Proceedings of ACDA 2021

Contributed Talks With Proceedings Papers

Contributed Talks Without Proceedings Papers

Contributed Posters

Invited Plenary Talks

Diffusion-based methods for biological network analysis
Lenore J. Cowen, Tufts University

Shortest path algorithms for road navigation
Andrew V. Goldberg, Amazon

Combinatorial optimization algorithms for clustering and machine learning
Dorit Hochbaum, University of California, Berkeley

Towards scalable and practical real-time computational epidemiology (joint plenary with SIAM Annual Meeting)
Madhav Marathe, University of Virginia

Scaling up network centrality computations
Henning Meyerhenke, Humboldt University Berlin

Matrix-free Jacobian chaining
Uwe Naumann, RWTH-Aachen

Minitutorials

An introduction to combinatorial scientific computing
Organizers: Rob Bisseling, Henning Meyerhenke and Uwe Naumann
Presenters: Assefaw Gebremedhin, Fredrick Manne and Christian Schulz

Combinatorial frontiers in computational biology
Organizers: Lenore J. Cowen and Christine Heitsch
Presenters: Christine Heitsch, Xiaozhe Hu, Smita Krishnaswamy and James Murphy

Industrial Problems Session

Performance analysis of HPC applications
Michael A. Frumkin, NVIDIA

What's left in dictionary design?
Rob Johnson, VMware

Next generation of parallel and distributed graph mining algorithms: theory and practice
Vahab Mirrokni, Google

Mixed integer programming - A confluence of algorithms
Edward Rothberg, Gurobi Optimization

Program Committee

Michael A. Bender (co-chair)	Stony Brook University
John R. Gilbert (co-chair)	University of California, Santa Barbara
David A. Bader	New Jersey Institute of Technology
Austin Benson	Cornell University
Jonathan Berry	Sandia National Laboratories
Aydin Buluc	Lawrence Berkeley National Laboratory
Umit Catalyurek	Georgia Institute of Technology
Tzu-Yi Chen	Pomona College
Alexander Conway	Rutgers University
Timothy A. Davis	Texas A&M University
Lori Diachin	Lawrence Livermore National Laboratory
Anne Driemel	University of Bonn
Martin Farach-Colton	Rutgers University
Sándor Fekete	Technische Universität Braunschweig
Jeremy Fineman	Georgetown University
Assefaw Gebremedhin	Washington State University
Phillip B. Gibbons	Carnegie Mellon University
Michael Goodrich	University of California, Irvine
Oded Green	NVIDIA/Georgia Institute of Technology
Laura Grigori	INRIA
Paul Hovland	Argonne National Laboratory
Rob Johnson	VMware Research Group
Jeremy Kepner	Massachusetts Institute of Technology
Stephen Kobourov	University of Arizona
Sherry Li	Lawrence Berkeley National Laboratory
Ivana Ljubic	ESSEC Business School
Kamesh Madduri	The Pennsylvania State University
Fredrik Manne	University of Bergen
Samuel McCauley	Williams College
Nicole Megow	Universität Bremen
Maryam Mehri Dehnavi	Rutgers University
Michael Mitzenmacher	Harvard University
Jose Moreira	IBM
Benjamin Moseley	Carnegie Mellon University
Jelani Nelson	University of California, Berkeley
Guillaume Pallez	INRIA
Prashant Pandey	Univerity of California, Berkeley
Robert Patro	University of Maryland
Richard Peng	Georgia Institute of Technology
Ali Pinar	Sandia National Laboratories
Alex Pothen	Purdue University
Emilie Purvine	Pacific Northwest National Laboratory
Eva Rotenberg	Technical University of Denmark
Peter Sanders	Karlsruhe Institute of Technology
Tao Schardl	Massachusetts Institute of Technology
Julian Shun	Massachusetts Institute of Technology

Sabine Storandt	University of Konstanz
Yihan Sun	University of California, Riverside
David Tench	Stony Brook University
Shang-Hua Teng	University of Southern California
Sivan Toledo	Tel Aviv University
Denis Trystram	Grenoble Alpes university
Richard Vuduc	Georgia Institute of Technology
Andrea Walther	Humboldt-Universität zu Berlin
Ulrike Yang	Lawrence Livermore National Laboratory

Organizing Committee

Bruce Hendrickson (co-chair)	Lawrence Livermore National Laboratory
Blair Sullivan (co-chair)	University of Utah
Rob Bisseling	University of Utrecht
Christine Heitsch	Georgia Institute of Technology
Monika Henzinger	University of Vienna
Cynthia Phillips	Sandia National Laboratories
Clifford Stein	Columbia University
David Williamson	Cornell University

Engagement Committee

Christian Schulz (chair)	Heidelberg University
Austin Benson	Cornell University
Cynthia Phillips	Sandia National Laboratories
Richard Vuduc	Georgia Institute of Technology

Prize Committee

Jonathan Berry (chair)	Sandia National Laboratories
Umit Catalyurek	Georgia Institute of Technology
Tzu-Yi Chen	Pomona College
Christine Klymko	Lawrence Livermore National Laboratory

Queues with Small Advice

Michael Mitzenmacher*
Harvard University

Abstract

Motivated by recent work on scheduling with predicted job sizes, we consider the performance of scheduling algorithms with minimal advice, namely a single bit. The analysis of such schemes, besides demonstrating the power of very limited advice, is quite natural. In the prediction setting, one bit of advice can be used to model a simple prediction as to whether a job is "large" or "small"; that is, whether a job is above or below a given threshold. Further, one-bit advice schemes can correspond to mechanisms that tell whether to put a job at the front or the back for the queue, a limitation which may be useful in many implementation settings. Finally, queues with a single bit of advice have a simple enough state that they can be analyzed in the limiting mean-field analysis framework for the power of two choices. Our work follows in the path of recent work by showing that even small amounts of even possibly inaccurate information can greatly improve scheduling performance.

1 Introduction

1.1 Motivation In queueing settings where the required service time for a job is known, strategies that take advantage of that information, such as Shortest Job First (SJF) or Shortest Remaining Processing Time (SRPT) can yield significant performance improvements over blind strategies such as First In First Out (FIFO). However, exact knowledge of the service times is a great deal to ask for in practice. Here we consider the setting where one is given much more limited information. Specifically, we consider the case where, for each job, a queue gets only one bit of information, or advice, regarding the job size.

While a one bit limitation may seem unusual, there are both theoretical and practical motivations for such a study. Online algorithms with small amounts of optimal advice has been a subject of study in the theoretical literature (see, e.g., the survey [4]); such work highlights the potential for additional information to improve performance. Considering the case of just one bit of information is an interesting limiting case. Further, one-bit advice can naturally correspond to informing whether a job should be placed at the front or the back of the queue; for some queue implementations, such as in hardware or other highly constrained settings, one may desire this simplicity over more complicated data structures for managing job placement in the queue.

However, as a more concrete practical motivation, recently researchers have studied queues with *predicted* service times, rather than exact service times, where such predictions might naturally be provided by a machine learning algorithm [6, 18, 19, 20, 22, 25]. Indeed, the queueing setting is one natural example of an expanding line of work where predictions can be used to improve algorithms, particularly in scheduling (e.g., [12, 14, 16, 20, 22]). Our setting here of one-bit predictions can model a natural setting where the prediction corresponds to whether a job's service time is believed to be above or below a fixed threshold. Such predictions may be simpler to implement or more accurate than schemes that attempt to provide a prediction of the exact service time.

For single-queue settings, our work uses standard queueing theoretic analysis techniques. Here we generally follow the (folklore) approach of using Kleinrock's Conservation Law to derive formulae for the conditional waiting time of a job according to its service time; this approach dates back to at least the work of O'Donovan [21], from whose framework and notation we borrow. The derivations can also be readily obtained using the analysis of priority systems, following the framework presented in for example [9]. The goal here is not to suggest new methods of analysis, but instead:

- show how the problem of scheduling with limited predicted information can naturally be analyzed;
- demonstrate how even limited advice and predictions can provide large performance gains; and
- show some interesting derivations for the special case we refer to as exponential predictions.

We also examine one-bit predictions schemes with large numbers of queues using the power of two choices. Here each arrival chooses the better of two randomly selected queues (or more generally from d randomly selected queues) from a large system of (homogeneous)

*Supported in part by NSF grants CCF-1563710, CCF-1535795, DMS-2023528, and by a gift to the Center for Research on Computation and Society at Harvard University.

queues. This study shares many of the same motivations as for single queues; moreover, it may offer a first step to some open questions in the area, such as analyzing the power of two choices when using Shortest Remaining Processing Time or related schemes at the queues (see e.g. [19]).

Finally, more generally, we believe this work also highlights some aspects of using machine learning predictions that may provide guidance for the design of machine learning prediction settings. For example, we see that some predictions may be much more important than others; in queueing settings, it seems generally much more important to identify long jobs correctly than short jobs, as long jobs will block many other jobs from service. One could therefore aim to design predictors that aim for accuracy in predicting long jobs over simply trying to maximize, for example, the number of correct predictions. (Related ideas on tailoring the machine learning component for its use in an algorithmic setting is discussed in for example [2].)

2 Single Queues and One-Bit Thresholds

To begin, we consider simple schemes where the single bit of advice associated with a job corresponds to whether its service time is below or above a threshold value. We emphasize that while these derivations follow from standard analysis techniques, they lay the groundwork for going forward.

2.1 Notation and Model We consider M/G/1 queueing systems; the (Kendall) notation M/G/1 means that the time between arrivals are exponentially distributed (that is, they form a Poisson process), the service times are governed by a specific distribution, and there is a single server. More specifically, we assume the arrival rate is $\lambda < 1$, and the processing times are independently sampled according to the cumulative distribution $F(x)$ with corresponding density $f(x)$. We follow some of the notation from [21]. We assume the expected service time has been scaled so the mean service time is 1 (that is, $\mathbf{E}[F] = 1$). Note $\mathbf{E}[F^2]$ is the second moment for the service time. We further let

$$V = \frac{\lambda \mathbf{E}[F^2]}{2}$$

be the expected remaining service time of the job being served at the time of a random arrival. We also let

$$\rho(t) = \lambda \int_0^t x f(x) dx$$

be the rate at which load is added to the queue from jobs with service time at most t, and let

$$\rho = \lambda \int_\infty^t x f(x) dx = \lambda$$

be the total rate at which load is added to the queue.

2.2 The Conservation Law As described in [21], Kleinrock's Conservation Law says that for a queue with Poisson arrivals satisfying basic assumptions (such as the queue is busy whenever there are jobs in the system), the expected load L on the system at a random time point (e.g., in the stationary distribution), satisfies

$$L(1 - \rho) = V,$$

where again V is the expected load due to the job in service and ρ is the total rate at which load is added to the system. (See also [13, 23].) The law allows simple derivations of conditional expected waiting times, by looking at appropriate subsystems of jobs.

2.3 Analysis of One-Bit Threshold Schemes We consider the case of an advisor that provides a single bit of advice per job. Specifically we consider the strategy where the advice bit is 0 if the job's service time is less than some threshold T, and 1 otherwise. The job is placed at the front of the queue if the advice bit is 0, and at the back of the queue otherwise. We consider preemptive and non-preemptive queues, where in the preemptive case a job placed at the front will preempt the job currently receiving service. We later generalize the one bit of advice to prediction-based systems, where the prediction is whether the service time for the job is larger or smaller than the threshold. We consider below the waiting time, by which we mean the time spent by an incoming job in the stationary distribution waiting before starting to obtain service, and the sojourn time, which similarly refers to the entire time spent by a job in the system.

2.3.1 The non-preemptive system We start with the following theorem.

THEOREM 2.1. *Let S_n (where the subscript n stands for non-preemptive) be the expected sojourn time for a job in an M/G/1 system with threshold T. Then*

$$S_n = \frac{V(1 - F(T)\rho)}{(1 - \rho)(1 - \rho(T))} + 1.$$

Proof. We first consider arriving jobs with service time at most T. Here we do not require the conservation rule; such a job is placed at the front of the queue, although it has to wait for the job, if any, in service to complete. Further, any additional jobs of service time at most T that arrive before this job starts service is placed ahead

of the arriving job being considered. We denote the expected waiting time for a job with service time t by $W_1(t)$. We denote the expected sojourn time by $S_1(t)$.

The expected time an arriving job has to wait for an existing job being processed is V. It follows from standard busy period analysis that additional incoming jobs increase the expected waiting time by a factor of $\rho(T)$, and so

$$W_1(t) = \frac{V}{1-\rho(T)}.$$

The expected sojourn time for such jobs is thus

$$S_1(t) = \frac{V}{1-\rho(T)} + t.$$

For jobs with service time t larger than T, we consider the subsystem of all jobs, and use the notation $W_2(t)$ and $S_2(t)$ for the corresponding quantities. Recall $L = V/(1-\rho)$ from the conservation law. For any job with service time larger than T, any new job with service time at most T that arrives will be placed ahead of this job until it is served. Hence

$$W_2(t) = \frac{L}{1-\rho(T)} = \frac{V}{(1-\rho)(1-\rho(T))},$$

and

$$S_2(t) = \frac{V}{(1-\rho)(1-\rho(T))} + t.$$

For a given service distribution F and threshold T, we have the total expected waiting time W_n is then

$$W_n = \frac{VF(T)}{1-\rho(T)} + \frac{V(1-F(T))}{(1-\rho)(1-\rho(T))} = \frac{V(1-F(T)\rho)}{(1-\rho)(1-\rho(T))}.$$

The expected sojourn time S_n satisfies $S_n = W_n + 1$. □

In general, minimizing W_n (or S_n) can be accomplished numerically. We look now at some interesting examples service distributions.

Example: Exponentially Distributed Service As an example we discuss through this work, for exponentially distributed service times, $V = \lambda$, $F(T) = 1 - e^{-T}$, $\rho = \lambda$, and $\rho(T) = (\lambda)(1-(T+1)e^{-T})$. We find the expected sojourn time for this case, which we refer to as $S_{e,n}$, is then

$$S_{e,n} = \frac{\lambda(1-\lambda+\lambda e^{-T})}{(1-\lambda)(1-(\lambda(1-(T+1)e^{-T})))} + 1.$$

Taking the derivative, we find the optimal T value occurs when

$$\frac{1}{\lambda} - 1 = \frac{e^{-T}}{T-1},$$

or equivalently we seek T that satisfies

$$\lambda = \frac{T-1}{e^{-T}+T-1},$$

In particular, as λ goes to 1, the optimal T increases to infinity, and as λ goes to 0, the optimal T goes to 1. It is perhaps worth noting that a threshold T of 4 corresponds to a λ larger than 0.99; that is, in this case, we do not see very large thresholds even under high load.

Example: Weibull Distributed Service As another example, we consider service distributions following the Weibull distribution with cumulative distribution $F(x) = 1 - e^{-\sqrt{2x}}$. The Weibull distribution is heavy-tailed; while the average service time of this distribution remains 1, the second moment is 6, so there are many more very long jobs as compared to the exponential distribution. Weibull distributions are commonly used for queueing simulations, as heavy-tailed service time distributions are more realistic for many scenarios.

For this Weibull distribution, $V = 3\lambda$ and $\rho(T) = \lambda(1 - e^{-\sqrt{2T}}(T + \sqrt{2T} + 1))$ are computed easily. The expected sojourn time in this case, which we denote by $S_{w,n}$, is then given by

$$S_{w,n} = \frac{3\lambda(1-\lambda+\lambda e^{-\sqrt{2T}})}{(1-\lambda)(1-(\lambda(1-e^{-\sqrt{2T}}(T+\sqrt{2T}+1))))} + 1.$$

2.3.2 The preemptive system For preemptive systems we have the corresponding result.

THEOREM 2.2. *Let S_p (where the subscript p stands for preemptive) be the expected sojourn time for a job in an M/G/1 system with threshold T. Then*

$$S_p = \frac{V(1-F(T)) + 1 - \rho}{(1-\rho)(1-\rho(T))}.$$

Proof. We first consider arriving jobs with service time at most T. Again, here we do not require the conservation rule; such a job is placed at the front of the queue, and any additional job of service time at most T that arrive before this job starts service is placed ahead of the arriving job being considered. We use $W_1(t)$ and $S_1(t)$ as before.

Clearly $W_1(t) = 0$. However, we also consider the effect of preemptions. Since any job of size at most T will preempt the job, the expected sojourn time is

$$S_1(t) = \frac{t}{1-\rho(T)}.$$

For jobs with service times larger than T, we consider the subsystem of all jobs, and again use the notation $W_2(t)$ and $S_2(t)$ for the corresponding quantities.

In this setting $L = V/(1-\rho)$ from the conservation law. While waiting any job of service time at most T is placed ahead of any job of size greater than T, so again

$$W_2(t) = \frac{L}{1-\rho(T)} = \frac{V}{(1-\rho)(1-\rho(T))}.$$

Because of the preemption, the expected time from the start of service until finishing service increases to $t/(1-\rho(T))$, and so

$$S_2(t) = \frac{V}{(1-\rho)(1-\rho(T))} + \frac{t}{1-\rho(T)}.$$

In this case the total expected waiting time is

$$W_p = \frac{V(1-F(T))}{(1-\rho)(1-\rho(T))},$$

and the total expected sojourn time S_p is

$$\begin{aligned} S_p &= \frac{V(1-F(T))}{(1-\rho)(1-\rho(T))} + \frac{1}{1-\rho(T)} \\ &= \frac{V(1-F(T)) + 1 - \rho}{(1-\rho)(1-\rho(T))}. \end{aligned}$$

□

Example: Exponentially Distributed Service The expected sojourn time for exponentially distributed service times, which we refer to as $S_{e,p}$, is then

$$S_{e,p} = \frac{1-\lambda+\lambda e^{-T}}{(1-\lambda)(1-(\lambda(1-(T+1)e^{-T})))}.$$

We can readily show that $S_{e,p} < S_{e,n}$ for any value of T, as long $S_{e,p} < 1/(1-\lambda)$. Indeed,

$$\lambda S_{e,p} = S_{e,n} - 1,$$

so

$$S_{e,n} - S_{e,p} = 1 - (1-\lambda)S_{e,p} > 0.$$

But $S_{e,p}$ can be shown to be less than $1/(1-\lambda)$, since the use of the threshold improves performance over a standard $M/M/1$ queue with expected sojourn time $1/(1-\lambda)$, which can be shown via standard approaches (such as coupling).

Also, the optimal value of T is again given by

$$\frac{1}{\lambda} - 1 = \frac{e^{-T}}{T-1}.$$

Example: Weibull Distributed Service For the Weibull distribution, we have the corresponding expression

$$S_{w,p} = \frac{1-\lambda+3\lambda e^{-\sqrt{2T}}}{(1-\lambda)(1-(\lambda(1-e^{-\sqrt{2T}}(T+\sqrt{2T}+1))))}.$$

3 Adding Predictions

We consider a simple model where the probability of a misprediction for a given item depends only on its service time, independent of other jobs and other considerations. Specifically, we suppose we have a desired threshold T, and our prediction is simply our best guess as to whether a job's service time is larger or less than T. We define $g_T(x)$ be the probability that a job of size x is predicted to be less than T. While one can imagine more complex prediction models, this model is quite natural, and is useful for examining the potential power of predictions.

3.1 The non-preemptive system To deal with the predictions, we now let

$$Q(T) = \int_0^\infty f(x) g_T(x) dx,$$

and

$$\rho'(T) = \lambda \int_0^\infty x f(x) g_T(x) dx.$$

Here $\rho'(T)$ can be interpreted as the rate load arrives to the system from jobs with *predicted service time* at most T, and similarly $Q(T)$ is the fraction of jobs predicted to have service time at most T.

We first consider arriving jobs with predicted service time at most T and actual service time t. Following the same reasoning we have previously used, the waiting time W'_t for such jobs is given by

$$W'_1(t) = \frac{V}{1-\rho'(T)}.$$

For jobs with predicted service time greater than T,

$$W'_2(t) = \frac{V}{(1-\rho)(1-\rho'(T))}.$$

The total expected waiting time per job is therefore given by

$$\begin{aligned} W'_n &= \frac{V\int_0^\infty f(x)g_T(x)dx}{1-\rho'(T)} + \frac{V\int_0^\infty f(x)(1-g_T(x))dx}{(1-\rho)(1-\rho'(T))} \\ &= \frac{V(1-\rho\int_0^\infty f(x)g_T(x)dx)}{(1-\rho)(1-\rho'(T))} \\ &= \frac{V(1-\rho Q(T))}{(1-\rho)(1-\rho'(T))}. \end{aligned}$$

In particular, we see that the only changes from the setting without the prediction is that in the $1-F(T)\rho$ term in the numerator, the $F(T)$ has been replaced by the more complex integral expression $Q(T)$, and similarly the $1-\rho(T)$ term in the denominator has become $1-\rho'(T)$. As before, we have the expected sojourn time is given by $S'_n = W'_n + 1$.

Example: Exponential Predictions A model suggested in [18, 19] considers the setting where a prediction for a job with service time z is itself exponentially distributed with mean z; we refer to this as the exponential prediction model. While not necessarily realistic, this model often allows for mathematical derivations, and provides a useful starting point for considering the effects of predictions. With this model,

$$g_T(x) = 1 - e^{-(T/x)},$$

and hence

$$\begin{aligned} Q(T) &= \int_0^\infty f(x) g_T(x) dx \\ &= 1 - \int_0^\infty e^{-x-(T/x)} dx \\ &= 1 - 2\sqrt{T} K_1(2\sqrt{T}), \end{aligned}$$

where K_1 is a modified Bessel function of the second kind. Also

$$\rho'(T) = \lambda \int_0^\infty (xe^{-x} - xe^{-x-(T/x)}) dx = \lambda(1 - 2TK_2(2\sqrt{T})),$$

where K_2 is a modified Bessel function of the second kind (with a different parameter).

The expected sojourn time for this case, which we refer to as $S_{e,n,e}$, is then

$$S_{e,n,e} = \frac{\lambda(1 - \lambda(1 - 2\sqrt{T}K_1(2\sqrt{T})))}{(1-\lambda)(1 - \lambda(1 - 2TK_2(2\sqrt{T})))} + 1.$$

There does not appear to be a simple form for the derivative of this expression that allows us to write a simple form for the optimal value fo T, although it can be found numerically.

We can perform similar calculations for our Weibull distribution. Here we have

$$\begin{aligned} Q(T) &= 1 - \int_0^\infty \frac{1}{\sqrt{2x}} e^{-\sqrt{2x}-(T/x)} dx \\ &= 1 - \sqrt{\frac{T}{2\pi}} G^{3,0}_{0,3}\left(\frac{T}{2} \mid -\frac{1}{2}, 0, \frac{1}{2}\right), \end{aligned}$$

where here G is the Meijer G-function.

Similarly

$$\begin{aligned} \rho'(T) &= \lambda \int_0^\infty \sqrt{\frac{x}{2}} \left(e^{-\sqrt{2x}} - e^{-\sqrt{2x}-(T/x)}\right) dx \\ &= \lambda \left(1 - \sqrt{\frac{T^3}{2\pi}} G^{3,0}_{0,3}\left(\frac{T}{2} \mid -\frac{3}{2}, 0, \frac{1}{2}\right)\right), \end{aligned}$$

where again G is the Meijer G-function.

The expected sojourn time for this case, which we refer to as $S_{w,n,e}$, is then

$$S_{w,n,e} = \frac{3\lambda\left(1 - \lambda\left(1 - \sqrt{\frac{T}{2\pi}} G^{3,0}_{0,3}\left(\frac{T}{2} \mid -\frac{1}{2}, 0, \frac{1}{2}\right)\right)\right)}{(1-\lambda)\left(1 - \lambda\left(1 - \sqrt{\frac{T^3}{2\pi}} G^{3,0}_{0,3}\left(\frac{T}{2} \mid -\frac{3}{2}, 0, \frac{1}{2}\right)\right)\right)} + 1.$$

Example: Uniform Predictions As an additional example, we consider the case where the prediction for a job with service time z is uniformly distributed over $[0, 2z]$. (Similar analysis would hold for predictions uniform over $[\alpha z, \beta z]$ for constants $\alpha < \beta$; we choose this specific example for convenience.) With this model,

$$g_T(x) = \min\left(1, \frac{T}{2x}\right),$$

and hence

$$\begin{aligned} Q(T) &= \int_0^\infty f(x) g_T(x) dx \\ &= \int_0^{T/2} e^{-x} dx + \frac{T}{2} \int_{T/2}^\infty \frac{e^{-x}}{x} dx \\ &= 1 - e^{-T/2} + \frac{T}{2} E_1(T/2) \end{aligned}$$

where E_1 is a standard exponential integral. (That is, $E_1(y) = \int_y^\infty \frac{e^{-x}}{x} dx$.) Also

$$\begin{aligned} \rho'(T) &= \lambda \left(\int_0^{T/2} xe^{-x} dx + \frac{T}{2} \int_{T/2}^\infty e^{-x} dx \right) \\ &= \lambda(1 - e^{-T/2}). \end{aligned}$$

The expected sojourn time for this case, which we refer to as $S_{e,n,u}$, is then

$$S_{e,n,u} = \frac{\lambda(1 - \lambda(1 - e^{-T/2} + (T/2)E_1(T/2)))}{(1-\lambda)(1 - \lambda(1 - e^{-T/2}))} + 1.$$

We can perform similar calculations for our Weibull distribution. Here we find

$$Q(T) = 1 - e^{-\sqrt{T}} + \sqrt{T}e^{-\sqrt{T}} - TE_1(\sqrt{T}),$$

and

$$\rho'(T) = \lambda(1 - \sqrt{T}e^{-\sqrt{T}} - e^{-\sqrt{T}}).$$

The expected sojourn time for this case, which we refer to as $S_{w,n,u}$, is then

$$S_{w,n,u} = \frac{3\lambda\left(1 - \lambda\left(1 - e^{-\sqrt{T}} + \sqrt{T}e^{-\sqrt{T}} - TE_1(\sqrt{T})\right)\right)}{(1-\lambda)\left(1 - \lambda(1 - \sqrt{T}e^{-\sqrt{T}} - e^{-\sqrt{T}})\right)} + 1.$$

3.2 The preemptive system For the preemptive system, again let

$$Q(T) = \int_0^\infty f(x)g_T(x)dx,$$

and

$$\rho'(T) = \lambda \int_0^\infty xf(x)g_T(x)dx.$$

We first consider arriving jobs with *predicted* service time at most T and actual service time t. Such jobs will have no waiting time, but their expected sojourn time is

$$S_1'(t) = \frac{t}{1-\rho'(T)}.$$

For jobs with predicted service time greater than or equal to T, following the same reasoning as in the case without predictions, we have

$$W_2'(t) = \frac{V}{(1-\rho)(1-\rho'(T))},$$

and

$$S_2'(t) = \frac{V}{(1-\rho)(1-\rho'(T))} + \frac{t}{1-\rho'(T)}.$$

We therefore find the total expected waiting time is

$$W_p' = \frac{V(1-Q(T))}{(1-\rho)(1-\rho'(T))},$$

and the total expected sojourn time is

$$\begin{aligned} S_p' &= \frac{V(1-Q(T))}{(1-\rho)(1-\rho'(T))} + \frac{1}{1-\rho'(T)} \\ &= \frac{V(1-Q(T)) + 1 - \rho}{(1-\rho)(1-\rho'(T))}. \end{aligned}$$

Example: Exponential Predictions For the exponential prediction model and exponentially distributed service times, the expected sojourn time, which we refer to as $S_{e,p,e}$, is then

$$S_{e,p,e} = \frac{1-\lambda+\lambda 2\sqrt{T}K_1(2\sqrt{T})}{(1-\lambda)(1-\lambda(1-2TK_2(2\sqrt{T})))}.$$

We again here have the relation

$$\lambda S_{e,p,e} = S_{e,n,e} - 1,$$

which implies $S_{e,p,e} < S_{e,n,e}$ as long as $S_{e,p,e} < 1/(1-\lambda)$. More generally, for Poisson arrivals and exponentially distributed service times,

$$\lambda S_p' = S_n' - 1,$$

so as long as predictions are sufficient to improve over not using predictions when using preemption (so that $S_p' < 1/(1-\lambda)$, using preemption will perform better than not using preemption.

For the exponential prediction model and Weibull service times, the expected sojourn time $S_{w,p,e}$ is given by

$$S_{w,p,e} = \frac{1-\lambda+3\lambda\sqrt{\frac{T}{2\pi}}G^{3,0}_{0,3}\left(\frac{T}{2} \mid -\frac{1}{2}, 0, \frac{1}{2}\right)}{(1-\lambda)\left(1-\lambda\left(1-\sqrt{\frac{T^3}{2\pi}}G^{3,0}_{0,3}\left(\frac{T}{2} \mid -\frac{3}{2}, 0, \frac{1}{2}\right)\right)\right)}.$$

Example: Uniform Predictions We can similarly determine the expected sojourn time when the prediction for the service time z is uniform over $[0, 2z]$ for exponentially distributed and Weibull distributed service times, given by

$$S_{e,p,u} = \frac{1-\lambda+\lambda(e^{-T/2} - (T/2)E_1(T/2))}{(1-\lambda)(1-\lambda(1-e^{-T/2}))}$$

and

$$S_{w,p,u} = \frac{1-\lambda+3\lambda\left(e^{-\sqrt{T}} - \sqrt{T}e^{-\sqrt{T}} + TE_1(\sqrt{T})\right)}{(1-\lambda)\left(1-\lambda(1-\sqrt{T}e^{-\sqrt{T}} - e^{-\sqrt{T}})\right)}.$$

4 One-Bit Advice with Multiple Queues

In this section, we consider one-bit threshold schemes for setting with multiple queues. In particular, we consider the "power of d choices" (also known as the "balanced allocations") setting, where we consider the limiting system as the number of queues growing to infinity, and each arrival chooses the best queue from a small constant-sized subset of randomly selected queues [3, 17, 24]. We are motivated by the fact that an advantage here of queues based on one bit of advice is their state can easily represented, allowing the type of mean-field analysis that is typical for such systems. We note that analysis of the power of d choices with for example exact job sizes using queueing schemes such as Shortest Remaining Processing Time remains an intriguing open question (see e.g. [19] for more discussion), although simply using FIFO queues and choosing the least loaded queue from a constant number of choices has recently been analyzed [10, 11].

As our purpose here is primarily to demonstrate how schemes utilizing one bit can be analyzed in this framework, we choose a relatively simple example, based on the anecdotal 80-20 rule, that 20% of the jobs cause 80% of the work. We assume that there are two types of jobs: long jobs have exponentially distributed service times with mean μ_1, and short jobs have exponentially distributed service times with mean $\mu_2 < \mu_1$. Long jobs arrive as a Poisson arrival process with rate $\lambda_1 n$ and

short jobs arrive as a Poisson arrival process with rate $\lambda_2 n$, where n is the number of queues in the system. While this model is general, it can encompass the 80-20 rule, where long jobs are less frequent but require much more work; for example, if $\lambda_1\mu_1 = 4\lambda_2\mu_2$, then long jobs are relatively rare but contribute 80% of the work.

In the prediction setting, we assume long jobs are misclassified as short jobs independently with probability q_1, and short jobs are misclassified as long jobs independently with probability q_2. (We may view the case without predictions, where the one bit of advice is accurate, as corresponding to $q_1 = q_2 = 0$, with the resulting equations.) We say that a job is labeled long if it is classified as a long job, and similarly use the term labeled short for jobs classified as short jobs. A more useful interpretation for our analysis is that a job that is labeled long is actually long with probability $p_L = \lambda_1(1-q_1)/(\lambda_1(1-q_1)+\lambda_2 q_2)$, and similarly a job that is labeled short is actually short with probability $p_S = \lambda_2(1-q_2)/(\lambda_2(1-q_2)+\lambda_1 q_1)$. Similarly, the arrival rate for jobs that labeled long is $\lambda_L = \lambda_1(1-q_1)+\lambda_2 q_2$ and the arrival rate for jobs that are labeled short is $\lambda_S = \lambda_2(1-q_2)+\lambda_1 q_1$.

For serving jobs, we give labeled short jobs priority, and serve them in FIFO fashion; similarly, labeled long jobs are served using FIFO. We suggest a simple, convenient method for choosing queues based on simple counts of labeled jobs, although other variations are possible and can be studied similarly. We choose the "best" of d queues chosen independently and uniformly at random for a constant d, where we determine the best as follows. First, an empty queue has highest priority; an empty queue will always be selected if it is one of the d chosen. Otherwise, we ignore the label and the time already spent being served of the job being served. Jobs that are predicted to be short shall choose the queue with the fewest queued labeled short jobs, breaking ties in favor of the queue with the fewest labeled long jobs (and then randomly if two queues match), and similarly jobs that are predicted to be long shall choose the queue with the smallest number of queued labeled long jobs, breaking ties in favor of the queue with fewer short jobs (and then randomly if two queues match). Again, one could imagine somewhat more complex policies based on minimizing the expected time until service; such policies can be studied using the same framework.

We derive equations describing the system state in the mean field limit, where the number of queues goes to infinity. (This approach can be formalized using the theory of density dependent jump Markov chains, following the work of Kurtz; see [7, 15, 26] for examples.) The state of a single queue can be represented by a triple (s,ℓ,c), where s is the number of jobs that are labeled short, ℓ is the number of jobs that are labeled long, and c is 1 if the current running job is long and 2 if it is short. The state $(0,0,0)$ is used for an empty queue; this is the only state where $c \neq 1,2$. Let $x_{(s,\ell,c)}(t)$ represent the fraction of queues in state (s,ℓ,c) at time t; we drop the t where the meaning is clear. We use $\hat{x}(s,\ell,c)$ to refer to the equilibrium values for these quantities.

