (Gurvits and Samorodnitsky, 2014; Jurkat and Ryser, 1966; Soules, 2000; Soules, 2003). Also, Soules (2005) has discussed extensively the history of permanental bounds.

3.4 Huber–Law Sampler

We consider next two rejection sampling methods. The first one is based on the Huber–Law bound and is straightforward to implement as every row partition is guaranteed to nest (Huber and Law, 2008). The main idea is that, for a matrix $A = (a_{ij})$, the upper bound $U^{\mathrm{S}}(f(A, i, j))$ can be written as

$$a_{ij} \cdot \prod_{\substack{s \in [n] \\ s \neq i}} \frac{h(r_s - a_{sj})}{\mathrm{e}}.$$

By defining values $x_0, x_1, \ldots, x_n$ and $y_1, y_2, \ldots, y_{n+1}$ for a fixed column j such that

$$x_0 \coloneqq 1, \qquad x_k \coloneqq \frac{h(r_k - a_{sj})}{\mathrm{e}} \cdot x_{k-1} \text{ for all } k \in [n],$$

and

$$y_{n+1} \coloneqq 1, \qquad y_k \coloneqq \frac{h(r_k - a_{sj})}{\mathrm{e}} \cdot y_{k+1} \text{ for all } k \in [n],$$

this can be further rewritten as

$$U^{\mathrm{S}}(f(A, i, j)) = a_{ij} x_{i-1} y_{i+1}.$$

If the row sums are known beforehand, we can precompute the values x_k and y_k and then use them for computing the upper bounds for all $i \in [n]$ in $O(n)$ time. The sampler proceeds column by column and computes the upper bounds whilst keeping track of the row sums. After the sampler selects row i for the column, A is replaced by $f(A, i, j)$. If a sample is not rejected in any of the columns, we accept the sample. This is described as a pseudocode in Algorithm 3. If a sample gets rejected, the code returns *slack*. The time

complexity of the algorithm is $O(n^2)$ due to the $O(n^2)$ precomputation of the row sums and $O(n)$ work per column.

Algorithm 3: Huber–Law rejection sampling

Input : An $n \times n$ nonnegative matrix A

Output: An accepted permutation σ or *slack*

$r_i \leftarrow \sum_{j=1}^{n} a_{ij}$ for $i = 1, \ldots, n$;
$ub \leftarrow U(A)$;
$\sigma \leftarrow (null, null, \ldots, null)$;
for $j = 1, \ldots, n$ **do**
  $x_0 \leftarrow 1$;
  $y_{n+1} \leftarrow 1$;
  for $i = 1, \ldots, n$ **do**
    $x_i \leftarrow h(r_i)/\mathrm{e} \cdot x_{i-1}$;
    $y_{n+1-i} \leftarrow h(r_i)/\mathrm{e} \cdot x_{n+2-i}$;
  Let $p(i)$ be a probability mass function for $i = 1, 2, \ldots, n$ and $i = slack$;
  for $i = 1, \ldots, n$ **do**
    $p(i) \leftarrow a_{ij} x_{i-1} y_{i+1}$;
  $p(slack) \leftarrow ub - \sum_{i=1}^{n} p(i)$;
  Select i randomly by using $p(i)$ as a probability mass function;
  if $i = slack$ **then**
    return *slack*;
  $\sigma(j) \leftarrow i$;
  $ub \leftarrow p(i)$;
  $A \leftarrow f(A, i, j)$;
  for $k = 1, \ldots, n$ **do**
    $r_k \leftarrow r_k - a_{kj}$;
  $r_i \leftarrow a_{ij}$;
return σ;

3.5 Adaptive partitioning method

The AdaPart method of Kuck et al. (2019) is another way of doing rejection sampling. It is based on the tighter Soules bound, whose main disadvantage is that it is not guaranteed to nest in every column, but, as Kuck et al. (2019) observed, there tends to be at least

column partition in which the bound nests. In AdaPart, we maintain a set of matrices, starting with $\{A\}$. Each iteration until the sum of the upper bounds for matrices in the set is at most $U^{\mathrm{S}}(A)$, we pick a random matrix B from the set, compute the sum of upper bounds for each column partition, and choose the column with the smallest sum. Then, we replace B by the matrices of that column partition. Finally, after the maintained set is a nesting partition of A, a matrix is sampled from the set with probability relative to its upper bound, and current A is replaced with it. This process is repeated until A contains exactly one permutation of positive weight or the sample is rejected. Algorithm 4 implements this as pseudocode.

Algorithm 4: Adaptive Partitioning

Input : An $n \times n$ nonnegative matrix A

Output: An accepted permutation σ or *slack*

while *there is more than one permutation of positive weight in* A **do**
- $P \leftarrow \{A\}$;
- $ub \leftarrow U^{\mathrm{S}}(A)$;
- **do**
  - Pick B randomly from P;
  - **for** $j = 1, \ldots, n$ **do**
    - **if** *there is more than* 1 *positive value in column* j **then**
      - $ub_j \leftarrow \sum_{i=1}^{n} U^{\mathrm{S}}(f(B, i, j))$;
    - **else**
      - $ub_j \leftarrow \infty$;
  - $j \leftarrow \arg\min_{i \in [n]} ub_i$;
  - $P \leftarrow (P \setminus \{B\}) \cup \{f(B, 1, j), f(B, 2, j), \ldots, f(B, n, j)\}$;
  - $ub \leftarrow ub - U^{\mathrm{S}}(B) + ub_j$;
- **while** $ub \leq U^{S}(A)$;
- Let A be equal to a matrix $C \in P$ with probability $U^{\mathrm{S}}(C)/U^{\mathrm{S}}(A)$ or *slack* with probability $1 - ub/U^{\mathrm{S}}(A)$;
- **if** $A =$ *slack* **then**
  - **return** *slack*;

Let σ be the only permutation of positive weight in A;

return σ;

To our understanding, one iteration of the do while loop cannot be implemented to be faster than $O(n^2)$. If $O(1)$ iterations are always enough, then this gives an algorithm with

$O(n^3)$ time complexity per trial, as at least one new column contains exactly one positive value each time after the do while loop is completed. It is still unclear if there exists a class of instances which can be used to force the do while loop to take arbitrarily many iterations until it completes.

The most complicated part of the implementation is efficiently computing upper bounds $U^{\mathrm{S}}(f(A,i,j))$ for a matrix A and all $i,j \in [n]$. Again, the basic idea is to write the upper bound in such a way that we can precompute some values to compute the upper bounds more efficiently. For this, we maintain a permutation π_i for each row i such that $\pi_i(t) = j$ if a_{ij} is the tth largest entry of the row t (ties can be broken arbitrarily). Then, we write

$$U^{\mathrm{S}}(f(A,i,j)) = a_{ij} \prod_{\substack{s\in[n]\\ s\neq i}} \left(\sum_{t=1}^{\pi_s^{-1}(j)-1} a_{s,\pi_s(t)}\delta(t) + \sum_{t=\pi_s^{-1}(j)+1}^{n} a_{s,\pi_s(t)}\delta(t-1) \right).$$

This allows us to use a similar method as with the Huber–Law bound. By defining

$$x_{0j} \coloneqq 1, \quad x_{sj} \coloneqq \left(\sum_{t=1}^{\pi_s^{-1}(j)-1} a_{s,\pi_s(t)}\delta(t) + \sum_{t=\pi_s^{-1}(j)+1}^{n} a_{s,\pi_s(t)}\delta(t-1) \right) \cdot x_{s-1,j} \text{ for all } s \in [n]$$

and

$$y_{n+1,j} \coloneqq 1, \quad y_{sj} \coloneqq \left(\sum_{t=1}^{\pi_s^{-1}(j)-1} a_{s,\pi_s(t)}\delta(t) + \sum_{t=\pi_s^{-1}(j)+1}^{n} a_{s,\pi_s(t)}\delta(t-1) \right) \cdot y_{s+1,j} \text{ for all } s \in [n],$$

we can write

$$U^{\mathrm{S}}(f(A,i,j)) = a_{ij} x_{i-1,j} y_{i+1,j}.$$

If we incrementally precompute the $2n^2$ sums

$$\sum_{t=1}^{k} a_{s,\pi_s(t)}\delta(t), \text{ for } k \in \{0,1,\ldots,n-1\}, s \in [n]$$

and

$$\sum_{t=k}^{n} a_{s,\pi_s(t)}\delta(t-1), \text{ for } k \in \{2,3,\ldots,n+1\}, s \in [n]$$

in $O(n^2)$ time, we can compute the values x_{sj} and y_{sj} in $O(n^2)$ time. After computing these, it is trivial to get any upper bound $U^{\mathrm{S}}(f(A,i,j))$ in constant time.

It is worth noting that we cannot actually keep the matrices in the set of matrices P. Instead, we need to keep track of how A and any $B \in P$ differ. Let $B_0, B_1, \ldots, B_\ell$ with $B_0 = A$ and $B_\ell = B$ be matrices such that each B_k belongs to the set of matrices to which

B_{k-1} partitioned at some point of time. This means that $B_k = f(B_{k-1}, i, j)$ for some i and j, so we can transform B_{k-1} into B_k in $O(n)$ time. Therefore, instead of keeping the matrices in P, we keep there lists of pairs of the form (i, j) required for transforming A into B. If ℓ is always guaranteed to be $O(1)$, then we do this transformation in $O(n)$ time as well.

However, we need to maintain the permutations π_i when transforming the matrices. Precomputing the necessary variables for computing the upper bounds requires $O(n^2)$ time, so if we can update the permutations faster than that, it does not become a bottleneck in the algorithm. The most efficient data structure for this seems to be a linked list containing the values $\pi_i(1), \pi_i(2), \ldots, \pi_i(n)$ for each i. Whenever we transform B_{k-1} into $B_k = f(B_{k-1}, i, j)$, we move the node containing $\pi_s^{-1}(j)$ to the end of the list if $s \neq i$. If $s = i$, we move the node containing $\pi_s^{-1}(j)$ to the beginning of the list. This requires $O(n)$ time and these operations are easy to undo in the same time complexity when we are done. Even in the worst case, where $\ell = n$, the total time required for maintaining the permutations whilst transforming A into B is $O(n^2)$. Transforming A into C in the while loop is done the same way. Overall, this gives a $O(n^3 f(n))$ time complexity per trial, where $f(n)$ is the worst possible number of iterations of the do while loop for the instance.

3.6 Dealing with Large Numbers

The order of magnitude of permanents and upper bounds can increase (or decrease) quite fast as n increases, and therefore we need to take this into consideration when implementing the algorithms. A natural solution for this is using logarithms as they allow us to present exponentially large (and small) numbers with a decent precision and have been researched extensively. However, this introduces new problems: We need a way for summing two variables when we only know their logarithms, and we need to be able to draw a weighted sample when only the logarithm of each probability is known. One possibility for solving these problems is using two well-known tricks called the log–sum–exp trick (Blanchard et al., 2019) and the Gumbel–Max trick (Gumbel, 1954; Maddison et al., 2014).