Note that our setting allows a relatively simple analysis by having service times be exponentially distributed and predictions depend only on the type of the job, instead of its running time. Because of this, to keep the state of a queue it suffices to keep the type of the running job, as this gives the distribution of the remaining time it is in service. This approach can be extended to more general service times and predictions; see e.g. [1, 10, 11], for example, for the appropriate framework. At a high level, for such generalizations, the state of the queue must track how long the current running job has been in the system; the distributions function for remaining time in service, which is derived by taking the proper weighted average over types, then determines whether the service will complete over the next time interval dt.

To write the equations describing the limiting behavior of these systems, we use some additional notation. Let $z_{(2,s,\ell)}$ be the fraction of queues with lower priority over a queue with s queued labeled short jobs and ℓ queue labeled large jobs when a job labeled short arrives (in terms of being chosen by our algorithm), and similarly define $z_{(1,s,\ell)}$ for when a job labeled long arrives. Here again we drop the implicit dependence on t. These z values can be readily computed by dynamic programming or even brute force given the $x_{(s,\ell,c)}$. For $c = 1,2$ and $c' = 1,2$, let $w_{(c',s,\ell,c)}$ be

$$\left(\left(z_{(c',s,\ell)} + x_{(s,\ell,1)} + x_{(s,\ell,2)}\right)^d - \left(z_{(c',s,\ell)}\right)^d\right) \frac{x_{(s,\ell,c)}}{x_{(s,\ell,1)} + x_{(s,\ell,2)}}.$$

Then $w_{(c',s,\ell,c)}$ gives the probability that an incoming job labeled c' chooses a queue in state (s,ℓ,c). For the empty queue, we have the special case

$$w_{(c',0,0,0)} = 1 - \left(1 - x_{(0,0,0)}\right)^d,$$

and is it convenient to let $w_{(c',s,\ell,c)} = 0$ if $s < 0$ or $\ell < 0$.

The limiting mean field equations when $s > 0$ are then

$$\begin{aligned}
\frac{dx_{(s,\ell,1)}}{dt} &= \lambda_S w_{(2,s-1,\ell,1)} + \lambda_L w_{(1,s,\ell-1,1)} \\
&\quad + \mu_1 x_{(s+1,\ell,1)}(1-p_S) + \mu_2 x_{(s+1,\ell,2)}(1-p_S) \\
&\quad - \left(\mu_1 x_{(s,\ell,1)} + \lambda_S w_{(2,s,\ell,1)} + \lambda_L w_{(1,s,\ell,1)}\right); \\
\frac{dx_{(s,\ell,2)}}{dt} &= \lambda_S w_{(2,s-1,\ell,2)} \\
&\quad + \lambda_L w_{(1,s,\ell-1,2)} + \mu_1 x_{(s+1,\ell,1)} p_S + \mu_2 x_{(s+1,\ell,2)} p_S \\
&\quad - \left(\mu_2 x_{(s,\ell,2)} + \lambda_S w_{(2,s,\ell,2)} + \lambda_L w_{(1,s,\ell,2)}\right).
\end{aligned}$$

The cases where $s = 0$ and $\ell > 0$ are given by

$$\begin{aligned}\frac{dx_{(0,\ell,1)}}{dt} &= \lambda_L w_{(1,0,\ell-1,1)} + \mu_1 x_{(0,\ell+1,1)} p_L + \mu_2 x_{(0,\ell+1,2)} p_L \\ &\quad + \mu_1 x_{(1,\ell,1)}(1-p_S) + \mu_2 x_{(1,\ell,2)}(1-p_S) \\ &\quad - \left(\mu_1 x_{(0,\ell,1)} + +\lambda_S w_{(2,0,\ell,1)} + \lambda_L w_{(1,0,\ell,1)}\right); \\ \frac{dx_{(0,\ell,2)}}{dt} &= \lambda_L w_{(1,0,l-1,2)} + \mu_1 x_{(1,\ell,1)} p_S + \mu_2 x_{(1,\ell,2)} p_S \\ &\quad + \mu_1 x_{(0,\ell+1,1)}(1-p_L) + \mu_2 x_{(0,\ell+1,2)}(1-p_L) \\ &\quad - \left(\mu_2 x_{(0,\ell,2)} + +\lambda_S w_{(2,0,\ell,2)} + \lambda_L w_{(1,0,\ell,2)}\right).\end{aligned}$$

For queues without any waiting jobs, we have

$$\begin{aligned}\frac{dx_{(0,0,1)}}{dt} &= \lambda_L p_L w_{(1,0,0,0)} + \lambda_S (1-p_S) w_{(1,0,0,0)} \\ &\quad + \mu_1 x_{(0,1,1)} p_L + \mu_2 x_{(0,1,2)} p_L \\ &\quad + \mu_1 x_{(1,0,1)}(1-p_S) + \mu_2 x_{(1,0,2)}(1-p_S) \\ &\quad - \left(\mu_1 x_{0,0,1} + \lambda_S w_{(2,0,0,1)} + \lambda_L w_{(1,0,0,1)}\right); \\ \frac{dx_{(0,0,2)}}{dt} &= \lambda_S p_S w_{(2,0,0,0)} + \lambda_L (1-p_L) w_{(2,0,0,0)} \\ &\quad + \mu_1 x_{(0,1,1)}(1-p_L) + \mu_2 x_{(0,1,2)}(1-p_L) \\ &\quad + \mu_1 x_{(1,0,1)} p_S + \mu_2 x_{(1,0,2)} p_S \\ &\quad - \left(\mu_1 x_{0,0,2} + \lambda_S w_{(2,0,0,2)} + \lambda_L w_{(1,0,0,2)}\right); \\ \frac{dx_{(0,0,0)}}{dt} &= \mu_1 x_{(0,0,1)} + \mu_2 x_{(0,0,2)} \\ &\quad - \left(\lambda_S w_{(2,0,0,0)} + \lambda_L w_{(1,0,0,0)}\right).\end{aligned}$$

These differential equations describe the limiting behavior as the number of queues go to infinity. Because of the complexity of the equations, we do not aim to prove here that the equations converge to a fixed point, although this variation is similar to many previous models and the formalization of the mean field analysis appears standard (see, e.g., [5, 8, 10, 11, 7, 15, 17]).

In section 5 below, we compare the results from calculating the differential equations results numerically with simulations.

5 Simulations and Numerical Results

5.1 Single Queues In the simulations for single queues, each data point is obtained by simulating initially empty queues over 1000000 units of time, and taking the average response time for all jobs that terminate after time 100000. We then take the average of over 100 simulations. Waiting for the first 10% allows the system to approach the stationary distribution, and we run for sufficient time that recording only completed jobs has small influence.

Before presenting results, we emphasize that we have checked the experimental results for single queues against the equations we have derived in sections 2 and 3. They match very closely; in general terms, nearly all the averaged simulation results presented are within 1% of the values derived from the equations. (Individual simulation runs can vary more significantly; the maximum and minimum times over our trials vary by 5-10% for exponentially distributed service times, and by 10-20% for Weibull service times.) As such, we do not present further results comparing equations to simulations here.[1]

Our first simulations are for exponentially distributed service times. While we have done more simulations at various arrival rates, we present results for $\lambda = 0.8, 0.9$, and 0.95. This focuses on the more interesting case of reasonably high arrival rates, while keeping the values within a reasonable range for presentation. As a baseline, when the arrival rate is λ, the expected time a job spends in such a system in equilibrium with a FIFO queue is $1/(1-\lambda)$.

We first show the results of the experiments, comparing the results with and without preemption, and with and without prediction. Figure 1 shows the results as the threshold varies given correct one-bit advice, and Figure 2 shows the results under the exponential prediction model. The figures are at the same scale so results can be compared. We see here that preemption, as expected, provides some gains, and the cost for using prediction is not too large. In particular, one bit of even sometimes incorrect advice substantially reduces the average time in system over simple FIFO queueing. The results show that in this setting choosing a threshold near the optimal rather than the optimal does not substantially affect the results.

We also do experiments for the Weibull distribution with cumulative distribution $1 - e^{-\sqrt{2x}}$. As a baseline, when the arrival rate is λ, the expected time a job spends in such a system in equilibrium with a FIFO queue under this Weibull distribution is $(1+\lambda)/(1-\lambda)$.

Similar to before, Figure 3 shows the results as the threshold varies given correct one-bit advice, and Figure 4 shows the results under the exponential prediction model. Note that the range of thresholds is much larger. One would expect larger thresholds would be optimal for a heavy-tailed distribution, as the downside of having a very large job at the front of the queue is more substantial. Further, in this setting, preemption offers more substantial gains, and clearly pushes the optimal threshold to larger values, as preemption significantly reduces the impact of a long job holding the queue. While the cost of using predicted advice over optimal advice is larger, the potential (and actual) gains from

[1] While arguably we could have simply trusted the equations, we find verifying results via simulation worthwhile.

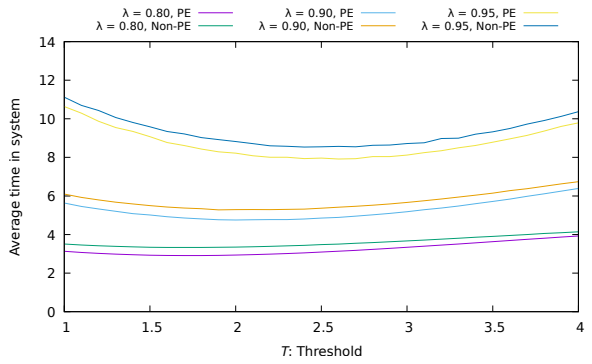


Figure 1: Performance under threshold schemes with exact information, exponentially distributed service times.

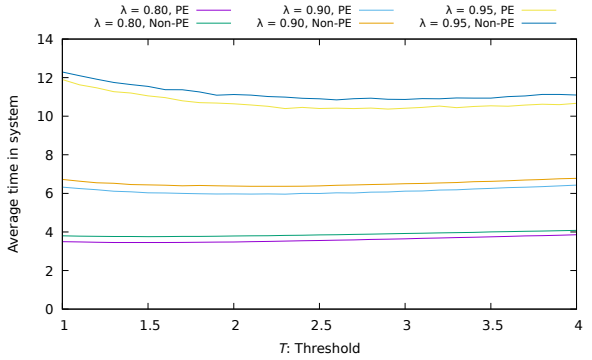


Figure 2: Performance under threshold schemes with predicted information, exponentially distributed service times, exponential predictions.

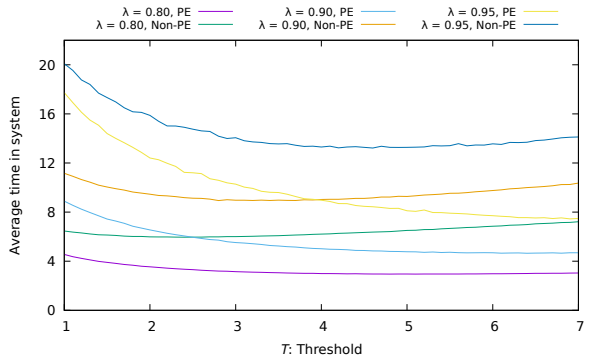


Figure 3: Performance under threshold schemes with exact information, Weibull distributed service times.

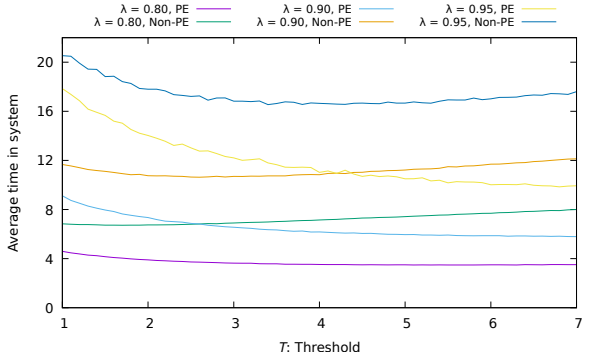


Figure 4: Performance under threshold schemes with predicted information, Weibull distributed service times, exponential predictions.

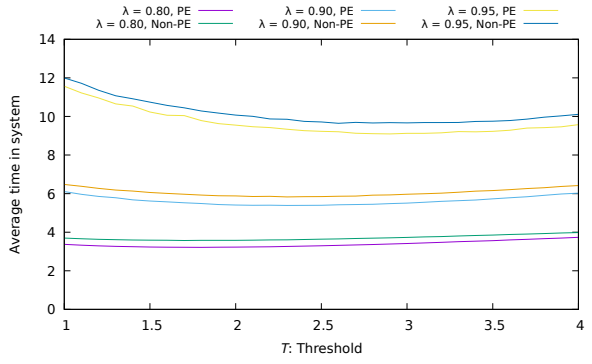


Figure 5: Performance under threshold schemes with predicted information, exponentially distributed service times, predictions uniform over $[0, 2z]$.

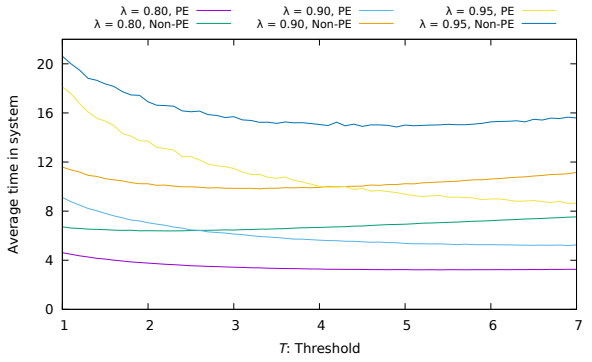


Figure 6: Performance under threshold schemes with predicted information, Weibull distributed service times, predictions uniform over $[0, 2z]$.

λ	FIFO	THRESHOLD NO PREEMPT	THRESHOLD PREEMPT	SRPT	PREDICTION NO PREEMPT	PREDICTION PREEMPT	SPRPT
0.50	2.000	1.783	1.564	1.425	1.850	1.698	1.659
0.60	2.500	2.089	1.814	1.604	2.209	2.013	1.940
0.70	3.333	2.542	2.203	1.875	2.761	2.517	2.369
0.80	5.000	3.329	2.910	2.355	3.757	3.451	3.143
0.90	10.00	5.278	4.755	3.552	6.366	5.960	5.097
0.95	20.00	8.535	7.914	5.532	10.848	10.372	8.424
0.98	50.00	16.495	15.735	10.436	22.418	21.909	16.696

Table 1: Results for exponentially distributed service times. Prediction results are using exponential predictions.

	FIFO	THRESHOLD NO PREEMPT	THRESHOLD PREEMPT	SRPT	PREDICTION NO PREEMPT	PREDICTION PREEMPT	SPRPT
0.50	4.000	3.012	1.608	1.411	3.155	1.736	1.940
0.60	5.500	3.676	1.867	1.574	3.918	2.062	2.280
0.70	8.000	4.565	2.258	1.813	4.983	2.568	2.750
0.80	13.00	5.955	2.951	2.217	6.721	3.481	3.519
0.90	29.00	8.940	4.649	3.154	10.630	5.790	5.224
0.95	58.00	13.223	7.448	4.517	16.546	9.846	7.788
0.98	148.0	22.451	15.194	7.666	29.346	20.918	13.404

Table 2: Results for Weibull distributed service times. Prediction results are using exponential predictions.

using predicted advice are even more substantial.

We show similar figures for the case where the prediction for a job with service time z is uniformly distributed over $[0, 2z]$ in Figures 5 and Figures 6. The results follow the same trends at a high level.

To shed additional light on our experiments for a single queue, we compare one-bit advice to no advice, in which case the queue uses FIFO, and full knowledge of the processing times, in which case the queue uses Shortest Remaining Processing Time (SRPT). In the case of predictions, we consider the exponential prediction model, comparing our schemes against FIFO and Shortest Predicted Remaining Processing Time (SPRPT) [18], which uses SRPT scheduling on the predicted times. For FIFO results we simply use the standard formula for expected time in the system; we could similarly use formulae for the other results, but present results from simulations. In particular, for our one-bit schemes, we choose the best threshold from simulation results.

As we can see in Tables 1 and 2, one-bit schemes greatly improve on FIFO across the board, with a greater improvement for the Weibull distribution, as one would expect. We also find that preemption is helpful, again moreso for the Weibull distribution, as one would expect. Perhaps somewhat surprisingly, for some of results one-bit predictions with preemption perform better than SPRPT in the Weibull distribution. This appears to be due to a known issue with SPRPT, namely that a large job with a small predicted service time can obtain a predicted remaining processing time of zero, and become a bottleneck. Previous work has shown that preemptive shortest predicted job first (PSPJF) can sometimes outperform SPRPT for the same reason [18, 19].

The main takeaways from these results are that one-bit threshold schemes can achieve a large fraction of the benefit that would arise from full knowledge of processing times, and even simple one-bit predictions can achieve a large fraction of the benefit that would arise from more detailed predictions. While these results are specifically for Poisson arrivals as well as specific service distributions and prediction models, they suggest generally that there is significant potential for even simple predictions to provide large value in scheduling in practice.

5.2 Multiple Queues We present results here for an example using the power of two choices, to demonstrate the results from differential equations match simulations, and to show the effectiveness of working with predictions. (As there is a large parameter space to choose from here, and this model is already rather idealized, we do not attempt larger-scale simulations; rather, we focus on the high level issues, such as the importance of predicting long jobs correctly.)

	Simulations	Diff. Eqns.
1 Choice	24.208	– – –
SRPT	2.366	– – –
Shorter Queue, FIFO	4.967	– – –
Pred 0.0, 0.0	3.394	3.392
Pred 0.1, 0.1	3.690	3.688
Pred 0.2, 0.2	4.010	4.007
Pred 0.3, 0.3	4.353	4.347
Pred 0.4, 0.4	4.717	4.711
Pred 0.5, 0.5	5.105	5.098
Pred 0.2, 0.4	4.280	4.276
Pred 0.4, 0.2	4.402	4.395
Pred 0.11, 0.61	4.617	4.611

Table 3: Results for queueing systems with 1000 queues, 2 choices and baseline comparisons.

In our example we follow the 80-20 rule; we choose parameters $\lambda_1 = 0.225$, $\mu_1 = 3.2$, $\lambda_2 = 0.90$, and $\mu_2 = 0.20$. The overall load on the system is therefore 0.9. For simulations, each data point is obtained by simulating systems of 1000 (initially empty) queues over 100000 units of time, and taking the average response time for all jobs that terminate after time 10000. We then take the average of over 100 simulations. For the differential equations, we simply used Euler's method over times steps of 10^{-5} over time 10^4. (This provides an accurate calculation for the "fixed point", or stationary distribution corresponding to the solution of these equations.) All experiments use two choices. We provide simulation results for randomly choosing a single queue and using FIFO, choosing a queue based on the least loaded and using SRPT within the queue, and choosing the shorter of two queues and FIFO processing for comparison. We provide simulation results for various predictions, where the two values after "Pred" in the table are q_1 (misprediction for long jobs) and q_2 (short jobs), respectively.

The main takeaways from Table 3, beyond the fact that the differential equations are quite accurate (less than 0.2% difference in these examples), are that one-bit predictions can provide benefits over the already excellent performance of choosing the shorter queue; even when all predictions are only 60% accurate, we see some gain in performance. Considering $q_1 = 0.2$ and $q_2 = 0.4$ along with $q_1 = 0.4$ and $q_2 = 0.2$ shows that predictions for long jobs are more important, even though there are much fewer long jobs. The effect of giving a long job higher priority, where it can block short jobs, has a more prominent effect than misclassifying a short job. This demonstrates that the goal of a machine learning algorithm in this setting should not be simply to maximize the number of correct predictions; a machine learning algorithm can do better by predicting the long jobs well. (See [19] for a similar discussion.)

As an extreme example of this, choosing $q_1 = 0.11$ and $q_2 = 0.61$ leads to a total error rate of 51% over all jobs, as short jobs have a much higher arrival rate than long jobs. But even though more than half the jobs types are predicted incorrectly, because long jobs are predicted correctly most of the time, such predictions still perform notably better than not using predictions and just choosing the shorter queue.

6 Conclusions

We have looked at the setting of queueing systems with one bit of advice, where a primary motivation is the potential for machine learning algorithms to provide simple but useful predictions to improve scheduling. In the case of single queues, we see that a natural probabilistic model for predictions leads to relatively straightforward equations that can be used to determine where one would ideally choose a threshold to separate long and short jobs. For large-scale queueing systems, where the power of two choices can be used, we have shown that one-bit prediction can allow for fluid limit analysis. We view this as a potential step forward for the interesting open problem of determining the behavior of systems using the power of two choices with scheduling via shortest remaining processing time or other scheduling schemes based on the service time.

We believe there remain several interesting directions to explore in this space. The use of predictions in more complex settings, such as call centers, may provide significant value. A challenging underlying questions, when "jobs" may correspond to people, is how to define appropriate notions of fairness, so that jobs that are mispredicted by a machine learning algorithm do not suffer overly from the algorithm's behavior.

References

[1] Reza Aghajani, Xingjie Li, and Kavita Ramanan. Mean-field dynamics of load-balancing networks with general service distributions. *arXiv preprint arXiv:1512.05056*, 2015.

[2] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ML Predictions For Online Algorithms. International Conference on Machine Learning, 2020.

[3] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999.

[4] Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online

algorithms with advice: a survey. *ACM SIGACT News*, 47(3):93–129, 2016.

[5] Maury Bramson, Yi Lu, and Balaji Prabhakar. Randomized load balancing with general service time distributions. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 275–286. ACM, 2010.

[6] Matteo Dell'Amico, Damiano Carra, and Pietro Michiardi. PSBS: Practical size-based scheduling. *IEEE Transactions on Computers*, 65(7):2199-2212, 2015.

[7] S. N. Ethier and T. G. Kurtz. *Markov Processes: Characterization and Convergence*. John Wiley and Sons, 1986.

[8] Nicolas Gast and Benny Van Houdt. On the power-of-*d*-choices with least loaded server selection. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2), pp. 1-28, 2017.

[9] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[10] Tim Hellemans and Benny Van Houdt. On the power-of-*d*-choices with least loaded server selection. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):27:1–27:22, 2018.

[11] Tim Hellemans, Tejas Bodas, and Benny Van Houdt. Performance analysis of workload dependent load balancing policies. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):35:1–35:35, 2019.

[12] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-Based Frequency Estimation Algorithms. In *7th International Conference on Learning Representations*, 2019.

[13] Leonard Kleinrock. A conservation law for a wide class of queueing disciplines. *Naval Research Logistics Quarterly*, 12.2, pp. 181-192, 1965.

[14] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[15] T. G. Kurtz. Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes. *Journal of Applied Probability*, Vol. 7, 1970, pp. 49-58.

[16] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 3302–3311, 2018.

[17] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12(10):1094–1104, 2001.

[18] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. Im *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, 14:1-14:18, 2020.

[19] Michael Mitzenmacher. The Supermarket Model with Known and Predicted Service Times. *arXiv preprint arXiv:1905.12155*, 2019.

[20] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with Predictions. In *Beyond the Worst-Case Analysis of Algorithms*, edited by Tim Roughgarden, Cambridge University Press, 2020.

[21] T.M. O'Donovan. Direct solutions of M/G/1 priority queueing models. *Revue française d'automatique, informatique, recherche opérationnelle*, 10.V1, 107-111, 1976.

[22] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*, pages 9684–9693, 2018.

[23] Linus Schrage. Letter to the editor—An alternative proof of a conservation law for the queue G/G/1. *Operations Research*, 18(1), pp.185-187.

[24] Nikita Dmitrievna Vvedenskaya, Roland L'vovich Dobrushin, and Fridrikh Izrailevich Karpelevich. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii*, 32(1):20–34, 1996.

[25] Adam Wierman and Misja Nuyens. Scheduling despite inexact job-size information. *Performance Evaluation Review*, 36(1):25-36, 2008.

[26] N.C. Wormald. Differential equations for random processes and random graphs. *The Annals of Applied Probability*, 5(1995), pp. 1217–1235.

The Quantile Matching Problem and Point Cloud Registration

Stéphane Chrétien* Oya Ekin Karaşan† Ecenur Oğuz‡ Mustafa Ç. Pınar§

Abstract

One of the fundamental problems in computer vision is the matching of two point clouds. For the case when the two point clouds do not match exactly we introduce a new approach based on quantile matching using curvature information. We define the quantile matching problem on a bipartite graph, the two parts of which represent two point clouds. The goal is to achieve an optimal registration of a point cloud with another point cloud. The problem is posed as the problem of computing a (perfect when possible) matching, which maximizes the α-quantile of affinity weights between the nodes of the graph. We prove that the problem is polynomially solvable in bipartite and non-bipartite graphs. Numerical illustrations are given. Implementations of the proposed algorithms in Python are described along with computational results with synthetic as well as real data from an optical coherence tomography application.

1 Introduction

The assignment problem is one of the oldest and most revered problems of combinatorial optimization as its history goes back at least to 19th century illustrious German mathematician Jacobi, c.f. [7]. It is polynomially solvable, see e.g., [1], and possesses several variants as discussed in [7]. We add yet another variant to this repertoire. We study a new problem called "quantile bipartite perfect matching" or simply "quantile assignment problem" (although we also cover the non-perfect matching case and non-bipartite graphs), which consists of maximizing the α-quantile of the affinity weights between nodes (the affinity weights serve as the entries of the matrix in the objective function of the assignment problem as we shall see below), a problem arising in a computer vision application [3]. We prefer to describe the problem on bipartite graphs as it is easier to do so (hence the term "the quantile assignment problem"), and prove that it is polynomially solvable using a 1978 matching algorithm given in [11], both on bipartite and non-bipartite graphs.

*University of Lyon II.
†Bilkent University.
‡Bilkent University.
§Bilkent University.

The rest of the paper is organized as follows. We describe briefly the computer vision application namely point cloud registration in section 2. Our contributions begin in section 2.1 where a description of the proposed method to point cloud registration is formulated as a problem of quantile matching in bipartite graphs is given. In section 3 we prove polynomial solvability of the problem while section 4 describes the numerical algorithms. Section 4 also contains an integer linear programming formulation in subsection 4.2, and a few simple numerical examples in subsection 4.3. In subsection 4.4, we give numerical results with implementations of the proposed algorithms in a Python environment on synthetic and real data. A full description of the computer vision application, which is in preparation, will be described in a subsequent and longer journal version. Section 5 describes the extension to the bipartite case where the two parts of the graph do not have equal node cardinality while section 6 summarizes the extension to the non-bipartite case.

2 Background on Point Cloud Registration.

In computer vision, point cloud registration or point set registration is a technique that consists in computing a transformation which aligns two sets of points lying in a two- or three-dimensional space. An example is obtained by considering the availability of two-dimensional images of a three dimensional scene. After obtaining two-dimensional coordinates of images, finding a transformation (e.g., rotation, scaling, translation) that maps one image to another allows to re-enact a three-dimensional representation of the scene. As a very elementary illustration, assume we have a three-dimensional object. Performing an orthogonal projection from the x-axis onto the yz-plane one obtains two-dimensional coordinates. From the z-axis another orthogonal projection to the xy-plane gives coordinates in the xy-plane. Merging these coordinates gives the 3D coordinates of the object. Obviously one obtains the same y-coordinates in the example. In a realistic application, these y-coordinates do not match since the transformation that maps coordinates between two images is different from orthogonal projection. A central problem of computer vision is to figure out precisely the transformation that maps one image to the other.

In contrast to existing approaches for registration in the literature (see e.g., [9] for a recent reference and application), in many industrial and medical applications the two point clouds at hand only correspond on restricted subregions, and these subregions are hard to specify precisely by hand. As a result, not all points from the first point cloud should necessarily be matched with another point in the target point cloud. In [3], a new approach to the problem of aligning two point clouds based on how well they can fit on the same manifold after transformation was proposed. In order to keep the problem simple, the study was restricted to the case where the transformation is assumed to be affine, which allows for anisotropic scaling, rotations and translations. The method introduced in [3] takes into account the following complicating factors:

- we do not have exact point matching
- there could be systematic errors
- there is random noise.

The main idea is that one should associate with each point in the cloud a number or a vector which does not change with the set of possible transformations that the point cloud may have to go through in order to achieve an optimal registration with another point cloud. Curvature is a quantity that is preserved under rigid transformation. Histogram shapes of local curvature for K-nearest neighbors is invariant under e.g., affine transformations. Curvature computation for point clouds is performed as follows. For each point x_i, $i = 1, \ldots, n$ in the point cloud, let M_i be the associated unit normal vector. Fix one point $P = x_{i_0}$ and let $N = M_{i_0}$ denote its normal vector. Let another point $Q_i = x_i$ be chosen in a close neighborhood of P. The normal curvature τ_i can be estimated at any point by the formula

$$\tau_i = \frac{\sin(\beta)}{|PQ_i| \cos(\alpha)}$$

where α denotes the angle between $-N$ and PQ_i, and β represents the angle between N and M_i (see [14] for details). Given the values of the curvatures at each point, one can match the points using bipartite matching, a well known problem in graph theory and combinatorial optimization. In order to address this problem, the curvature matching approach based on subpatches of one of the pointclouds is used. This subpatch will be optimally matched with the corresponding region in the overlapping domain of the two point clouds. In order to proceed, one uses a representation of the matching problem as a purely combinatorial optimization problem known as bipartite matching. Let us consider K patches from point cloud X_1, denoted by $X_1^{(1)}, \ldots, X_1^{(K)}$, respectively corresponding to the index sets $I_1^{(1)}, \ldots, I_1^{(K)}$, e.g., obtained by sampling K points from X_1 and for each of them, searching their L nearest neighbors. Given the value of the curvature at every point in the local patch $X_1^{(k)}$ we can match these points to the points in the second point cloud using the rectangular affinity coefficients $A_{ii'}$ defined as

$$A_{ii'} = \exp(-\gamma(\tau_i - \tau_{i'})^2)$$

for $i \in I_1^{(k)}, k = 1, \ldots, K$, and a suitable constant γ. The affinity coefficients $A_{ii'}$ constitute the entries of the affinity matrix that is the basis of the quantile matching approach. Define the binary decision variable $X_{ii'}$ to be equal to one if i and i' are matched, we have the following matching constraints:

$$\sum_{i=1}^{|I_1^{(k)}|} X_{ii'} = 1, \forall i' = 1, \ldots, n_2, \sum_{i'=1}^{n_2} X_{ii'} = 1, \forall i \in I_1^{(k)},$$

where n_2 is the number of points in the second cloud. Recall that, in our problem, the point clouds do not match exactly, and might only have partial overlap. The approach considered in the present work consists in finding adaptively a subset of points which can be matched safely and help avoiding automatically the areas that do not belong to the intersection of the observed areas in the two samples. In the next section, we present the approach based on quantile matching.

2.1 The Quantile Bipartite Perfect Matching Problem. Given a random variable X taking values in $\mathbb{R}$, the α-quantile is the quantity

$$q_\alpha := \inf \{q \mid \mathbb{P}(X \leq q) \geq \alpha\}. \tag{2.1}$$

When the measure $\mathbb{P}$ puts a weight $\frac{1}{N}$ over N real values x_n, $n = 1, \ldots, N$, i.e. $\mathbb{P}$ is the empirical measure associated to the sample set $\{x_1, \ldots, x_N\}$, the probability is replaced with an average and one obtains

$$q_\alpha(\{x_n,\ n = 1, \ldots, N\}) := \inf \left\{ q \mid \frac{1}{N} \sum_{n=1}^{N} I_{x_n \leq q} \geq \alpha \right\}. \tag{2.2}$$

Quantiles are well-studied in statistics and in finance. An authoritative account of quantile regression can be found in [6] while one can consult [4] for its uses in mathematical finance.

For simplicity, let $\mathcal{N} = \{1, \ldots, N\}$. The standard maximum weight assignment problem or equivalently the maximum weight bipartite perfect matching problem in graph $G = (S \cup T, \mathcal{A})$ where $S = \{s_1, \ldots, s_N\}$,

$T = \{t_1, \ldots, t_N\}$ and $\mathcal{A} = \{(s_m, t_n) : m, n \in \mathcal{N}\}$ consists in solving

$$\begin{aligned} \max \quad & \sum_{m=1}^{N}\sum_{n=1}^{N} A_{mn}X_{mn} & \\ \text{s.t.} \quad & \sum_{m=1}^{N} X_{mn} = 1, & n \in \mathcal{N} \\ & \sum_{n=1}^{N} X_{mn} = 1, & m \in \mathcal{N} \\ & X \in \{0,1\}^{N\times N} & \end{aligned}$$

where $A = [A_{mn}]$ is an affinity matrix that quantifies how close m is to n, i.e., the correspondng weight on arc (s_m, t_n) in G, for $m, n \in \mathcal{N}$. As is well known (see c.f. [7, 8]), the following relaxation defined over the polytope of doubly stochastic matrices is exact for this problem

$$\begin{aligned} \max \quad & \sum_{m=1}^{N}\sum_{n=1}^{N} A_{mn}X_{mn} & \\ \text{s.t.} \quad & \sum_{m=1}^{N} X_{mn} = 1, & n \in \mathcal{N} \\ & \sum_{n=1}^{N} X_{mn} = 1, & m \in \mathcal{N} \\ & X_{mn} \geq 0 \text{ for } m, n \in \mathcal{N}. & \end{aligned}$$

Given an affinity matrix A, let $\mathcal{X} = \{X \in \{0,1\}^{N\times N} : \sum_{m=1}^{N} X_{mn} = 1, n \in \mathcal{N}$ and $\sum_{n=1}^{N} X_{mn} = 1, m \in \mathcal{N}\}$ be the set of all perfect matchings, i.e., the set of permutation matrices.

The goal of the present study is to solve the problem of maximizing the α-quantile of the weights associated with a bipartite perfect matching problem, i.e, solve the following optimization problem

$$\max_{X\in\mathcal{X}} \quad q_\alpha\left(\{A_{mn}X_{mn},\ m, n \in \mathcal{N}\}\right). \tag{2.3}$$

We shall establish below that the problem is solvable in polynomial time by a suitable algorithm. Based on this development, we shall also give an integer linear programming formulation for completeness.

3 Complexity Status of the Quantile Bipartite Perfect Matching Problem.

LEMMA 3.1. *Given $X \in \mathcal{X}$ and $\alpha \in [0,1]$, $q_\alpha\left(\{A_{mn}X_{mn},\ m, n \in \mathcal{N}\}\right)$ is the k_α^{th} smallest affinity value in $\{A_{mn} : X_{mn} = 1,\ m, n \in \mathcal{N}\}$ where*

$$k_\alpha = \max(0, \lceil (N - (1-\alpha)N^2 \rceil).$$

Proof. In any perfect matching $X \in \mathcal{X}$, only at most N elements, namely the matched ones, can take positive values in set $\{A_{mn}X_{mn},\ m, n \in \mathcal{N}\}$. Thus, for any $q \geq 0$, we have $N^2 - N$ unmatched elements having values vacuously less than or equal to q. Out of the matched ones, if we guarantee that there are at least k where

$$\frac{N^2 - N + k}{N^2} \geq \alpha$$

with values less or equal to q, then we shall have $\frac{1}{N^2}\sum_{m=1}^{N}\sum_{n=1}^{N} I_{A_{mn}X_{mn}\leq q} \geq \alpha$.

Minimizing the q value while finding the α-quantile is equivalent with choosing the minimum number of small valued matched edges that would suffice for the minimum average requirement. Ultimately, given a matching and an α value, the k_α^{th} smallest affinity value in the matched edge set is the α-quantile value of this matching. Note that when α is small, $k_\alpha = 0$ and no matched elements are necessary to reach the minimum average requirement. ∎

With the optimization problem in (2.3), out of all perfect matchings, we seek to find the one giving the highest α-quantile value for a specific α value. Since the α-quantile value is the k_α^{th} smallest affinity value in a matching, a perfect matching between the rows and columns of the affinity matrix that gives the maximum k_α^{th} smallest entry must be found to solve the problem. Ultimately, the highest possible value in (2.3) will be one of the distinct N^2 values in the affinity matrix.

LEMMA 3.2. *Given α, q and A, deciding whether there exists a perfect matching such that the k_α^{th} smallest entry in this matching is at least q can be done in $O(N^{2.5})$ time.*

Proof. It has been shown in [11] that given a bipartite graph, finding a perfect cardinality matching (assuming all entries in the affinity matrix have value equal to one) with the additional restriction that no more than a given number of edges from a specified subset are in the matching can be done in $O(N^3)$ time. We can adapt their algorithm to solve the problem at hand. Let the subset of edges be those having affinity values strictly less than the given q value, and the restriction be that no more than $k_\alpha - 1$ edges should be used from this set. Then, any perfect cardinality matching in this graph will correspond to a perfect matching in the underlying weighted bipartite graph with the k_α^{th} smallest entry at least as large as the given q value. This bound can further be improved by eliminating the

edges with affinity values less than q from the bipartite graph and solving for a maximum cardinality matching via Hopcropt Karp algorithm [10]. If the number of edges to complement the matching to a perfect one is no more than $k_\alpha - 1$, then the answer is yes. Since the complexity of the Hopcropt-Karp maximum cardinality finding algorithm is $O((\#edges + \#nodes)\sqrt{\#nodes})$ [10], our result follows. □

Now, our main result simply follows from a binary search conducted on possible q values, i.e., the distinct entries in the affinity matrix.

THEOREM 3.1. *Finding a perfect matching that maximizes the α-quantile can be done in $O(N^{2.5} \log N)$ time.*

Proof. Sorting N^2 elements can be done in $O(N^2 \log N^2)$ time. Applying a binary search to these elements, after at most $\log N^2$ attempts, the largest such value will be determined. Since each requires $O(N^{2.5})$ time, the result follows. □

The above result can also be alternatively reached. In [5], the authors show that the problem of maximizing the k^{th} smallest entry in a combinatorial optimization problem can be achieved in time complexity $O(T \log p)$ where p is the number of variables in the optimization problem and $O(T)$ corresponds to solving the underlying optimization problem with unitary data. In the special case when the combinatorial optimization problem is that of finding a perfect cardinality matching, the same complexity is attained.

4 Solving the Quantile Bipartite Perfect Matching Problem

4.1 Algorithms For a given affinity matrix A and a particular $\alpha \in [0, 1]$, our approaches based on [5] and [11] for solving the quantile bipartite perfect matching problem all entail solving the assignment problem for a sequence of potential distinct q values in $\{A_{mn},\ m, n \in \mathcal{N}\}\}$. We shall say that a particular q value is α-*feasible* if there exists $X \in \mathcal{X}$ such that the k_α^{th} smallest affinity value in $\{A_{mn}X_{mn},\ m, n \in \mathcal{N}\}$ is at least as high as q. Our aim is to find the largest such α-*feasible* q value. Deciding whether a particular q value is α-*feasible* amounts to solving a minimum cost assignment (perfect matching) problem in a square matrix (balanced bipartite graph) with cost values as 0 or 1. Let C be this cost matrix where

$$C_{mn} = \begin{cases} 1 \text{ if } A_{mn} < q \\ 0 \text{ otherwise} \end{cases} \quad m, n \in \mathcal{N}.$$

Now using Lemma 3.2, q value is α-*feasible* if the minimum cost assignment problem has an optimal value less than or equal to $k_\alpha - 1$. We depict the pseudocode for solving the quantile bipartite perfect matching problem in Algorithm 1.

Algorithm 1 quantile assignment for square matrices

Input: $N \times N$ affinity matrix A and $\alpha \in [0, 1]$
Output: $q^* \leftarrow \max_{X \in \mathcal{X}} q_\alpha\ (\{A_{mn}X_{mn},\ m, n \in \mathcal{N})$

Initialize:
$q^* \leftarrow 0$
$k_\alpha \leftarrow \max(0, \lceil (N - (1 - \alpha)N^2 \rceil)$
Sort distinct entries of $(\{A_{mn},\ m, n \in \mathcal{N}\})$ into $q_1 < q_2, \ldots, < q_s$
$left \leftarrow 1$
$right \leftarrow s$
while $left \leq right$ **do**
  $mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$
  $C_{mn} = \begin{cases} 1 \text{ if } A_{mn} < q_{mid} \\ 0 \text{ otherwise} \end{cases} \quad m, n \in \mathcal{N}.$
  $cost \leftarrow$ minimum cost perfect matching value for data $C = [C_{mn}]$
  if $cost \leq k_\alpha - 1$ **then**
    $q^* \leftarrow q_{mid}$
    $left \leftarrow mid + 1$
  else
    $right \leftarrow mid - 1$
  end if
end while
return q^*

The minimum cost perfect matching in the complete, balanced bipartite graph $G = (S \cup T, \mathcal{A})$ with cost data $C = [C_{mn}]$, more specifically, with arc weight for arc (s_m, t_n) taking value C_{mn} can be found through several methods. One choice is a direct application of the well known Hungarian algorithm [12] that takes as input a square cost matrix data and outputs a minimum cost perfect matching. A second choice is to apply the Hopcroft-Karp algorithm [10] to find a maximum cardinality matching in a bipartite graph that is not necessarily complete or balanced. The edges of this bipartite graph will correspond to pairs in $\mathcal{A}$ that have zero corresponding costs in $C = [C_{mn}]$. Augmenting the maximum cardinality matching edges in this graph with enough edges with cost values as one will result in a minimum cost perfect matching for G. A third possibility is to adopt the method proposed in [11]. The approach utilizes the fact that the maximum cardinality matching in a bipartite graph can be found through an application of a minimum cost maximum flow in an appropriately defined capacitated network with the addition of source and sink nodes. Given $G = (S \cup T, \mathcal{A})$,

let this network be $\hat{G} = (\hat{\mathcal{N}}, \hat{A})$ where

$$\hat{\mathcal{N}} = S \cup T \cup \{s, t\},$$

$$\hat{A} = \{(s, s_m) : m \in \mathcal{N}\} \cup \{(t_n, t) : n \in \mathcal{N}\} \cup \mathcal{A}$$

$$capacity_{mn} = 1 \text{ for each } (u, v) \in \hat{A}.$$

The aim is to find a maximum flow from s to t in $\hat{G}$ with the minimum cost where cost data is defined as follows:

$$cost_{uv} = \begin{cases} C_{mn} \text{ if } u = s_m, v = t_n \text{ for } m, n \in \mathcal{N} \\ 0 \text{ otherwise} \end{cases} \quad (u, v) \in \hat{A}.$$

Ford and Fulkerson's augmenting path algorithm [13] through shortest paths in the residual network is applied where the Bellman-Ford algorithm [2] is utilized for finding shortest paths since the residual network will have negative costs.

4.2 Integer Linear Programming Formulation

Even though we have established above that the problem is solvable in polynomial time, the problem is immediately amenable to a mixed integer linear programming formulation that can be processed by off-the-shelf solvers. For the interested reader, we provide this formulation below. Let $A* = \max\{A_{mn} : m, n \in \mathcal{N}\}$ be the largest affinity value. Let us partition the matched edge variables defined for all $m, n \in \mathcal{N}$ into two, namely,

$$x_{mn} = \begin{cases} 1 \text{ if matched edge } \{m, n\} \text{ has } A_{mn} \geq q \\ 0 \text{ otherwise} \end{cases}$$

and

$$y_{mn} = \begin{cases} 1 \text{ if matched edge } \{m, n\} \text{ has } A_{mn} < q \\ 0 \text{ otherwise} \end{cases}.$$

Then the following model will solve the quantile bipartite perfect matching problem.

$$\max \quad q$$

$$\text{s.t.} \quad \sum_{m=1}^{N} (x_{mn} + y_{mn}) = 1 \qquad n \in \mathcal{N} \tag{4.4}$$

$$\sum_{n=1}^{N} (x_{mn} + y_{mn}) = 1 \qquad m \in \mathcal{N} \tag{4.5}$$

$$A_{mn} x_{mn} \geq q - \mathrm{A}^*(1 - x_{mn}) \quad m, n \in \mathcal{N} \tag{4.6}$$

$$A_{mn} y_{mn} \leq q - 1 \qquad m, n \in \mathcal{N} \tag{4.7}$$

$$\sum_{m=1}^{N} \sum_{n=1}^{N} y_{mn} \leq k_\alpha - 1 \tag{4.8}$$

$$x_{mn}, y_{mn} \in \{0, 1\} \qquad m, n \in \mathcal{N} \tag{4.9}$$

Constraints (4.4) and (4.5) ensure that there will be a perfect matching. The matching edges indicated with x and y variables are forced to have corresponding affinity values at least and strictly less than q values, respectively through constraints (4.6) and (4.7). With constraint (4.8) the number of matched edges with small values is limited with the maximum allowed value for the given α. Finally, the objective function finds the maximum such possible q, i.e. the maximum α-quantile value.

4.3 Nontrivial Cases and Numerical Examples

If $\alpha \leq \frac{N-1}{N}$ then $k_\alpha = 0$ so any matching will solve the optimization problem (2.3) with zero objective function value. On the other hand, if $\alpha > \frac{N^2-1}{N^2}$, then $k_\alpha = N$ and any matching which includes element $A*$ will solve (2.3) with the highest possible value of $A*$.

In other words, the nontrivial cases for the optimization problem (2.3) are when α and N are such that

$$\frac{N-1}{N} < \alpha \leq \frac{N^2-1}{N^2}.$$

Consider the following instance with $N = 5$, the affinity matrix

$$A = \begin{bmatrix} 19 & 13 & 8 & 1 & 14 \\ 9 & 3 & 18 & 2 & 18 \\ 17 & 15 & 7 & 14 & 19 \\ 2 & 1 & 9 & 6 & 13 \\ 17 & 20 & 13 & 14 & 15 \end{bmatrix}.$$

and $\alpha = 0.92$. One can easily find that $k_\alpha = 3$. Consider the following perfect matching

$$A = \begin{bmatrix} \boxed{19} & 13 & 8 & 1 & 14 \\ 9 & 3 & 18 & 2 & \boxed{18} \\ 17 & 15 & 7 & \boxed{14} & 19 \\ 2 & 1 & \boxed{9} & 6 & 13 \\ 17 & \boxed{20} & 13 & 14 & 15 \end{bmatrix}.$$

The q value in this matching is the third smallest value in the set of matched elements, and is equal to 18. However, the following assignment

$$A = \begin{bmatrix} \boxed{19} & 13 & 8 & 1 & 14 \\ 9 & 3 & 18 & \boxed{2} & 18 \\ 17 & 15 & 7 & 14 & \boxed{19} \\ 2 & 1 & \boxed{9} & 6 & 13 \\ 17 & \boxed{20} & 13 & 14 & 15 \end{bmatrix}.$$

is preferable since it gives a q value equal to 19. In fact, the latter is the optimal value for (2.3).

Now, consider the following 4×4 example with the affinity matrix

$$A = \begin{bmatrix} 60 & 87 & 91 & 94 \\ 47 & 55 & 31 & 38 \\ 1 & 37 & 43 & 91 \\ 92 & 62 & 52 & 56 \end{bmatrix}.$$

For $\alpha = 0.9375 = \frac{N^2-1}{N^2}$ an optimal assignment is

$$A = \begin{bmatrix} 60 & 87 & 91 & \boxed{94} \\ 47 & 55 & \boxed{31} & 38 \\ 1 & \boxed{37} & 43 & 91 \\ \boxed{92} & 62 & 52 & 56 \end{bmatrix}.$$

with value 92. The same assignment will also solve (2.3) with value 94 for any $\alpha > 0.9375$. Actually, it is one of the six alternate optimal assignments including the maximum affinity value 94.

4.4 Computational Performance of the Algorithms The three algorithms described in Subsection 4.1 are assessed based on computational performance. All tests were performed on a laptop with 64-bit Operating System, x64-based Intel(R) Core(TM) i5-8265U processor, CPU @1.60GHz to 1.80 GHz, and with installed memory (RAM) equal to 8.00 GB. Out of the three, the slowest performing method turned out to be the minimum cost maximum flow algorithm implemented through Ford and Fulkerson and Bellman-Ford methods. Indeed, the computational performance when contrasted with the other two algorithms is orders of magnitude slower. Figure 1 depicts the performance (in CPU seconds) of this method for a range of N values till 300. Each point in the figure corresponds to the average performance of 7 instances. Note that solving the quantile perfect matching problem for a 300x300 affinity matrix instance took 25 minutes on the average. The performances of the Hungarian and the Hopcroft-Karp approaches are shown in Figure 2. Clearly, solving the minimum cost perfect matching with these two approaches is much faster. Each dot in this figure again corresponds to an average of 7 instances. We notice that both approaches can solve the quantile assignment problem for a 4000x4000 affinity matrix in around seven minutes on the average. An example of application pertains to point cloud registration as illustrated in Figures 3 and 4 where the point clouds were acquired using Optical Coherence Tomography.

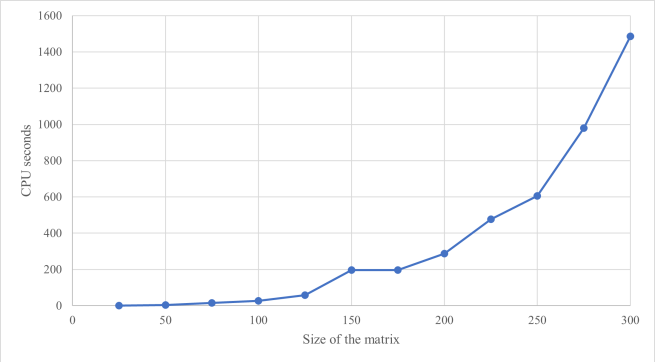


Figure 1: Results of Minimum Cost Maximum Flow Algorithm

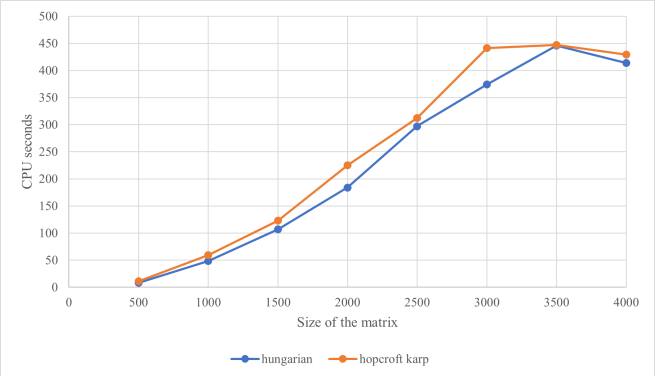


Figure 2: Results of Hungarian and Hopcroft-Karp Algorithms

5 The Quantile Assignment Problem for Non-Square Matrices

The quantile assignment problem is also valid when the affinity matrix is not a square matrix. Let us assume that the given affinity matrix A is $M \times N$ where $M > N$. Note that assuming that the number of columns is less than the number of rows is without loss of generality as one simply can take the transpose of the given affinity matrix otherwise. Let $\mathcal{M} = \{1, \ldots, M\}$.

In the rectangular affinity matrix case, the set of matchings becomes $\mathcal{X} = \{X \in \{0,1\}^{M \times N} : \sum_{m=1}^{M} X_{mn} = 1, n \in \mathcal{N} \text{ and } \sum_{n=1}^{N} X_{mn} \leq 1, m \in \mathcal{M}$ and the maximization of the α-quantile problem can be casted as

$$\max_{X \in \mathcal{X}} \quad q_\alpha \left(\{ A_{mn} X_{mn},\ m \in \mathcal{M}, n \in \mathcal{N} \} \right). \tag{5.10}$$

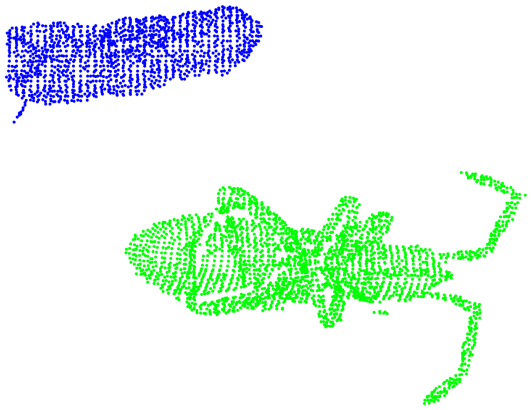


Figure 3: Experiment using two ants with missing parts: initial position of the two point clouds

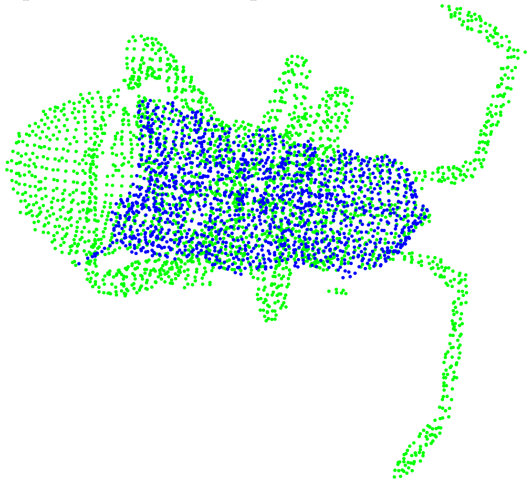


Figure 4: Experiment using two ants with missing parts: final alignment of the two point clouds.

Proceeding as we did for the square affinity matrices, the value k_α is defined by the formula,

$$k_\alpha = \max\left(0, \left\lceil N - (1-\alpha)NM \right\rceil\right).$$

Note that the Hungarian algorithm can only be used for square matrices. One can complement any rectangular matrix to a square one by adding appropriate dummy rows or columns, however, since the performances of the Hungarian and the Hopcroft-Karp approaches are similar, we chose to find the minimum cost maximum matching with the Hopcroft-Karp approach. One can easily adapt Algorithm 1 to rectangular matrices. Obviously, the resulting 0-1 cost matrix C for the minimum cost maximum matching problem will be a rectangular matrix of the same dimensions as the affinity matrix A. The mathematical model can easily be adapted to the rectangular case as well as follows.

$$\begin{aligned}
\max \quad & q \\
\text{s.t.} \quad & \sum_{m=1}^{M}(x_{mn}+y_{mn}) = 1 && n \in \mathcal{N} \\
& \sum_{n=1}^{n}(x_{mn}+y_{mn}) \le 1 && m \in \mathcal{M} \\
& A_{mn}x_{mn} \ge q - \mathrm{A}^*(1-x_{mn}) && m \in \mathcal{M}, n \in \mathcal{N} \\
& A_{mn}y_{mn} \le q - 1 && m \in \mathcal{M}, n \in \mathcal{N} \\
& \sum_{m=1}^{M}\sum_{n=1}^{N} y_{mn} \le k_\alpha - 1 \\
& x_{mn}, y_{mn} \in \{0,1\} && m \in \mathcal{M}, n \in \mathcal{N}
\end{aligned}$$

For a non-square matrix example consider the following 7×4 affinity matrix

$$A = \begin{bmatrix} 36 & 50 & 68 & 69 \\ 72 & 98 & 8 & 22 \\ 79 & 90 & 6 & 25 \\ 2 & 26 & 70 & 26 \\ 62 & 36 & 38 & 35 \\ 27 & 28 & 28 & 10 \\ 55 & 22 & 21 & 41 \end{bmatrix}.$$

For $\alpha = 0.9254$, k_α can be calculated as 2. The following maximum cardinality matching gives the optimal q value for this affinity matrix, which is equal to 70.

$$A = \begin{bmatrix} 36 & 50 & 68 & 69 \\ \boxed{72} & 98 & 8 & 22 \\ 79 & \boxed{90} & 6 & 25 \\ 2 & 26 & \boxed{70} & 26 \\ 62 & 36 & 38 & 35 \\ 27 & 28 & 28 & \boxed{10} \\ 55 & 22 & 21 & 41 \end{bmatrix}.$$

6 Generalization to Non-Bipartite Perfect Matchings

Though the motivating point cloud registration is valid for bipartite graphs, one can generalize the quantile matching problem as a combinatorial optimization problem to the case of arbitrary graphs. In this section we extend our results on bipartite graphs to general graphs.

Given a weighted graph $G = (V, E)$ that is not necessarily bipartite with $w_e \ge 0$ a given weight for each $e \in E$, the non-bipartite perfect matching problem is to find a perfect matching $\tilde{M} \subseteq E$ such that $|\tilde{M}| = \frac{|V|}{2}$, $e_1 \cap e_2 = \emptyset \quad \forall e_1, e_2 \in \tilde{M} : e_1 \ne e_2$ and $\sum_{e\in\tilde{M}} w_e$ is the largest possible among all such perfect matchings.

As in the bipartite perfect matching case let $\mathcal{X} = \{X \in \{0,1\}^{|E|} : X_e = 1 \iff e \in \tilde{M}$ and $\tilde{M}$ indicates a perfect matching in $G\}$ be the set of all perfect matchings.

The problem of maximizing the α-quantile of the weights associated with a non-bipartite perfect matching problem becomes the following optimization problem

$$\max_{X \in \mathcal{X}} \quad q_\alpha \left(\{ w_e X_e : e \in E \} \right). \tag{6.11}$$

Proceeding as we did in section 3, the following results can be attained through similar arguments hence we provide them without proofs.

LEMMA 6.1. *Given $X \in \mathcal{X}$ and $\alpha \in [0,1]$, $q_\alpha(\{w_e X_e, e \in E\})$ is the k_α^{th} smallest affinity value in $\{w_e : X_e = 1, e \in E\}$ where*

$$k_\alpha = \max\left(0, \left\lceil \frac{|V|}{2} - (1-\alpha)|E| \right\rceil \right).$$

It has been shown in [11] that finding a perfect cardinality matching with the additional restriction that no more than a given number of edges from a specified subset are in the matching can be done in $O(|V|^3)$ time for general graphs as well. Thus, the following results are readily attainable.

LEMMA 6.2. *Given α, q and weighted graph $G = (V, E)$, finding a perfect matching such that the k_α^{th} smallest entry in this matching is no smaller than q can be done in $O(|V|^3)$ time.*

THEOREM 6.1. *Finding a perfect matching that maximizes the α-quantile in general graphs can be done in $O(|V|^3 \log |V|)$ time.*

References

[1] M. Akgül, A Genuinely Polynomial Primal Simplex Algorithm for the Assignment Problem, Discrete Applied Mathematics, 45: 93-115, 1993.

[2] R. Bellman, On a Routing Problem Quarterly of Applied Mathematics, vol. 16, no. 1, 1958, pp. 87–90.

[3] S. Chrétien, K. Cross, J. McBride and W. Sun, Curvature and Real Algebraic Manifold Matching Approach to 3D Approximation and Affine Point Cloud Registration, National Physical Laboratory, Teddington, UK, 2020.

[4] H. Föllmer and A. Schied, Stochastic Finance, 4th Edt., De Gruyter, Berlin, 2016.

[5] J. Gorski and S. Ruzika, On k-Max-Optimization, Operations Research Letters, vol. 37, no. 1, 2009, pp. 23–26.

[6] R. Koenker, Quantile Regression, Econometric Society Monographs, Cambridge University Press, Cambridge, 2005.

[7] R. Burkard, M. Dell'Amico and S. Martello, Assignment Problems, Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[8] G.B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, 1963.

[9] D. Eppstein, M. T. Goodrich, J. Jorgensen, M. R. Torres, Geometric Fingerprint Recognition via Oriented Point-Set Pattern Matching, Canadian Conference on Computational Geometry, 98-113, 2018.

[10] J. E. Hopcroft and R. M. Karp, An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs, SIAM J. Comput., 2(4):225–231, 1973.

[11] A. Itai, M. Rodeh and S. L. Tanimoto, Some Matching Problems for Bipartite Graphs, Journal of the Association for Computing Machinery, 25 (4), 517-525, 1978.

[12] H.W. Kuhn, The Hungarian Method for the Assignment Problem, Naval Research Logistics Quarterly, vol. 2, no. 1-2, 1955, pp. 83–97.

[13] L. R. Ford, D. R. Fulkerson, Maximal Flow Through a Network, Canadian Journal of Mathematics, 8: 399-404, 1956.

[14] X. Zhang, H. Li, and Z. Cheng, Curvature Estimation of 3D Point Cloud Surfaces through the Fitting of Normal Section Curves, Proceedings of ASIAGRAPH, 2008:23–26, 2008.

Using Predicted Weights for Ad Delivery*

Thomas Lavastida[1], Benjamin Moseley[1], R. Ravi†[1], and Chenyang Xu‡[2]

[1]Carnegie Mellon University, USA
tlavasti,moseleyb,ravi@andrew.cmu.edu
[2]Zhejiang University, China
xcy1995@zju.edu.cn

Abstract

We study the performance of a proportional weights algorithm for online capacitated bipartite matching modeling the delivery of impression ads. The algorithm uses predictions on the advertiser nodes to match arriving impression nodes fractionally in proportion to the weights of its neighbors. This paper gives a thorough empirical study of the performance of the algorithm on a data-set of ad impressions from Yahoo! and shows its superior performance compared to natural baselines such as a greedy water-filling algorithm and the ranking algorithm.

The proportional weights algorithm has recently received interest in the theoretical literature where it was shown to have strong guarantees beyond the worst-case model of algorithms augmented with predictions. We extend these results to the case where the advertisers' capacities are no longer stationary over time. Additionally, we show the algorithm has near optimal performance in the random-order arrival model when the number of impressions and the optimal matching are sufficiently large.

1 Introduction

There has been recent interest in augmenting online algorithms with machine-learned prediction. This line of work has lead to new models of algorithm analysis for going beyond worst-case analysis [23, 21, 17, 3]. The theoretical models considered in these works have led to the development of new algorithms which incorporate learned parameters (i.e. predictions) along with theoretical guarantees depending on the quality of the predictions. Typically, an algorithm's performance is parameterized by the error in the prediction. With a perfect prediction, an algorithm's performance should be stronger than the best worst-case algorithm. Additionally the algorithm's performance should be robust to moderate error in the predicted parameters.

An exciting question is how close the model is to practice and how to leverage it to develop improved practical algorithms. The goal of this paper is to show that the model is closely tied to practice for the online matching problem that arises in impression ad allocation and to demonstrate the empirical efficiency of a recently proposed algorithm based on proportional weights.

Capacitated Online Matching: Capacitated online matching is a fundamental problem faced in practice and is a special case of the Adwords problem [25]. In the problem, there is a set of known advertisers (offline) that each have a capacity (budget). Impressions arrive online that have edges to an arbitrary subset of advertisers. An impression can be matched (fractionally) to at most one advertiser. The goal is to match as many impressions as possible under the capacity constraints.

It is well-known that the best deterministic (greedy) algorithm is $\frac{1}{2}$-competitive. Allowing for randomization, the ranking algorithm is known to be $(1 - \frac{1}{e})$-competitive and this is the best possible competitive ratio for any online algorithm [19].

The best worst-case algorithm is far from optimal. Squeezing out the best performance in practice is fundamental in applications such as the Adwords problem [10].

An emerging line of work [21, 22, 5] has suggested that perhaps better matching algorithms exist if they use learned information about known real world match-

*Thomas Lavastida, and Benjamin Moseley: Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.

†This material is based upon work supported in part by the U. S. Office of Naval Research under award number N00014-21-1-2243 and the Air Force Office of Scientific Research under award number FA9550-20-1-0080.

‡The corresponding author. Supported in part by Science and Technology Innovation 2030 –"The Next Generation of Artificial Intelligence" Major Project No.2018AAA0100902 and China Scholarship Council No.201906320329.

ing instances. That is, in practice there is often lots of data available on prior matching instances (e.g. the instance from yesterday). The idea is that an algorithm can use information from prior instances to perform better in future problem instances. Information learned from past data is referred to as a prediction. This work has the potential to offer new algorithmic ideas for solving matching instances in practice.

We consider the proportional weights algorithm, which has been suggested in this line of work.

Proportional Weights for Online Matching: This paper considers a recently proposed proportional weights algorithm for online matching. The algorithm was first proposed by Agrawal et al. [2] and further developed in [21, 22].

Let $G = (I, A, E)$ be a bipartite graph with capacities $C \in \mathbb{Z}_+^A$ on A. We refer to I as impressions and A as advertisers. We use $m = |I|$ and $n = |A|$ for the number of impressions and advertisers. Each advertiser (impression) has a subset N_a (N_i) of neighbors in the graph. In this paper we consider the fractional matching problem that is represented by the following linear program[1].

$$\begin{aligned} \max \quad & \sum_{ia \in E} x_{ia} \\ \text{s.t.} \quad & \sum_{a \in N_i} x_{ia} \leq 1 \qquad \forall i \in I \\ & \sum_{i \in N_a} x_{ia} \leq C_a \quad \forall a \in A \\ & x_{ia} \geq 0 \qquad \forall ia \in E \end{aligned} \tag{1.1}$$

The proportional weights algorithm assigns each advertiser $a \in A$ a weight $\alpha_a > 0$. The vector of weights $\alpha \in \mathbb{R}_+^A$ on the advertisers encodes a fractional assignment of impressions to advertisers in the following way.

$$x_{ia}(\alpha) = \frac{\alpha_a}{\sum_{a' \in N_i} \alpha_{a'}} \tag{1.2}$$

That is, an impression is assigned to the advertisers in its neighborhood proportionally according to α. Notice that the allocation of each impression is independent of the others. Thus, if a set of weights is given a priori then the weights can be used to assign impressions online.

This algorithm does not consider if an advertiser has been saturated and may assign extra impressions to an advertiser above its capacity. These impressions are effectively not allocated. In the online setting, we consider an improved version that uses the weights to assign the impression proportionally, but only among the neighborhood of advertisers that have remaining capacity. This is the natural adaptation to the case where an advertiser becomes saturated.

Agrawal et al. [2] showed that for any $\epsilon > 0$, there exists a set of weights $\alpha \in \mathbb{R}_+^A$ such that the allocation given by (1.2) is a $(1-\epsilon)$-approximate solution to (1.1): the running time to arrive at such weights is inversely proportional to ϵ^2. This establishes that there exists a set of weights giving a high quality matching; notice that it is not obvious that such weights exist in the first place.

The work of Agrawal et al. [2] was interested in this proportional weights algorithm because they give a static assignment of impressions (order independent) as well as allowing for each impression to be assigned only knowing the neighborhood of the impressions, which is useful for distributed algorithms.

Later these weights were considered in the algorithms augmented with predictions model [21, 22]. Lavastida et al. [22] showed that predicting these weights can be used to go beyond the worst-case for online matching. This prediction could come from computing the weights from prior instances of matching. The work of Agrawal et al. [2] imply the weights give near optimal performance if predicted perfectly. [22] showed that the weights are instance-robust. Informally, this guarantees that if the weights give good performance on one instance, then they have strong performance on similar instances. Moreover, [22] showed that if the matching instance is drawn from an unknown product distribution then weights that give a near optimal solution are learnable in the PAC learning model (learnability). This suggests that weights can be learned in practice from prior matching instances and used on future instances to get strong online performance.

This line of work begs the question, does the proportional weights algorithm perform well empirically? In particular, if weights are computed from prior instances of online matching can they be used to give strong performance on future instances of the matching problem in practice? For instance, can learning the weights on yesterdays data be used on today's online instance? Understanding these questions has the potential to influence matching algorithms in numerous applications.

Results: This paper's goal is to demonstrate the empirical effectiveness of the proportional weights algorithm. To do so, we consider a data set obtained from Yahoo! [29] on the Adwords problem [25]. This data set gives instances of advertisers and impressions over multiple days. We propose two algorithms utilizing pre-

[1] The more general AdWords problem has an objective coefficient for each allocated impression representing different values of an impression for different advertisers.

dictions, one is the standard proportional weights algorithm while the other is an improved version.

We use the following strong benchmarks. One is the water-filling algorithm [18], which fractionally allocates the current impression so that it maximizes the minimum occupied proportion of its capacity among its neighbours. The other is the randomized ranking algorithm, which uses a single random ordering of the advertisers and assigns each impression to the available advertiser in its neighborhood with highest priority. In the following results we consider several methods for setting the advertisers' capacities and the impression arrival orders.