Assume that we only know logarithms $\ln x$ and $\ln y$ for some values x and y, and we wish to compute the logarithm of their sum $\ln(x + y)$. If x and y are large numbers, naively computing $\ln(\exp(\ln x) + \exp(\ln y))$ may overflow. However, note that

$$\exp(\ln x) + \exp(\ln y) = \exp(z)(\exp(\ln(x) - z) + \exp(\ln(y) - z))$$

for any z. Without loss of generality, assume that $x \geq y$. By choosing $z = \ln x$, we get that

$$\ln(x + y) = \ln(x) + \ln(1 + \exp(\ln(y) - \ln(x))).$$

Because $x \geq y$, the difference $\ln(y) - \ln(x)$ has to be nonpositive, and thus the natural exponent of it is between zero and one and cannot overflow. This log–sum–exp trick generalizes for any number of logarithms.

The other problem caused by logarithms is that we need a way for doing weighted sampling while only having the logarithms of the weights. In other words, we have logarithms of weights $w_1, w_2, \ldots, w_n$, and we want to select an item number k with a probability $w_k / \left(\sum_{i=1}^{n} w_i\right)$. This problem is solvable with the Gumbel-Max trick (Gumbel, 1954; Maddison et al., 2014), in which we take n samples $u_1, u_2, \ldots, u_n$ from the Uniform$[0, 1]$ distribution, and let K be the value that maximizes $-\ln(-\ln(u_K)) + \ln(w_K)$. It can be shown that then $\Pr(K = k)$ is equal to $w_k / \left(\sum_{i=1}^{n} w_i\right)$, which provides us a straightforward way for weighted sampling without having to worry about large numbers.

3.7 Tassa's Algorithm

Sometimes a matrix contains entries that do not belong to any permutation of positive weight. Replacing any such entries by zeros does not change the permanent of the matrix, and doing so has been suggested in (Chen and Liu, 2007; Sullivan and Beichl, 2014). Before explaining an efficient way for doing this, we give several definitions to simplify further discussion:

Definition 6. A positive entry a_{ij} in a nonnegative $n \times n$ matrix $A = (a_{ij})$ is said to be *supported* if there exists a permutation $\sigma \in S_n$ that has a positive weight in A and $\sigma(j) = i$.

Definition 7. A nonnegative $n \times n$ matrix $A = (a_{ij})$ is said to have *total support* if every positive entry in A is supported.

The most efficient known way to remove entries that are not supported is to transform the matrix into its corresponding bipartite graph and run Tassa's algorithm (Tassa, 2012), which deletes all edges that are not a part of any perfect matching. Tassa's algorithm works in linear time with respect to the number of vertices and edges of the graph assuming that we know one perfect matching beforehand. There exists a randomized algorithm for

finding a perfect bipartite matching in $\tilde{O}\left(m + n^{1.5}\right)$ time (van den Brand et al., 2020), but often the simpler $O\left(m\sqrt{n}\right)$-time Hopcroft–Karp algorithm (Hopcroft and Karp, 1973) is fast enough.

Let $G = (V, W, E)$ be a bipartite graph with independent vertex sets $V = \{v_1, v_2, \ldots, v_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$ and a set of edges E connecting them. Without loss of generality, assume that our perfect matching is $\{\{v_1, w_1\}, \{v_2, w_2\}, \ldots, \{v_n, w_n\}\}$, which is easy to achieve for any perfect matching by simply permuting the indices of W. Algorithm 5 describes Tassa's algorithm for G under this assumption. The pseudocode requires finding the strongly connected components of a directed graph, and this can be done in linear time with, for example, Tarjan's algorithm (Tarjan, 1972) or the Kosaraju–Sharir algorithm (Sharir, 1981).

Algorithm 5: Tassa's algorithm

Input : A bipartite graph $G = (V, W, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$

Output: A bipartite graph G' containing only the edges that are in some perfect matching in G

Let $G' = (V', W', E')$ with $V' = V$, $W' = W$ and $E' = \emptyset$;
Let $H = (U, F)$ be a directed graph with $U = \{u_1, u_2, \ldots, u_n\}$ and $F = \emptyset$;
for $\{v_i, w_j\} \in E$ **do**
    if $i \neq j$ **then**
        Add a directed edge (u_i, u_j) to H;
Find strongly connected components of H;
for $\{v_i, w_j\} \in E$ **do**
    if $i = j$ or u_i and u_j are in the same strongly connected component **then**
        Add an undirected edge $\{v_i, w_j\}$ to G';
return G';

Consider for example an $n \times n$ matrix $A = (a_{ij})$ in which $a_{ij} = 1$ if $j \leq i$ and $a_{ij} = 0$ otherwise. Such matrices are suspected to have one of the worst possible fit ratios on some upper bounds (Soules, 2003), as their permanent is equal to one but the number of ones grows quadratically in n. However, by applying Tassa's algorithm, all ones that are not on the main diagonal are removed so most upper bounds become equal to one.

3.8 Doubly Stochastic Matrices

Matrices whose column and row sums are all equal to one are called *doubly stochastic*. The permanent of a doubly stochastic $n \times n$ matrix is at least $n!/n^n \approx e^n$, as conjectured by van der Waerden and later proven in (Egorychev, 1981; Falikman, 1981). This lower bound can be used for proving time complexities of rejection sampling algorithms (see for example (Huber, 2006; Huber and Law, 2008)). The properties of doubly stochastic matrices have proven to be useful in importance sampling as well (Beichl and Sullivan, 1999; Sullivan and Beichl, 2014).

Sinkhorn balancing, the method of making a matrix nearly doubly stochastic by alternating between dividing each row and each column by their sums, has been studied in for example (Brualdi et al., 1966; Sinkhorn, 1964; Sinkhorn and Knopp, 1967). The most important result for our purposes is that a sufficient condition for that the Sinkhorn balancing converges linearly is that the matrix has total support (Soules, 1991). Tassa's algorithm is a straightforward way to ensure that the matrix has total support without changing the value of the permanent (Sullivan and Beichl, 2014). The fit ratio can be super-exponential in n for some classes of matrices, so making the matrix nearly doubly stochastic guarantees an exponential fit ratio.

The ellipsoid method is an alternative method for making the matrix nearly doubly stochastic, and it is faster than Sinkhorn balancing. However, Sullivan and Beichl (2014) suggest that repeating Sinkhorn balancing for n^2 iterations should be enough for obtaining a nearly doubly stochastic matrix, so we will use that method in our tests due to its simplicity.

After making the matrix nearly doubly stochastic, Huber and Law (2008) also divide each row vector by its largest entry l. This is done because multiplying that row vector by $1/l$ increases the permanent by a factor of $1/l$ and the Huber–Law bound by a factor of at most $1/l$, and therefore it can only tighten the fit ratio. For the Soules bound, this last step has no effect.

As an example, consider a class of $n \times n$ $(0,1)$-matrices $A_n = (a_{ij})$ where $a_{ij} = 1$ if and only if $i + j \leq n + 2$, which were analyzed by Soules (2003). The permanent of any A_n is equal to 2^{n-1} (Soules, 2003), but both the Huber–Law bound and the Soules bound for them are super-exponential in n. Any such matrix has total support, so Tassa's algorithm does not simplify it, but this means that the Sinkhorn balancing for A_n converges to some

matrix $B_n = (b_{ij})$ such that

$$(b_{ij}) = \begin{cases} 2^{i+j-n-3}, & i > 1, j > 1, \\ 2^{j-n-1}, & i = 1, j > 1, \\ 2^{i-n-1}, & i > 1, j = 1, \\ 2^{1-n}, & i = 1, j = 1. \end{cases}$$

This matrix can be obtained by dividing each row i by $2^{(n+3)/2-\max\{2,i\}}$ and each column j by $2^{(n+3)/2-\max\{2,j\}}$. A simple induction on n shows that the permanent of B_n is equal to $1/2^{n-1}$, and Huber–Law upper bound states that the permanent is at most 1.

The largest entry of each row in B_n is $1/2$, so we divide each row vector by it to get a matrix C_n. The permanent of C_n is equal to 2 and its Huber–Law upper bound is $h(2)^n < 1.623^n$. Hence, by making the matrix nearly doubly stochastic and multiplying each row by 2, we get the ratio of the upper bound and the permanent from super-exponential to $O(2^n)$ and finally to $O(1.623^n)$. Even though the improvements are not always this good, this example shows how significant doubly stochastic matrices and dividing the rows vectors can be in speeding up the rejection sampling.

However, this can be harmful as well sometimes. Consider a 3×3 matrix

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

Its fit ratio with respect to the Huber–Law bound is approximately 1.44. On the other hand, it can be shown that the matrix after making it doubly stochastic and dividing the row vectors is

$$\begin{bmatrix} 1 & 1 & 1/\phi \\ 0 & 1 & 1/\phi \\ 1 & 0 & 1/\phi \end{bmatrix},$$

where ϕ is the golden ratio. The fit ratio of this matrix is approximately 1.66, which is greater than the fit ratio of the original matrix.

4 Deep Rejection Sampling

In this chapter, we introduce our deep rejection sampling method, whose basic idea is to try improving the permanental upper bound by replacing it with a linear combination of upper bounds for subproblems. We start by discussing the Gamma Bernoulli approximation scheme used for getting a desired accuracy and certain preprocessing algorithms we use in some of the tests. After that, we present the deep sampling table, which is the main component of our method. We show how the table is computed, how its values are interpreted, and how it can be used to sample a set of rows in linear time to jump-start the permutation sampling process.

4.1 Deep Sampling Table

The main new idea of the present work is the deep sampling table (DST), which speeds up the rejection sampling in two ways: it gives a better initial upper bound and enables faster sampling for the first columns. Let A be an $n \times n$ matrix, U a permanental upper bound, d a fixed positive integer, and $J \subseteq [n]$ with $|J| = d$. Denote a submatrix of A that contains only its rows $R \subseteq [n]$ and columns $C \subseteq [n]$ by A_{RC}. For a set $S \subseteq [n]$, denote the set $[n] \setminus S$ by $\bar{S}$. Then,

$$\begin{aligned}
\operatorname{per} A &= \sum_{\substack{I \subseteq [n] \\ |I| = d}} \operatorname{per} A_{IJ} \operatorname{per} A_{\bar{I}\bar{J}} \\
&\leq \sum_{\substack{I \subseteq [n] \\ |I| = d}} \operatorname{per} A_{IJ}\ U\left(A_{\bar{I}\bar{J}}\right) \\
&=: U_d(A),
\end{aligned}$$

which we call the *depth-d variant of the bound U*. The bound is a linear combination of upper bounds for a set of subproblems. Denote the ith row of $A_{[n]\bar{J}}$ by a'_i and assume that the upper bound for $A_{\bar{I}\bar{J}}$ is of the form $\prod_{i \in \bar{I}} \gamma(a'_i)$ for some function γ. Next, define the entries $D_{S,i}$ of the deep sampling table recursively as follows for all nonempty sets $S \subseteq J$

and integers $i \in [n]$:

$$\begin{aligned}
D_{\{j\},1} &= a_{1,j}, && \text{if } S = \{j\}, i = 1,\\
D_{S,1} &= 0, && \text{if } S \text{ is not of the form } \{j\}, i = 1,\\
D_{\{j\},i} &= \gamma(a'_i) \cdot D_{S,i-1} + a_{ij} \prod_{k=1}^{i-1} \gamma(a'_k), && \text{if } S = \{j\}, i > 1,\\
D_{S,i} &= \gamma(a'_i) \cdot D_{S,i-1} + \sum_{j \in S} a_{ij} \cdot D_{S\setminus\{j\},i-1}, && \text{if } S \text{ is not of the form } \{j\}, i > 1.
\end{aligned}$$

This is somewhat similar to what Björklund et al. (2010) compute for getting the permanent of a rectangular matrix. It is easy to see that the table can be computed in $O(2^d nd)$ operations. The following theorem proves some of the properties of the table.

Theorem 1. For any nonempty $S \subseteq J$ and $i \in [n]$, it holds that

$$D_{S,i} = \sum_{\substack{I \subseteq [i] \\ |I| = |S|}} \left(\operatorname{per} A_{IS} \prod_{k \in [i]\setminus I} \gamma(a'_k) \right).$$

Especially,

$$D_{J,n} = \sum_{\substack{I \subseteq [n] \\ |I| = d}} \operatorname{per} A_{IJ} \; U\left(A_{\bar{I}\bar{J}}\right) = U_d(A).$$

Proof. Proof by induction on i. Consider first the case that $i = 1$. If S is not of the form $\{j\}$, then $D_{S,1} = 0$ as there can be no sets $I \subseteq [1]$ of size S because $|S| > 1$. For a set $S = \{j\}$ with $j \in J$, the value of $D_{S,i}$ is clearly $a_{1,j} \cdot 1 = a_{1,j}$ since I has to be $\{1\}$.

Assume now that $i > 1$ and $S = \{j\}$ for some $j \in J$. Then, each I is either equal to $\{i\}$ or equal to $\{k\}$ for some $k < i$. In the former case,

$$\operatorname{per} A_{IS} \prod_{k \in [i]\setminus I} \gamma(a'_k) = a_{ij} \cdot \prod_{k=1}^{i-1} \gamma(a'_k),$$

and in the latter case

$$\operatorname{per} A_{IS} \prod_{k \in [i]\setminus I} \gamma(a'_k) = \gamma(a'_i) \operatorname{per} A_{IS} \prod_{k \in [i-1]\setminus I} \gamma(a'_k).$$

Therefore,

$$\begin{aligned}
\sum_{\substack{I \subseteq [i] \\ |I| = |S|}} \left(\operatorname{per} A_{IS} \prod_{k \in [i]\setminus I} \gamma(a'_k) \right) &= a_{ij} \cdot \prod_{k=1}^{i-1} \gamma(a'_k) + \gamma(a'_i) \cdot \sum_{\substack{I \subseteq [i-1] \\ |I| = |S|}} \left(\operatorname{per} A_{IS} \prod_{k \in [i-1]\setminus I} \gamma(a'_k) \right)\\
&= a_{ij} \cdot \prod_{k=1}^{i-1} \gamma(a'_k) + \gamma(a'_i) \cdot D_{S,i-1}\\
&= D_{S,i}.
\end{aligned}$$

Finally, consider the case in which $i > 1$ and S is not of the form $\{J\}$. Again, either $i \in I$ or $i \notin I$. If $i \in I$, then, by self-reducibility,

$$\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) = \sum_{j\in S} a_{ij} \operatorname{per} A_{I\setminus\{i\},S\setminus\{j\}} \prod_{k\in[i-1]\setminus(I\setminus\{i\})} \gamma(a'_k).$$

The case $i \notin I$ goes similarly as to when $S = \{j\}$, i.e.,

$$\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) = \gamma(a'_i) \operatorname{per} A_{IS} \prod_{k\in[i-1]\setminus I} \gamma(a'_k).$$

Consider now the sums

$$\sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\in I}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right)$$

and

$$\sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\notin I}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right).$$

The former one becomes

$$\begin{aligned}
\sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\in I}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right) &= \sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\in I}} \sum_{j\in S} a_{ij} \operatorname{per} A_{I\setminus\{i\},S\setminus\{j\}} \prod_{k\in[i-1]\setminus(I\setminus\{i\})} \gamma(a'_k) \\
&= \sum_{j\in S} a_{ij} \sum_{\substack{I\subseteq[i-1]\\|I|=|S\setminus\{j\}|}} \operatorname{per} A_{I,S\setminus\{j\}} \prod_{k\in[i-1]\setminus I} \gamma(a'_k) \\
&= \sum_{j\in S} a_{ij} \cdot D_{S\setminus\{j\},i-1},
\end{aligned}$$

and the latter one is simply

$$\begin{aligned}
\sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\notin I}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right) &= \gamma(a'_i) \cdot \sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\notin I}} \operatorname{per} A_{IS} \prod_{k\in[i-1]\setminus I} \gamma(a'_k) \\
&= \gamma(a'_i) \cdot D_{S,i-1}.
\end{aligned}$$

Hence,

$$\begin{aligned}
\sum_{\substack{I\subseteq[i]\\|I|=|S|}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right) &= \sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\in I}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right) + \sum_{\substack{I\subseteq[i]\\|I|=|S|\\i\notin I}} \left(\operatorname{per} A_{IS} \prod_{k\in[i]\setminus I} \gamma(a'_k) \right) \\
&= \sum_{j\in S} a_{ij} \cdot D_{S\setminus\{j\},i-1} + \gamma(a'_i) \cdot D_{S,i-1} \\
&= D_{S,i}.
\end{aligned}$$

This proves that the recurrence formula computes the values of $D_{S,i}$ as is stated in the theorem. Most notably,

$$D_{J,n} = \sum_{\substack{I \subseteq [n] \\ |I|=d}} \operatorname{per} A_{IJ}\ U\left(A_{\bar{I}\bar{J}}\right) = U_d(A).$$

□

4.2 Stochastic Bactracking

We use standard stochastic backtracking to build a set of rows I based on the values in the deep sampling table. This is presented as pseudocode in Algorithm 6. The algorithm is presented in a bit more general way than what is actually necessary: usually we are only interested in the case in which $S = J$ and $i_0 = n$. The more general form is given to simplify proving the theorem related to it. In Theorem 2, we prove that the algorithm can be used to sample I with a probability relative to how much it contributes to the upper bound of A. If we set $\sigma(i) = j$ whenever we replace S by $S \setminus \{j\}$, we can use the algorithm to sample a value $\sigma^{-1}(j)$ for every $j \in J$.

Theorem 2. For a set of rows $I \subseteq [i_0]$, with a slight abuse of earlier notation, denote $[i_0] \setminus I$ by $\bar{I}$. Assume that $|S| \leq i_0$ and the sum

$$\sum_{\substack{R \subseteq [i_0] \\ |R|=|S|}} \operatorname{per} A_{RS}\ U\left(A_{\bar{R}\bar{J}}\right)$$

is positive. Then, Algorithm 6 samples $I \subseteq [i_0]$ with probability

$$\frac{\operatorname{per} A_{IS}\ U\left(A_{\bar{I}\bar{J}}\right)}{\sum\limits_{\substack{R \subseteq [i_0] \\ |R|=|S|}} \operatorname{per} A_{RS}\ U\left(A_{\bar{R}\bar{J}}\right)}.$$

Most notably, when $i_0 = n$ and $S = J$, the algorithm samples I with probability

$$\frac{\operatorname{per} A_{IJ}\ U\left(A_{\bar{I}\bar{J}}\right)}{\sum\limits_{\substack{R \subseteq [n] \\ |R|=|J|}} \operatorname{per} A_{RJ}\ U\left(A_{\bar{R}\bar{J}}\right)}.$$

Algorithm 6: Stochastic backtracking in DST

Input : A nonnegative $n \times n$ matrix $A = (a_{ij})$, a set of columns J, their deep sampling table D, a nonempty set of columns $S \subseteq J$, and a positive integer $i_0 \leq n$

Output: A set of rows I

$I \leftarrow \emptyset$;
for $i = i_0, \ldots, 1$ **do**
  if $S \neq \emptyset$ **then**
    if $i \neq 1$ **then**
      $X \leftarrow \text{Bernoulli}(D_{S,i-1} \cdot \gamma(a'_i)/D_{S,i})$;
      if $X = 0$ **then**
        $I \leftarrow I \cup \{i\}$;
        if S *is not of the form* $\{j\}$ **then**
          Let $p(j)$ be a probability mass function for every $j \in S$;
          for $j \in S$ **do**
            $p(j) \leftarrow a_{ij} \cdot D_{S\setminus\{j\},i-1}$;
          Select j randomly by using $p(j)$ as the probability mass function;
          $S \leftarrow S \setminus \{j\}$;
        else
          $S \leftarrow S \setminus \{j\}$, where j is the only element of S;
    else
      $I \leftarrow I \cup \{1\}$;
      $S \leftarrow S \setminus \{j\}$, where j is the only element of S;
return I;

Proof. Proof by induction on i_0. Consider first the case $i_0 = 1$. As S is nonempty and $|S| \leq i_0$, $|S|$ has to contain exactly one element j. By the pseudocode of the algorithm, the set I returned by it has to be $\{1\}$. It is also the only set for which $I \subseteq [i_0]$ and $|I| = |S|$, so the ratio

$$\frac{\operatorname{per} A_{IS}\ U\left(A_{\bar{I}\bar{J}}\right)}{\sum\limits_{\substack{R\subseteq[i_0]\\|R|=|S|}} \operatorname{per} A_{RS}\ U\left(A_{\bar{R}\bar{J}}\right)}$$

is indeed equal to 1.

Assume now that $i_0 > 1$. Two factors need to be considered: is $i_0 \in I$ or not and whether S is of the form $\{j\}$ for some $j \in J$ or not? If $i_0 \in I$ and S is of the form $\{j\}$, the

probability of sampling I is

$$1 - \frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}}.$$

By the recurrence formulas, this is equal to

$$1 - \frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0-1} \cdot \gamma(a'_{i_0}) + a_{i_0,j} \prod_{k=1}^{i_0-1} \gamma(a'_k)} = \frac{a_{i_0,j} \prod_{k=1}^{i_0-1} \gamma(a'_k)}{D_{S,i_0-1} \cdot \gamma(a'_{i_0}) + a_{i_0,j} \prod_{k=1}^{i_0-1} \gamma(a'_k)}.$$

It is easy to see

$$a_{ij} \prod_{\substack{k \in [i_0] \\ k \neq i}} \gamma(a'_k) = \operatorname{per} A_{\{i\},S} \cdot U\left(A_{\bar{\{i\}},\bar{J}}\right)$$

and, by induction, that

$$D_{S,i_0-1} \cdot \gamma(a'_{i_0}) + a_{i_0,j} \prod_{k=1}^{i_0-1} \gamma(a'_k) = \sum_{i=1}^{i_0} a_{ij} \prod_{\substack{k \in [i_0] \\ k \neq i}} \gamma(a'_k).$$

Hence, the statement of the theorem holds for this case.