- Our improved proportional weights algorithm significantly outperforms the standard version.
- Fix a single day. Consider weights that are computed from a random sample of the day's impressions and use them for the remaining impressions online. The improved weights algorithm is consistently ahead of the baselines, often giving a near optimal matching. This shows the impressions can be learned from a sample of a day's impressions and used on the remaining impressions effectively. This empirically demonstrates that the weights are learnable.
- Next we consider learning weights on prior days and using them on future days. The impression distribution varies drastically from day to day. Despite this high variance, our improved weights algorithm has stronger performance than the baselines on every day tested, demonstrating robustness.

To complement our results, we give two theoretical results. First, we consider the weights in the random order arrival model. We show that the weights give a $(1-\epsilon)$-approximate online matching in the random order model for any constant $\epsilon > 0$. This algorithm uses the first σ fraction of the arriving impressions to compute the weights (i.e. learn the weights) and then applies them to the remaining instance. In the result below, m refers to the number of impressions arriving online while n is the number of (offline) advertisers. This gives a similar result to prior work in the random order model [10], but uses proportional weights instead of a primal-dual scheme.

THEOREM 1.1. *There exists an algorithm which is* $(1-\epsilon)$*-competitive with probability* $1-\delta$ *for online matching in the random order model whenever* $m = \Omega(\frac{n^2}{\sigma\epsilon^2}\log(\frac{n}{\delta}))$ *and* $\mathrm{OPT} \geq \epsilon m$ *for any* $\epsilon, \delta, \sigma \in (0,1)$.

Next, we give a theorem that demonstrates the robustness of the weights. This is an extension of the robustness result in [22]. Intuitively, we show that predicted weights are robust to modest changes in the input, including changes in advertiser capacity.

Consider a problem instance where advertiser a has capacity C_a and there is a set of impression types, where each type is defined by a subset of advertisers. Each impression of the same type has the same set of advertisers as neighbors. Let C_i be the number of impressions of type i.

In Section 5 (Theorem 5.2), we show that if a fixed set of weights can match at least $(1-\epsilon)\mathrm{OPT}$ impressions on a given instance defined by C, then the same weights have value at least $(1-\epsilon)\mathrm{OPT} - 2\eta$ on any instance C' where $\eta = \sum_i |C_i - C'_i| + \sum_a |C'_a - C_a|$. Here η measures the difference in the two instances. This gives a theoretical explanation for the strong empirical performance of the weights even when advertiser capacities change and the impression volumes vary.

2 Related Work

Algorithms with Predictions: In this paper we do a practical evaluation of online matching algorithms using predictions learned from past data. There has been significant recent interest in analyzing online algorithms in the presence of erroneous predictions [23, 28, 16, 4, 27, 13, 3, 21, 8, 31, 32, 5]

Antoniadis et al. [5] looks at online weighted bipartite matching problems with predictions in the random order model. In this setting each offline node can be matched at most once and the edges are weighted. This differs from our setting where we consider the unweighted problem with capacities on the offline nodes.

Data Driven Algorithm Design: Using past data to learn the weights for our algorithm can be seen as a case of data driven algorithm design [6, 15, 7]. This line of work is concerned with using past problem instances to learn an algorithm from some class of algorithms which will perform well on future instances drawn from the same population as the past instances. In particular, the sample complexity (i.e. the number of past instances used by the learning algorithm) is of particular importance.

Practical Algorithms for Online Matching: In addition to the deep theoretical understanding of online matching algorithms, there has been effort to develop algorithms that work well on real data sets. Zhou et al. [30] develop a robust online weighted matching algorithm based on primal-dual schemes for the random order model which account for changes in the underlying distribution and evaluate it on a display ad data set. Ma et al. [24] develop an algorithm for online assortment

optimization and evaluate it on data from a hotel chain, but their emphasis is on revenue maximization rather than capacity allotment. Chen et al. [9] develop a real-time bidding algorithm for display ad allocation and evaluate it on a proprietary display ad data set; their methods closely mirror the water-filling algorithm we study.

Random Order Model: There has been a line of work in analyzing online algorithms in the random order model for matching problems and more general packing integer linear programs [10, 12, 26, 1, 11, 20, 14]. These algorithms are usually based off of some sort of primal-dual approach. We give an alternative approach for online capacitated matching in the random order model based on using proportional weights.

3 Preliminaries

Recall the capacitated online matching problem represented by the linear program (1.1). In the online version of the problem, only the advertisers and their capacities are initially known to the algorithm. The impressions I (along with their neighbors) are revealed to the algorithm one at a time and it must commit to an assignment $\{x_{ia}\}_{a \in N_i}$ satisfying the constraints in (1.1). We consider the case when the set of impressions I are decided by an adversary, but revealed to the algorithm in random order.

Proportional Weights: We consider fractional solutions to (1.1) parameterized by weights $\alpha \in \mathbb{R}_+^A$, using the proportional allocation scheme in (1.2).

Observe that given a fixed set of weights α, the allocation $x_{ia}(\alpha)$ is order independent and thus is suitable for the online setting. Next, note that $x_{ia}(\alpha)$ always satisfies the first constraint of (1.1) with equality, but it may not satisfy the second constraint. Any reasonable way of decreasing the allocation to satisfy the second constraint suffices, so we define $R_a(\alpha) = \min\{\sum_{i \in I} x_{ia}(\alpha), C_a\}$ to be a's contribution to the size of the fractional matching. We utilize the following theorem due to Agrawal et al. [2].

DEFINITION 3.1. *For $T \in \mathbb{Z}_+$ and $\epsilon > 0$, define $\mathcal{A}(T, \epsilon) = \{\alpha \in \mathbb{R}_+^A \mid \alpha_a = (1+\epsilon)^k, k \in [T]\}$.*

THEOREM 3.1. ([2]) *For any bipartite graph $G = (I, A, E)$, capacities $C \in \mathbb{Z}_+^A$, and $\epsilon > 0$ there exists $T \in \mathbb{Z}_+$ and $\alpha \in \mathcal{A}(T, \epsilon)$ such that*

$$\sum_{a \in A} R_a(\alpha) \geq (1-\epsilon)\mathrm{OPT}.$$

In particular, we can take $T = O(\frac{1}{\epsilon^2}\log(\frac{n}{\epsilon}))$. Moreover, there is a polynomial time algorithm which computes α.

4 Proportional Weights under Random Orders

This section considers the capacitated online matching problem in the random order model and shows that the proportional weights are learnable in this model. Under some mild assumptions, the weights computed by the first small portion of impressions can obtain a good performance for the whole instance.

We first state the definition of the random order model. Suppose that $I = \{i_1, i_2, \ldots, i_m\}$ and let $\gamma : [m] \to [m]$ be a permutation. Denote $I(\gamma)$ to be the sequence $(i_{\gamma(1)}, i_{\gamma(2)}, \ldots, i_{\gamma(m)})$, i.e. consider revealing the impressions I in the order induced by γ. If γ is a permutation drawn uniformly at random and $\delta \in (0,1)$, then we say that an algorithm is c-competitive with high probability if for all impression sets I:

$$\Pr[\mathrm{ALG}(I(\gamma)) \geq c\mathrm{OPT}(I)] \geq 1 - \delta. \tag{4.3}$$

where $\mathrm{ALG}(I(\gamma))$ is the size of the matching when $I(\gamma)$ is given as input to the online algorithm and $\mathrm{OPT}(I)$ is the optimal value of (1.1). Note that $\mathrm{OPT}(I)$ does not depend on the ordering γ. We will use OPT instead of $\mathrm{OPT}(I)$ when the context is clear.

Our algorithm takes the first σm impressions for some $\sigma \in (0,1)$ and reduces the capacity of each advertiser by a σ factor, then computes proportional weights α using the algorithm of Theorem 3.1.

Denote the first σm impressions by $S \subseteq I$. Let $R_a(\alpha, S) = \min\{\sum_{i \in S} x_{ia}(\alpha), \frac{|S|}{m} C_a\}$. Similarly, let $\mathrm{OPT}(S)$ be the size of a maximum cardinality matching on the graph $G' = (S, A, E)$ with capacities $C'_a = \frac{|S|}{m} C_a$. We think of $\sum_a R_a(\alpha, S)$ as the value obtained by weights α on an instance restricted to the impressions in S and the capacities scaled down appropriately.

Algorithm 1 Proportional Weights in the Random Order Model

Input: $G = (I, A, E), C, \gamma, \sigma, \epsilon$.
Let S be the first σm impressions in $I(\gamma)$
Set $x_{ia} = 0$ for each $i \in S, a \in N_i$
Set $T = \Theta(\frac{1}{\epsilon^2}\log(\frac{n}{\epsilon}))$
Compute weights $\alpha \in \mathcal{A}(T, \epsilon)$ on $G' = (S, A, E)$ with capacities $C'_a = \sigma C_a \ \forall a \in A$
for each remaining impression i **do**
  For each $a \in N_i$, let $x_{ia}(\alpha) = \frac{\alpha_a}{\sum_{a' \in N_i} \alpha_{a'}}$
end for

We show that this algorithm performs well when the number of impressions m is large relative to the number of advertisers.

THEOREM 4.1. *Algorithm 1 is $(1-\epsilon)$-competitive with probability $1-\delta$ for online matching in the random order*

Algorithm 2 Proportional Weights (PW)

Input: $G = (I, A, E)$ where I and E arrive online, $\{C_a\}_{a\in A}$, predicted weights $\{\hat{\alpha}\}$
while an impression i comes **do**
For each $a \in N_i$, let $x_{ia} = \frac{\hat{\alpha}_a}{\sum_{a'\in N_i} \hat{\alpha}_{a'}}$.
end while

Algorithm 3 Improved Proportional Weights (IPW)

Input: $G = (I, A, E)$ where I and E arrive online, $\{C_a\}_{a\in A}$, predicted weights $\{\hat{\alpha}\}$
while an impression i comes **do**
Let $N^* \subseteq N_i$ be the unfull advertisers in its neighbourhood (A full advertiser is one whose capacity is equal to its current total allocation).
For each $a \in N^*$, let $x_{ia} = \frac{\hat{\alpha}_a}{\sum_{a'\in N^*} \hat{\alpha}_{a'}}$.
end while

model whenever $m = \Omega(\frac{n^2}{\sigma\epsilon^2}\log(\frac{n}{\delta}))$ *and* $\mathrm{OPT} \geq \epsilon m$ *for any* $\epsilon, \delta, \sigma \in (0, 1)$.

Our analysis applies two probabilistic arguments. First, we show that $\sum_a R_a(\alpha, S) \approx \sigma \sum_a R_a(\alpha)$ for the computed weights α with high probability. This involves a union bound over all possible weights in $\mathcal{A}(R, \epsilon)$. Second, we show that $\mathrm{OPT}(S) \approx \sigma\mathrm{OPT}$ with high probability. This involves a union bound over cuts in the bipartite graph G. Formal versions of these two statements imply the theorem. Proofs have been omitted from this version of the paper.

5 Robustness of Proportional Weights on Similar Instances

A learning-augmented algorithm can be given directly if we can predict the proportional weights. See Algorithm 2 for the description. For this algorithm, Lavastida et al. [22] give a theoretical result of its competitive ratio under an assumption about advertiser capacities. This section extends this result by relaxing that assumption.

Define an impression vector for an instance to be the m-dimensional vector $(\ldots, C_i, \ldots)$, where each component corresponds to the supply of each impression i. In [22], Lavastida et al. prove that when the capacity of each advertiser is fixed, for any constant $\epsilon > 0$, the $(1-\epsilon)$-approximated weights $\hat{\alpha}$ for instance $\hat{\mathcal{I}}$ has a competitive ratio at least $1 - \epsilon - 2\eta/\mathrm{OPT}$ on instance $\mathcal{I}$, where OPT is the optimal value of instance $\mathcal{I}$ and η is ℓ_1 norm between the impression vectors of these two instances. In this paper, we relax the condition that the capacity needs to be fixed, and obtain a similar theorem. Define an advertiser vector, similarly, to be the n-dimensional vector $(\ldots, C_a, \ldots)$, where each component corresponds to the capacity of each advertiser a.

THEOREM 5.1. *For any constant* $\epsilon > 0$*, with the* $(1-\epsilon)$*-approximated weights* $\hat{\alpha}$ *for instance* $\hat{\mathcal{I}}$*, algorithm PW has a competitive ratio at least*

$$1 - \epsilon - \frac{2\eta}{\mathrm{OPT}}$$

on instance $\mathcal{I}$*, where* OPT *is the optimal value of instance* $\mathcal{I}$ *and* η *is* ℓ_1 *norm between the impression vectors of these two instances plus the* ℓ_1 *norm between two advertiser vectors.*

The basic idea of this proof is first showing the difference between the performances of weights $\hat{\alpha}$ in the two instances is at most η and then proving the difference of these two instances' optimal values is also at most η with the help of a vertex cut. The details are deferred to a full version of this paper.

This simple algorithm is consistent, but not robust, which means that it will achieve good results if the prediction is accurate, but will perform badly if the prediction error is large. Several ideas are introduced in [22] to obtain robust algorithms with proportional weights. In this paper, instead of using the complicated algorithms in [22], we propose a very simple algorithm called *Improved Proportional Weights* (IPW) that can achieve a very competitive and robust performance in the experiments. The description is given in Alg. 3.

Clearly, algorithm IPW always matches more (fractional) impressions than algorithm PW and obtains a maximal matching, whose competitive ratio is at least 1/2. Thus, we have the following theorem.

THEOREM 5.2. *For any constant* $\epsilon > 0$*, with the* $(1-\epsilon)$*-approximated weights* $\hat{\alpha}$ *for instance* $\hat{\mathcal{I}}$*, algorithm IPW has a competitive ratio at least*

$$\max\left\{1 - \epsilon - \frac{2\eta}{\mathrm{OPT}}, \frac{1}{2}\right\}$$

on instance $\mathcal{I}$*, where* OPT *is the optimal value of instance* $\mathcal{I}$ *and* η *is* ℓ_1 *norm between the impression vectors of these two instances plus the* ℓ_1 *norm between two advertiser vectors.*

6 Experimental Results

In this section, we validate the performance of PW and IPW on realistic data empirically. We investigate two main aspects of applying predicted weights in practice:

- Learnability - we sample a small fraction of the data we assembled as training data (motivated by the

results in Section 4) and observe that this provides enough information to set weights that are superior in performance to the benchmarks.

- Robustness - we use data obtained over several days and examine the performance of our weights from previous day(s) on the current day's set of impression arrivals, and show consistent improved performance over the benchmarks across time.

First, we describe our data source and how we set up instances, such as the definition of impressions, the bipartite graph between impressions and advertisers, and how the capacities of the advertisers are set based on the supply of impressions.

6.1 Experimental Setup We used the Yahoo! Search Marketing Advertiser Bid-Impression-Click data, provided as part of their Advertising and Market data [29][2]. The data contains nearly 78 million records each containing an allocated advertising impression. Each record provides a day d, an anonymized account id (for the advertiser) a, a rank r, a set of anonymized keyphrases P, average bid b, the number of impressions C_P of the keyphrase P allocated to this advertiser, and clicks k out of these impressions. A row (d, a, r, P, b, C_p, k) indicates that on day d, advertiser a was allocated C_P copies of impression P at (average monetary) cost b and obtained k clicks. Since the impressions are presumably made available in a ranked order of preference of placement, an advertiser's placement for each of these impressions is also accompanied by its rank.

Our maximum cardinality matching instances are built from this data set, one for each day. The average bid and click data are ignored. We first define the vertex set, then the edge set and finally the supply and capacity functions.

Vertex Sets: The advertiser set A is the set of advertiser account ids. Keyphrases in the data are defined as a combination of a set of elementary (anonymized) keywords. We use this to construct the impression set I as follows. The keyphrase base set S is obtained by taking the union of all elementary keyphrases in P. We count the number of occurrences of each elementary keyphrase $p \in S$ and select the top 20 most frequent keyphrases (based on the supply of the impressions they are part of). Let S^* denote the set of these most popular keyphrases and define the impression set I to be the power set of S^* excluding the empty set, i.e., $I := 2^{S^*} \setminus \emptyset$. Each vertex $i \in I$ can thus be viewed as an impression type. Different keyphrases that have the same intersection with S^* are this 'contracted' into the same impression vertex in this formulation.

Edge Set: Next we construct the edges. Define a mapping f from the given keyphrase sets $\{P\}$ to the impression set I by $f(P) = P \cap S^*$. For each row, add edge $(f(P), a)$ into the edge set. Additionally, for any subset $i \in I$ of $f(P)$, the edge (i, a) is also added. Although these keyphrases might not be assigned to advertiser a in this table, we assume that they are relevant and hence can be assigned to advertiser a[3].

Impression supply: Intuitively, on each day, the total impression sizes corresponding to different ranks should be the same. However, this is not the case in the provided data set, potentially due to sampling effects. Thus, for each day and for each keyphrase, we only retain the rows with the rank that contains the *maximum* total impression size and remove the rows with other ranks [4]. For each vertex $i \in I$, its impression supply C_i is the total size of the keyphrase sets that belongs to type i, i.e., $C_i = \sum_{f(P)=i} C_P$.

Advertiser Capacities: The appropriate capacity function of the advertiser set for creating challenging instances is not readily apparent. Thus, we consider three ways to set advertiser capacities.

1. **Random Quota:** For each impression vertex i, split its size C_i among its neighbourhood randomly. The capacity of each advertiser is set to be its final allocation obtained in this way.

2. **Max-min Quota:** Sort all impressions lexicographically and process each impression one by one: for each impression, allocate its supply to its neighbourhood such that the minimum allocation in the neighborhood is maximized. The capacity of each advertiser is set to be its final allocation at the end of this process.

3. **Least-degree Quota:** For each impression, assign its size equally to the advertisers with the least degree[5] in its neighbourhood.

The random method above tries to de-correlate an impression's capacity and its neighborhood's eventual demands. The second and the third are also designed

[2]The data set is available from Yahoo! upon request.

[3]While some advertisers may only seek out very specific combinations of keyphrases in their bidding, we assume that most advertisers are interested in impressions broadly matching their keyphrases of interest in our formulation, and ignore the existence of the former type of advertisers.

[4]The experimental results do not vary significantly if selecting other ranks in this process of defining the impression supply.

[5]In the graph constructed from the data set, the proportion of the least-degree neighbours is roughly half its total neighborhood for each impression.

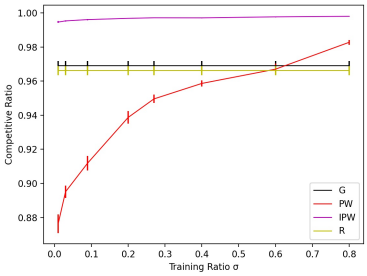
to deliberately avoid correlations between the neighborhood's demands.

For the daily instance constructed in this way, there are roughly 4500 advertisers and 85 impressions with non-zero sizes, with roughly 8000 edges between them that can be used to allocate a total supply of about 1.8 million copies of these impressions.

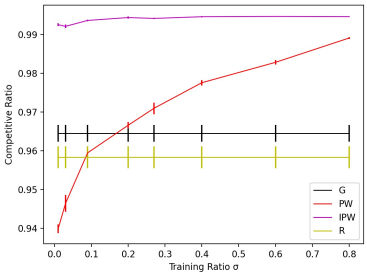
Computational Setup and Methods. We conducted the experiments[6] on a machine running Ubuntu 18.04 with 12 i7-7800X CPUs and 48 GB memory. In the experiments, our algorithms (PW and IPW) are compared to the competitive greedy/water-filling algorithm (G) [18] and ranking algorithm (R) [19]. The greedy 'water-filling' algorithm (G) fractionally allocates the current impression so that the proportion of capacities of all its neighbors capacities that are filled are as equal as possible (Imagining these filled proportions to be water levels, the allocation fills the lowest levels until they all rise to include another in this set, and so on). The ranking algorithm (R) uses a single random permutation of the advertisers to set a priority order among the neighborhoods of any arriving impressions and allocates the impression in this order. All algorithms are implemented in Python 3.6.11. All results are averaged over 4 runs.

6.2 Learnability To test the learnability of our algorithms, for each daily instance we sample a σ proportion of impressions (for a sampling parameter $\sigma \in [0, 1]$) to construct the training instance. The graph connectivity is the same as in the original data set while the advertiser capacities are constructed using these impressions and one of the three rules above. We compute proportional weights on this training instance and use them in Algorithms 2 and 3 for that day's whole instance.

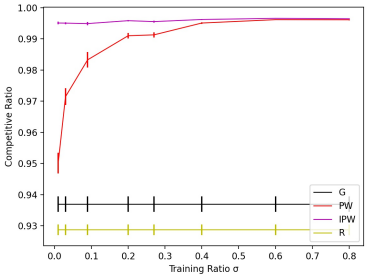
We investigate the performance when impressions arrive in a random order or in an adversarial order. We measure performance by the traditional measure of competitive ratio. Since the instances were engineered so that all impressions are allocable, this is simply the fraction of all impressions that were assigned by each of the methods. Since it is hard to find the most adversarial order for each algorithm for the given impressions, we set up five different arrival orders and use the worst performance of an algorithm among these orders to approximate its performance in an adversarial order. Finally, we illustrate the performance of the algorithms across the daily order, which we think is closest to the order which occurs in practice. The description of the daily order is given later.



(a) Random Quota



(b) Max-min Quota



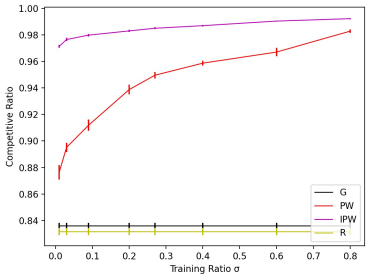
(c) Least-degree Quota

Figure 1: The performance of each algorithm on the test data when impressions arrive in a random order, plotted as a function of the training ratio.

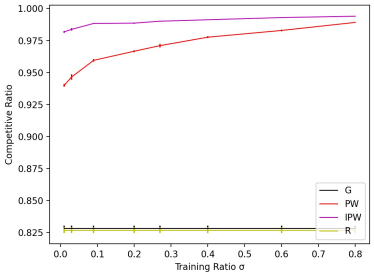
Random Order. The performance of each algorithm in random order is shown in Figure 1, where each plot corresponds to one of the three quota allocation rules[7]. All results are obtained by taking the average performance of ten days' instances, while the performance of each day is evaluated on 4 separate runs. The x-axis of each plot is the training ratio σ, indicating that σ proportion of impressions are sampled to serve as the training data. Note that in these figures, the learned weights from the σ proportion are evaluated on the whole data set for the same day. As σ increases, the baseline greedy (G) and ranking (R) algorithms'

[6]Code is available at `https://github.com/Chenyang-1995/PredictiveWeights`

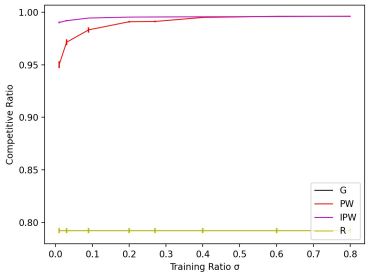
[7]Note the varying scales in the Y-axes in many of the figures.



(a) Random Quota

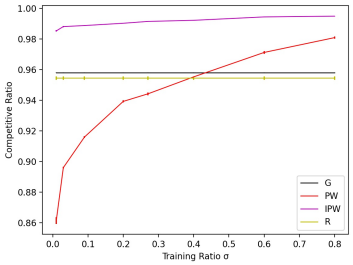


(b) Max-min Quota

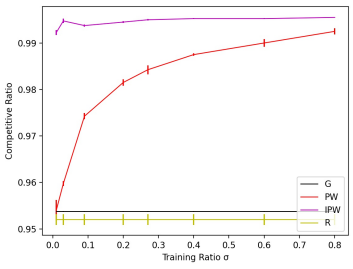


(c) Least-degree Quota

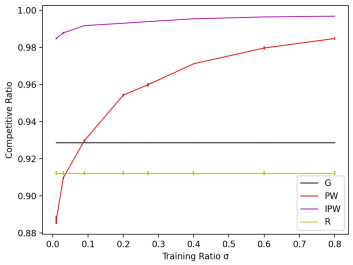
Figure 2: The worst performance of each algorithm on the test data among five impression arriving orders tested, plotted as a function of the training ratio.



(a) Random Quota



(b) Max-min Quota



(c) Least-degree Quota

Figure 3: The performance of each algorithm when impression arrives in a daily order, as a function of the training ratio. Impressions within a day are assumed to arrive in random order.

performance remain unchanged since they do not use the training data, while algorithms PW and IPW show improving performances. In Figure 1a and Figure 1b, algorithm PW has a worse performance than algorithm G and algorithm R initially, but obtains a better performance when σ becomes 0.6 and 0.1 respectively. Compared to other algorithms, IPW gives the best performance. We see that even with one percent of the data ($\sigma = 0.01$), IPW outperforms the traditional online algorithms.

Approximating Adversarial Orders. To study the performance of these algorithms when impressions arrive in an adversarial order, we set up following five arriving orders and check the worst performance.

1. **Random Order:** Impressions arrive randomly.
2. **C_i-descending Order:** Sort all impressions in the descending order of their capacities and let impressions arrive in this order.
3. **C_i-ascending Order:** All impressions arrive in the opposite order of the C_i-descending order (i.e. non-descending order of their capacities).
4. **C_a-descending Order:** Define the neighbourhood capacity of an impression to be the sum of the supplies of its neighbouring advertisers. Sort all impressions in the descending order of their neighbourhood capacities and let impressions arrive in this order.

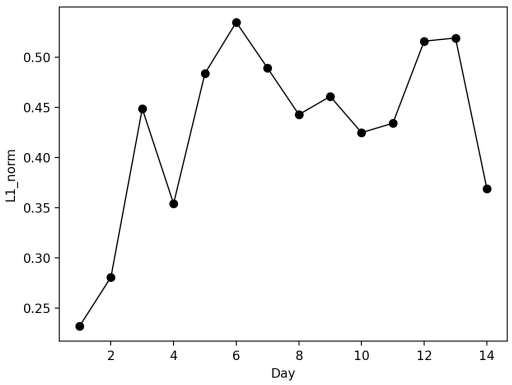


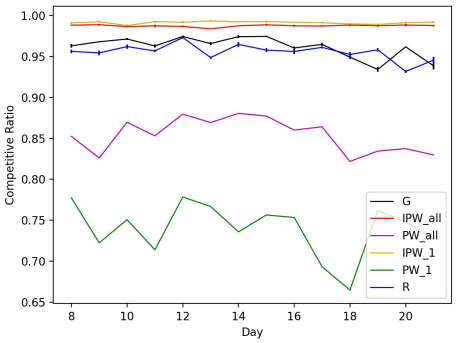
Figure 4: The ℓ_1 norm between the impression vectors on day 0 and day $i > 0$.

5. **C_a-ascending Order:** All impressions arrive in the opposite order of the C_a-descending order.

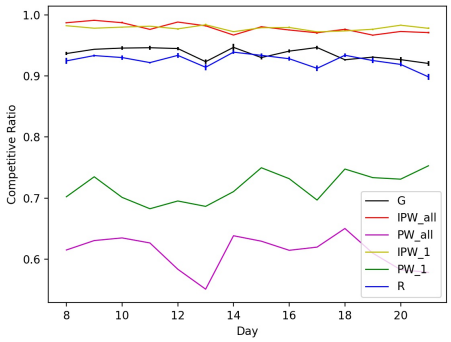
The worst performance of each algorithm (over these five orders) is shown in Figure 2. We observe that the algorithms based on proportional weights are much more stable than other algorithms over different arrival orders (More experimental results can be found in a full version of this paper). The performance of algorithms PW and IPW mostly remain unchanged because their allocation policies are unrelated to the impression arrival order, unlike greedy and ranking, and IPW remains the best. The performances of algorithm G and R vary significantly when the arriving order changes. Their worst performances are usually realized in the C_i-descending order and C_a-descending order respectively.

Daily Order. Finally, we show each algorithm's performance when impressions arrive in a daily order in Figure 3. We combine the instances on 7 days and get a stacked instance, where the vertex set is the union of vertex sets in these graphs and each vertex's capacity is the sum of its capacities in the instances. Say that impressions arrive in daily order if all impressions on day d arrive before the impressions on day $d+1$ while inside any day d, impressions arrive in a random order. The results show a similar trend as in Figure 1 with one difference being that the performances of algorithm G and R decrease slightly. Thus, the value σ where algorithm PW surpasses algorithm G and R becomes slightly smaller.

6.3 Robustness. This subsection considers the robustness. As mentioned above, the instances on different days are quite different from each other. We visualize this by showing the ℓ_1 norm between the impressions



(a) Max-min Quota



(b) Least-degree Quota

Figure 5: The performance of algorithms on different days where PW and IPW use weights from previous days on the current day, and the impression arrival order is random within each day. Note that the starting day is set to be day 8 in order to collect more training data for algorithm PW_all and IPW_all.

on the first day (Day 0) and the normalized impression vectors on following days, using the impressions from the given data set across the first 15 days. In our setting, the vector has dimension $2^{20} - 1$. As shown in Figure. 4, the difference between two days could be very large. This suggests it could be hard for proportional weights to work well when learned on one day and used on another. Surprisingly, our experiments shows empirically that weights are robust across days despite these large differences in the problem instances. Since we wish to model some level of correlation between the instances on different days, the random quota is not tested in this experiment.

The robustness experiment simulates how the proportional weights may be used in practice. We predict the weights based on previous day(s) and use the weights to allocate impressions for a different day. We

consider either computing weights on an instance that is just the previous day or computing the weights from *all* prior days impressions. We use "_all" and "_1" to denote the weights of all previous days and yesterday respectively. The results are shown in Figure 5 [8].

As mentioned in the beginning of this section, algorithm PW will not be robust if the predicted weights are inaccurate. The performance of algorithm PW_all and PW_1 imply a large error in the prediction. However, even with large prediction error IPW achieves the best performance.

6.4 Conclusions We see the following trends from the experiments.

- The Improved Proportional Weights algorithm is consistently the best algorithm considered, giving near optimal performance on all instances tested.
- The weights are learnable given a random sample of a problem instance. In particular, such weights leads to the improved proportional weights algorithm having near-optimal performance.
- The weights are robust to large changes in the problem instance. Across days, the advertiser capacities and the supply of the impressions change by large margins. Still, the Improved Proportional Weights algorithm has strong performance.

These experiments show that (1) the proportional weights algorithm is a strong algorithm for online matching and (2) the theoretical results on the weights can be seen empirically. This empirically shows a connection between the algorithms augmented with predictions model and practice. We hope these results stimulate further experimental investigation of algorithms augmented with predictions.

References

[1] Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Oper. Res.*, 62(4):876–890, 2014.

[2] Shipra Agrawal, Morteza Zadimoghaddam, and Vahab Mirrokni. Proportional allocation: Simple, distributed, and diverse matching with high entropy. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 99–108, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[3] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ml predictions for online algorithms. *ICML 2020*, 2020.

[4] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 345–355. PMLR, 13–18 Jul 2020.

[5] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7933–7944. Curran Associates, Inc., 2020.

[6] Maria-Florina Balcan. Data-driven algorithm design, 2020.

[7] Maria-Florina Balcan, Dan F. DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? *CoRR*, abs/1908.02894, 2019.

[8] Aditya Bhaskara, Ashok Cutkosky, Ravi Kumar, and Manish Purohit. Online learning with imperfect hints. *CoRR*, abs/2002.04726, 2020.

[9] Ye Chen, Pavel Berkhin, Bo Anderson, and Nikhil R. Devanur. Real-time bidding algorithms for performance-based display ad allocation. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 1307–1315, New York, NY, USA, 2011. Association for Computing Machinery.

[10] Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings 10th ACM Conference on Electronic Commerce (EC-2009), Stanford, California, USA, July 6–10, 2009*, pages 71–78, 2009.

[11] Nikhil R. Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. *J. ACM*, 66(1):7:1–7:41, 2019.

[12] Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online stochastic packing applied to display ad allocation. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010.

[13] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of*

[8] If one considers the worst performance among the five orders, the performance of algorithm G and R will decrease significantly as in previous experiments, while PW and IPW remain stable.

Machine Learning Research, pages 2319–2327. PMLR, 2019.

[14] Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Math. Oper. Res.*, 41(4):1404–1431, 2016.

[15] Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM J. Comput.*, 46(3):992–1017, 2017.

[16] Piotr Indyk, Frederik Mallmann-Trenn, Slobodan Mitrovic, and Ronitt Rubinfeld. Online page migration with ML advice. *CoRR*, abs/2006.05028, 2020.

[17] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[18] Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000.

[19] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 352–358. ACM, 1990.

[20] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. *SIAM J. Comput.*, 47(5):1939–1964, 2018.

[21] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020.

[22] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing, 2020.

[23] Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3302–3311, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[24] Will Ma and David Simchi-Levi. Algorithms for online matching, assortment, and pricing with tight weight-dependent competitive ratios. *Oper. Res.*, 68(6):1787–1803, 2020.

[25] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

[26] Marco Molinaro and R. Ravi. The geometry of online packing linear programs. *Math. Oper. Res.*, 39(1):46–59, 2014.

[27] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 9684–9693, 2018.

[28] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1834–1845. SIAM, 2020.

[29] Yahoo! Webscope. Yahoo! search marketing advertiser bid-impression-click dataset version 1.0. `http://research.yahoo.com/Academic_Relations`. Accessed 2020-12-1.

[30] Yu-Hang Zhou, Chen Liang, Nan Li, Cheng Yang, Shenghuo Zhu, and Rong Jin. Robust online matching with user arrival distribution drift. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):459–466, Jul. 2019.

[31] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling, 2020.

[32] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms, 2020.

Fully-dynamic Weighted Matching Approximation in Practice

Eugenio Angriman* Henning Meyerhenke* Christian Schulz† Bora Uçar‡

Abstract

Finding large or heavy matchings in graphs is a ubiquitous combinatorial optimization problem. In this paper, we engineer the first non-trivial implementations for approximating the dynamic weighted matching problem. Our first algorithm is based on random walks/paths combined with dynamic programming. The second algorithm has been introduced by Stubbs and Williams without an implementation. Roughly speaking, their algorithm uses dynamic unweighted matching algorithms as a subroutine (within a multilevel approach); this allows us to use previous work on dynamic unweighted matching algorithms as a black box in order to obtain a fully-dynamic weighted matching algorithm. We empirically study the algorithms on an extensive set of dynamic instances and compare them with optimal weighted matchings. Our experiments show that the random walk algorithm typically fares much better than Stubbs/Williams (regarding the time/quality tradeoff), and its results are often not far from the optimum.

1 Introduction

A matching in a graph is a set of pairwise vertex-disjoint edges. Alternatively, a matching can be seen as a subgraph (restricted to its edges) with degree at most 1. A matching is *maximal* if no edges can be added to it without violating the matching property that no two matching edges share a common vertex. A matching of a graph G is *maximum*, in turn, if there exists no matching in G with higher cardinality. Computing (such) matchings in a graph is a ubiquitous combinatorial problem that appears in countless applications [31]. Two popular optimization problems in this context are (i) the maximum cardinality matching (MCM) problem, which seeks a matching with maximum cardinality, and (ii) the maximum weighted matching (MWM) problem, i.e., to find a matching (in a weighted graph) whose total edge weight is maximum. Micali and Vazirani [43] solve MCM in $\mathcal{O}(m\sqrt{n})$ time, Mucha and Sankowski [44] in $\mathcal{O}(n^\omega)$, where $\omega < 2.373$ is the matrix multiplication exponent [1], m is the number of edges, and n is the number of vertices. Concerning MWM, the best known algorithm is by Galil et al. [27], it takes $\mathcal{O}(mn \log n)$ time. For integral edge weights up to W, Gabow and Tarjan [26] proposed an $\mathcal{O}(m\sqrt{n} \log nW)$ time algorithm. Sankowski's algorithm [50], in turn, requires $\tilde{\mathcal{O}}(Wn^\omega)$ time, where $\tilde{\mathcal{O}}$ hides a polylogarithmic factor.

Real-world graphs occurring in numerous scientific and commercial applications are not only massive but can also change over time [42]. Finding matchings in dynamic networks has numerous applications, e.g. in kidney exchange [21] or when maintaining dynamic multilevel hierarchies that are frequently used for partitioning or clustering [14, 16]. For example, edges can be inserted, deleted, or their weight can be updated – think of a road network where edge weights represent the traffic density, the constantly evolving web graph, or the virtual friendships between the users of a social network. Hence, recomputing a (weighted) matching from scratch each time the graph changes is expensive. With many frequent changes, it can even be prohibitive, even if we are using a polynomial-time algorithm. Further, when dealing with large graphs, computing an optimal matching can be too time-consuming and thus one often resorts to approximation to trade running time with solution quality. In recent years, several fully-dynamic algorithms for both exact and approximate MCM [4, 6, 7, 9–12, 17, 28, 30, 35, 36, 45, 46, 49, 51] and MWM [2, 29, 52] have been proposed. These algorithms exploit previously computed information about the matching to handle graph updates more efficiently than a static recomputation. Yet, limited effort has been invested to actually implement these algorithms in executable code and test their performance on real-world instances. Only recently, Henzinger et al. [33] engineered several fully-dynamic algorithms for MCM and investigated their practical performance on large real-world graphs. Similar results have not been produced yet for fully-dynamic MWM algorithms (neither exact nor approximate) and therefore their practical performance is still unknown.

Contributions. In this paper we focus on the fully-dynamic MWM problem and explore the gap between theoretical results and practical performance for two algorithms. The first one, inspired by Maue and Sanders [41], combines random walks and dynamic pro-

*Humboldt-Universität zu Berlin, Berlin, Germany
†Heidelberg University, Heidelberg, Germany
‡CNRS and LIP, ENS Lyon, France

gramming to compute augmenting paths in the graph; if the random walks are of appropriate length and repeated sufficiently often, the algorithm maintains a $(1+\epsilon)$-approximation in $\mathcal{O}(\epsilon^{-1}\Delta^{2/\epsilon+3}\log n)$ time per update, where Δ is the maximum node degree observed throughout the algorithm. The second algorithm, introduced by Stubbs and Williams [52], uses existing fully-dynamic α-approximation algorithms for MCM as subroutines to maintain a $2\alpha(1+\epsilon)$-approximation for MWM in fully-dynamic graphs. We provide the first implementations of the aforementioned algorithms, analyze their performance in systematic experiments on an extensive set of dynamic instances, and compare the quality of the computed matchings against the optimum. The best algorithm is very often less than 10% away from the optimum and at the same time very fast.

2 Preliminaries

Basic Concepts. Let $G = (V, E)$ be a simple *undirected graph* with edge weights $\omega : E \to \mathbb{R}_{>0}$. We extend ω to sets, i.e., $\omega(E') := \sum_{e\in E'} \omega(e)$. We set $n = |V|$, and $m = |E|$; $N(v) := \{u : \{v, u\} \in E\}$ denotes the *neighbors* of v. The (unweighted) degree of a vertex v is $d(v) := |N(v)|$. A matching $\mathcal{M} \subset E$ in a graph is a set of edges without common vertices. The *cardinality* or *size* of a matching is simply the cardinality of the edge subset $\mathcal{M}$. We call a matching *maximal* if there is no edge in E that can be added to $\mathcal{M}$. A *maximum cardinality matching* $\mathcal{M}_{\text{opt}}$ is a matching that contains the largest possible number of edges of all matchings. A *maximum weight matching* $\mathcal{M}'_{\text{opt}}$ is a matching that maximizes $\omega(\mathcal{M}'_{\text{opt}})$ among all possible matchings. An *α-approximate* maximum (weight) matching is a matching that has weight at least $\frac{\omega(\mathcal{M}_{\text{opt}})}{\alpha}$. A vertex is called *free* or *unmatched* if it is not incident to an edge of the matching. Otherwise, we call it *matched*. For a matched vertex u with $\{u, v\} \in \mathcal{M}$, we call vertex v the *mate* of u, which we denote as $\text{mate}(u) = v$. For an unmatched vertex u, we define $\text{mate}(u) = \perp$. An *augmenting path* is defined as a cycle-free path in the graph G that starts and ends on a *free* vertex and where edges from $\mathcal{M}$ alternate with edges from $E\backslash\mathcal{M}$. The *trivial augmenting path* is a single edge with both endpoints free. Throughout this paper, we call such an edge a *free* edge. If we take an augmenting path and resolve it by matching every unmatched edge and unmatching every matched edge, we increase the cardinality of the matching by one. In the maximum cardinality case, any matching without *augmenting paths* is a maximum matching [8] and any matching with no augmenting paths of length at most $2k-3$ is a $(k/(k-1))$-approximate maximum matching [34]. A *weight-augmenting path* $\mathcal{P}$ with respect to a matching $\mathcal{M}$ is an alternating path whose free edges are heavier than its edges in M: $w(\mathcal{M} \oplus \mathcal{P}) > w(\mathcal{M})$, where $\oplus$ denotes the symmetric difference. A weight-augmenting path with k edges outside $\mathcal{M}$ is called *weight-augmenting k-path*. Note that a weight-augmenting k-path can have $2k-1, 2k$, or $2k+1$ edges, whereas in the unweighted case an augmenting path with k edges outside of a given matching has exactly $2k-1$ edges. Note that a weight-augmenting path does not have to start or end in a free node.

Our focus in this paper are *fully-dynamic graphs*, where the number of vertices is fixed, but edges can be added and removed. All the algorithms evaluated can handle edge insertions as well as edge deletions. In the following, Δ denotes the maximum degree that can be found in any state of the dynamic graph.

Related Work. Dynamic algorithms are a widely researched topic. We refer the reader to the recent survey by Hanauer et al. [32] for most material related to theoretical and practical dynamic algorithms. Many of the numerous matching applications require matchings with certain properties, like maximal (no edge can be added to $\mathcal{M}$ without violating the matching property) or maximum cardinality matchings. Edmonds's blossom algorithm [25] computes a maximum cardinality matching in a static graph in time $\mathcal{O}(mn^2)$. This result was later improved to $\mathcal{O}(m\sqrt{n})$ by Micali and Vazirani [43]. Some recent algorithms use simple data reductions rules [37] or shrink-trees instead of blossoms [24] to speed up computations in static graphs. In practice, these algorithms can still be time-consuming for many applications involving large graphs. Hence, several practical approximation algorithms with nearly-linear running time exist such as the local max algorithm [15], the path growing algorithm [23], global paths [41], and suitor [40]. As this paper focuses on dynamic graphs, we refer the reader to the quite extensive related work section of [24] for more recent static matching algorithms.

In the dynamic setting, the maximum matching problem has been prominently studied ensuring α-approximate guarantees. A major exception is the algorithm by van den Brand et al. [53], which maintains the *exact* size of a maximum matching in $\mathcal{O}(n^{1.407})$ update time. One can trivially maintain a maximal (2-approximate) matching in $\mathcal{O}(n)$ update time by resolving all trivial augmenting paths of length one. Ivković and Llyod [35] designed the first fully-dynamic algorithm to improve this bound to $\mathcal{O}((n+m)^{\sqrt{2}/2})$ update time. Later, Onak and Rubinfeld [46] presented a randomized algorithm for maintaining an $\mathcal{O}(1)$-approximate matching in a dynamic graph that takes $\mathcal{O}(\log^2 n)$ expected amortized time for each edge update. This result led to a flurry of results in this area.

Baswana, Gupta and Sen [7] improved the approximation ratio of [46] from $\mathcal{O}(1)$ to 2 and the amortized update time to $\mathcal{O}(\log n)$. Further, Solomon [51] improved the update time of [7] from amortized $\mathcal{O}(\log n)$ to *constant*. However, the first deterministic data structure improving [35] was given by Bhattacharya et al. [11]; it maintains a $(3+\epsilon)$ approximate matching in $\tilde{O}(\min(\sqrt{n}, m^{1/3}/\epsilon^2))$ amortized update time, which was further improved to $(2+\epsilon)$ requiring $\mathcal{O}(\log n/\epsilon^2)$ update time by Bhattacharya et al. [12]. Recently, Bhattacharya et al. [10] achieved the first $\mathcal{O}(1)$ amortized update time for a deterministic algorithm but for a weaker approximation guarantee of $\mathcal{O}(1)$. For worst-case bounds, the best results are by (i) Gupta and Peng [29] requiring $\mathcal{O}(\sqrt{m}/\epsilon)$ update time for a $(1+\epsilon)$-approximation, (ii) Neiman and Solomon [45] requiring $\mathcal{O}(\sqrt{m})$ update time for a (3/2)-approximation, and (iii) Bernstein and Stein [9] requiring $\mathcal{O}(m^{1/4}/\epsilon^{2.5})$ update time for a $(3/2+\epsilon)$-approximation. Recently, Charikar and Solomon [17] as well as Arar et al. [4] (using [13]) presented independently the first algorithms requiring $\mathcal{O}(poly\log n)$ worst-case update time while maintaining a $(2+\epsilon)$-approximation. Recently, Grandoni et al. [28] gave an incremental matching algorithm that achieves a $(1+\epsilon)$-approximate matching in constant deterministic amortized time. Barenboim and Maimon [6] present an algorithm that has $\tilde{\mathcal{O}}(\sqrt{n})$ update time for graphs with constant neighborhood independence. Kashyop and Narayanaswamy [36] give a conditional lower bound for the update time, which is sublinear in the number of edges for two subclasses of fully-dynamic algorithms, namely lax and eager algorithms.

Despite this variety of different algorithms, to the best of our knowledge, very limited efforts have been made so far to engineer these dynamic algorithms and to evaluate them on real-world instances. Henzinger et al. [33] have started to evaluate algorithms for the dynamic maximum cardinality matching problem in practice. To this end, the authors engineer several dynamic maximal matching algorithms as well as an algorithm that is able to maintain the maximum matching. The algorithms implemented in their work are Baswana, Gupta and Sen [7], which performs edge updates in $\mathcal{O}(\sqrt{n})$ time and maintains a 2-approximate maximum matching, the algorithm of Neiman and Solomon [45], which takes $\mathcal{O}(\sqrt{m})$ time to maintain a (3/2)-approximate maximum matching, as well as two *novel* dynamic algorithms: a random walk-based algorithm as well as a dynamic algorithm that searches for augmenting paths using a (depth bounded) blossom algorithm. Experiments indicate that maintaining optimum matchings can be done much more efficiently than the naive algorithm that recomputes maximum matchings from scratch (more than an order of magnitude faster). Second, all inexact dynamic algorithms that have been considered in that work are able to maintain near-optimum matchings in practice while being multiple orders of magnitudes faster than the naive optimum dynamic algorithm. The study concludes that in practice an extended random walk-based algorithms should be the method of choice.

For the *weighted* dynamic matching problem, Anand et al. [2] propose an algorithm that can maintain an 4.911-approximate dynamic maximum weight matching that runs in amortized $\mathcal{O}(\log n \log D)$ time, where D is the ratio between the highest and the lowest edge weight. Gupta and Peng [30] maintain a $(1+\epsilon)$-approximation under edge insertions/deletions that runs in time $\mathcal{O}(\sqrt{m}\epsilon^{-2-\mathcal{O}(1/\epsilon)}\log N)$ time per update. Their result is based on several ingredients: (i) rerunning a static algorithm from time to time, (ii) a trimming routine that trims the graph to a smaller equivalent graph whenever possible and (iii) in the weighted case, a partition of the edges (according to their weights) into geometrically shrinking intervals.

Stubbs and Williams [52] present metatheorems for dynamic weighted matching. They reduce the dynamic maximum weight matching problem to the dynamic maximum cardinality matching problem in which the graph is unweighted. The authors prove that using this reduction, if there is an α-approximation for maximum cardinality matching with update time T, then there is also a $2\alpha(1+\epsilon)$-approximation for maximum weight matching with update time $\mathcal{O}(\frac{T}{\epsilon^2}\log^2 D)$. Their basic idea is an extension/improvement of the algorithm of Crouch and Stubbs [18] who tackled the problem in the streaming model. We go into more detail in Section 4. None of these algorithms have been implemented.

3 Algorithm DynMWMRandom: Random Walks + Dynamic Programming

Random walks have already been successfully employed for dynamic maximum cardinality matching by Henzinger et al. [33]; however, in their current form they cannot be used for dynamic maximum weight matching. The main idea of our first dynamic algorithm for the weighted dynamic matching problem is to find random augmenting paths in the graph. To this end, our algorithm uses a random walk process in the graph and then uses dynamic programming on the computed random path $\mathcal{P}$ to compute the best possible matching on $\mathcal{P}$.We continue by explaining these concepts in our context and how we handle edge insertions and deletions.

3.1 Random Walks For Augmenting Paths.

Our first dynamic algorithm is called DynMWMRan-

DOM and constructs a random (cycle-free) path $\mathcal{P}$ as follows: initially, all nodes are set to be eligible. Whenever an edge is added to the path $\mathcal{P}$, we set the endpoints to be not eligible. The random walk starts at an arbitrary eligible vertex u. If u is free, the random walker tries to randomly choose an eligible neighbor w of u. If successful, the corresponding edge e is added to the path $\mathcal{P}$ and the walk is continued at w (otherwise the random walk stops and returns $\mathcal{P}$). Moreover, u is set to not eligible. On the other hand, if u is matched and mate(u) is eligible, we add the edge $\{u, \text{mate}(u)\}$ to $\mathcal{P}$, set u to not eligible and continue the walk at $v := \text{mate}(u)$. There the algorithm tries to randomly choose an eligible neighbor w of v. If successful, the random walker adds the corresponding edge to the path, sets v to not eligible and continues the walk at w. If at any point during the algorithm execution, there is no adjacent eligible vertex, then the algorithm stops and returns the path $\mathcal{P}$. Note that, by construction, no vertex on the path is adjacent to a matched edge that is not on the path. We stop the algorithm after $\mathcal{O}(\frac{1}{\epsilon})$ steps for a given ϵ.

The algorithm tries to find a random eligible neighbor by sampling a neighbor u uniformly at random. If u is eligible, then we are done. If u is not eligible, we repeat the sampling step. We limit, however, the number of unsuccessful repetitions to a constant. If our algorithm did not find an eligible vertex, we stop and return the current path $\mathcal{P}$. Overall, this ensures that picking a random neighbor can be done in constant time. Hence, the time to find a random path is $\mathcal{O}(\frac{1}{\epsilon})$. Here, we assume that the eligible state of a node is stored in an array of size n that is used for many successive random walks. This array can be reset after the random walk has been done in $\mathcal{O}(\frac{1}{\epsilon})$ by setting all nodes on the path to be eligible again. The length of the random walk is a natural parameter of the algorithm that we will investigate in the experimental evaluation.

After the path has been computed, we run a dynamic program on the path to compute the optimum weighted matching on the path in time $\mathcal{O}(\frac{1}{\epsilon})$. If the weight of the optimum matching on $\mathcal{P}$ is larger than the weight of the current matching on the path, we replace the current matching on $\mathcal{P}$. Note that, due to the way the path has been constructed, this yields a feasible matching on the overall graph.

Dynamic Programming on Paths. It is well-known that the weighted matching problem can be solved to optimality on paths using dynamic programming [41]. In order to be self-contained, we outline briefly how this can be done. The full description of the algorithm can be found in the full version of the paper [3]. The algorithms is due to Maue and Sanders [41]. Given the path $\mathcal{P} = \langle e_1, \ldots, e_k \rangle$, the main idea of the dynamic programming approach is to scan the edges in the given order. Whenever it scans the next edge e_i, there are two cases: either that edge is used in an optimum matching of the subproblem $\langle e_1, \ldots, e_i \rangle$ or not. To figure this out, the algorithm takes the weight of the current edge e_i and adds the weight of the optimum subsolution for the path $\langle e_1, \ldots, e_{i-2} \rangle$. If this is larger than the weight of the optimum subsolution for the path $\langle e_1, \ldots, e_{i-1} \rangle$, then the edge e_i is used in an optimum solution for $\langle e_1, \ldots, e_i \rangle$; otherwise it is not.

Optimizations. The random walk can be repeated multiple times, even if a weight-augmenting path has been found. This is because, even if the random walker found an improvement, it can be possible that there is another augmenting path left in the graph. In our experiments, we use ℓ walks, where ℓ is a tuning parameter. However, repeating the algorithm too often can be time-consuming. Hence, we optionally use a heuristic to break early if we already performed a couple of unsuccessful random walks: we stop searching for augmenting paths if we made β consecutive random walks that were all unsuccessful. In our experiments, we use $\beta = 5$; similar choices of parameters should work just as well. We call this the *"stop early" heuristic*. We now explain how we perform edge insertions and deletions using the augmented random walks introduced above.

Edge Insertion. Our algorithm handles edge insertions $\{u, v\}$ as follows: if both u and v are free, we start a random walk at either u or v (randomly chosen) and make sure that the new edge is always included in the path $\mathcal{P}$. That means if we start at u, then we add the edge $\{u, v\}$ to the path and start the random walk described above at v (and vice versa). If one of the endpoints is matched, say u, we add the edges $\{u, \text{mate}(u)\}$ and $\{u, v\}$ to the path $\mathcal{P}$ and continue the walk at v. If both endpoints are matched, we add $\{u, \text{mate}(u)\}$, $\{u, v\}$ and $\{v, \text{mate}(v)\}$ to the path and continue the random walk at mate(v). Note that it is necessary to include matched edges, since the optimum weight matching on the path that we are constructing with the random walk may match the newly inserted edge. Hence, if we did not added matched edges incident to u and v, we may end up with an infeasible matching. After the full path is constructed, we run the dynamic program as described above and augment the matching if we found a better one.

Edge Deletion. In case of the deletion of an edge $\{u, v\}$, we start two subsequent random walks as described above, one at u and one at v. Note that if u or v are matched, then the corresponding matched edges will be part of the respective paths $\mathcal{P}$ that are created by the two random walks.

3.2 Analysis. The algorithm described above, called DynMWMRandom, can maintain a $(1+\epsilon)$-approximation if the random walks are of appropriate length and repeated sufficiently often. To this end, we first show a relation between the existence of a weight-augmenting k-path and the quality of a matching. For the proofs, see [3]. We note that result of proposition 3.1 is known. Pothen et al. [47, Lemma 3.1] show the result given in proposition 3.1. Our proof is somewhat different from the previous one, and is closer to the way we make us of the result in the random walk based algorithm, and therefore we include it in the full version for completeness.

Proposition 3.1. *Let $\mathcal{M}$ be a matching and $k \geq 1$ be the smallest number such that there is a weight-augmenting k-path with respect to $\mathcal{M}$. Then $w(\mathcal{M}) \geq \frac{k-1}{k} \cdot w(\mathcal{M}^*)$, where $\mathcal{M}^*$ is a matching with the maximum weight.*

Theorem 3.1. *Algorithm* DynMWMRandom *maintains a $(1+\epsilon)$-approximate maximum weight matching with high probability if the length of each path is $\lceil 2/\epsilon+3 \rceil$ and the walks are repeated $\lceil \Delta^{2/\epsilon+3} \log n \rceil$ times.*

4 Algorithm Family DynMWMLevel based on Results by Crouch, Stubbs and Williams

The algorithm by Stubbs and Williams [52] can use dynamic *unweighted* matching algorithms as a black box to obtain a weighted dynamic matching algorithm. We outline the details of the meta-algorithm here, as this is one of the algorithms that we implemented. Consider a dynamic graph that has integer weights in $[L, N]$. Moreover, let $C = N/L$. The basic idea of the algorithm, due to Crouch and Stubbs [18], is to maintain $\mathcal{O}(\epsilon^{-1} \log C)$ levels. Each of the levels only contains a subset of the overall set of edges. More precisely, the i-th level contains all edges of weight at least $(1+\epsilon)^i$ for $i \in \{\lfloor \log_{1+\epsilon} L \rfloor, \lfloor \log_{1+\epsilon} L \rfloor + 1, \ldots, \lfloor \log_{1+\epsilon} N \rfloor\}$. The algorithm then maintains unweighted matchings on each level by employing a dynamic maximum cardinality matching algorithm for each level when an update occurs. The output matching is computed by including all edges from the matching in the highest level, and then by adding matching edges from lower levels in descending order, as long as they are not adjacent to an edge that has been already added. Stubbs and Williams [52] extend the result by Crouch and Stubbs [18] and show how the greedy merge of the matchings can be updated efficiently.

Theorem 4.1. (Stubbs and Williams [52]) *If the matchings on every level are α-approximate maximum cardinality matchings, then the output matching of the algorithm is a $2\alpha(1+\epsilon)$-approximate maximum weight matching. Moreover, the algorithm can be implemented such that the update time is $\mathcal{O}(t(m,n)(\log_{1+\epsilon} N/L)^2)$, where $t(m,n)$ is the update time of the dynamic cardinality matching algorithm used.*

In our implementation, we use the recent dynamic maximum cardinality matching algorithms due to Henzinger et al. [33] that fared best in their experiments. The first algorithm is a random walk-based algorithm, the second algorithm is a (depth-bounded) blossom algorithm – leading to the dynamic counterparts DynMWMLevelR and DynMWMLevelB, respectively. The latter also has the option to maintain optimum maximum cardinality matchings. Both are described next to be self-contained. We refer to [33] for more details.

Random Walk-based MCM. In the update routines of the algorithm (*insert* and *delete*), the algorithm uses a random walk that works as follows: The algorithms starts at a free vertex u and chooses a random neighbor w of u. If the chosen neighbor is free, the edge $\{u, w\}$ is matched and the random walk stops. If w is matched, then we unmatch the edge $\{w, \text{mate}(w)\}$ and match $\{u, w\}$ instead. Note that $u \neq \text{mate}(w)$ since u is free in the beginning and therefore $\text{mate}(u) = \perp$, but $\text{mate}(\text{mate}(w)) = w$ and $w \neq \perp$. Afterwards, the previous mate of w is free. Hence, the random walk is continued at this vertex. The random walk performs $\mathcal{O}(\frac{1}{\epsilon})$ steps. If unsuccessful, the algorithm can perform multiple repetitions to find an augmenting path. Moreover, the most successful version of the algorithm uses Δ-*settling*, in which the algorithm tries to settle visited vertices by scanning through their neighbors. The running time of the Δ-settling random walk is then $\mathcal{O}(\Delta/\epsilon)$.

The algorithm handles edge insertions $\{u, v\}$ as follows: if both endpoints are free, the edge is matched and the algorithm returns. The algorithm does nothing if both endpoints are matched. If only one of the endpoints is matched, say u, then the algorithm unmatches u, sets $w := \text{mate}(u)$, and matches $\{u, v\}$. Then a random walk as described above is started from w to find an augmenting path (of length at most $\mathcal{O}(1/\epsilon)$). If the random walk is unsuccessful, all changes are undone. Deleting a matched edge $\{u, v\}$ leaves the two endpoints u and v free. Then a random walk as described above is started from u if it is free and from v if v is free.

Lemma 4.1. (Henzinger et al. [33]) *The random walk based algorithm maintains a $(1+\epsilon)$-approximate maximum matching if the length of the walk is at least $2/\epsilon - 1$ and the walks are repeated $\Delta^{2/\epsilon-1} \log n$ times.*

Corollary 4.1. *If the algorithm by Stubbs and Williams uses the random walk-based MCM algorithm on each level s.t. the length of the walks*

Table 1: Basic properties of the benchmark set of static graphs obtained from [5, 19, 39].

graph	n	m	graph	n	m
144	144 649	1 074 393	eu-2005	862 664	16 138 468
3elt	4 720	13 722	fe_4elt2	11 143	32 818
4elt	15 606	45 878	fe_body	45 087	163 734
598a	110 971	741 934	fe_ocean	143 437	409 593
add20	2 395	7 462	fe_pwt	36 519	144 794
add32	4 960	9 462	fe_rotor	99 617	662 431
amazon-2008	735 323	3 523 472	fe_sphere	16 386	49 152
as-22july06	22 963	48 436	fe_tooth	78 136	452 591
as-skitter	554 930	5 797 663	finan512	74 752	261 120
auto	448 695	3 314 611	in-2004	1 382 908	13 591 473
bcsstk29	13 992	302 748	loc-brightkite_edges	56 739	212 945
bcsstk30	28 924	1 007 284	loc-gowalla_edges	196 591	950 327
bcsstk31	35 588	572 914	m14b	214 765	1 679 018
bcsstk32	44 609	985 046	memplus	17 758	54 196
bcsstk33	8 738	291 583	p2p-Gnutella04	6 405	29 215
brack2	62 631	366 559	PGPgiantcompo	10 680	24 316
citationCiteseer	268 495	1 156 647	rgg_n_2_15_s0	32 768	160 240
cnr-2000	325 557	2 738 969	soc-Slashdot0902	28 550	379 445
coAuthorsCiteseer	227 320	814 134	t60k	60 005	89 440
coAuthorsDBLP	299 067	977 676	uk	4 824	6 837
coPapersCiteseer	434 102	16 036 720	vibrobox	12 328	165 250
coPapersDBLP	540 486	15 245 729	wave	156 317	1 059 331
crack	10 240	30 380	web-Google	356 648	2 093 324
cs4	22 499	43 858	whitaker3	9 800	28 989
cti	16 840	48 232	wiki-Talk	232 314	1 458 806
data	2 851	15 093	wing	62 032	121 544
email-EuAll	16 805	60 260	wing_nodal	10 937	75 488
enron	69 244	254 449	wordassociation-2011	10 617	63 788

is at least $2/\epsilon - 1$ *and the walks are repeated* $\Delta^{2/\epsilon-1} \log n$ *times, the resulting* DynMWMLevelR *algorithm is* $2(1+\epsilon)^2$*-approximate with update time* $\mathcal{O}(\epsilon^{-1}\Delta^{2/\epsilon-1} \log n(\log_{1+\epsilon} N/L)^2)$.

Blossom-based MCM. Roughly speaking, the algorithm runs a modified BFS – as in the blossom-algorithm – that is bounded in depth by $2/\epsilon - 1$ to find augmenting paths from a free node. This has a theoretically faster running time of $\mathcal{O}(\Delta^{2/\epsilon-1})$ per free node and still guarantees a $(1+\epsilon)$ approximation. The algorithm handles insertions as follows. Let $\{u, v\}$ be the inserted edge. If u and v are free, then the edge is matched directly and the algorithm stops. Otherwise, an augmenting path search is started from u if u is free and from v if v is free. If both u and v are not free, then a breadth-first search from u is started to find a free node reachable via an alternating path. From this node an augmenting path search is started. If the algorithm does not handle this case, i.e., if both u and v are not free, it does nothing and is called *unsafe*. If the algorithm handles the this case and the augmenting path search is not depth-bounded, then it maintains a maximum matching. The algorithm handles deletions as follows: let $\{u, v\}$ be the deleted edge. After the deletion, an augmenting path search is started from any free endpoint u or v. The algorithm uses two optimizations: *lazy augmenting path search*, in which an augmenting path search from u and v is only started if at least x edges have been inserted or deleted since the last augmenting path search from u or v or if no augmenting path search has been started so far. The second optimization limits the search depth of the augmenting path search to $2/\epsilon - 1$. This ensures that there is no augmenting path of length at most $2/\epsilon - 1$, thus yielding a deterministic $(1+\epsilon)$-approximate matching algorithm (if the *safe* option is used).

Corollary 4.2. *If the algorithm by Stubbs and Williams uses the blossom-based MCM algorithm on each level s. t. the depth is bounded by* $2/\epsilon - 1$*, then the resulting* DynMWMLevelB *algorithm is* $2(1+\epsilon)^2$*-approximate with update time* $\mathcal{O}(\Delta^{2/\epsilon-1}(\log_{1+\epsilon} N/L)^2)$.

5 Experiments

Implementation and Compute System. We implemented the algorithms described above in `C++` compiled with `g++-9.3.0` using the optimization flag

`-O3`. For the dynamic MCM algorithms, we use the implementation by Henzinger et al. [33]. All codes are sequential. We intend to release our implementation after paper acceptance.

Our algorithms use the following dynamic graph data structure. For each node u, we maintain a vector L_u of adjacent nodes, and a hash table $\mathcal{H}_u$ that maps a neighbor v of u to its position in L_u. This data structure allows for expected constant time insertion and deletion as well as a constant time operation to select a random neighbor of u. The deletion operation on $\{u, v\}$ is implemented as follows: get the position of v in L_u via a lookup in $\mathcal{H}_u(v)$. Swap the element in L_u with the last element w in the vector and update the position of w in $\mathcal{H}_u$. Finally, pop the last element (now v) from L_u and delete its entry from $\mathcal{H}_u$.

Our experiments are conducted on one core of a machine with a AMD EPYC 7702P 64-core CPU (2 GHz) and 1 TB of RAM. In order to be able to compare our results with optimum weight matchings, we use the LEMON library [20], which provides algorithms for static weighted maximum matching.

Methodology. By default we perform ten repetitions per instance. We measure the total time taken to compute all edge insertions and deletions and generally use the *geometric mean* when averaging over different instances in order to give every instance a comparable influence on the final result. In order to compare different algorithms, we use *performance profiles* [22]. These plots relate the matching weight of all algorithms to the optimum matching weight. More precisely, the y-axis shows $\#\{\mathit{objective} \geq \tau \times \mathit{OPT}\}/\#\text{instances}$, where *objective* corresponds to the result of an algorithm on an instance and OPT refers to the optimum result. The parameter $\tau \leq 1$ in this equation is plotted on the x-axis.

Instances. We evaluate our algorithms on graphs collected from various resources [5, 19, 32, 38, 39, 48], which are also used in related studies [33]. Table 1 summarizes the main properties of the benchmark set. Our benchmark set includes a number of graphs from numeric simulations as well as complex networks. These include static graphs as well as real dynamic graphs. As our algorithms only handle undirected simple graphs, we ignore possible edge directions in the input graphs, and we remove self-loops and parallel edges. We perform *two different types* of experiments. First, we use the algorithms using insertions only, i.e., we start with an empty graph and insert all edges of the static graph in a random order. We do this with all graphs from Table 1. Second, we use real dynamic instances from Table 2. Most of these instances, however, only feature insertions (exceptions: `dewiki` and `wiki-simple-en`). Hence, we perform additional experiments with fully-dynamic graphs from these inputs, by undoing x percent of the update operations performed last (in the opposite order of their appearance). As the instances are unweighted, we generated weights for edges uniformly at random in $[1, 100]$.