If $i_0 \in I$ and S is not of the form $\{j\}$, then, by the induction assumption, the probability of sampling I is the probability that $X = 0$ on the first iteration of the for-loop and the algorithm samples $I \setminus \{i_0\}$ if the initial parameter i_0 is decreased by one. This is equal to

$$\left(1 - \frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}}\right) \cdot \sum_{j \in S} \frac{a_{i_0,j} \cdot D_{S\setminus\{j\},i_0-1}}{\sum_{k \in S} a_{i_0,k} \cdot D_{S\setminus\{k\},i_0-1}} \cdot \frac{\operatorname{per} A_{(I\setminus\{i_0\})(S\setminus\{j\})} \; U\left(A_{\bar{I}\bar{J}}\right)}{\sum\limits_{\substack{R \subseteq [i_0-1] \\ |R| = |S\setminus\{j\}|}} \operatorname{per} A_{R(S\setminus\{j\})} \; U\left(A_{\bar{R}\bar{J}}\right)}.$$

By applying Theorem 1, this simplifies to

$$\begin{aligned}
&\left(1 - \frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}}\right) \cdot \sum_{j \in S} \frac{a_{i_0,j} \cdot D_{S\setminus\{j\},i_0-1}}{\sum_{k \in S} a_{i_0,k} \cdot D_{S\setminus\{k\},i_0-1}} \cdot \frac{\operatorname{per} A_{(I\setminus\{i_0\})(S\setminus\{j\})} \; U\left(A_{\bar{I}\bar{J}}\right)}{D_{S\setminus\{j\},i_0-1}} \\
=&\left(1 - \frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}}\right) \cdot \frac{\sum_{j \in S} a_{i_0,j} \cdot \operatorname{per} A_{(I\setminus\{i_0\})(S\setminus\{j\})} \; U\left(A_{\bar{I}\bar{J}}\right)}{\sum_{k \in S} a_{i_0,k} \cdot D_{S\setminus\{k\},i_0-1}}.
\end{aligned}$$

By the self-reducibility of the permanent, this is equal to

$$\left(1 - \frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}}\right) \cdot \frac{\operatorname{per} A_{IS} \; U\left(A_{\bar{I}\bar{J}}\right)}{\sum_{k \in S} a_{i_0,k} \cdot D_{S\setminus\{k\},i_0-1}}.$$

Next, we apply the recurrence equations to get

$$\begin{aligned}
&\left(1 - \frac{D_{S,i_0} - \sum_{j \in S} a_{i_0,j} \cdot D_{S\setminus\{j\},i_0-1}}{D_{S,i_0}}\right) \cdot \frac{\operatorname{per} A_{IS} \; U\left(A_{\bar{I}\bar{J}}\right)}{\sum_{k \in S} a_{i_0,k} \cdot D_{S\setminus\{k\},i_0-1}} \\
=&\frac{\sum_{j \in S} a_{i_0,j} \cdot D_{S\setminus\{j\},i_0-1}}{D_{S,i_0}} \cdot \frac{\operatorname{per} A_{IS} \; U\left(A_{\bar{I}\bar{J}}\right)}{\sum_{k \in S} a_{i_0,k} \cdot D_{S\setminus\{k\},i_0-1}} \\
=&\frac{\operatorname{per} A_{IS} \; U\left(A_{\bar{I}\bar{J}}\right)}{D_{S,i_0}}
\end{aligned}$$

Applying Theorem 1 on D_{S,i_0} proves the statement for this case.

Finally, if $i_0 \notin I$, the probability of sampling I is equal to the probability that X is 1 on the first iteration of the for-loop and the algorithm samples I when the initial parameter i_0 is decreased by one. By the induction assumption, this is equal to

$$\frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}} \cdot \frac{\operatorname{per} A_{IS}\ U\left(A_{(\bar{I}\setminus\{i_0\})\bar{J}}\right)}{\sum\limits_{\substack{R\subseteq[i_0-1]\\|R|=|S|}} \operatorname{per} A_{RS}\ U\left(A_{(\bar{R}\setminus\{i_0\})\bar{J}}\right)}.$$

By applying Theorem 1, this becomes

$$\frac{D_{S,i_0-1} \cdot \gamma(a'_{i_0})}{D_{S,i_0}} \cdot \frac{\operatorname{per} A_{IS}\ U\left(A_{(\bar{I}\setminus\{i_0\})\bar{J}}\right)}{D_{S,i_0-1}} = \frac{\operatorname{per} A_{IS}\ U\left(A_{\bar{I}\bar{J}}\right)}{D_{S,i_0}}.$$

By applying it once more on D_{S,i_0}, we can see that the statement of the theorem holds here. □

After sampling I using Algorithm 6, we only need to run the regular rejection sampling on $A_{\bar{I}\bar{J}}$ and accept or reject the sample depending on its result. Algorithm 7 combines all discussed algorithms together

Algorithm 7: Deep Rejection Sampler

Input : A nonnegative $n \times n$ matrix A, precision (ϵ, δ), a depth d, a permanental upper bound U, a set of columns J

Output: An (ϵ, δ)-approximation of the permanent of A

Compute the value of k used by GBAS;

Compute the deep sampling table $D_{S,j}$;

$s \leftarrow 0$;

$r \leftarrow 0$;

while $s \neq k$ **do**

- Run Algorithm 6 to sample a set of rows I;
- Run a rejection sampler based on U on $A_{\bar{I}\bar{J}}$;
- Let X be 1 if the sample was accepted 0 otherwise;
- $A \leftarrow \text{Exp}(1)$;
- $s \leftarrow s + X$;
- $r \leftarrow r + A$;

$\hat{p} \leftarrow (k-1)/r$;

return $\hat{p} \cdot U_d(A)$;

Now, all that remains to be shown is that Algorithm 7 returns an (ϵ, δ)-approximation.

Theorem 3. Algorithm 7 returns an (ϵ, δ)-approximation of the permanent of A with expected running time $O(2^d dn + U_d(A)/\operatorname{per} A \cdot f(n)\epsilon^{-2} \log \delta^{-1})$ time, where $f(n)$ is the time needed for one trial of rejection sampling with bound U.

Proof. We need to show that the variable X follows a Bernoulli distribution with parameter $\operatorname{per} A/U_d(A)$. The probability of sampling I is

$$\frac{\operatorname{per} A_{IJ}\ U\left(A_{\bar{I}\bar{J}}\right)}{U_d(A)}$$

and the probability of getting an accepted sample from $A_{\bar{I}\bar{J}}$ is $\operatorname{per} A_{\bar{I}\bar{J}}/U(A_{\bar{I}\bar{J}})$, so the probability that they both happen is

$$\frac{\operatorname{per} A_{IJ} \operatorname{per} A_{\bar{I}\bar{J}}}{U_d(A)}.$$

By the law of total probability, the probability that X is 1 is $\operatorname{per} A/U_d(A)$. Therefore, GBAS is guaranteed to give an (ϵ, δ)-approximation of $\operatorname{per} A/U_d(A)$, and multiplying this by the precomputed value of $U_d(A)$ gives an (ϵ, δ)-approximation of $\operatorname{per} A$.

All instances have to do $O(2^d dn)$ precomputing for the deep sampling table. Factors ϵ^{-2} and $\log \delta^{-1}$ come from the upper bound for the number of accepted samples required in GBAS,

$$k \geq 2\epsilon^{-2}(1 - (4/3)\epsilon)^{-1}\ln(2\delta^{-1}).$$

The expected number of trials required for an accepted sample is $U_d(A)/\operatorname{per} A$, and each of those requires $f(n)$ time. Combining all of these gives us the expected running time $O(2^d dn + U_d/\operatorname{per} A \cdot f(n)\epsilon^{-2} \log \delta^{-1})$. □

One may wish to apply certain preprocessing on A, that is, run Tassa's algorithm on A so that it has total support, apply Sinkhorn balancing n^2 times on A, and divide each row vector by its largest entry. In such a case, we need to divide the estimate returned by the algorithm by the product of values by which the rows and columns were scaled. This is due to the fact that multiplying a row or a column of a matrix by some value multiplies its permanent by the same value, and all changes to the permanent of A in preprocessing come from multiplying its rows and columns, so doing this in a reverse direction negates these changes. The preprocessing has time complexity $O(n^4)$.

5 Running Time for Random Matrices

In this chapter, we will give a high probability upper bound for the expected time complexity of getting an (ϵ, δ)-approximation for the permanent of a random matrix whose entries are Bernoulli distributed with parameter p. We prove the bound by giving a high probability upper bound for the ratio $U_d^{\mathrm{HL}}(A)/\operatorname{per} A$ and then combining it with Theorem 3. Our bound is equal to $O(n^{3/2+c/p})$, where c is any number greater than $1/2$. This improves the earlier bound $O(n^{1-1/p}\omega)$ of Fürer and Kasiviswanathan (2004) when $p < 1/5$, where ω is any function that tends to infinity. Our bound loses to the $O(ng(n))$-time bound of Godsil–Gutman type estimators (Frieze and Jerrum, 1995) that holds for almost all matrices, where $g(n)$ is the time required for computing the determinant. However, in the same spirit as Frieze and Jerrum argued, our bound is more trustworthy.

We structure the proof in three parts. First, we introduce the inequalities used in the proofs. Then, we give an upper bound for the expected value of the depth-d variant of the Huber–Law bound U_d^{HL} and the expected value for the permanent. Finally, we combine these results to give a high probability upper bound for the expected time complexity.

In all following proofs, let $A = (a_{ij})$ be a random $n \times n$ $(0,1)$-matrix whose entries are independent and follow Bernoulli distribution with parameter $p = p(n)$ such that $p^2 n$ tends to infinity as n tends to infinity. We use M to denote the number of ones in A. Frieze and Jerrum (1995) observed that the number of ones in A is more significant to the magnitude of permanent than their locations in A, so we try working with M instead of p.

5.1 Inequalities

In this section, we give a list of several well-known inequalities that are used in the later proofs. Unless otherwise specified, the lemmas in this section are from a textbook by Mitzenmacher and Upfal (2017). We start with the exponential inequality, which is sometimes useful in simplifying inequalities.

Lemma 1 (*Exponential inequality*)**.** For any x,

$$1 + x \leq e^x.$$

The next two inequalities are concentration inequalities for a random variable.

Lemma 2 (*Markov's inequality*)**.** For a nonnegative random variable X and $a > 0$,

$$\Pr(X \geq a) \leq \frac{\mathbf{E}[X]}{a}.$$

Lemma 3 (*Chebyshev's inequality*)**.** For a random variable X and $a > 0$,

$$\Pr(|X - \mathbf{E}[X]| \geq a) \leq \frac{\mathbf{Var}[X]}{a^2}.$$

For a concave function f, Jensen's inequality gives an upper bound for its expected value.

Definition 8. A function $f(x)$ is *concave* if for any x, y and $\alpha \in [0, 1]$,

$$f(\alpha x + (1 - \alpha)y) \geq \alpha f(x) + (1 - \alpha)f(y).$$

Lemma 4 (*Jensen's inequality*)**.** For a concave function f and a random variable X,

$$\mathbf{E}[f(x)] \leq f(\mathbf{E}[X]).$$

Stirling's formula can be used to get rid of factorials.

Lemma 5 (*Stirling's formula*)**.** For $n > 0$,

$$\sqrt{2\pi n}\frac{n^n}{e^n} \leq n! \leq 2\sqrt{2\pi n}\frac{n^n}{e^n}.$$

Finally, we need the multiplicative form of *Chernoff bound*.