Table 2: Basic properties of the benchmark set of dynamic graphs with number of update operations $\mathcal{U}$. Most of the graphs only feature insertions. The only two exceptions are marked with a *. All of these graphs have been obtained from the KONECT graph database [48].

graph	n	$\mathcal{U}$
amazon-ratings	2 146 058	5 838 041
citeulike_ui	731 770	2 411 819
dewiki*	2 166 670	86 337 879
dnc-temporalGraph	2 030	39 264
facebook-wosn-wall	46 953	876 993
flickr-growth	2 302 926	33 140 017
haggle	275	28 244
lastfm_band	174 078	19 150 868
lkml-reply	63 400	1 096 440
movielens10m	69 879	10 000 054
munmun_digg	30 399	87 627
proper_loans	89 270	3 394 979
sociopatterns-infections	411	17 298
stackexchange-overflow	545 197	1 301 942
topology	34 762	171 403
wikipedia-growth	1 870 710	39 953 145
wiki_simple_en*	100 313	1 627 472
youtube-u-growth	3 223 590	9 375 374

5.1 DynMWMRandom. First, we look at DynMWMRandom (in various configurations) on insertions-only instances. We start with the effect of the number of walks done after each update and the impact of the "stop early" heuristic. In Fig 1 (left), we look at different numbers of walks when (not) using the "stop early" heuristic (if used, ϵ is set to 10^{-3}). Geometric mean values for running time and achieved matching weight are in Table 3. As expected, increasing the number of walks increases the weight of the computed matchings (with and without "stop early" heuristic). With the "stop early" heuristic (in our experiments $\beta = 5$), increasing the number of walks more than ten does not significantly improve the solutions. When disabling the "stop early" heuristic, solutions improve again, even for the same number of 20 walks. The strongest configuration in this setting uses 100 walks with the "stop early" heuristic being enabled. The configuration is as close as 4% to the optimum on more than 95% of the instances. Note that the number of walks done here is much less than what Lemma 4.1 suggests to achieve a $(1 + \epsilon)$-approximation. Moreover, using more walks

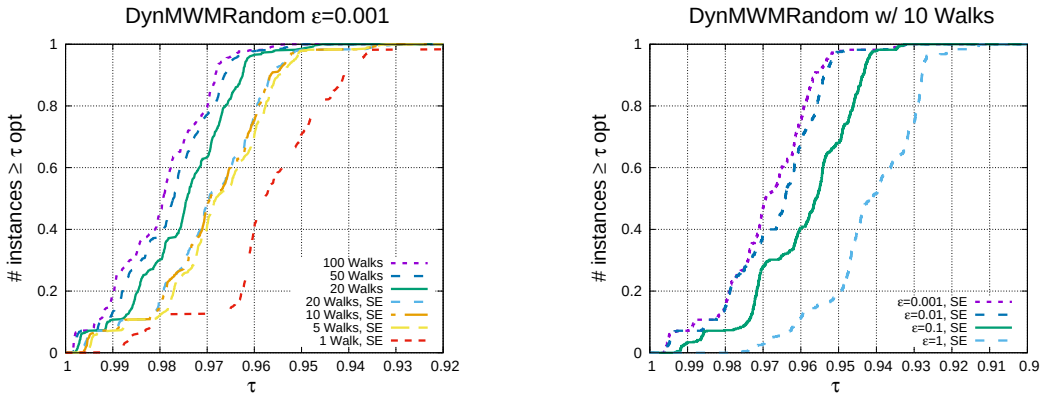


Figure 1: Left: Number of random walks and random walks with "stop early" heuristic (SE) for fixed ϵ. Right: Various values of ϵ for fixed number of walks.

Table 3: Top: Geometric mean running times and weight of the final matching for fixed $\epsilon = 0.001$ and varying number of walks. Bottom: Geometric mean running times and weight of final matching for fixed number of walks (10) and varying ϵ.

# walks	geo. mean t [s]	geo. mean $\omega(\mathcal{M})$
with SEHeuristic		
1	4.0	1572285.92
5	16.4	+1.15%
10	20.5	+1.28%
20	21.8	+1.31%
without SEHeuristic		
20	56.4	+1.91%
50	128.6	+2.24%
100	241.6	+2.44%
ϵ	geo. mean t [s]	geo. mean $\omega(\mathcal{M})$
with SEHeuristic		
1	2.4	1547129.56
10^{-1}	5.9	+1.84%
10^{-2}	13.7	+2.71%
10^{-3}	20.5	+2.93%

as well as not using the "stop early" heuristic also comes at the cost of additional running time. For example, for $\epsilon = 0.001$ and 20 walks, using the "stop early" heuristic saves roughly a factor of three in running time. We conclude that for a fixed ϵ, the number of walks and the "stop early" heuristic yield a favorable speed quality trade-off.

Next, we fix the use of the "stop early" heuristic and the number of walks per update to ten. Figure 1 (right) shows the performance compared to the optimum solution value and Table 3 gives geometric mean values for running time and matching weight again. When using $\epsilon = 1$, the algorithm computes in all cases at least a matching that has 91.2% of the weight of the optimum matching. As soon as we decrease ϵ, solutions improve noticeably. For the case $\epsilon = 10^{-1}$, solutions improve by 1.84% on average. The algorithm always computes a matching which has at least 93.3% of the optimum weight matching. However, the parameter ϵ seems to be diminishing in the sense that decreasing it from 10^{-2} to 10^{-3} does not significantly improve solutions anymore. Also note that the running time grows much slower than $1/\epsilon$. This is due to the "stop early" heuristic. Overall, we conclude here that both the number of walks and the parameter ϵ are very important to maintain heavy matchings.

On the other hand, it is important to mention that in the unweighted case the quality of the algorithms based on random walks is not very sensitive to both parameters, ϵ and the number of walks [33]. That is why Henzinger et al. [33] use only one random walk and vary only ϵ in their final MCM algorithm. The different behavior comes from two observations: first, in the unweighted case many subpaths of an augmenting path are often interchangeable. In the weighted case, this may not be the case and hence it can be hard to find some of the remaining augmenting paths in later stages of the algorithms. Moreover, the algorithms differ in the way the random walkers work. Our weighted algorithm builds a path and then solves a dynamic program on it (hence, the length of a random walker is limited by n). In the unweighted case, a random walker does changes to the graph on the fly and undoes changes in the end if it was unsuccessful. Hence, it can in principle run much longer than n steps.

In terms of running time, we can give a rough estimate of a speedup that would be obtained when running the optimal algorithm after each update. Note that the comparison is somewhat unfair since we compare an optimal algorithm with a heuristic algorithm and since we did not run the optimum algorithm after each update step due to excessive time necessary to run this experiment. On the largest instance of the set, eu-2005,

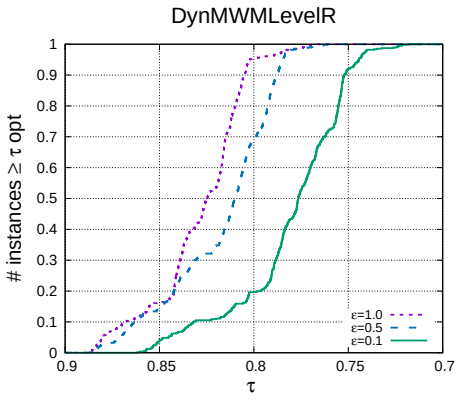


Figure 2: Performance profile of DynMWMLevelR instantiated with random-walk based algorithms as MCM algorithms for different values of ϵ.

the time per update of the slowest dynamic algorithm in these experiments ($\epsilon = 0.001$, 100 walks, no stop heuristic) is 0.00227s on average. On the other hand, the optimum algorithm needs 79.8998s to compute the solution on the final graph. This is more than four orders of magnitude difference. On the other hand, the fastest algorithms in this experiment are at least one additional order of magnitude faster.

5.2 DynMWMLevel. We now evaluate the performance of our implementation DynMWMLevel of the meta-algorithm by Crouch, Stubbs, and Williams. First, we use $\alpha = (1+\epsilon)$, so the algorithms under investigation have approximation guarantee of $2(1+\epsilon)^2$. Note that decreasing ϵ increases both, the number of levels in the algorithm necessary as well as the work performed by the unweighted matching algorithms on each level of the hierarchy. The results for the meta-algorithm are roughly the same when using the unweighted random walk algorithm compared to using the unweighted blossom-based algorithm for the same value of ϵ on each level. For example, the difference between DynMWMLevelR and DynMWMLevelB is smaller than 0.5% on average for $\epsilon = 0.1$ (with random walks having the slight advantage). Moreover, both instantiations have roughly the same running time on average (less than 1.6% difference on average). Hence, we focus our presentation on DynMWMLevelR. Figure 2 shows three different configurations of the algorithm using different values of $\epsilon \in \{1, 0.5, 0.1\}$. We do not run smaller values of ϵ as the algorithm creates a lot of levels for small ϵ and thus needs a large amount of memory. This is due to the large amount of levels created for small values of ϵ. To give a rough estimate, on the smallest graph from the collection, add20, DynMWMLevelR with $\epsilon = 0.1$ needs more than a factor 40 more memory than DynMWMRandom.

First, note that the results are significantly worse than the results achieved by DynMWMRandom: DynMWMLevelR computes at least a weighted matching that has 71.8% of the optimum weight matching for $\epsilon = 0.1$. Note that this is very well within the factor 2 that theory suggests. However, for larger values of ϵ, the algorithm computes even better results. For $\epsilon = 0.5$ the algorithm computes a weighted matching of at least 76.1% of the optimum weight matching, and for $\epsilon = 1$, the algorithm computes 76.8% of the optimum weight matching. We believe that this somewhat surprising behavior is due to two reasons. First, if one only increases the work done by the MCM algorithms on each level, then the cardinality of the matchings on each level increases as expected. However, the weight of the computed matching on each level does not necessarily increase. Hence, the level-based meta-algorithm has a too strong preference on the cardinality as it ignores the weights on each level. Moreover, when increasing the number of levels, the priority of heavy edges increases, as there are only very few of them at the highest levels in the hierarchy. Hence, the algorithm becomes closer to a greedy algorithm that picks heavy edges first.

In terms of running time, DynMWMLevelR uses 1.53s, 2.66s, and 14.61s on average for $\epsilon = 1$, 0.5, and 0.1, respectively. Moreover, the best and fastest configuration in this category uses $\epsilon = 1$, computes 11% smaller matchings than the augmented random walk for $\epsilon = 1$ (recall that the number of steps of the walk is $2/\epsilon + 3$)) and the geometric mean running time of the random walk is 0.61s. We conclude here that DynMWMRandom is better and faster than DynMWMLevel. Moreover, augmented random walks need significantly less memory.

5.3 Real-World Dynamic Instances. We now switch to the real-world dynamic instances. Note that most of these instances are insertion-only (with two exceptions); however, the insertions are ordered in the way they appear in the real world. Hence, we perform additional experiments with fully-dynamic graphs from these inputs, by undoing x percent of the update operations performed last (call them $\mathcal{U}_x$). More precisely, we perform the operations in $\mathcal{U}_x$ in reverse order. If an edge operation was an insertion in $\mathcal{U}_x$, we perform a delete operation and if it was a delete operation, we insert it. However, when reinserting an edge, we use a new random weight. In our experiments, we perform 0%, 10% and 25% undo operations. As before, we compute the update on the graph after each removal/insertion. Table 4 summarizes the results of the experiment and Figure 3 compares the final quality of the dynamic algorithms with the optimum on the fi-

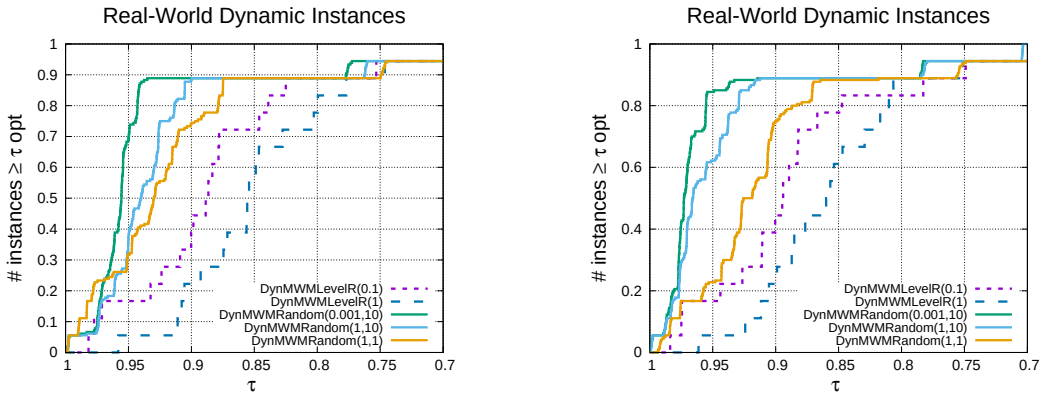


Figure 3: Comparison on real-world dynamic instances without (left) and with (right) 25% undo operations.

Table 4: Geometric mean results for real-world dynamic instances with and without undo operations depending on ϵ and walks per update (w). Smaller is better.

Algorithm	$\overline{t[s]}$	$\overline{OPT/\omega(\mathcal{M})}$
without undo ops		
Random($\epsilon = 10^{-3}$, w=10)	11.2	7.3%
Random($\epsilon = 1$, w=10)	3.9	9.0%
Random($\epsilon = 1$, w=1)	1.0	10.0%
LevelR($\epsilon = 10^{-1}$)	17.3	14.0%
LevelR($\epsilon = 1$)	1.7	18.7%
with 10% undo ops		
Random($\epsilon = 10^{-3}$, w=10)	14.3	6.2%
Random($\epsilon = 1$, w=10)	5.0	7.2%
Random($\epsilon = 1$, w=1)	1.2	10.7%
LevelR($\epsilon = 10^{-1}$)	18.6	13.5%
LevelR($\epsilon = 1$)	1.9	17.9%
with 25% undo ops		
Random($\epsilon = 10^{-3}$, w=10)	18.1	6.0%
Random($\epsilon = 1$, w=10)	6.3	7.0%
Random($\epsilon = 1$, w=1)	1.5	10.9%
LevelR($\epsilon = 10^{-1}$)	20.5	13.8%
LevelR($\epsilon = 1$)	2.1	17.5%

nal graph after all updates have been performed. For our experiments with DynMWMRandom, we use the configurations $\epsilon = 10^{-3}$ with 10 walks, $\epsilon = 1$ with 10 walks, and $\epsilon = 1$ with 1 walk. In any case, for these algorithms the "stop early" heuristic is enabled. For the DynMWMLevelR algorithm, we use $\epsilon = 10^{-1}$ and $\epsilon = 1$. Overall, the results confirm the results of the previous section. For example, DynMWMRandom ($\epsilon = 0.001$, 10 walks per update), computes matchings with 93.5% weight of the optimum weight matching on more than 89% of the instances. The fastest configuration DynMWMRandom ($\epsilon = 1$, 1 walk per update), is also already very good. It computes 90% of the optimum weight matching on 89% of the instances. On the other hand, DynMWMLevelR computes 82.5% and 77.7% of the optimum weight matching on 88.9% of the instances for $\epsilon = 0.1$ and $\epsilon = 1$, respectively.

Moreover, the experiments indicate that DynMWMLevelR is always outperformed in terms of quality and speed by a configuration of DynMWMRandom. This is true with and without running undo operations. For example, without running undo operations, DynMWMLevelR ($\epsilon = 10^{-1}$) computes solutions that are 14% smaller than the optimum matching weight on average while taking 17.3s running time. On the other hand, already DynMWMRandom ($\epsilon = 1$, 1 walk per update) computes weighted matchings that are 10% worse than the optimum solution while needing only 1s running time on average. The same comparison can be done when undo operations are performed, i.e., DynMWMLevelR ($\epsilon = 10^{-1}$) is always outperformed by DynMWMRandom ($\epsilon = 1$, 1 walk).

When considering undo operations, the quality of all algorithms is stable, i.e., there is no reduction in quality observable over the case without undo operations. Note that in contrast the quality in terms of distance to OPT often improves when undo operations are performed. This is due to the fact that the algorithms perform additional work when the undo operations are called. Hence, the results may find additional augmenting paths that had not been found when the algorithm was at the original state of the graph.

6 Conclusions

Our experiments show that fully-dynamic approximation algorithms for the MWM problem are viable alternatives compared to an exact approach. Of the algorithms we implemented, DynMWMRandom fares best overall and should be the method of choice whenever exact solutions are not really necessary. After all, the solutions are often close to the optimum.

References

[1] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. *CoRR*, abs/2010.05846, 2020. URL: https://arxiv.org/abs/2010.05846, arXiv:2010.05846.

[2] Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, LIPIcs, pages 257–266. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.257.

[3] Eugenio Angriman, Henning Meyerhenke, Christian Schulz, and Bora Uçar. Fully-dynamic weighted matching approximation in practice. *CoRR*, abs/2104.13098, 2021. URL: https://arxiv.org/abs/2104.13098, arXiv:2104.13098.

[4] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.7.

[5] D. Bader, A. Kappes, H. Meyerhenke, P. Sanders, C. Schulz, and D. Wagner. Benchmarking for Graph Clustering and Partitioning. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 2018. doi:10.1007/978-1-4939-7131-2_23.

[6] Leonid Barenboim and Tzalik Maimon. Fully dynamic graph algorithms inspired by distributed computing: Deterministic maximal matching and edge coloring in sublinear update-time. *ACM J. Exp. Algorithmics*, 24(1):1.14:1–1.14:24, 2019. doi:10.1145/3338529.

[7] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in $O(\log n)$ update time. *SIAM J. Comput.*, 44(1):88–113, 2015. doi:10.1137/16M1106158.

[8] Claude Berge. Two theorems in graph theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957. URL: http://www.pnas.org/content/43/9/842, arXiv:http://www.pnas.org/content/43/9/842.full.pdf, doi:10.1073/pnas.43.9.842.

[9] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *Proceedings of the 27th Symposium on Discrete Algorithms SODA*, pages 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.ch50.

[10] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *19th Intl. Conf. on Integer Programming and Combinatorial Optimization IPCO*, volume 10328 of *LNCS*, pages 86–98. Springer, 2017. doi:10.1007/978-3-319-59250-3_8.

[11] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018. doi:10.1137/140998925.

[12] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual Symposium on Theory of Computing*, pages 398–411. ACM, 2016. doi:10.1145/2897518.2897568.

[13] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In Philip N. Klein, editor, *Proc. of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 470–489. SIAM, 2017. doi:10.1137/1.9781611974782.30.

[14] Sonja Biedermann, Monika Henzinger, Christian Schulz, and Bernhard Schuster. Vienna graph clustering. In Stefan Canzar and Francisca Rojas Ringeling, editors, *Protein-Protein Interaction Networks, Methods and Protocols*, volume 2074 of *Methods in Molecular Biology*, pages 215–231. Springer, 2020. doi:10.1007/978-1-4939-9873-9_16.

[15] Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava. Efficient parallel and external matching. In *Euro-Par 2013*, volume 8097 of *LNCS*, pages 659–670. Springer, 2013. doi:10.1007/978-3-642-40047-6_66.

[16] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent Advances in Graph Partitioning. In *Algorithm Engineering - Selected Results and Surveys*, pages 117–158. Springer, 2016.

[17] Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *45th International Colloquium on Automata, Languages, and Programming ICALP*, volume 107 of *LIPIcs*, pages 33:1–33:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.33.

[18] Michael Crouch and Daniel S. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014*, volume 28 of *LIPIcs*, pages 96–104, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.96.

[19] T. Davis. The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices, 2008. URL: http://www.cise.ufl.edu/research/sparse/matrices/.

[20] Balázs Dezső, Alpár Jüttner, and Péter Kovács. LEMON – an Open Source C++ Graph Template Library. *Electronic Notes in Theoretical Computer Science*, 264(5):23 – 45, 2011. URL: http://www.sciencedirect.com/science/article/pii/S1571066111000740, doi:https://doi.org/10.1016/j.entcs.2011.06.003.

[21] John P. Dickerson, Ariel D. Procaccia, and Tuomas Sandholm. Dynamic matching via weighted myopia with application to kidney exchange. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5031.

[22] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, 2002. doi:10.1007/s101070100263.

[23] D. Drake and S. Hougardy. A Simple Approximation Algorithm for the Weighted Matching Problem. *Information Processing Letters*, 85(4):211–213, 2003. doi:10.1016/S0020-0190(02)00393-9.

[24] Andre Droschinsky, Petra Mutzel, and Erik Thordsen. Shrinking trees not blossoms: A recursive maximum matching approach. In *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020*, pages 146–160. SIAM, 2020. doi:10.1137/1.9781611976007.12.

[25] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[26] Harold N. Gabow and Robert Endre Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991. doi:10.1145/115234.115366.

[27] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, 1986. doi:10.1145/6462.6502.

[28] Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1 + \epsilon)$-approximate incremental matching in constant deterministic amortized time. In *Proceedings of the 20th Symposium on Discrete Algorithms*, pages 1886–1898. SIAM, 2019. doi:10.1137/1.9781611975482.114.

[29] Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Symposium on Foundations of Computer Science, FOCS*, pages 548–557. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.65.

[30] Manoj Gupta and Richard Peng. Fully Dynamic $(1 + \epsilon)$-Approximate Matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science*, pages 548–557. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.65.

[31] Mahantesh Halappanavar. *Algorithms for Vertex-Weighted Matching in Graphs*. PhD thesis, Old Dominion University, May 2009.

[32] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. Recent advances in fully dynamic graph algorithms. *CoRR*, abs/2102.11169, 2021. URL: https://dyngraphlab.github.io/, arXiv:2102.11169.

[33] Monika Henzinger, Shahbaz Khan, Richard Paul, and Christian Schulz. Dynamic Matching Algorithms in Practice. In *28th Annual European Symposium on Algorithms*, volume 173 of *LIPIcs*, pages 58:1–58:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.58.

[34] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite. In *12th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 122–125, 1971. doi:10.1109/SWAT.1971.1.

[35] Zoran Ivkovic and Errol L. Lloyd. Fully dynamic maintenance of vertex cover. In *19th International Workshop Graph-Theoretic Concepts in Computer Science*, volume 790 of *LNCS*, pages 99–111, 1993. doi:10.1007/3-540-57899-4_44.

[36] Manas Jyoti Kashyop and N.S. Narayanaswamy. Lazy or eager dynamic matching may not be fast. *Information Processing Letters*, 162:105982, 2020. URL: http://www.sciencedirect.com/science/article/pii/S0020019020300697, doi:https://doi.org/10.1016/j.ipl.2020.105982.

[37] Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data Reduction for Maximum Matching on Real-World Graphs: Theory and Experiments. In *26th European Symposium on Algorithms ESA*, volume 112 of *LIPIcs*, pages 53:1–53:13, 2018. doi:10.4230/LIPIcs.ESA.2018.53.

[38] Jérôme Kunegis. KONECT: the koblenz network collection. In *22nd World Wide Web Conference, WWW '13*, pages 1343–1350, 2013. doi:10.1145/2487788.2488173.

[39] J. Lescovec. Stanford Network Analysis Package (SNAP). http://snap.stanford.edu/index.html.

[40] Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 519–528. IEEE Computer Society, 2014. doi:10.1109/IPDPS.2014.61.

[41] J. Maue and P. Sanders. Engineering Algorithms for Approximate Weighted Matching. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA'07)*, volume 4525 of *LNCS*, pages 242–255. Springer, 2007. URL: http://dx.doi.org/10.1007/978-3-540-72845-0_19.

[42] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized on-line matching. In *46th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 264–273. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.12.

[43] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *21st Symposium on Foundations of Computer Science*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

[44] Marcin Mucha and Piotr Sankowski. Maximum matchings in planar graphs via gaussian elimination. *Algorithmica*, 45(1):3–20, 2006. doi:10.1007/s00453-005-1187-5.

[45] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. doi:10.1145/2700206.

[46] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464, 2010. doi:10.1145/1806689.1806753.

[47] Alex Pothen, S M Ferdous, and Fredrik Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019. doi:10.1017/S0962492919000035.

[48] Julia Preusse, Jérôme Kunegis, Matthias Thimm, Thomas Gottron, and Steffen Staab. Structural dynamics of knowledge networks. In *Proc. Int. Conf. on Weblogs and Social Media*, 2013. URL: http://www.aaai.org/ocs/index.php/ICWSM/ICWSM13/paper/view/6076.

[49] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 118–126. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283397.

[50] Piotr Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theor. Comput. Sci.*, 410(44):4480–4488, 2009. doi:10.1016/j.tcs.2009.07.028.

[51] Shay Solomon. Fully dynamic maximal matching in constant update time. In *57th Symposium on Foundations of Computer Science FOCS*, pages 325–334, 2016. doi:10.1109/FOCS.2016.43.

[52] Daniel Stubbs and Virginia Vassilevska Williams. Metatheorems for dynamic weighted matching. In *8th Innovations in Theoretical Computer Science Conference*, volume 67 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ITCS.2017.58.

[53] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.

A Parallel Approximation Algorithm for Maximizing Submodular b-Matching *

S M Ferdous† Alex Pothen† Arif Khan‡ Ajay Panyala§ Mahantesh Halappanavar‡

Abstract

We design new serial and parallel approximation algorithms for computing a maximum weight b-matching in an edge-weighted graph with a submodular objective function. This problem is NP-hard; the new algorithms have approximation ratio 1/3, and are relaxations of the Greedy algorithm that rely only on local information in the graph, making them parallelizable. We have designed and implemented Local Lazy Greedy algorithms for both serial and parallel computers. We have applied the approximate submodular b-matching algorithm to assign tasks to processors in the computation of Fock matrices in quantum chemistry on parallel computers. The assignment seeks to reduce the run time by balancing the computational load on the processors and bounding the number of messages that each processor sends. We show that the new assignment of tasks to processors provides a four fold speedup over the currently used assignment in the NWChemEx software on 8000 processors on the Summit supercomputer at Oak Ridge National Lab.

1 Introduction

We describe new serial and parallel approximation algorithms for computing a maximum weight b-Matching in an edge-weighted graph with a submodular objective function. This problem is NP-hard; the new algorithms have approximation ratio 1/3, and are variants of the Greedy algorithm that rely only on local information in the graph, making them parallelizable. We apply the approximate submodular b-Matching algorithm to assign tasks to processors in the computation of Fock matrices in quantum chemistry on parallel computers, in order to balance the computational load on the processors and bound the number of messages that a processor sends.

A b-Matching is a subgraph of the given input graph, where the degree of each vertex v is bounded by a given natural number $b(v)$. In linear (or modular) b-Matching the objective function is the sum of the weights of the edges in a b-Matching, and we seek to maximize this weight. The well-known maximum edge-weighted matching problem is the 1-matching problem. Although these problems can be solved in polynomial time, in recent years a number of approximation algorithms have been developed since the run time of exact algorithms can be impractical on massive graphs. These algorithms are based on the paradigms of short augmentations (paths that increase the cardinality or weight of the matching) [32]; relaxations of a global ordering (by non-increasing weights) of edges to local orderings [34]; partitioning heavy weight paths in the graph into matchings [12]; proposal making algorithms similar to stable matching [18, 26], etc. Some, though not all, of these algorithms are concurrent and can be implemented on parallel computers; a recent survey is available in [33].

Our algorithm employs the concept of an edge being locally dominant in its neighborhood that was first employed by Preis [34] to design the 1/2-approximate matching algorithm for (modular or linear) maximum weighted 1-matching; the approximation ratio is as good as the Greedy algorithm, and Preis showed that the algorithm could be implemented in time linear in the size of the graph. Since then there has been much work in implementing variants of the locally dominant edge algorithm for 1-matching and b-matching on both serial and parallel computational models (e.g., [18, 26]). More details are included in [33].

In this paper we employ the local dominance technique, relaxing global orderings to local orderings, to the b-Matching problem with submodular objective. We exploit the fact that the b-Matching problem may be viewed as a 2-extendible system, which is a relaxation of a matroid. We show that any algorithm that adds locally optimal edges, with respect to the marginal gain for a submodular objective function, to the matching preserves the approximation ratio of the corresponding global Greedy algorithm. This result offers a blueprint to design many approximation algorithms, of which we develop one: Local Lazy Greedy algorithm. Testing

*Research supported by NSF grant CCF-1637534, DOE grant DE-FG02-13ER26135, and the Exascale Computing Project (17-SC-20- SC), a collaborative effort of the DOE Office of Science and the NNSA.

†Computer Science Department, Purdue University, West Lafayette IN 47907 USA. {sferdou,apothen}@purdue.edu

‡Data Science and Machine Intelligence, Pacific Northwest National Lab, Richland WA 99352 USA. {ariful.khan,mahantesh.halappanavar}@pnnl.gov

§High Performance Computing, Pacific Northwest National Lab, Richland WA 99352 USA. ajay.panyala@pnnl.gov

for local optimality in the submodular objective is more expensive than in the linear case due to the variability of the marginal gain. To efficiently maintain marginal gains, we borrow an idea from the lazy evaluation of the Greedy algorithm. Combining local dominance and lazy Greedy techniques, we develop a Local Lazy Greedy algorithm, which is theoretically and practically faster than the Lazy Greedy algorithm. The runtime of both these algorithms are analyzed under a natural local dependence assumption on the submodular function. Since our algorithm is parallelizable thanks to the local orderings, we show good scaling performance on a shared memory parallel environment. To the best of our knowledge, this is the first parallel implementation of a submodular b-Matching algorithm.

Submodular b-Matching has applications in many real-life problems. Among these are content spread maximization in social networks [8], peptide identification in tandem mass spectrometry [2, 3], word alignment in natural language processing [25], and diversity maximizing assignment [1, 11]. Here we show another application of submodular b-Matching in load balancing the Fock matrix computation in quantum chemistry on a multiprocessor environment. Our approach enables the assignment of tasks to processors leading to scalable Fock matrix computations.

We highlight the following contributions:

- We show that any b-Matching that is ϵ-locally dominant w.r.t the marginal gain is $\frac{\epsilon}{2+\epsilon}$-approximate for submodular objective functions, and devise an algorithm, Local Lazy Greedy to compute such a matching. Under a natural local dependence assmption on the submodular function, this algorithm runs in $O(\beta\, m \log \Delta)$ time and is theoretically and practically faster than the popular Lazy Greedy algorithm. (Here m is the number of edges, Δ is the maximum degree of a vertex, and β is the maximum value of $b(v)$ over all vertices v.)

- We provide a shared memory parallel implementation of the Local Lazy Greedy algorithm. Using several real-world and synthetic graphs, we show that our parallel implementation scales to more than sixty-five cores.

- We apply submodular b-Matching to generate an assignment of tasks to processors for building Fock matrices in the NWChemEx quantum chemistry software. The current assignment method used there does not scale beyond 3000 processors, but our assignment shows a four-fold speedup per iteration of the Fock matrix computation, and scales to 8000 cores of the Summit supercomputer at ORNL.

2 Background

In this section we describe submodular functions and their properties, formulate the submodular b-Matching problem, and discuss approximation algorithms for the problem.

2.1 Submodular b-Matching

Definition 2.1. (Marginal gain) *Given a ground set X, the marginal gain of adding an element $e \in X$ to a set $A \subseteq X$ is*

$$\rho_e(A) = f(A \cup \{e\}) - f(A).$$

The marginal gain may be viewed as the discrete derivative of the set A for the element e. Generalizing, the marginal gain of adding a subset Q to another subset A of the ground set X is

$$\rho_Q(A) = f(A \cup Q) - f(A).$$

Definition 2.2. (Submodular set function) *Given a set X, a real-valued function f defined on the subsets of X is submodular if*

$$\rho_e(A) \geq \rho_e(B)$$

for all subsets $A \subseteq B \subseteq X$, and elements $e \in X \setminus B$. The function f is monotone if for all sets $A \subseteq B \subseteq X$, we have $f(A) \leq f(B)$; it is normalized if $f(\emptyset) = 0$.

We will assume throughout this paper that f is normalized. The concept of submodularity also extends to the addition of a set. Formally, for $Q \subseteq X \setminus B$, f is submodular if $\rho_Q(A) \geq \rho_Q(B)$. If f is monotone then $\rho_e(A) \geq 0$, $\forall A \subseteq X$ and $\forall e \in X$.

Proposition 2.1. *Let $S = \{e_1, \ldots, e_k\}$, S_i be the subset of S that contains the first i elements of S, and f be a normalized submodular function. Then $f(S) = \sum_{i=1}^{k} \rho_{e_i}(S_{i-1})$.*

Proposition 2.2. *For sets $A \subseteq B \subseteq X$, and $e \in X$, a monotone submodular function f defined on X satisfies $\rho_e(A) \geq \rho_e(B)$.*

Proof. There are three cases to consider. i) $e \in X \setminus B$: The inequality holds by definition of a submodular function. ii) $e \in A$: Then both sides of the inequality equal zero and the inequality holds again. iii) $e \in B \setminus A$: Then $\rho_e(B) = 0$, and since f is monotone, $\rho_e(A)$ is non-negative. ∎

Proposition 2.2 extends to a set, i.e., monotonicity of f implies that for every $A \subseteq B \subseteq X$, and $Q \subseteq X$, $\rho_Q(A) \geq \rho_Q(B)$. Informally Proposition 2.2 states that if f is monotone then the diminishing marginal gain property holds for every subset of X.

We are interested in maximizing a monotone submodular function with b-MATCHING constraints. Let $G(V, E, W)$ be a simple, undirected, and edge-weighted graph, where V, E, W are the set of vertices, edges, and non-negative edge weights, respectively. For each $e \in E$ we define a variable $x(e)$ that takes values from $\{0, 1\}$. Let $M \subseteq E$ denote the set of edges for which $x(e)$ is equal to 1, and let $\delta(v)$ denote the set of edges incident on the vertex $v \in V$. The submodular b-MATCHING problem is

$$\begin{aligned} &\max f(M) \\ &\text{subject to} \\ &\sum_{e \in \delta(v)} x(e) \leq b(v) \quad \forall v \in V, \\ &x(e) \in \{0, 1\}. \end{aligned} \tag{2.1}$$

Here f is a non-negative monotone submodular set function, and $0 \leq b(v) \leq |\delta(v)|$ is the integer degree bound on v. Denote $\beta = \max_{v \in V} b(v)$.

We now consider the b-MATCHING problem on a bipartite graph with two parts in the vertex set, say, U and V, where the objective function is a concave polynomial.

$$\begin{aligned} f = \max &\sum_{u \in U} \left(\sum_{e \in \delta(u)} W(e)x(e) \right)^{\alpha} \\ &+ \sum_{v \in V} \left(\sum_{e \in \delta(v)} W(e)x(e) \right)^{\alpha} \end{aligned} \tag{2.2}$$

$$\begin{aligned} &\text{subject to} \\ &\sum_{e \in \delta(u)} x(e) \leq b(u) \quad \forall u \in U, \\ &\sum_{e \in \delta(v)} x(e) \leq b(v) \quad \forall v \in V, \\ &x(e) \in \{0, 1\}. \end{aligned}$$

The objective function Problem 2.2 becomes submodular when $\alpha \in [0, 1]$. This formulation has been used for peptide identification in tandem mass spectrometry [2, 3], and word alignment in natural language processing [25].