Lemma 6 (Angluin and Valiant, 1979)**.** For any $a > 0$ and independent random variables $X_1, X_2, \ldots, X_\ell$ following Bernoulli distribution with parameter p,

$$\Pr\left(\sum_{k=1}^{\ell} X_k \leq (1 - a)\ell p\right) \leq \exp\left\{-a^2 \ell p/2\right\}.$$

5.2 Expected Time Complexity

In this section, we give a high probability upper bound for the expected time complexity of the algorithm. We start by giving a probabilistic lower bound for M using a Chernoff bound.

Lemma 7. The probability that M is below $m_0 \coloneqq n^2p - a$ is at most δ if we choose $a \coloneqq n\sqrt{2p\ln(\delta^{-1})}$.

Proof. We note that M is a sum of n^2 random variables following Bernoulli distribution with parameter p. Therefore, by Lemma 6,

$$\Pr\left(M \leq (1 - an^{-2}p^{-1})n^2p\right) \leq \exp\left\{-a^2n^{-2}p^{-1}/2\right\} = \delta.$$

□

In the rest of the proofs, unless otherwise specified, we are assuming $M = m$ for some $m > m_0$. Next, given the previous assumption, we will show that n^3m^{-2} tends to zero as n tends to infinity.

Lemma 8. The product n^3m^{-2} tends to zero as n tends to infinity.

Proof. By the assumption, $m > m_0 = n^2p - a$. Therefore,

$$\begin{aligned} n^3m^{-2} &= (n^{-3/2}m)^{-2} \\ &\leq (n^{-3/2}m_0)^{-2} \\ &= (n^{1/2}p - n^{-3/2}a)^{-2}. \end{aligned}$$

We note that $n^{1/2}p$ tends to infinity by the assumption that p^2n tends to infinity and that $n^{-3/2}a$ tends to zero because $a \leq n\sqrt{2\ln(\delta^{-1})}$. Hence, the square of the inverse of the subtraction has to tend to zero as n tends to infinity. □

To be able to analyze the upper bound for the expected value of the Huber–Law bound, we need two more lemmas to allow us to move from a fixed number of ones back to a binomially distributed number of ones.

Lemma 9. Let X be a discrete nonnegative random variable and let $\tilde{X}$ be its median. Then, $\tilde{X} \leq 2\mathbf{E}[X]$.

Proof. By definition, $\tilde{X}$ is a value for which $\Pr(X \leq \tilde{X}) \geq 1/2$ and $\Pr(X \geq \tilde{X}) \geq 1/2$. Then, by Markov's inequality,

$$\begin{aligned}\tilde{X} &\leq 2\tilde{X} \Pr(X \geq \tilde{X}) \\ &\leq 2\tilde{X} \cdot \frac{\mathbf{E}[X]}{\tilde{X}} \\ &= 2\mathbf{E}[X]\end{aligned}$$

□

Lemma 10. Let $p = mn^{-2}$. Then,

$$\mathbf{E}\left[U_d^{\mathrm{HL}}(A) \mid M = m\right] \leq 2\mathbf{E}\left[U_d^{\mathrm{HL}}(A)\right].$$

Proof. The median of a binomial distribution with parameters n^2 and p is $n^2 p = m$, so the median of M is m. By applying Lemma 9, we get that

$$\mathbf{E}\left[U_d^{\mathrm{HL}}(A) \mid M = m\right] \leq 2\mathbf{E}\left[U_d^{\mathrm{HL}}(A)\right].$$

□

We are now able to give an upper bound for the expected value of the Huber–Law bound for A.

Lemma 11. Let $p = mn^{-2}$. Then, for the depth-d variant of the Huber–Law bound U_d^{HL}, it holds that

$$\mathbf{E}\left[U_d^{\mathrm{HL}}(A) \mid M = m\right] \leq 2\binom{n}{d} d! \mathrm{e}^{d-n} p^d h\left(p(n-d)\right)^{n-d}$$

Proof. From Lemma 10 we get immediately that

$$\mathbf{E}\left[U_d^{\mathrm{HL}}(A) \mid M = m\right] \leq 2\mathbf{E}\left[U_d^{\mathrm{HL}}(A)\right].$$

By the definition of U_d^{HL} and linearity of expectation,

$$\begin{aligned}2\mathbf{E}\left[U_d^{\mathrm{HL}}(A)\right] &= 2\mathbf{E}\left[\sum_{\substack{I \subseteq [n] \\ |I| = k}} \operatorname{per} A_{IJ} \cdot U^{\mathrm{HL}}(A_{\bar{I}\bar{J}})\right] \\ &= 2\sum_{\substack{I \subseteq [n] \\ |I| = k}} \mathbf{E}\left[\operatorname{per} A_{IJ} \cdot U^{\mathrm{HL}}(A_{\bar{I}\bar{J}})\right].\end{aligned}$$

The expected value is the same for all sets I, and therefore it is enough to do the analysis for a fixed set $I \subseteq [n]$ with $|I| = k$. Then,

$$2\mathbf{E}[U_d^{\mathrm{HL}}(A)] = 2\binom{n}{k}\mathbf{E}\left[\operatorname{per} A_{IJ} \cdot U^{\mathrm{HL}}(A_{\bar{I}\bar{J}})\right].$$

As all entries are independent,

$$\mathbf{E}[\operatorname{per} A_{IJ} \cdot U^{\mathrm{HL}}(A_{\bar{I}\bar{J}})] = \mathbf{E}[\operatorname{per} A_{IJ}] \cdot \mathbf{E}\left[U^{\mathrm{HL}}(A_{\bar{I}\bar{J}}).\right]$$

Now $\mathbf{E}[\operatorname{per} A_{IJ}]$ is simply $d!(m/n^2)^d$ by linearity of expectation. Let R be a random variable equal to the row sum of the first row of $A_{\bar{I}\bar{J}}$. Because the row sums of $A_{\bar{I}\bar{J}}$ are independent,

$$\mathbf{E}\left[U^{\mathrm{HL}}(A_{\bar{I}\bar{J}})\right] = \mathbf{E}[h(R)/\mathrm{e}]^{n-d}.$$

Note that h is a concave function. It is clearly concave in $[0, 1)$ and $(1, \infty)$. By letting $f(r) = 1 + (\mathrm{e} - 1)r$ and $g(r) = \mathrm{e} - 1 + r + \ln(r)/2$, we have $f(1) = g(1)$ and $f'(r) = \mathrm{e} - 1$. For all $r \geq 1$, we have $g'(r) = 1 + 1/(2r) \leq 3/2 < \mathrm{e} - 1$, so h is concave in $[0, \infty)$. Thus, we can apply Jensen's inequality to get

$$\mathbf{E}[h(R)/\mathrm{e}]^{n-d} \leq \mathrm{e}^{d-n} h\left(\frac{m(n-d)}{n^2}\right)^{n-d}.$$

Hence,

$$\mathbf{E}[U_d^{\mathrm{HL}}(A) \mid M = m] \leq 2\binom{n}{d} d!\mathrm{e}^{d-n} p^d h\left(p(n-d)\right)^{n-d}$$

□

Next, we combine these lemmas to give an upper bound for the ratio of the expected value of the upper bound and the permanent.

Lemma 12. Let $p_* = mn^{-2}$. Then,

$$\frac{\mathbf{E}\left[U_d^{\mathrm{HL}}(A) \mid M = m\right]}{\mathbf{E}\left[\operatorname{per} A \mid M = m\right]} \leq \exp\left\{O(n^3 m^{-2})\right\} (\pi \mathrm{e}(n-d)/2)^{-1/2} \left(\mathrm{e}^{2\mathrm{e}-1} p_*(n-d)\right)^{1/(2p_*)}.$$

Proof. Given that $m = \sigma(n^{3/2})$, Frieze and Jerrum (1995) show that

$$\mathbf{E}\left[\operatorname{per} A \mid M = m\right] = n! \left(\frac{m}{n^2}\right)^n \exp\left\{-\frac{n^2}{2m} + \frac{1}{2} + O\left(\frac{n^3}{m^2}\right)\right\}.$$

This and Lemma 11 give us

$$\frac{\mathbf{E}[U_d^{\mathrm{HL}}(A) \mid M=m]}{\mathbf{E}[\operatorname{per} A \mid M=m]} \le \frac{2\binom{n}{d} d! p_*^d h\left(p_*(n-d)\right)^{n-d}}{n!\left(\frac{m}{n^2}\right)^n \exp\left\{-\frac{n^2}{2m}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}$$
$$= \frac{2}{(n-d)!\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}\left(\frac{h\left(p_*(n-d)\right)}{p_*}\right)^{n-d}.$$

Applying Stirling's lower bound $n! \ge \sqrt{2\pi n} n^n e^{-n}$ results in

$$\frac{2}{(n-d)!\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}\left(\frac{h\left(p_*(n-d)\right)}{p_*}\right)^{n-d}$$
$$\le \sqrt{\frac{2}{\pi(n-d)}}\cdot\frac{1}{\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}\left(\frac{\mathrm{e}h\left(p_*(n-d)\right)}{p_*(n-d)}\right)^{n-d}$$
$$= \sqrt{\frac{2}{\pi(n-d)}}\cdot\frac{1}{\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}\left(1+\frac{\mathrm{e}-1+\frac{1}{2}\ln\left(p_*(n-d)\right)}{p_*(n-d)}\right)^{n-d}.$$

Using the exponential inequality $1+x \le \mathrm{e}^x$ gives us that

$$\sqrt{\frac{2}{\pi(n-d)}}\cdot\frac{1}{\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}\left(1+\frac{\mathrm{e}-1+\frac{1}{2}\ln\left(p_*(n-d)\right)}{p_*(n-d)}\right)^{n-d}$$
$$\le \sqrt{\frac{2}{\pi(n-d)}}\cdot\frac{\exp\left\{\mathrm{e}-1+\frac{1}{2}\ln\left(p_*(n-d)\right)\right\}^{1/p_*}}{\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}.$$

Finally, after some rewriting,

$$\sqrt{\frac{2}{\pi(n-d)}}\cdot\frac{\exp\left\{\mathrm{e}-1+\frac{1}{2}\ln\left(p_*(n-d)\right)\right\}^{1/p_*}}{\exp\left\{-\frac{1}{2p_*}+\frac{1}{2}+O\left(\frac{n^3}{m^2}\right)\right\}}$$
$$= \exp\left\{O(n^3m^{-2})\right\}(\pi\mathrm{e}(n-d)/2)^{-1/2}\left(\mathrm{e}^{2\mathrm{e}-1}p_*(n-d)\right)^{1/(2p_*)}.$$

□

The next lemmas are used to show that our estimates of the expected values tend to be close enough to the actual values.

Lemma 13. Assume that $M=m$. Then, for all α,β, with $0<\alpha,\beta<1$, we have with probability at least $1-\alpha-\beta^{-2}O\left(n^3m^{-2}\right)$ that

$$U_d^{\mathrm{HL}}(A) \le \alpha^{-1}\mathbf{E}[U_d^{\mathrm{HL}}(A)|M=m]$$

and

$$\operatorname{per} A \ge (1-\beta)\mathbf{E}[U_d^{\mathrm{HL}}(A)|M=m].$$

Proof. Applying Markov's inequality gives us that

$$\Pr(U_k(A) \geq \alpha^{-1}\mathbf{E}[U_k(A)] \mid M = m) \leq \alpha.$$

Assuming that $m = \omega(n^{3/2})$, as is the case because n^3m^{-2} tends to zero as n tends to infinity, Frieze and Jerrum (1995) show that

$$\frac{\mathbf{E}[(\operatorname{per} A)^2 \mid M = m]}{\mathbf{E}[\operatorname{per} A \mid M = m]^2} = 1 + O\left(\frac{n^3}{m^2}\right).$$

This immediately implies that

$$\mathbf{Var}[\operatorname{per} A \mid M = m] = \mathbf{E}[\operatorname{per} A \mid M = m]^2 \cdot O\left(\frac{n^3}{m^2}\right),$$

and, by Chebyshev's inequality, we get that

$$\begin{aligned}
&\Pr(\operatorname{per} A \leq (1-\beta)\mathbf{E}[\operatorname{per} A] \mid M = m)\\
=&\Pr\left(\mathbf{E}[\operatorname{per} A] - \operatorname{per} A \geq \beta\mathbf{E}[\operatorname{per} A] \mid M = m\right)\\
\leq&\Pr\left(|\mathbf{E}[\operatorname{per} A] - \operatorname{per} A| \geq \beta\mathbf{E}[\operatorname{per} A] \mid M = m\right)\\
\leq&\frac{\mathbf{Var}[\operatorname{per} A \mid M = m]}{\beta^2\mathbf{E}[\operatorname{per} A \mid M = m]^2}\\
=&\beta^{-2}\left(\frac{\mathbf{E}[(\operatorname{per} A)^2 \mid M = m]}{\mathbf{E}[\operatorname{per} A \mid M = m]^2} - 1\right).
\end{aligned}$$

By applying the result of Frieze and Jerrum again, we get that

$$\Pr(\operatorname{per} A \leq (1-\beta)\mathbf{E}[\operatorname{per} A] \mid M = m) \leq \beta^{-2}O\left(\frac{n^3}{m^2}\right).$$

Finally, we get that the probability that neither of the events happen is at least

$$1 - \alpha - \beta^{-2}O\left(\frac{n^3}{m^2}\right),$$

by Boole's inequality. □

Lemma 14. For any $\delta > 0$, we can select α and β with $0 < \alpha, \beta < 1$ such that

$$\alpha + \beta^{-2}O\left(\frac{n^3}{m^2}\right) \leq \delta,$$

and

$$\alpha^{-1}(1-\beta)^{-1}\exp\left\{O(n^3m^{-2})\right\}(e/2)^{-1/2} \leq \delta^{-1},$$

for all sufficiently large values of n.

Proof. By Lemma 8, the product n^3m^{-2} tends to zero as n tends to infinity. Therefore, for any $c > 0$, there exists n_0 such that for all $n > n_0$ it holds that $n^3m^{-2} < c$. By the definition of O notation, the same property holds for any nonnegative function $f(n) = O(n^3m^{-2})$. Similarly, we have that $\exp\{f(n)\}$ tends to one as n tends to infinity.

Let $\alpha \leq \delta$. Whichever constant β we choose, the first inequality will always hold for sufficiently large values of n. For the second inequality, we note that $(e/2)^{-1/2} < 0.86$. Thus, for example, if we let $\alpha = 0.9\delta$, we have

$$\alpha^{-1}(e/2)^{-1/2} < \delta^{-1}.$$

Finally, we can choose β arbitrarily close to 0 to have $(1-\beta)^{-1}$ be as close to 1 as we want. The factor $\exp\{O(n^3m^{-2})\}$ gets arbitrarily close to 1 as n increases, which completes the proof. □

Theorem 4. Let $\delta > 0$ and d be a constant with $0 \leq d \leq n$. Then, it holds with probability $1 - 2\delta$ for sufficiently large n that

$$\frac{U_d^{\mathrm{HL}}(A)}{\operatorname{per} A} \leq \delta^{-1} \left(\pi(n-d)\right)^{-1/2} \left(e^{2e-1}(n-d)p_0\right)^{1/(2p_0)},$$

where $p_0 \coloneqq p - n^{-1}\sqrt{2p \ln \delta^{-1}}$.

Proof. Let $\delta > 0$ and $p_* = mn^{-2}$. Assume that $M = m$ and that α and β are numbers that fulfill both inequalities of Lemma 14. Then, with probability $1 - \delta$,

$$\frac{U_d^{\mathrm{HL}}(A)}{\operatorname{per} A} \leq \frac{\alpha^{-1}\mathbf{E}[U_d^{\mathrm{HL}}(A) \mid M = m]}{(1-\beta)\mathbf{E}[\operatorname{per} A \mid M = m]}.$$

By Lemma 12, this is at most

$$\alpha^{-1}(1-\beta)^{-1} \exp\left\{O(n^3m^{-2})\right\} \left(\pi \mathrm{e}(n-d)/2\right)^{-1/2} \left(\mathrm{e}^{2\mathrm{e}-1}p_*(n-d)\right)^{1/(2p_*)},$$

which is at most

$$\delta^{-1} \left(\pi(n-d)\right)^{-1/2} \left(e^{2e-1}(n-d)p_*\right)^{1/(2p_*)}$$

by the assumptions. Note that the bound increases asymptotically as m decreases.

According to Lemma 7, M is at least $m_0 \coloneqq n^2p - a$ with probability at least $1 - \delta$ if we choose $a \coloneqq n\sqrt{2p_* \ln(\delta^{-1})}$. We note that

$$\begin{aligned} m_0 n^{-2} &= p_* - an^{-2} \\ &= p_* - n^{-1}\sqrt{2p_* \ln(\delta^{-1})}, \end{aligned}$$

which is p_0 if we substitute m_0 to m. Hence, by Boole's inequality and substituting m_0 to m, we have with probability at least $1 - 2\delta$ that for sufficiently large values of n,

$$\frac{U_d^{\mathrm{HL}}(A)}{\operatorname{per} A} \leq \delta^{-1} \left(\pi(n-d)\right)^{-1/2} \left(e^{2e-1}(n-d)p_0\right)^{1/(2p_0)}.$$

□

Theorem 5. For constants δ, ϵ, d, and c, with $\delta, \epsilon > 0$, $0 \leq d \leq n$, and $c > 1/2$, the expected time complexity for computing an (ϵ, δ)-approximation of the permanent of A using the depth-d variant of the Huber–Law bound is with high probability $O(n^{1.5+c/p})$.

Proof. By Theorem 3, the expected running time for computing an (ϵ, δ)-approximation is $O(2^d dn + U_d^{\mathrm{HL}}(A)/\operatorname{per} A \cdot n^2 \epsilon^{-2} \log \delta^{-1})$, and as d, ϵ, and δ are constants, this simplifies to $O(U_d^{\mathrm{HL}}(A)/\operatorname{per} A \cdot n^2)$.

On the other hand, with probability $1 - 2n^{(1/2-c)/p}$, we have that

$$\frac{U_d^{\mathrm{HL}}(A)}{\operatorname{per} A} \leq n^{(c-1/2)/p} \left(\pi(n-d)\right)^{-1/2} \left(e^{2e-1}(n-d)p_0\right)^{1/(2p_0)},$$

where $p_0 \coloneqq p - n^{-1}\sqrt{2p \ln \delta^{-1}}$. The value $1/(2p_0)$ gets arbitrarily close to $1/(2p)$ as n increases, and thus it is at most $1/(2p) + (c-1/2)/p$ after a certain point. Therefore, the probabilistic fit ratio can be written as

$$O(n^{(1/2-c)/p} \cdot n^{-1/2} \cdot n^{1/(2p)+(c-1/2)/p})$$

and further simplified to $O(n^{-1/2+c/p})$. Combining these two results yields the high probability expected running time $O(n^{1.5+c/p})$. □

6 Empirical Results

In this chapter we compare the deep rejection sampling method with the rejection samplers of Huber and Law (2008) and Kuck et al. (2019) and the Godsil–Gutman type estimators based on real numbers (Godsil and Gutman, 1981), complex numbers (Karmarkar et al., 1993), and quaternions (Chien et al., 2003). The schemes are tested on multiple classes of matrices and some instances based on real-world data. We start by discussing the test setup, after which we present our empirical results.

6.1 Test Setup

We will consider three types of approximation schemes HL-d, AdaPart-d, and GG-t. HL-d is a rejection sample using the depth-d Huber–Law bound, and it is implemented like Algorithm 3. Similarly, AdaPart-d uses the depth-d Soules bound and is implemented like Algorithm 4. GG-t approximates the permanent using Godsil–Gutman type estimators based on reals, complex numbers, and quaternions, and these are denoted by replacing t with r, c, and q, respectively. We also tested the variants of AdaPart-d and HL-d in which we apply the preprocessing on the input as discussed at the end of Section 4.1. These schemes are denoted by "-DS" at the end of the name of the scheme. Finally, we compared the original implementation AdaPart of the AdaPart scheme by Kuck et al. (2019) with AdaPart-0.

In our tests, we tried approximating the time required for getting an (ϵ, δ)-approximation with $\epsilon = 0.1$ and $\delta = 0.05$. For the rejection samplers, getting this precision is equivalent to getting 388 successful samples using the aforementioned GBAS (Huber, 2017). However, to speed up the testing and to be able to estimate the time usage for larger values of n, we stopped sampling after getting 65 successful samples and multiplied the time used for sampling by 388/65.

Let X be the output of one trial of a Godsil–Gutman type estimator for an $n \times n$ matrix A. Then, the number of trials required for achieving an $(\epsilon, 1/4)$-approximation is equal to

$$\frac{4\mathbf{E}[X^2]}{\epsilon^2 \mathbf{E}[X]^2},$$

which can be proven with Chebyshev's inequality (Karmarkar et al., 1993). Chernoff

bounds give that repeating the experiment $22\ln(\delta^{-1})$ times and taking the median results in an (ϵ, δ)-approximation algorithm. The fraction $\mathbf{E}[X^2]/\mathbf{E}[X]^2$ is called the critical ratio of the estimator (Karmarkar et al., 1993), and it is at most $3^{n/2}$ for the estimator based on real numbers, $2^{n/2}$ for complex numbers (Karmarkar et al., 1993), and $(3/2)^{n/2}$ for quaternions (Chien et al., 2003). For computing the determinant, we use Gaussian elimination, whose time complexity is $O(n^3)$, despite the fact that there are some faster methods (for example, Strassen's algorithm (Strassen, 1969)). This was done because the rejection samplers outperform the Godsil–Gutman type estimators so considerably that a slightly faster implementation would not have changed the results significantly, as we shall soon see. Similarly to rejection samplers, we take only 65 samples and scale the time required for this to be an estimate of taking all required samples.

We tested the algorithms with randomly generated instances from several classes of matrices as well as with non-random instances. The considered matrix classes were *Uniform*, *Block Diagonal*, and *Bernoulli(p)*. In *Uniform*, each entry is chosen uniformly at random from $[0, 1]$. The main diagonal of matrices in *Block Diagonal* consists of blocks (last of which might not be whole) of 5×5 matrices with entries uniformly at random from $[0, 1]$ and the other entries are zeros. *Bernoulli(p)* is the class of matrices whose entries follow Bernoulli distribution with parameter p.