2.2 Complexity of Submodular b-MATCHING and Approximation A *subset system* is a pair $(X, \mathcal{I})$, where X is a finite set of elements and $\mathcal{I}$ is a collection of subsets of X with the property that if $A \in \mathcal{I}$ and $A' \subseteq A$ then $A' \in \mathcal{I}$. A *matroid* is a subset system $(X, \mathcal{I})$ which satisfies the property that $\forall A, B \in \mathcal{I}$ and $|A| < |B|$, $\exists e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$. Here the sets in $\mathcal{I}$ are called *independent sets*. A *subset system* is *k-extendible* [27] if the following holds: let $A \subseteq B$, $A, B \in \mathcal{I}$ and $A \cup \{e\} \in \mathcal{I}$, where $e \notin A$, then there is a set $Y \subseteq B \setminus A$ such that $|Y| \leq k$ and $B \setminus Y \cup \{e\} \in \mathcal{I}$.

Maximizing a monotone submodular function with constraints is NP-hard in general; indeed, it is NP-hard for the simplest constraint of cardinality for many classes of submodular functions [13, 21]. A Greedy algorithm that repeatedly chooses an element with the maximum marginal gain is known to achieve $(1 - 1/e)$-approximation ratio [31], and this is tight [30]. The Greedy algorithm with matroid constraints is 1/2-approximate. More generally, with k-matroid intersection and k-extendible system constraints, the approximation ratio of the Greedy algorithm becomes $1/(k + 1)$ [7].

3 Related Work

Here we situate our contributions to submodular b-MATCHING in the broader context of earlier work in submodular maximization. A reader who is eager to get to the algorithms and results in this paper could skip this section on a first reading.

The maximum k-cover problem can be reduced to submodular b-MATCHING [16]. Feige [13] showed that there is no polynomial time algorithm for approximating the max k-cover within a factor of $(1 - 1/e + \epsilon)$ for any $\epsilon > 0$. Thus we obtain an immediate bound on the approximation ratio of submodular b-MATCHING.

New approximation techniques have been developed to expedite the greedy algorithm, especially for cardinality and matroid constraints. Surveys on submodular function maximization under different constraints may be found in [6, 20, 36].

Several approximation algorithms have been proposed for maximizing monotone submodular functions with b-MATCHING constraints. If the graph is bipartite, then the b-MATCHING constraint can be represented as the intersection of two partition matroids, and the Greedy algorithm provides a 1/3-approximation ratio. But b-MATCHING forms a 2-extendible system and the Greedy algorithm yields a 1/3-approximation ratio for non-bipartite graphs. Feldman *et al.* [14] showed that b-MATCHING is also a 2-exchange system, and they provide a $1/(2 + \frac{1}{p} + \epsilon)$-approximation algorithm based on local search, with time complexity $O(\beta^{p+1}(\Delta - 1)^p nm\epsilon^{-1})$. (Here p is a parameter to be chosen.) The continuous

greedy and randomized LP rounding algorithms have been used in [8] to compute a submodular b-MATCHING algorithm that produces a $(\frac{1}{3+2\epsilon})(1-\frac{1}{e})$ approximate solution in $O(m^5)$ time.

Recently Fuji [16] developed two algorithms for the problem. One of these, Find Walk, is a modified version of the Path Growing approximation algorithm [12] proposed for 1-matching with linear weights. Mestre [27] extended the idea to b-MATCHING. In [16], Fuji extended this further to the submodular objectives. They showed an approximation ratio of 1/4 with time complexity $O(\beta\, m)$. The second algorithm uses randomized local search, has an approximation ratio of $1/(2+\frac{1}{p})-\epsilon$, and runs in $O(\beta^{p+1}(\Delta-1)^{p-1}m\log\frac{1}{\epsilon}$ time in expectation. Here a vertex is chosen uniformly at random in each iteration, and the algorithm searches for a certain length alternating path with increasing weights. This algorithm is similar to the $2/3-\epsilon$ approximation algorithm for linear weighted matching in [32] and the corresponding b-MATCHING variant in [27]. We list several approximation algorithms for submodular b-MATCHING in Table 1.

Now we consider several related problems that do not have the b-MATCHING constraint.

Assigning tasks to machines is a classic scheduling problem. The most studied objective here is minimizing the makespan, i.e., the maximum total time used by any machine. The problem of makespan minimization can be generalized to a General Assignment Problem(GAP), where there is a fixed processing time and a cost associated with each task and machine pair. The goal is to assign the tasks into available machines with the assignment cost bounded by a constant C and makespan at most T. Shmoys and Tardos [35] extended the LP relaxation and rounding approach [24] to GAP. The makespan objective can be a surrogate to the load balancing that we are seeking, but the GAP problem does not encode the b-matching constraints on the machines. Computationally solving a GAP problem entails computing an LP relaxation that is expensive for large problems.

Alg.	Appx. Ratio	Time	Conc.?
Greedy[31]	1/3	$O(\beta nm)$	N
Lazy Greedy [28]	1/3	$O(\beta m\log m)$ assuming 4.1	N
Local Search [14]	$1/(2+\frac{1}{p}+\epsilon)$	$O(\beta^{p+1}(\Delta-1)^p nm\epsilon^{-1})$	N
Cont. Grdy+ Rand. Round[8]	$(\frac{1}{3+2\epsilon})(1-\frac{1}{e})$	$O(m^5)$	N
Path Growing [16]	1/4	$O(\beta m)$	N
Rand LS [16]	$1/3-\epsilon$	$O(\beta^2 m\log 1/\epsilon)$ in expectation	N
Local Lazy Greedy	$\frac{\epsilon}{2+\epsilon}$	$O(\beta m\log\Delta)$ assuming 4.1	Y

Table 1: Algorithms for the submodular b-MATCHING problem. The last column lists if the algorithm is concurrent or not.

Another possible approach is to model our load balancing problem as a multiple knapsack problem (MKP). In an MKP, we are given a set of n items and m knapsacks such that each item i has a weight (profit) w_i and a size s_i, and each knapsack j has a capacity c_j. The goal here is to find a subset of items of maximum weight such that they have a feasible packing in the knapsacks. MKP is a special case of GAP [9], and like the GAP, we cannot model the $b(v)$ constraints by MKP.

Our formulation of load balancing has the most similarity with the Submodular Welfare Maximization (SWM) problem [23]. In the SWM problem, the input consists of a set of n items to be assigned to one of m agents. Each agent j has a submodular function v_j, where $v_j(S)$ denotes the utility obtained by this agent if the set of items S is allocated to her. The goal is to partition the n items into m disjoint subsets $S_1,\ldots,S_m$ to maximize the total welfare, defined as $\sum_{j=1}^{m} v_j(S_j)$. The greedy algorithm achieves 1/2- approximation ratio [23]. Vondrak's $(1-1/e)$-approximation [37] is the best known algorithm for this problem. This algorithm uses continuous greedy relaxation of the submodular function and randomized rounding. Although we have modeled our objective as the sum of submodular functions, unlike the SWM, we have the same submodular function for each machine; our approach could be viewed as Submodular Welfare Maximization with b-matching constraints. In the original SWM problem, there are no constraints on the partition size, but in our problem we are required to set an upper bound on the individual partition sizes.

4 Greedy and Lazy Greedy Algorithms

A popular algorithm for maximizing submodular b-MATCHING is the GREEDY algorithm [31], where in each iteration, an edge with the maximum marginal gain is added to the matching. In its simplest form the GREEDY algorithm could be expensive to implement, but submodularity can be exploited to make it efficient. The efficient implementation is known as the LAZY GREEDY algorithm [22, 28]. As the maximum gain of each edge decreases monotonically in the course of the algorithm, we can employ a maximum heap to store the gains of the edges. Since the submodular function is normalized, the initial gain of each edge is just the function applied on the edge, and at each iteration we pop an edge e from the heap. If e is an *available edge*, i.e., e can be added to the current matching without violating b-MATCHING constraints,

we update its marginal gain $g(e)$. We compare $g(e)$ with the next best marginal gain of an edge, available as the heap's current top. If $g(e)$ is greater than or equal to the marginal gain of the current top, we add e to the matching; otherwise we push e to the heap. We iterate on the edges until the heap becomes empty. Algorithm 1 describes the Lazy Greedy approach.

Algorithm 1 Lazy Greedy Algorithm ($G(V, E, W)$)

```
pq = max heap of the edges keyed by marginal gain
while pq is not empty do
  Edge e = pq.pop()
  Update marginal gain of e
  if e is available then
    if marg_gain of e ≥ marg_gain of pq.top() then
      Add e to the matching
      update b(.) values of endpoints of e
    else
      push e and its updated gain into pq
    end if
  end if
end while
```

The maximum cardinality of a b-Matching is bounded by $\beta\, n$. In every iteration of the Greedy algorithm, an edge with maximum marginal gain can be chosen in $O(m)$ time. Hence the time complexity of the Greedy algorithm is $O(\beta\, n\, m)$. The worst-case running time of the Lazy Greedy algorithm is no better than the Greedy algorithm [28]. However, by making a reasonable assumption we can show a better time complexity bound for the Lazy Greedy algorithm.

The adjacent edges of an edge $e = (u, v)$ constitute the set $N(e) = \{e' : e' \in \delta(u) \text{ or } e' \in \delta(v)\}$. Likewise, the adjacent vertices of a vertex u are defined as the set $N(u) = \{v : (u, v) \in \delta(u)\}$.

Assumption 4.1. *The marginal gain of an edge e depends only on its adjacent edges.*

With this assumption, when an edge is added to the matching only the marginal gains of adjacent edges change. We make this assumption only to analyze the runtime of the algorithms but not to obtain the quality of the approximation. This assumption is applicable to the objective function in Problem 2.2 that has been used in many applications, including the one considered in this paper of load balancing Fock matrix computations.

Lemma 4.1. *Under Assumption 4.1, the time complexity of Algorithm 1 is $O(\beta\, m \log m)$.*

Proof. The time complexity of Algorithm 1 depends on the number of push and pop operations in the max heap. We bound how many times an edge e is pushed into the heap. The edge e is pushed when its updated marginal gain is less than the current top's marginal gain, and thus the number of times the marginal gain of e is updated is an upper bound on the number of push operations on it. From our assumption, the update of the marginal gain of an edge e can happen at most 2β times. Hence an edge is pushed into the priority queue $O(\beta)$ times, and each of these pushes can take $O(\log m)$ time. Thus the runtime for the all pushes is $O(\beta\, m \log m)$. The number of pop operations are at most the number of pushes. Thus the overall runtime of the Lazy Greedy algorithm for b-Matching is $O(\beta\, m \log m)$. ∎

5 Locally Dominant Algorithm

We introduce the concept of ϵ-local dominance, use it to design an approximation algorithm for submodular b-Matching, and prove the correctness of the algorithm.

5.1 ϵ-Local Dominance and Approximation Ratio The Lazy Greedy algorithm presented in Algorithm 1 guarantees a $\frac{1}{3}$ approx. ratio [7, 15] by choosing an edge with the highest marginal gain at each iteration, and thus it is an instance of a globally dominant algorithm. We will show that it is unnecessary to select a globally best edge because the same approximation ratio could be achieved by choosing an edge that is best in its neighborhood.

Recall that given a matching M, an edge e is *available w.r.t M* if both of its end-points are unsaturated in M.

Definition 5.1. (Locally dominant matching) *An edge e is* locally dominant *if it is available w.r.t a matching M, and the marginal gain of e is greater than or equal to all available edges adjacent to it. Similarly, for an $\epsilon \in (0, 1]$, an edge e is ϵ-* locally dominant *if its marginal gain is at least ϵ times the marginal gain of any of its available adjacent edges. A matching M is ϵ-*locally dominant *if every edge of M is ϵ-locally dominant when it is added to the matching.*

A globally dominant algorithm is also a locally dominant one. Thus our analysis of locally dominant matchings would establish the same approximation ratio for the Greedy and Lazy Greedy algorithms.

Theorem 5.1. *Any algorithm that produces an ϵ-locally dominant b-Matching is $\frac{\epsilon}{2+\epsilon}$-approximate for a submodular objective function.*

Proof. Let M^* denote an optimal matching and M be a matching produced by an ϵ-locally dominant algorithm.

Denote $|M| = k$. We order the elements of M such that when the edge e_i is included in M, it is an ϵ-locally dominant edge. Let M_i denote the locally dominant matching after adding e_i to the set, where $M_0 = \emptyset$ and $M_k = M$.

Our goal is to show that for each edge in the locally dominant algorithm, we may charge at most two distinct elements of M^*. At the ith iteration of the algorithm when we add e_i to M_{i-1}, we will show that there exists a distinct subset $M_i^* \subset M^*$ with $|M_i^*| \leq 2$ such that $\rho_{e_i}(M_{i-1}) \geq \epsilon\rho_{e_j^*}(M_{i-1})$, for all $e_j^* \in M_i^*$. We will achieve this by maintaining a new sequence of sets $\{T_j\}$, where T_{i-1} is the reservoir of potential edges that e_i could be charged to. The initial set of this sequence of sets is T_0, which holds the edges in the optimal matching M^*. The sequence of T-sets shrink in every iteration by removing the elements charged in the previous iteration, so that it stores only the candidate elements that could be charged in this and future iterations. Formally, $M^* = T_0 \supseteq T_1 \supseteq \cdots \supseteq T_k = \emptyset$ such that for $1 \leq i \leq k$, the following two conditions hold.
i) $M_i \cup T_i$ is also a b-Matching and
ii) $M_i \cap T_i = \emptyset$.
The two conditions are satisfied for M_0 and T_0 because $M_0 \cup T_0 = M^*$ and $M_0 \cap T_0 = \emptyset \cap M^* = \emptyset$.

Now we will describe the charging mechanism at each iteration. We need to construct the reservoir set T_i from T_{i-1}. Recall that e_i is added at the ith step of the ϵ-locally dominant matching to obtain M_i. There are two cases to consider:
i) If $e_i \in T_{i-1}$, the charging set $M_i^* = \{e_i\}$, $M_i = M_{i-1} \cup \{e_i\}$, and $T_i = T_{i-1} \setminus \{e_i\}$.
ii) Otherwise, let M_i^* be a smallest subset of T_{i-1} such that $(M_{i-1} \cup \ldots \cup \{e_i\} \cup T_{i-1}) \setminus M_i^*$ is a b-Matching. Since a b-matching is a 2-extendible system, we know $|M_i^*| \leq 2$. Then $M_i = M_{i-1} \cup \{e_i\}$; and $T_i = T_{i-1} \setminus M_i^*$. Note that the two conditions on M_i and T_i from the previous paragraph are satisfied after these sets are computed from M_{i-1} and T_{i-1}. Since M is a maximal matching, we have $T_k = \emptyset$; otherwise we could have added any of the available edges in T_k to M.

Now when e_i is added to M_{i-1}, all the elements of M_i^* are available. This set M_i^* must be the adjacent edges of e_i. Thus $\forall e_j^* \in M_i^*$, we have $\epsilon\rho_{e_j^*}(M_{i-1}) \leq \rho_{e_i}(M_{i-1})$. We can sum the inequality for each element of $e_j^* \in M_i^*$, leading to $\sum_j \rho_{e_j^*}(M_{i-1}) \leq \frac{2}{\epsilon}\rho_{e_i}(M_{i-1})$.

Rewriting the summation we have,

$$\begin{aligned}
\rho_{e_i}(M_{i-1}) \geq & \frac{\epsilon}{2}\sum_j \rho_{e_j^*}(M_{i-1}) \\
\geq & \frac{\epsilon}{2}\sum_j \rho_{e_j^*}(M_{i-1} \cup \{e_1^*, \ldots, e_{j-1}^*\}) \\
= & \frac{\epsilon}{2}\sum_j (f(M_{i-1} \cup \{e_1^*, \ldots, e_j^*\}) \\
& - f(M_{i-1} \cup \{e_1^*, \ldots, e_{j-1}^*\})) \\
= & \frac{\epsilon}{2}(f(M_{i-1} \cup \{e_1^*, \ldots, e_{|M_i^*|}^*\}) - f(M_{i-1})) \\
= & \frac{\epsilon}{2}(f(M_{i-1} \cup M_i^*) - f(M_{i-1})) \\
\geq & \frac{\epsilon}{2}(f(M \cup M_i^*) - f(M)).
\end{aligned}$$

In line 2, each of the summands is a superset of M_{i-1}, and the inequality follows from submodularity of f (Proposition 2.2). Line 3 expresses the marginal gains in terms of the function f. The fourth equality is due to telescoping of the sums, the fifth equality replaces the set M_i^* for its elements, and the last inequality follows by monotonicity of f (from Proposition 2.2).

We now sum over all the elements in M as follows.

$$\begin{aligned}
\sum_i \rho_{e_i}(M_{i-1}) \geq & \frac{\epsilon}{2}\sum_i (f(M \cup M_i^*) - f(M)), \\
f(M) \geq & \frac{\epsilon}{2}\sum_i (f(M \cup \{M_1^* \cup \ldots M_i^*\}) \\
& - f(M \cup \{M_1^*, \ldots, M_{i-1}^*\})) \\
= & \frac{\epsilon}{2}(f(M \cup M^*) - f(M)) \\
\geq & \frac{\epsilon}{2}(f(M^*) - f(M)). \\
f(M) \geq & \frac{\epsilon}{2+\epsilon} f(M^*).
\end{aligned}$$

The left side of the second line of the above equations is due to Proposition 2.1, while the right side comes from Proposition 2.2. The next equality telescopes the sum, and the fourth inequality is due to monotonicity of f. Finally the last line is a restatement of the inequality above it. ∎

Corollary 5.1. *Any algorithm that produces an ϵ-locally dominant semi-matching is $\frac{\epsilon}{1+\epsilon}$-approximate for a submodular objective function.*

Proof. A semi-matching (there are matching constraints on only one vertex part in a bipartite graph) forms a matroid, which is a 1-extendible system [27]. So by definition of 1-extendible system, $|M_i^*| \leq 1$. We can substitute this value in appropriate places in the proof of Lemma 5.1 and get the desired ratio. ∎

5.2 Local Lazy Greedy Algorithm Now we design a locally dominant edge algorithm to compute a b-MATCHING, outlining our approach in Algorithm 3. We say that a vertex v is *available* if there is an available edge incident on it, i.e., adding the edge to the matching does not violate the $b(v)$ constraint.

For each vertex $v \in V$, we maintain a priority queue that stores the edges incident on v. The key value of the queue is the marginal gain of the adjacent edges. At each iteration of the algorithm we alternate between two operations: *update* and *matching*. In the *update* step, we update a best incident edge of an unmatched vertex v. Similar to LAZY GREEDY, we can make use of the monotonicity of the marginal gains, and the lazy evaluation process is shown in Algorithm 2. After this step, we can consider a best incident edge for each vertex as a candidate to be matched. We also maintain an array (say *pointer*) of size $|V|$ that holds the best vertex found in the *update* step. The next step is the actual *matching*. We scan over all the available vertices $v \in V$ and check whether $pointer(v)$ also points to v (i.e., $pointer(pointer(v)) = v$). If this condition is true, we have identified a locally dominant edge, and we add it to the matching. We continue the two steps until no available edge remains.

Algorithm 2 Lazy Evaluation (Max Heap pq)

```
1: while pq is not empty do
2:    Edge e = pq.pop()
3:    Update marginal gain of e
4:    if e is available then
5:       if marg_gain of e ≥ marg_gain of pq.top() then
6:          break
7:       else
8:          push e and its updated gain into pq
9:       end if
10:   end if
11: end while
```

We omit the short proofs of the following two results.

LEMMA 5.1. *The* LOCAL LAZY GREEDY *algorithm is locally dominant.*

COROLLARY 5.2. *For the* b-MATCHING *problem with submodular objective, the* LOCAL LAZY GREEDY *algorithm is* $1/3$*-approximate.*

LEMMA 5.2. *Under Assumption 4.1, the time complexity of Algorithm 3 is* $O(\beta\, m \log \Delta)$.

Proof. As for the Lazy Greedy algorithm, the number of total push operations is $O(m\beta \log \Delta)$ (the argument of the logarithm is Δ instead of m because the maximum size of a priority queue is Δ). We maintain two arrays, say *PotentialU* and *PotentialM*, of vertices that hold the candidate vertices for iteration in the *update* and *matching* step, respectively. Initially all the vertices are in *PotentialU* and *PotentialM* is empty. The two arays are set to empty after their corresponding step. In the *update* phase, we insert the vertices for which the marginal gain changed into *PotentialM*. In the *matching* step, we iterate only over the vertices in *PotentialM* array. When an edge (u, v) is matched in the *matching* step, we insert u, v if they are unsaturated and all their available neighboring vertices into the *PotentialU*. This is the array on which in the next iteration, *update* would iterate. Since a vertex u can be inserted at most $b(u) + \sum_{v \in N(u)} b(v)$ times into the array, the overall size of *PotentialU* array during the execution of the algorithm is $O(m\beta)$. The *PotentialM* is always a subset of *PotentialU*. So it is also bounded by $O(m\beta)$.Combining all these we get, an $O(\beta\, m \log \Delta)$ time complexity. ∎

Algorithm 3 Local Lazy Greedy Algorithm

```
   ▷ Initialization
1: for v ∈ V do
2:    pq(v) := max-heap of the incident edges keyed by marginal gain
3:    pointer(v) = pq(v).top
4: end for

   ▷ Main Loop
5: while ∃ an edge with both its endpoints available do
   ▷ Updating
6:    for v ∈ V such that u is available do
7:       Update pq(v) using Lazy Evaluation (pq(v))
8:       pointer(v) = pq(v).top
9:    end for

   ▷ Matching
10:   for u ∈ V such that u is available do
11:      v = pointer(u)
12:      if v is available and pointer(v) == u then
13:         M = M ∪ {u, v}
14:      end if
15:   end for
16: end while
```

5.3 Parallel Implementaion of Local Lazy Greedy Both the standard GREEDY and LAZY GREEDY algorithm offer little to no concurrency. The GREEDY algorithm requires global ordering of

Algorithm 4 Parallel Local Lazy Greedy

▷ `Initialization`
1: **for** $v \in V$ **in parallel do**
2: pq(v) ≔ max-heap of the incident edges keyed by marginal gain
3: pointer(v) = pq(v).top
4: **end for**

▷ `Main Loop`
5: **while** ∃ an edge where both endpoints are available **do**
▷ `Updating`
6: **for** $v \in V$ such that v is available **in parallel do**
7: Update pq(v) according to Lazy Evaluation (pq(v))
8: pointer(v) = pq(v).top
9: **end for**

▷ `Matching`
10: **for** $u \in V$ such that u is available **in parallel do**
11: v = pointer(u)
12: **if** v is available and $u < v$ and pointer(v) == u **then**
13: Mark (u, v) as a matching edge
14: **end if**
15: **end for**
16: **end while**

the gains after each iteration, and the Lazy Greedy has to maintain a global priority queue. On the other hand, the Local Lazy Greedy algorithm is concurrent. Here local dominance is sufficient to maintain the desired approximation ratio. We present a shared memory parallel algorithm based on the serial Local Lazy Greedy in Algorithm 4.

One key difference between the parallel and the serial algorithms is on maintaining the *potentialU* and *potentialM* arrays. One option is for each of the processors to maintain individual *potentialU* and *potentialM* arrays and concatenate them after the corresponding steps. These arrays may contain duplicate vertices, but they can be handled as follows. We maintain a bit array of size of n initialized to 0 in each position. This bit array would be reset to 0 at every iteration. We only process vertices that have 0 in its corresponding position in the array. To make sure that only one processor is working on the vertex, we use an atomic `test-and-set` instruction to set the corresponding bit of the array. Thus the total work in the parallel algorithm is the same as of that the serial one i.e., $O(\beta\, m \log \Delta)$. Since the fragment inside the while loop is embarrassingly parallel, the parallel runtime depends on the number of iterations. This number depends on the weights and the edges in the graph, but in the worst case, could be $O(\beta n)$. We leave it for future work to bound the number of iterations under different weight distributions (say random) and different graph structures.

6 Experimental Results

The experiments on the serial algorithm were run on an Intel Haswell CPUs with 2.60 GHz clock speed and 512 GB memory. The parallel algorithm was executed on an Intel Knights Landing node with a Xeon Phi processor (68 physical cores per node) with 1.4 GHz clock speed and 96 GB DDR4 memory.

6.1 Dataset We tested our algorithm on both real-world and synthetic graphs shown in Table 2. (All Tables and Figures from this section are at the end of the paper.) We generated two classes of RMAT graphs: (a) G500, representing graphs with skewed degree distributions from the Graph 500 benchmark [29] and (b) SSCA, from the HPCS Scalable Synthetic Compact Applications graph analysis (SSCA#2) benchmark using the following parameter settings: (a) $a = 0.57$, $b = c = 0.19$, and $d = 0.05$ for G500, and (b) $a = 0.6$, and $b = c = d = 0.4/3$ for SSCA. Moreover, we considered eight problems taken from the SuiteSparse Matrix Collection [10] covering application areas such as medical science, structural engineering, and sensor data. We also included a large web-crawl graph(*eu-2015*) [4] and a movie-interaction network(*hollywood-2011*) [5].

6.2 Serial Performance In Table 3 we compare the Local Lazy Greedy algorithm with the Lazy Greedy algorithm. Each edge weight is chosen uniformly at random from the set $[1, 5]$. The submodular function employed here is the concave polynomial with $\alpha = 0.5$, and $b = 5$ for each vertex. Since both Lazy Greedy and Local Lazy Greedy algorithms have equal approximation ratios, the objective function values computed by them are equal, but the Local Lazy Greedy algorithm is faster. For the largest problem in the dataset, the Local Lazy Greedy algorithm is about five times faster than the Lazy Greedy, and it is about three times faster in geometric mean.

6.3 Parallel Performance Performance of the parallel implementations of the Local Lazy Greedy algorithm is shown by a scalibility plot in Figure 1. Figure 1 reports results from a machine with 68 threads, with all the cores on a single socket. We see that

all problems show good speedups, and all but three problems show good scaling with high numbers of threads.

7 Load Balancing in Quantum Chemistry

We show an application of submodular b-MATCHING in Self-Consistent Field (SCF) computations in computational chemistry [17].

7.1 Background The SCF calculation is iterative, and we focus on the computationally dominant kernel that is executed in every SCF iteration: the two-electron contribution to the Fock matrix build. The algorithm executes forty to fifty iterations to converge to a predefined tolerance.

The two-electron contribution involves a $\Theta(n^4)$ calculation over $\Theta(n^2)$ data elements, where n is the number of basis functions. The computation is organized as a set of n^4 tasks, where only a small percentage ($< 1\%$) of tasks contribute to the Fock matrix build. Before starting the main SCF iterative loop, the work required for the Fock matrix build in each iteration is computed from the number of nonzeros in the matrix, which is proportional to the work across all SCF iterations. This step is inexpensive since it only captures the execution pattern of the Fock matrix build algorithm without performing other computations. The task assignment is recorded prior to the first iteration and then reused across all SCF iterations.

The Fock matrix build itself is also iterative (written as a $\Theta(n^4)$ loop), where each iteration represents a task that computes some elements of the Fock matrix. For a given iteration, a task is only executed upon satisfying some domain constraints based on the values in two other pre-computed matrices, the Schwarz and density matrices.

The default load balancing used in NWChemEx [19] is to assign iteration indices of the outermost two loops in the Fock matrix build across MPI ranks using an atomic counter based work sharing approach. All MPI ranks atomically increment a global shared counter to identify the loop iterations to execute. This approach limits scalability of the Fock build since the work and number of tasks across MPI ranks are not guaranteed to be balanced.

The task assignment problem here naturally corresponds to a b-MATCHING problem. Let $G(U, V, W)$ be a complete bipartite graph, where U, V, W represent the sets of blocks of the Fock matrix, the set of machines, and the load of the (block,machine) pairs, respectively. The b value for each vertex in U is set to 1; for each vertex in V, it is set to $\lceil |U|/|V| \rceil$ in order to balance the number of MPI messages that each processor needs to send. We will show that a submodular objective with these b-MATCHING constraints implicitly encodes the desired load balance. To motivate this, we use the square root function ($\alpha = 0.5$) as our objective function in Eqn. (2.2).

We consider the execution of the Greedy algorithm for Submodular b-MATCHING on a small example consisting of four tasks with work loads of 300, 200, 100 and 50 on two machines $M1$ and $M2$. The b-MATCHING constraint requires each processor to be assigned two tasks. At the first iteration, we assign the first block (load 300) to machine $M1$. Note that assigning the second block to machine $M1$ would have the same marginal gain as assigning it to $M2$ if the objective function were linear. But since the square root objective function is submodular, the marginal gain of assigning the second block to the second machine is higher than assigning it to the first machine. So we will assign the second block (load 200) to machine $M2$. Then the third block of work 100 would be assigned to $M2$ rather than $M1$, due to the higher marginal gain, and finally the last block with load 50 would be assigned to $M1$ due to the b-MATCHING constraint. We see that modeling the objective by a submodular function implicitly provides the desired load balance, and the experimental results will confirm this.

7.2 Performance Results As a representative bio-molecular system we chose the Ubiquitin protein to test performance, varying the basis functions used in the computation to represent molecular orbitals, and to demonstrate the capability of our implementation to handle large problem sizes. The assignment algorithm is general enough to be applied to any scenario where such computational patterns exist, and does not depend on the molecule or the basis functions used.

We visualize the load on the processors in Fig. 2. The standard deviation for the current assignment is 10^5, and the coefficient of variation (Std./Avg.) is 7.5×10^{-2}; while these quantities for the submodular assignment are 436 and 3×10^{-4}, respectively. It is clear that the latter assignment achieves much better load balance than the former. The run time is plotted against the number of processors in Figure 3. It can be seen that the current assignment does not scale beyond 3000 processors, where as the submodular assignment scales to 8000 processors of Summit. The better load balance also leads to a four-fold speedup over the default assignment. Since the Fock matrix computation takes about fifty iterations, we reduce the total run time from 30 minutes to 8 minutes on Summit.

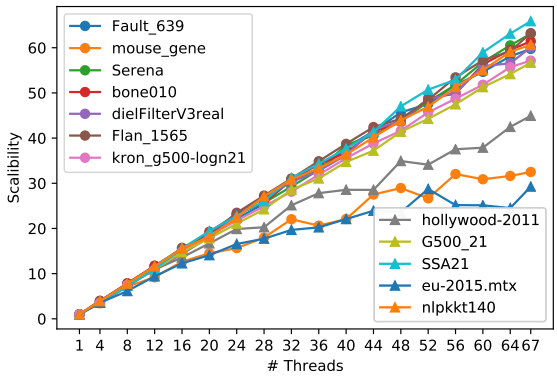


Figure 1: Scalability of the Local Lazy Greedy algorithm for submodular b-matching with 67 threads.

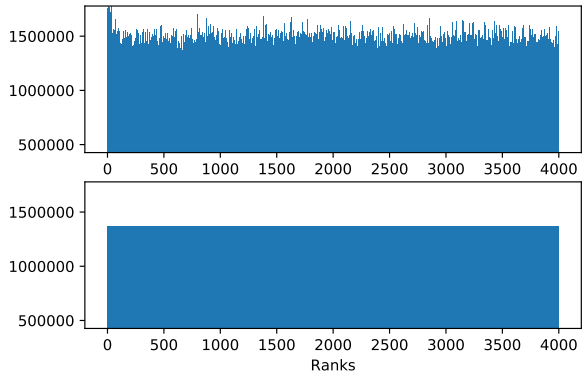


Figure 2: Visualizing the load distribution for the Fock matrix computation for the Ubiquitin protein. Results from: Top, current assignment on NWChemEx. Bottom, submodular assignment.

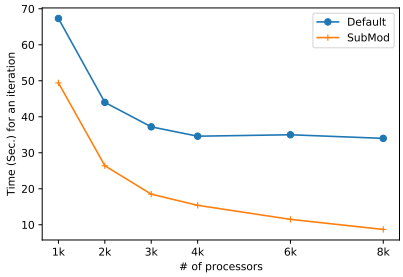


Figure 3: Runtime per iteration for the current (default) and submodular assignments with the 6-31g basis functions for the Ubiquitin protein in NWChemEx on Summit.

Problems	Vertices	Edges	Mean Degree
Fault_639	638,802	13,987,881	44
mouse_gene	45,101	14,461,095	641
Serena	1,391,349	31,570,176	45
bone010	986,703	35,339,811	72
dielFilterV3real	1,102,824	44,101,598	80
Flan_1565	1,564,794	57,920,625	74
kron_g500-logn21	2,097,152	91,040,932	87
hollywood-2011	2,180,759	114,492,816	105
G500_21	2,097,150	118,594,475	113
SSA21	2,097,152	123,097,397	117
eu-2015	11,264,052	257,659,403	46
nlpkkt240	27,993,600	373,239,376	27

Table 2: The properties of the test graphs listed by increasing number of edges.

Problems	Weight	Time (sec.)		Rel. Perf
		LG	LLG	LG/LLG
Fault_639	3.07E+06	61.05	16.83	3.63
mouse_gene	1.90E+05	50.68	22.41	2.26
Serena	6.69E+06	155.81	40.27	3.87
bone010	4.80E+06	177.37	44.15	4.02
dielFilterV3real	5.35E+06	221.92	62.22	3.57
Flan_1565	7.63E+06	310.31	72.00	4.31
kron_g500-logn21	3.69E+06	304.85	105.58	2.89
hollywood-2011	8.59E+06	622.73	163.26	3.81
G500_21	3.93E+06	344.13	137.06	2.51
SSA21	9.46E+06	588.16	285.79	2.06
eu-2015	2.40E+07	1098.40	396.16	2.77
nlpkkt240	1.31E+08	2456.34	465.30	5.28
Geo. Mean				**3.29**

Table 3: The objective function values and comparison of the serial run times for the Lazy Greedy and Local Lazy Greedy algorithms.

References

[1] Saba Ahmadi, Faez Ahmed, John P. Dickerson, Mark Fuge, and Samir Khuller. An algorithm for multi-attribute diverse matching. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3–9, 2020.

[2] Wenruo Bai, Jeffrey Bilmes, and William S Noble. Bipartite matching generalizations for peptide identification in tandem mass spectrometry. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 327–336, 2016.