Five of the non-random instances *ENZYMES-g192*, *ENZYMES-g230*, *ENZYMES-g479*, *cage5*, and *bcspwr01* are real-world instances are from the Network Repository (http://networkrepository.com, licensed under CC BY-SA) of Rossi and Ahmed (2015). They are the same instances Kuck et al. (2019) tested, and albeit their connection to the permanents does not seem obvious, we included them so that one may compare our and their times if one so desires. In addition to that, we tested the algorithms on instances *Staircase-n* with $n = 30$ and $n = 45$ that are defined such that $A = (a_{ij})$ is a $n \times n$ binary matrix that has $a_{ij} = 1$ if and only if $i + j \leq n + 2$. These serve as an extremely hard case for rejection samplers as discussed in Section 3.8, and around $n = 30$ computing their exact value using Ryser's algorithm starts becoming impractical.

Each test instance had a time limit of 4825 second. This number is motivated by the fact that the estimated running time would be over eight hours if this time limit is exceeded. For the randomly generated instances, we sampled one instance for each value of n between 1 and 200 and ran the approximation scheme on them starting from $n = 1$. If the scheme exceeded the time limit for some $n \times n$ matrix, it was not tested on matrices of the same matrix class larger than that to save computational resources. Tests for random instances

excluding the comparison of AdaPart implementations were run on a computer cluster on one CPU (Sandy Bridge, 2.66 GHz) with a memory limit of 2 GiB. The non-random instances and comparison of AdaPart implementations were run on a laptop with one CPU (Kaby Lake R, 1.60 GHz) with a memory limit of 8 GiB.

6.2 Estimated Running Times

Figure 6.1 contains the comparison of AdaPart and AdaPart-0 for matrices whose entries are uniformly distributed. As both schemes use the Soules bound, the differences in the estimated running times are mostly due to the implementation details. Our AdaPart-0 is about 50 times faster than AdaPart, so we will not include AdaPart in further testing.

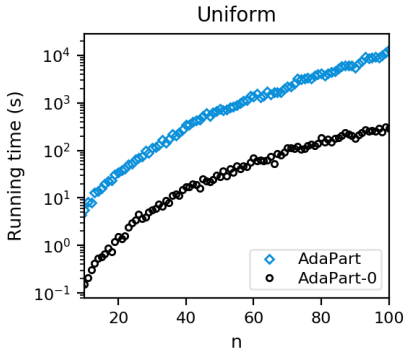

Figure 6.1: Comparison of AdaPart and AdaPart-0.

Similar results can be observed for the Godsil–Gutman type estimators. The performance of schemes GG-r, GG-c, and GG-q is compared with HL-0 on three classes of randomly generated matrices in Figure 6.2. HL-0 was chosen because it performed the worst among the rejection samplers for these classes. These results suggest that Godsil–Gutman type estimators are not worth comparing in further testing.

In Figure 6.3 we compare the rejection sampling models AdaPart-d, AdaPart-d-DS, HL-d, and HL-d-DS with $d = 0$ and $d = 20$ on three classes of matrices. The columns represent the classes of matrices and rows represent the schemes based on the Huber–Law bound and the Soules bound, respectively. For matrices in *Uniform*, the deep rejection sampling does not appear to be useful at the depth 20. This is largely due to the fact their fit ratio tends to increase rather slowly. However, schemes based on the Huber–Law bound seem to be one order of magnitude faster than the ones based on the Soules bound.

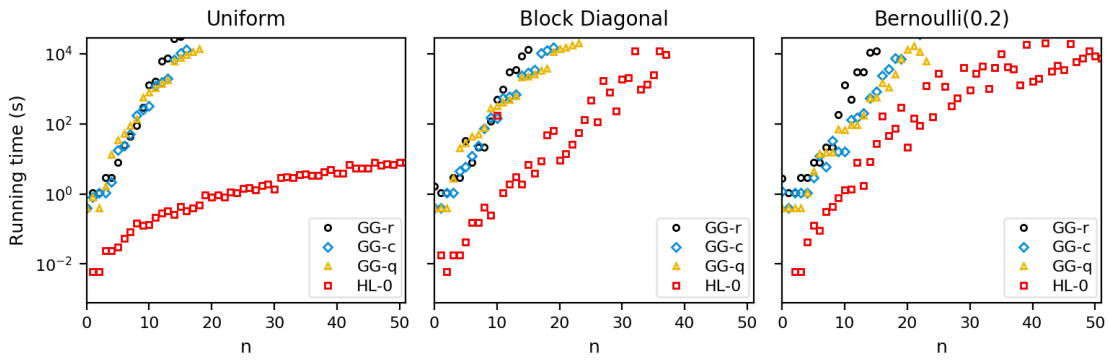

Figure 6.2: Comparison of Godsil–Gutman type estimators and HL-0.

For classes *Block Diagonal* and *Bernoulli(0.2)*, the depth-20 bounds yield improvements of two orders of magnitude. Quite interestingly, the schemes with preprocessing outperform their respective schemes without preprocessing by approximately 1–2 orders of magnitude in *Block Diagonal* whilst in *Bernoulli(0.2)* the preprocessing seems to have a negative impact on the time consumption. The AdaPart method appears advantageous in *Block Diagonal*, and in *Bernoulli(0.2)* models of both bounds perform approximately the same.

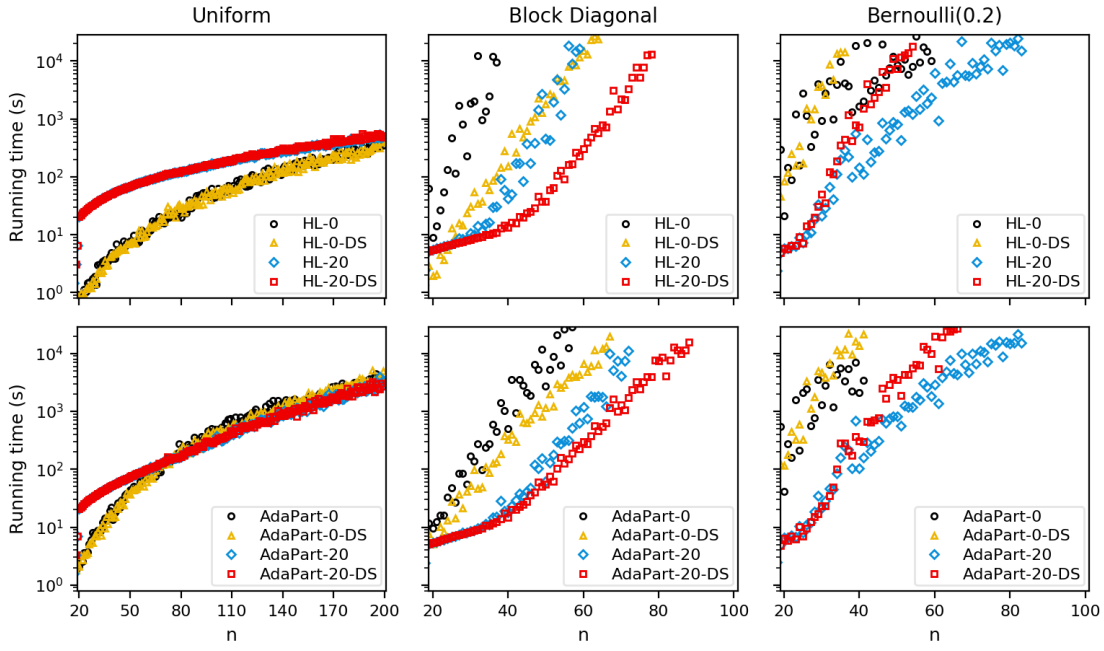

Figure 6.3: Comparison of rejection sampling schemes.

Finally, Table 6.1 contains the estimated running times for the non-random instances. The fastest models are bolded on each row. Depth-20 models beat their depth-0 counterparts by at least 1–3 orders of magnitude in every tested instance. The differences between

different models of the same depth are rather small in all tested instances but *cage5* and *Staircase*-45. The former one is due to *cage5* being a matrix with some row sums less than 1, so HL-d suffers from its relaxed permanental upper bound. The latter one is caused by the preprocessing which decreases the fit ratio from super-exponential to exponential, like we discussed in Section 3.8.

Table 6.1: Estimates of expected running times for non-random instances.

		AdaPart-d		AdaPart-d-DS		HL-d		HL-d-DS	
Instance	n	$d=0$	$d=20$	$d=0$	$d=20$	$d=0$	$d=20$	$d=0$	$d=20$
ENZYMES-g192	31	$2\cdot10^2$	$\mathbf{1\cdot10^1}$	$8\cdot10^2$	$\mathbf{1\cdot10^1}$	$1\cdot10^2$	$\mathbf{1\cdot10^1}$	$6\cdot10^2$	$2\cdot10^1$
ENZYMES-g230	32	$3\cdot10^2$	$2\cdot10^1$	$1\cdot10^3$	$\mathbf{1\cdot10^1}$	$2\cdot10^2$	$\mathbf{1\cdot10^1}$	$1\cdot10^3$	$2\cdot10^1$
ENZYMES-g479	28	$3\cdot10^2$	$8\cdot10^0$	$1\cdot10^3$	$8\cdot10^0$	$1\cdot10^2$	$\mathbf{7\cdot10^0}$	$7\cdot10^2$	$9\cdot10^0$
cage5	37	$2\cdot10^3$	$2\cdot10^1$	$1\cdot10^3$	$\mathbf{1\cdot10^1}$	$>10^4$	$5\cdot10^3$	$9\cdot10^2$	$2\cdot10^1$
bcspwr01	39	$6\cdot10^2$	$\mathbf{1\cdot10^1}$	$4\cdot10^3$	$2\cdot10^1$	$3\cdot10^2$	$\mathbf{1\cdot10^1}$	$2\cdot10^3$	$2\cdot10^1$
Staircase-30	30	$>10^4$	$2\cdot10^1$	$9\cdot10^3$	$8\cdot10^0$	$>10^4$	$2\cdot10^1$	$3\cdot10^3$	$\mathbf{7\cdot10^0}$
Staircase-45	45	$>10^4$	$>10^4$	$>10^4$	$2\cdot10^3$	$>10^4$	$>10^4$	$>10^4$	$\mathbf{8\cdot10^2}$

7 Conclusions

In this thesis, we developed the deep rejection sampling method based on deep variants of the permanental upper bounds, continuing the recent trend of applying rejection sampling on approximating the permanent. We proved a new high probability upper bound for the expected time complexity of computing an (ϵ, δ)-approximation of a random matrix whose entries follow Bernoulli distribution with parameter p using the depth-d Huber–Law bound. We noted that this bound improves an earlier result when $p < 1/5$, and, albeit Godsil–Gutman estimators need only a linear number of samples with high probability (Frieze and Jerrum, 1995), our bound is more "trustworthy".

In empirical testing, our deep rejection sampling method proved to be superior compared to other rejection sampling methods and Godsil–Gutman type estimators, beating them by multiple orders of magnitude especially on harder instances, in which the ratio of the permanental upper bound and the permanent is large. To our knowledge, this was the first time the effects of making a matrix nearly doubly stochastic and dividing the rows by their largest entries have been tested empirically. The results for this showed that this preprocessing can be both helpful and harmful depending on the instance.

We also gave the first detailed implementation of the adaptive partitioning method of Kuck et al. (2019) that has $O(n^3)$ time complexity per trial assuming there is a column partition on which the Soules bound is nesting. Our implementation was 1–2 orders of magnitude faster than their implementation.

Still, many intriguing questions remain. For example, different values of d work the best in different instances. When the upper bound and the permanent are close to each other, a smaller d tends to be better, while in harder instances higher precomputation time caused by a higher d is often worth taking. A natural question is that could a nearly optimal value of d be guessed efficiently. The set of columns J affects the running time as well and thus choosing it optimally should be explored more.

Kuck et al. (2019) explored the possibility of improving rejection sampling by tightening a bound whenever slack was encountered and propagating that upwards in the partition tree. They reported that this improved the bound to roughly two thirds (64–89%) of the original bound after 1000 trials. This technique seems hard to apply on the deep rejection sampling method, as there is no obvious way for updating the deep sampling

table efficiently, but might still be worth investigating.

Sharpened upper bounds discussed by Soules (2003, 2005) provide another potential way of improving rejection sampling, as they can decrease the fit ratio of a matrix massively. Unfortunately, it seems to be rather hard to find any nesting partition trees for such bounds in feasible time.

Bibliography

Angluin, D. and Valiant, L. G. (1979). "Fast probabilistic algorithms for Hamiltonian circuits and matchings". *J. Comput. Syst. Sci.* 18.2, pp. 155–193.

Arvind, V. and Srinivasan, S. (2010). "On the hardness of the noncommutative determinant". In: *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC 2010*. Association for Computing Machinery, pp. 677–686.

Barvinok, A. (1999). "Polynomial time algorithms to approximate permanents and mixed discriminants within a simply exponential factor". *Random Struct. Algorithms* 14.1, pp. 29–61.

Beichl, I. and Sullivan, F. (1999). "Approximating the permanent via importance sampling with application to the dimer covering problem". *J. Comput. Phys.* 149.1, pp. 128–147.

Ben-Dor, A. and Halevi, S. (1993). "Zero-one permanent is $\#P$-complete, a simpler proof". In: *Proceedings of the Second Israel Symposium on Theory of Computing Systems, ISTCS 1993*. IEEE Computer Society, pp. 108–117.

Bezáková, I., Štefankovič, D., Vazirani, V. V., and Vigoda, E. (2008). "Accelerating simulated annealing for the permanent and combinatorial counting problems". *SIAM J. Comput.* 37.5, pp. 1429–1454.

Björklund, A., Husfeldt, T., Kaski, P., and Koivisto, M. (2010). "Evaluation of permanents in rings and semirings". *Inf. Process. Lett.* 110.20, pp. 867–870.

Blanchard, P., Higham, D. J., and Higham, N. J. (2019). "Accurate computation of the log-sum-exp and softmax functions". *math.NA* abs/1909.03469.

Brègman, L. M. (1973). "Some properties of nonnegative matrices and their permanents". In: vol. 211. 1, pp. 27–30.

Brualdi, R. A., Parter, S. V., and Schneider, H. (1966). "The diagonal equivalence of a nonnegative matrix to a stochastic matrix". *J. Math. Anal. Appl.* 16.1, pp. 31–50.

Chen, Y. and Liu, J. (2007). "Sequential Monte Carlo Methods for permutation tests on truncated data". *Stat. Sin* 17.3, pp. 857–872.

Chien, S., Harsha, P., Sinclair, A., and Srinivasan, S. (2011). "Almost settling the hardness of noncommutative determinant". In: *Proceedings of the Forty-Third ACM Symposium on Theory of Computing, STOC 2011*. ACM, pp. 499–508.

Chien, S., Rasmussen, L. E., and Sinclair, A. (2003). "Clifford algebras and approximating the permanent". *J. Comput. Syst. Sci.* 67.2, pp. 263–290.

Egorychev, G. P. (1981). “The solution of van der Waerden’s problem for permanents”. *Adv. Math.* 42.3, pp. 299–305.

Falikman, D. I. (1981). “Proof of the van der Waerden conjecture regarding the permanent of a doubly stochastic matrix”. *Math. Notes* 29.6, pp. 475–479.

Frieze, A. and Jerrum, M. (1995). “An analysis of a Monte Carlo algorithm for estimating the permanent”. *Combinatorica* 15.1, pp. 67–83.

Fürer, M. and Kasiviswanathan, S. P. (2004). “An almost linear time approximation algorithm for the permanent of a random (0-1) matrix”. In: *Proceedings of the Twenty-Fourth International Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science*. Vol. 3328. Springer, pp. 263–274.

Glynn, D. G. (2010). “The permanent of a square matrix”. *Eur. J. Comb.* 31.7, pp. 1887–1891.

Godsil, C. D. and Gutman, I. (1981). “On the matching polynomial of a graph”. In: *Algebraic Methods in Graph Theory, Colloquia Mathematica Societatis János Bolyai*. 25. North-Holland, pp. 241–249.

Gumbel, E. J. (1954). *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. U.S. Government Printing Office.

Gurvits, L. and Samorodnitsky, A. (2014). “Bounds on the permanent and some applications”. In: *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*. IEEE Computer Society, pp. 90–99.

Harviainen, J., Röyskö, A., and Koivisto, M. (2021). “Approximating the Permanent with Deep Rejection Sampling”. unpublished.

Hopcroft, J. E. and Karp, R. M. (1973). “An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs”. *SIAM J. Comput.* 2.4, pp. 225–231.

Huber, M. (2006). “Exact sampling from perfect matchings of dense regular bipartite graphs”. *Algorithmica* 44.3, pp. 183–193.

– (2017). “A Bernoulli mean estimate with known relative error distribution”. *Random Struct. Algorithms* 50.2, pp. 173–182.

Huber, M. and Law, J. (2008). “Fast approximation of the permanent for very dense problems”. In: *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008*. Society for Industrial and Applied Mathematics, pp. 681–689.

Jerrum, M., Sinclair, A., and Vigoda, E. (2004). “A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries”. *J. ACM* 51.4, pp. 671–697.

Jurkat, W. B. and Ryser, H. J. (1966). "Matrix factorizations of determinants and permanents". *J. Algebra* 3.1, pp. 1–27.

Karmarkar, N., Karp, R., Lipton, R., Lovász, L., and Luby, M. (1993). "A Monte-Carlo algorithm for estimating the permanent". *SIAM J. Comput.* 22.2, pp. 284–293.

Kasteleyn, P. (1967). "Graph theory and crystal physics". In: *Graph Theory and Theoretical Physics*. Ed. by F. Harary. Academic Press, pp. 43–110.

Kuck, J., Dao, T., Rezatofighi, H., Sabharwal, A., and Ermon, S. (2019). "Approximating the permanent by sampling from adaptive partitions". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8860–8871.

Kuznetsov, N. Y. (1996). "Computing the permanent by importance sampling method". *Cybern. Syst. Anal.* 32.6, pp. 749–755.

Maddison, C. J., Tarlow, D., and Minka, T. (2014). "A* sampling". In: *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., pp. 3086–3094.

Minc, H. (1963). "Upper bounds for permanents of (0, 1)-matrices". *Bull. Amer. Math. Soc* 69.6, pp. 789–791.

Mitzenmacher, M. and Upfal, E. (2017). *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press.

Moore, C. and Russell, A. (2012). "Approximating the permanent via nonabelian determinants". *SIAM J. Comput.* 41.2, pp. 332–355.

Newman, J. E. and Vardi, M. Y. (2020). "FPRAS approximation of the matrix permanent in practice". *CoRR* abs/2012.03367.

Radhakrishnan, J. (1997). "An entropy proof of Bregman's theorem". *J. Comb. Theory, Ser. A* 77.1, pp. 161–164.

Rasmussen, L. E. (1994). "Approximating the permanent: A simple approach". *Random Struct. Algorithms* 5.2, pp. 349–362.

Rossi, R. A. and Ahmed, N. K. (2015). "The network data repository with interactive graph analytics and visualization". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 4292–4293.

Ryser, H. J. (1963). *Combinatorial mathematics*. Vol. 14. Mathematical Association of America.

Schrijver, A. (1978). "A short proof of Minc's conjecture". *J. Comb. Theory, Ser. A* 25.1, pp. 80–83.

Sharir, M. (1981). "A strong-connectivity algorithm and its applications in data flow analysis". *Comput. Math. Appl.* 7.1, pp. 67–72.

Sinkhorn, R. (1964). “A relationship between arbitrary positive matrices and doubly stochastic matrices”. *Ann. Math. Stat.* 35.2, pp. 876–879.

Sinkhorn, R. and Knopp, P. (1967). “Concerning nonnegative matrices and doubly stochastic matrices”. *Pac. J. Math.* 21.2, pp. 343–348.

Smith, P. J. and Dawkins, B. (2001). “Estimating the permanent by importance sampling from a finite population”. *J. Stat. Comput. Simul.* 70.3, pp. 197–214.

Smith, P. J., Gao, H., and Clark, M. V. (2002). “Performance bounds for MMSE linear macrodiversity combining in Rayleigh fading, additive interference channels”. *J. Commun. Networks* 4.2, pp. 102–107.

Soules, G. W. (1991). “The rate of convergence of Sinkhorn balancing”. *Linear Algebra Appl.* 150, pp. 3–40.

– (2000). “Extending the Minc–Brègman upper bound for the permanent”. *Linear Multilinear Algebra* 47.1, pp. 77–91.

– (2003). “New permanental upper bounds for nonnegative matrices”. *Linear Multilinear Algebra* 51.4, pp. 319–337.

– (2005). “Permanental bounds for nonnegative matrices via decomposition”. *Linear Algebra Appl.* 394, pp. 73–89.

Strassen, V. (1969). “Gaussian elimination is not optimal”. *Numer. Math.* 13.4, pp. 354–356.

Sullivan, F. and Beichl, I. (2014). “Permanents, α-permanents and Sinkhorn balancing”. *Comput. Stat.* 29, pp. 1793–1798.

Tarjan, R. E. (1972). “Depth-first search and linear graph algorithms”. *SIAM J. Comput.* 1.2, pp. 146–160.

Tassa, T. (2012). “Finding all maximally-matchable edges in a bipartite graph”. *Theor. Comput. Sci.* 423, pp. 50–58.

Uhlmann, J. K. (2004). “Matrix permanent inequalities for approximating joint assignment matrices in tracking systems”. *J. Frankl. Inst.* 341.7, pp. 569–593.

Valiant, L. G. (1979). “The complexity of computing the permanent”. *Theor. Comput. Sci.* 8.2, pp. 189–201.

van den Brand, J., Lee, Y. T., Nanongkai, D., Peng, R., Saranurak, T., Sidford, A., Song, Z., and Wang, D. (2020). “Bipartite matching in nearly-linear time on moderately dense graphs”. In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*. IEEE, pp. 919–930.