[3] Wenruo Bai, Jeffrey Bilmes, and William S Noble. Submodular generalized matching for peptide identification in tandem mass spectrometry. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(4):1168–1181, 2018.

[4] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUbiNG: Massive crawling for the masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*, pages 227–228, 2014.

[5] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, pages 595–602, 2004.

[6] Niv Buchbinder and Moran Feldman. Submodular functions maximization problems. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 753–788. Chapman and Hall/CRC, 2018.

[7] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[8] Vineet Chaoji, Sayan Ranu, Rajeev Rastogi, and Rushi Bhatt. Recommendations to boost content spread in social networks. In *Proceedings of the 21st International Conference on World Wide Web*, pages 529–538, 2012.

[9] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[10] Tim Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1:1–1:25, 2011.

[11] John P Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Balancing relevance and diversity in online bipartite matching via submodularity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1877–1884, 2019.

[12] Doratha E Drake and Stefan Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003.

[13] Uriel Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[14] Moran Feldman, Joseph Seffi Naor, Roy Schwartz, and Justin Ward. Improved approximations for k-exchange systems. In *Proceedings of the European Symposium on Algorithms*, pages 784–798. Springer, 2011.

[15] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions—II. In M. L. Balinski and A. J. Hoffman, editors, *Polyhedral Combinatorics: Dedicated to the memory of D.R. Fulkerson*, pages 73–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[16] Kaito Fujii. Faster approximation algorithms for maximizing a monotone submodular function subject to a b-matching constraint. *Information Processing Letters*, 116(9):578–584, 2016.

[17] Tracy P Hamilton and Henry F Schaefer III. New variations in two-electron integral evaluation in the context of direct SCF procedures. *Chemical Physics*, 150(2):163–171, 1991.

[18] Arif Khan, Alex Pothen, Md Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Fredrik Manne, Mahantesh Halappanavar, and Pradeep Dubey. Efficient approximation algorithms for weighted b-matching. *SIAM Journal on Scientific Computing*, 38(5):S593–S619, 2016.

[19] Karol Kowalski, Raymond Bair, Nicholas P. Bauman, Jeffery S. Boschen, Eric J. Bylaska, Jeff Daily, Wibe A. de Jong, Thom Dunning, Niranjan Govind, Robert J. Harrison, Murat Keçeli, Kristopher Keipert, Sriram Krishnamoorthy, Suraj Kumar, Erdal Mutlu, Bruce Palmer, Ajay Panyala, Bo Peng, Ryan M. Richard, T. P. Straatsma, Peter Sushko, Edward F. Valeev, Marat Valiev, Hubertus J. J. van Dam, Jonathan M. Waldrop, David B. Williams-Young, Chao Yang, Marcin Zalewski, and Theresa L. Windus. From NWChem to NWChemEx: Evolving with the computational chemistry landscape. *Chemical Reviews*, 121(8):4962–4998, 2021. PMID: 33788546.

[20] Andreas Krause and Daniel Golovin. Submodular function maximization. In Lucas Bordeaux, Youssef Hamadi, and Pushmeet Kohli, editors, *Tractability: Practical Approaches to Hard Problems*, pages 71–104. Cambridge University Press, 2014.

[21] Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence*, pages 324–331, 2005.

[22] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(8):235–284, 2008.

[23] Benny Lehmann, Daniel Lehmann, and Noam

Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, 2006.

[24] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46(1):259–271, 1990.

[25] Hui Lin and Jeff Bilmes. Word alignment via submodular maximization over matroids. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 170–175, 2011.

[26] Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *Proceedings of the 28th International Parallel and Distributed Processing Symposium*, pages 519–528. IEEE, 2014.

[27] Julián Mestre. Greedy in approximation algorithms. In *Proceedings of the 14th European Symposium on Algorithms*, pages 528–539. Springer, 2006.

[28] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Proceedings of the 8th IFIP Conference on Optimization Techniques*, pages 234–243. Springer, 1977.

[29] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. Introducing the Graph 500. *Cray User's Group*, 2010.

[30] George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[31] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, 1978.

[32] Seth Pettie and Peter Sanders. A simpler linear time 2/3-ε approximation for maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.

[33] Alex Pothen, S M Ferdous, and Fredrik Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019.

[34] Robert Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science*, pages 259–269, 1999.

[35] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474, 1993.

[36] Ehsan Tohidi, Rouhollah Amiri, Mario Coutino, David Gesbert, Geert Leus, and Amin Karbasi. Submodularity in action: From machine learning to signal processing applications. *IEEE Signal Processing Magazine*, 37(5):120–133, 2020.

[37] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pages 67–74, 2008.

Non-monotone Adaptive Submodular Meta-Learning

Shaojie Tang * Jing Yuan †

Abstract

The core idea of meta-learning is to leverage prior experience to design solutions that can be quickly adapted to new, unseen tasks. Most of existing studies consider the case where the feasible parameter space is continuous. Recently, [1] develops the framework of a discrete variant of meta-learning, called submodular meta-learning, and they treat each task as a discrete optimization problem, i.e., they intend to select a group of items that maximizes the average expected utility of all tasks. Motivated by their framework, we consider the submodular meta-learning problem under the adaptive setting. In particular, we assume that each item has a random state, which is drawn from some known prior distribution. One must select an item before observing its realized state. Given a task, the utility function is defined over items and states. Our goal is to adaptively select a group items, each selection is based on the feedback from the past, to maximize the average expected utility of all tasks. Following the framework of standard meta-learning, we propose an effective policy that is composed of two stages: We first pre-compute an initial set of items, called initial solution set, based on previously visited tasks, then, once a new task is revealed, we add more items to the initial solution set to complete the selection process. We show that our policy achieves a $1/32$ approximation ratio if the utility function of each task is adaptive submodular. Our policy enjoys the benefits of providing a personalized solution to each task while reducing the computation cost at test time.

1 Introduction

Meta-learning, whose goal is to leverage a few examples to develop a solution that can be rapidly adapted to new, unseen tasks, has attracted increased attention in many domains, including reinforcement learning [5, 6] and one-shot learning [12]. Most of previous studies focus on the continuous domain. For example, Model-Agnostic Meta-Learning (MAML) [7], one of the most popular meta-learning framework in continuous domain, intends to provide a good initialization of a model's parameters such that they can be adapted to a new task through a few gradient steps. Recently, [1] extends this study to the discrete domain by considering a discrete variant of meta-learning, called submodular meta-learning. They treat each task as maximizing a monotone and submodular function, and they intend to select a group of k items that maximizes the average expected utility of all tasks. Motivated by their framework, we consider the submodular meta-learning problem under the adaptive setting. We assume that each item belongs to a particular state drawn from some known prior distribution. However, the actual state of an item is unknown initially, one must select an item before observing its realized state. We treat each task as maximizing an adaptive submodular function which is defined over items and states. Following the framework of standard meta-learning, we propose an effective policy that is composed of two stages: We first pre-compute an initial set of at most l items based on previously visited tasks, then, after observing a new task, we add a group of at most $k - l$ items adaptively, each selection is based on the feedback from the past, to the initial solution set to complete the selection process. Note that the first batch of up to l items are selected before observing the incoming task, thus it does not consume the computation resource at the test time. One can balance the tradeoff between the degree of personalization of the solution and the computation overhead at the test time by adjusting the value of l. In particular, choosing a smaller l leads to a more personalized solution for each new task, however, the additional degree of personalization comes at the cost of higher computation cost at the test time. Note that when $l = k$, our framework reduces to the non-adaptive setting where all items must be selected before observing the incoming task, and when $l = 0$, our framework reduces to the fully adaptive setting where all selections are made in a closed-loop manner after observing the incoming task. The goal of this study is to compute the best learning policy for a fixed l, while leaving the decision of the best l to the decision-maker.

Our framework can be applied in many domains such as interactive recommendations [9], viral marketing [16, 15], machine learning [4], and link prediction [10]. Consider the example of adaptive viral marketing [8], where we would like to choose influential sets of

*Naveen Jindal School of Management, University of Texas at Dallas

†Department of Computer Science, University of Texas at Dallas

individuals to promote a particular product through a social network. In this context, we consider the promotion of a particular product as a task. And we treat each individual as an item, and the state of an item is the actual set of individuals it influences. A typical adaptive meta-learning based solution first selects a group of up to l influential individuals based on previously visited products, then after observing the incoming product, it selects the remaining individuals, whose size is at most $k - l$, adaptively to provide a product-specific solution.

We next summarize the main contribution of this paper. We show that the new objective function defined in our framework does *not* satisfy the property of adaptive submodularity, despite the utility function of each task is adaptive submodular. This makes the existing results on adaptive submodular maximization not applicable to our setting. We overcome this challenge by proposing a randomized policy that achieves a 1/32 approximation ratio for (non-monotone) adaptive submodular meta-learning problem.

2 Related Work

Meta-learning has been successfully applied to many domains, including reinforcement learning [5, 6] and one-shot learning [12]. Model Agnostic Meta-learning [7] is one of the most popular forms of meta-learning, it aims at learning an initial model that can easily adapt to the new task from few examples. Most of existing studies, including MAML, consider the case where the feasible parameter space is continuous. Very recently, [1] extends this study to the discrete domain, i.e., they consider the case when the parameter space is discrete. Our study follows their work by considering a discrete variant of meta-learning. In [1], they assume that each task can be represented using a monotone and submodular utility function. As a result, their objective is to find a good initial solution set that can quickly adapt to a new monotone and submodular function. In this work, we generalize their study by introducing an adaptive variant of submodular meta-learning. In particular, we assume that each item is associated with a random state whose realization is initially unknown. One must select an item in order to reveal its realized state. The utility function of each task is defined over sets of items as well as their realized states. One natural approach to maximize a utility function under the above setting is to sequentially select a group of items, each selection is based on the feedback from previous selections. The previous submodular meta-learning framework falls short in adaptive settings as it requires the decision-maker to make selections regardless of the realization of items' states. To circumvent this issue, we adopt the notation of adaptive submodularity [8], which generalize the classic notations of submodularity from sets to policies. We assume that each task can be represented as an adaptive submodular function. Thus, our study is related to non-monotone adaptive submodular maximization [13, 14]. Our study is also closely related to batch model active learning [3], where the selection is performed in batches. In each batch, they select multiple items in an offline fashion, and receive feedback only after all items from a batch have been selected. They develop a constant factor approximate solution to their problem. Their basic idea is to treat each possible batch as a virtual item, then apply the adaptive greedy algorithm over virtual items to obtain an approximate solution. Our work is different from theirs in two ways: As discussed in Section 3.5, our utility function is not adaptive submodular, thus the standard analysis used in adaptive submodular maximization does not apply to our framework. Moreover, [3] assume that each batch has the same size, while under our setting, we select at most l items in the first batch (training stage) and each of the following batches contains a single item.

3 Preliminaries

We start by introducing some important notations. In the rest of this paper, we use $[m]$ to denote the set $\{1, 2, \cdots, m\}$, and we use $|X|$ to denote the cardinality of a set X.

3.1 Items and States The input of our problem is a set E of n items. Each item is in a particular state from O. We use a function $\phi : E \to O$, called a *realization*, to represent the states of all items, where $\phi(e)$ denotes the realization of e's state. Let $\Phi = \{\Phi(e) \mid e \in E\}$ denote the random realizations of E, where $\Phi(e) \in O$ is a random realization of e. For any $Y \subseteq E$, let $\Phi(Y) = \cup_{e \in Y} \Phi(e)$ denote the random realizations of Y. There is a known prior probability distribution $p = \{\Pr[\Phi = \phi] : \phi \in U\}$ over realizations U. The state $\Phi(e)$ of each item $e \in E$ is initially unknown, and we must select e before observing the value of $\Phi(e)$. After selecting a set of items, we are able to observe a *partial realization* of those items' states. The *domain* $\mathrm{dom}(\psi)$ of a partial realization ψ is the set of all items involved in ψ. A partial realization ψ is said to be consistent with a realization ϕ, denoted $\phi \sim \psi$, if they are equal everywhere in $\mathrm{dom}(\psi)$. A partial realization ψ is said to be a *subrealization* of another partial realization ψ', denoted $\psi \subseteq \psi'$, if $\mathrm{dom}(\psi) \subseteq \mathrm{dom}(\psi')$ and they are equal everywhere in the domain $\mathrm{dom}(\psi)$ of ψ. Given a partial realization ψ, denote by $p(\phi \mid \psi)$ the conditional distribution over realizations conditioned on ψ: $p(\phi \mid \psi) = \Pr[\Phi = \phi \mid \Phi \sim \psi]$.

3.2 Training and Test Tasks We consider a family G of tasks, where the size of G could be infinite. Each task $i \in G$ is represented via a utility function f^i from a subset of items and their states to a non-negative real number: $f^i : 2^E \times O^E \to \mathbb{R}_{\geq 0}$. We assume that tasks from G arrive randomly according to an underlying probability distribution θ. Although θ is not always available, we assume that we have observed a group M of m training tasks. Each of the training tasks is sampled independently according to the distribution θ. Our ultimate goal is to maximize the utility function at test time. That is, we aim at achieving the best performance for an incoming task that is sampled independently from the distribution θ.

Consider the task of adaptive viral marketing whose objective is to adaptively select a set of *seed* users from a social network to help promote some product through the word-of-mouth effect. A social network can be represented as a graph with vertices representing users and edges representing their relations. The information propagation process is governed by some product-specific stochastic cascade model. Notably, Independent Cascade model assigns a product-specific propagation probability to each edge, say (u, v), and it represents the probability that user u successfully promotes a product to user v. One approach for solving this problem is to find a fixed group of seed users that can generate the largest average influence over all observed products, and let them promote all other products arriving in the future. As this group of users is pre-computed offline, the aforementioned approach has zero computational overhead at test time, however, it fails to provide a personalized solution for each new task. Another approach is to adaptively select a group of seed users with respect to each new task. Clearly, this approach leads to better performance, but at the cost of spending longer time and higher computational power on selecting all seed users at the test time.

3.3 Policies We consider an adaptive optimization problem where we sequentially select a group of items, after each selection, we observe the partial realization of the states of those items which have been previously selected. We define an adaptive policy using a function π that maps a set of observations to a distribution $\mathcal{P}(E)$ of E, specifying which item to select next based on the current task and the partial realization observed so far: $\pi : 2^E \times O^E \times 2^G \to \mathcal{P}(E)$.

Definition 1. (Policy Concatenation) *Given two policies π and π', let $\pi@\pi'$ denote a policy that runs π first, and then runs π', ignoring the observation obtained from running π.*

Definition 2. (Level-t-Truncation of a Policy) *Given a policy π, we define its level-t-truncation π_t as a policy that runs π until it selects t items.*

For each task $i \in G$ and each realization ϕ, let $E(\pi, \phi, i)$ denote the subset of items selected by π under realization ϕ for task i. Note that $E(\pi, \phi, i)$ is a random variable. The expected utility $f^i_{avg}(\pi)$ of a policy π for task i can be written as $f^i_{avg}(\pi) = \mathbb{E}_{\Phi \sim p, \Pi} f^i(E(\pi, \Phi, i), \Phi)$, where the expectation is taken over the realization and the random output of the policy. For any set of items $Y \subseteq E$ and any task $i \in G$, let $f^i(Y) = \mathbb{E}_{\Phi \sim p} f^i(Y, \Phi)$.

3.4 Adaptive Submodularity and Monotonicity We next introduce several important notations.

Definition 3. *(Conditional Expected Marginal Utility of an Item) Given a utility function f^i, the conditional expected marginal utility $\Delta_i(e \mid \psi)$ of an item e conditioned on ψ is $\Delta_i(e \mid \psi) = \mathbb{E}_\Phi[f^i(\mathrm{dom}(\psi) \cup \{e\}, \Phi) - f^i(\mathrm{dom}(\psi), \Phi) \mid \Phi \sim \psi]$, where the expectation is taken over Φ with respect to $p(\phi \mid \psi) = \Pr(\Phi = \phi \mid \Phi \sim \psi)$.*

Definition 4. *(Conditional Expected Marginal Utility of a Policy) Given a utility function f^i, the conditional expected marginal utility $\Delta_i(\pi \mid \psi)$ of a policy π conditioned on a partial realization ψ is $\Delta_i(\pi \mid \psi) = \mathbb{E}_{\Phi,\Pi}[f^i(\mathrm{dom}(\psi) \cup E(\pi, \Phi, i), \Phi) - f^i(\mathrm{dom}(\psi), \Phi) \mid \Phi \sim \psi]$, where the expectation is taken over (1) Φ with respect to $p(\phi \mid \psi) = \Pr(\Phi = \phi \mid \Phi \sim \psi)$, and (2) the random output of the policy. For any set of items $Y \subseteq E$, we define the conditional expected marginal utility $\Delta_i(\pi \mid Y)$ of a policy π conditioned on Y as*

$$\Delta_i(\pi \mid Y) = \mathbb{E}_{\Phi,\Pi}[f^i(Y \cup E(\pi, \Phi, i), \Phi) - f^i(Y, \Phi) \mid \Phi \sim p]$$

We assume that the utility functions of all tasks in M are *adaptive submodular* [8]. That is, for any two partial realizations ψ and ψ' such that $\psi \subseteq \psi'$, the following holds for each $i \in M$ and $e \in E \setminus \mathrm{dom}(\psi')$:

$$\Delta_i(e \mid \psi) \geq \Delta_i(e \mid \psi') \tag{3.1}$$

3.5 Adaptive Submodular Meta-Learning We next formally introduce the adaptive submodular meta-learning framework. Our framework is done in two stages: In the first (training) stage, we select a set $S \subseteq E$ of size at most l as the initial solution set, then, in the second (test) stage, we adaptively add the remaining items, whose size is at most $k-l$, to S after observing the task at hand. The motivation behind pre-computing an initial solution S is twofold: 1) in some cases it is time-consuming to acquire the partial realization of an item's state, it is more time-effective to acquire the partial

realization of a batch of items' states at once. As in our case, the selection of S is pre-computed regardless of the realization, we can safely select all items from S at once and acquire their partial realization simultaneously, which significantly reduces the response time at the test time compared to fully adaptive approach, i.e., $l = 0$. 2) our second motivation inherits from the one behind the non-adaptive submodular meta-learning framework [1]. Since computing S does not consume any computational resource such as power in the test stage, our framework helps to reduce the resource consumption in the test stage. We next consider two extreme cases in terms of l:

• $l = 0$ means that we select all items adaptively after observing the incoming task. Clearly, this fully adaptive policy, which computes a fully personalized solution for each incoming task, can not perform worse than the case when $l > 0$. However, implementing such a fully adaptive policy is more expensive than implementing a non-adaptive solution where all items are selected offline at once. For example, if acquiring the partial realization of an item's state is time-consuming, it is clearly more cost-effective to acquire the partial realization of a batch of items' states at once.

• $l = k$ means that we select all items offline before observing the incoming task. This non-adaptive policy is less computational power- and time-consuming since it requires zero computation at the test time, however, because such a non-adaptive solution can not adapt to the incoming task, its performance could be less satisfactory than the one of an adaptive policy.

One important question that arises from the above discussion is what would be the best l. The answer to this question is application-specific, one can balance the computational overhead at test time and the degree of personalization of the solution by tuning the value of l. We leave that decision to the decision-maker, and in this paper, we focus on finding the best learning policy for a given l.

3.6 Problem Statement For any $S \subseteq E$ and k, we use $\Omega(S, k, l)$ to denote the set of all policies that 1) pick S as the initial set, i.e., for each $\pi \in \Omega(S, k, l)$ and each $i \in G$, it holds that $\pi(\emptyset, i) = S$, and 2) $|E(\pi, \phi, i) \setminus S| \leq k - l$ for all $i \in G$ and all ϕ with $p(\phi) > 0$. Hence, the expected utility $f^i_{avg}(\pi)$ of any $\pi \in \Omega(S, k, l)$ conditioned on observing task i is

$$f^i_{avg}(\pi) = f^i(S) + \Delta_i(\pi|S)$$

The expected utility $\mathbb{E}_{i\sim\theta}[f^i_{avg}(\pi)]$ of any $\pi \in \Omega(S, k, l)$ over the distribution θ of tasks can be written as

$$\mathbb{E}_{i\sim\theta}[f^i_{avg}(\pi)] = \mathbb{E}_{i\sim\theta}[f^i(S)] + \mathbb{E}_{i\sim\theta}[\Delta_i(\pi|S)]$$

Since the underlying probability distribution θ of the tasks is not always available, we often collect a group of tasks that are sampled independently according to the distribution θ. As a result, we focus on optimizing the sample average approximation of $\mathbb{E}_{i\sim\theta}[f^i_{avg}(\pi)]$ given by

$$f_{avg}(\pi) \quad = \quad \frac{1}{m} \sum_{i \in M} f^i_{avg}(\pi)$$

Thus, our problem can be written as

$$\max_{S\subseteq E, |S|\leq l} \max_{\pi \in \Omega(S,k,l)} f_{avg}(\pi) \tag{3.2}$$

Remark 1: It was worth noting that Problem (3.2) will be solved at training time to obtain a task-independent set S of size at most l. The remaining items are added after observing the incoming task to form a task-specific solution of size at most k. When $l = 0$, i.e., all items are selected adaptively after observing the incoming task, our problem is reduced to the adaptive submodular maximization problem. This is because if the incoming task, say i, is known, our objective is reduced to maximizing $f^i_{avg}(\pi)$ subject to $|E(\pi, \phi, i)| \leq k$ for all ϕ with $p(\phi) > 0$. [8] show that the following simple adaptive greedy algorithm achieves a $1 - 1/e$ approximation ratio for the monotone case. It starts with a empty set, and in each round, it selects an item that maximizes the marginal utility on top of the current partial realization. For the non-monotone case, [13] develop a randomized policy that achieves a $1/e$ approximation ratio. Their policy starts with an empty set, and in each round, it selects an item uniformly at random from a set of k items (which may contain some dummy items whose marginal utility is zero on top of any partial realization) that have the largest marginal utility on top of the current partial realization. When $l = k$, i.e., all items are selected non-adaptively before observing the incoming task, our problem is reduced to the non-adaptive submodular maximization problem which has been studied in [11, 2]. Hence, in the rest of this paper, we assume that $l > 0$ and $k - l > 0$.

Remark 2: We next show that the training objective in Problem (3.2) does not satisfy the property of diminishing returns as specified in (3.1), making the existing results on adaptive submodular maximization [3, 13] not applicable to our setting. Recall that we must select the first group of at most l items before observing the incoming task. Because of the uncertainty associated with the incoming task, our utility function no longer satisfies the property of diminishing returns. We next use an example to show that selecting an item at a later stage could increase its marginal utility. Consider a toy example with two items $E = \{1, 2\}$,

two tasks $M = \{1, 2\}$, and one state $O = \{o\}$, i.e., the state of each item is deterministic, and the utility functions are defined as follows: $f^1(\emptyset, \{o, o\}) = f^1(\{1\}, \{o, o\}) = f^1(\{2\}, \{o, o\}) = f^1(\{1, 2\}, \{o, o\}) = 0$ and $f^2(\emptyset, \{o, o\}) = 0, f^2(\{1\}, \{o, o\}) = f^2(\{2\}, \{o, o\}) = 1, f^2(\{1, 2\}, \{o, o\}) = 2$. Clearly, the utility functions of both tasks are adaptive submodular. If we select item 1 in the training stage before observing the task, then the marginal utility of item 1 is $\frac{1}{2} \times f^1(\{1\}, \{o, o\}) + \frac{1}{2} \times f^2(\{1\}, \{o, o\}) = \frac{1}{2}$. Meanwhile, if we select item 1 in the test stage after observing the incoming task, say 2, then the marginal utility of item 1 to an existing set $\{2\}$ is $f^2(\{1, 2\}, \{o, o\}) - f^2(\{2\}, \{o, o\}) = 1$, which is larger than $\frac{1}{2}$. This clearly violates the property of diminishing returns.

4 Two-phase Randomized Greedy Policy

We next explain the design of our *Two-phase Randomized Greedy policy* π^r for the non-monotone adaptive meta-learning problem. π^r is composed of two phases: *initialization phase* and *execution phase*. The initialization phase is done at the training stage to find a good initial set S^r of size at most l, the execution phase is conducted after observing the incoming task. We first add a set D of k dummy items to the ground set, such that, for any $e \in D$, any partial realization ψ, and any $i \in G$, we have $\Delta_i(d \mid \psi) = 0$. Let $E' = E \cup D$. We introduce D to ensure that our policy never adds an item with negative marginal utility to the solution. Moreover, it is safe to assume that we select exactly l items at the training stage and select exactly $k - l$ items at the test stage. This is because we can always add enough dummy items to make this happen, and later we can safely remove those dummy items from the solution without affecting its utility. For any $t \in [0, k]$, let ψ^{r_t} denote the partial realization of the first t items selected by π^r. A detailed description of π^r is listed in Algorithm 1.

• **Initialization Phase:** Computing a task-independent initial set S^r of size l according to the following non-adaptive random greedy algorithm: It starts with $S^r = \emptyset$, and then adds a group of l items to S^r iteratively. At each step, we select an item uniformly at random from the set which is composed of the l items with the largest marginal utility to the current solution. This process iterates until all l items are selected.

• **Execution Phase:** When a task $i \in G$ arrives, π^r first selects S^r and observes their states, then selects the rest of the $k - l$ items according to an adaptive greedy algorithm. In particular, π^r runs in k rounds. The first l rounds are performed non-adaptively for selecting S^r and observing the partial realization ψ^{r_l} of S^r. At each of the remaining $k - l$ rounds $t \in [l + 1, k]$, π^r selects an item e_t uniformly at random from the set $U(\psi^{r_{t-1}})$, where $U(\psi^{r_{t-1}})$ contains the $k-l$ items with the largest marginal utility on top of the current partial realization $\psi^{r_{t-1}}$. After observing the state $\phi(e_t)$ of e_t, update the current partial realization ψ^{r_t} using $\psi^{r_{t-1}} \cup \{(e_t, \phi(e_t))\}$. This process iterates until all the remaining $k - l$ items have been selected.

Algorithm 1 Two-phase Randomized Greedy policy π^r

$S^r = \emptyset, t = 1, b = 1, \psi^{r_0} = \emptyset$.
{Initialization Phase}
while $b \leq l$ **do**
  $U(S^r) \leftarrow \arg\max_{V \subseteq E' : |V| = l} \frac{1}{m} \sum_{e \in V} \sum_{i \in M} (f^i(S^r \cup \{e\}) - f^i(S^r))$;
  sample e uniformly at random from $U(S^r)$;
  $S^r \leftarrow S^r \cup \{e\}$
  $b \leftarrow b + 1$;
{Execution Phase}
{The first l rounds are performed non-adaptively for selecting S^r.}
for $e \in S^r$ **do**
  $e_t \leftarrow e$; select e_t;
  $t \leftarrow t + 1$;
observe the partial realization

$$\psi^{r_l} = \cup_{t \in [l]} \{(e_t, \phi(e_t))\};$$

{The remaining $k - l$ rounds are performed adaptively.}
$t = l + 1$;
while $t \leq k$ **do**
  observe $\psi^{r_{t-1}}$;
  $U(\psi^{r_{t-1}}) \leftarrow \arg\max_{V \subseteq E' : |V| = k-l} \sum_{e \in V} \Delta_i(e \mid \psi^{r_{t-1}})$;
  sample e uniformly at random from $U(\psi^{r_{t-1}})$;
  $e_t \leftarrow e$;
  select e_t and observe $\phi(e_t)$;
  $\psi^{r_t} = \psi^{r_{t-1}} \cup \{(e_t, \phi(e_t))\}$; $t \leftarrow t + 1$;

The rest of this section is devoted to proving the performance bound of π^r. We use π^o to denote the optimal policy and use S^o to denote the initial solution set adopted by π^o. Here, we assume that S^o is deterministic since a probabilistic initial solution set can be expressed as the weighted sum of deterministic initial solution sets. Before presenting the main theorem, we first present four technical lemmas.

LEMMA 4.1. *For any training task $i \in M$, we have $f^i_{avg}(\pi^r @ \pi^o) - f^i_{avg}(\pi^r @ \pi^o_l) \leq f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l)$, where π^o_l (resp. π^r_l) denotes the level-l-truncation of π^o (resp. π^r).*

Proof: Define $\psi^{r_0} = \emptyset$, let $\overrightarrow{\psi^r} = \{\psi^{r_0}, \psi^{r_1}, \psi^{r_2}, \cdots, \psi^{r_k}\}$

denote the sequence of partial realizations obtained after running π^r, where ψ^{r_t} is the partial realization observed after selecting the t-th item. Conditioned on a sequence of partial realizations $\overrightarrow{\psi^r}$, we first give an upper bound on the value of $f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l)$. Let A_e be an indicator that e is selected by π^o at some step t such that $t > l$, and let x_e denote the expected marginal contribution of e to $f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l)$ conditioned on $A_e = 1$ and a partial realization ψ^{r_k}.

$$\begin{aligned}
&\mathbb{E}[f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l) \mid \overrightarrow{\psi^r}] \\
= &\sum_{e\in E'} \Pr[A_e = 1 \mid \psi^{r_k}] \cdot x_e \\
\leq &\sum_{e\in E'} \Pr[A_e = 1 \mid \psi^{r_k}] \cdot \Delta_i(e \mid \psi^{r_k}) \\
\leq &\max_{V\subseteq E':|V|=k-l} \sum_{e\in V} \Delta_i(e \mid \psi^{r_k})
\end{aligned}$$

Consider an arbitrary item $e \in E'$, assume e is selected by π^o at some step t such that $t > l$, the first inequality is due to f^i is adaptive submodular and ψ^{r_k} is a subrealization of the realization observed after running $\pi^r@\pi^o_{t-1}$. The second inequality is due to $\sum_{e\in E'} \Pr[A_e = 1 \mid \psi^{r_k}] \leq k - l$ for all ψ^{r_k}, this is because π^o selects at most $k-l$ items at the test stage. Hence,

$$\begin{aligned}
&f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l) \\
&= \sum_{\overrightarrow{\psi^r}} \Pr[\overrightarrow{\psi^r}]\mathbb{E}[f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l) \mid \overrightarrow{\psi^r}] \\
&\leq \sum_{\overrightarrow{\psi^r}} \Pr[\overrightarrow{\psi^r}] \max_{V\subseteq E':|V|=k-l} \sum_{e\in V} \Delta_i(e \mid \psi^{r_k}) \qquad (4.3)
\end{aligned}$$

where $\Pr[\overrightarrow{\psi^r}]$ denotes the probability that $\overrightarrow{\psi^r}$ is realized. We next provide a lower bound of $f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l)$.

$$\begin{aligned}
&f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l) \\
= &\sum_{\overrightarrow{\psi^r}} \Pr[\overrightarrow{\psi^r}] \sum_{t\in[l+1,k]} \sum_{e\in U(\psi^{r_{t-1}})} \frac{1}{k}\Delta_i(e \mid \psi^{r_{t-1}}) \\
\geq &\sum_{\overrightarrow{\psi^r}} \Pr[\overrightarrow{\psi^r}] \sum_{t\in[l+1,k]} \sum_{e\in U(\psi^{r_k})} \frac{1}{k}\Delta_i(e \mid \psi^{r_k}) \\
= &\sum_{\overrightarrow{\psi^r}} \Pr[\overrightarrow{\psi^r}] \sum_{e\in U(\psi^{r_k})} \Delta_i(e \mid \psi^{r_k}) \\
= &\sum_{\overrightarrow{\psi^r}} \Pr[\overrightarrow{\psi^r}] \max_{V\subseteq E':|V|=k-l} \sum_{e\in V} \Delta_i(e \mid \psi^{r_k}) \qquad (4.4)
\end{aligned}$$

The first equality is due to the design of π^r, i.e., at each round $t \in [l+1, k]$, π^r selects an item e_t uniformly at random from the set $U(\psi^{r_{t-1}})$. The inequality is due to f^i is adaptive submodular, $\psi^{r_k} \supseteq \psi^{r_{t-1}}$ for all $t \in [l+1, k]$. The third equality is due to the definition of U. (4.3) and (4.4) together imply that $f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l) \leq f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l)$. This finishes the proof of this lemma. □

LEMMA 4.2. *For any training task $i \in M$, we have $f^i_{avg}(\pi^o_l@\pi^r) - f^i_{avg}(\pi^o_l@\pi^r_l) \leq f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l)$.*

Proof: We first bound the expected marginal utility $\Delta_i(e_t \mid \psi^{r_{t-1}})$ of e_t conditioned on partial realization $\psi^{r_{t-1}}$ for all $t \in [k]$: $\Delta_i(e_t \mid \psi^{r_{t-1}}) \geq \mathbb{E}[\Delta_i(e_t \mid \Phi(S^o) \cup \psi^{r_{t-1}}) \mid \Phi \sim \psi^{r_{t-1}}]$. This inequality is due to $\psi^{r_{t-1}} \subseteq \Phi(S^o) \cup \psi^{r_{t-1}}$ for any Φ such that $\Phi \sim \psi^{r_{t-1}}$, and f^i is adaptive submodular.

Unfixing $\psi^{r_{t-1}}$ and take the expectation over $\Psi^{r_{t-1}}$, we have $\mathbb{E}_{\Psi^{r_{t-1}}}[\Delta_i(e_t \mid \Psi^{r_{t-1}})] \geq \mathbb{E}_{\Psi^{r_{t-1}}}[\mathbb{E}[\Delta_i(e_t \mid \Phi(S^o) \cup \Psi^{r_{t-1}}) \mid \Phi \sim \Psi^{r_{t-1}}]]$. Hence, the following inequality holds for all $t \in [k]$:

$$\begin{aligned}
&f_{avg}(\pi^r_t) - f_{avg}(\pi^r_{t-1}) \\
&\quad \geq f_{avg}(\pi^r_t@\pi^o_l) - f_{avg}(\pi^r_{t-1}@\pi^o_l) \qquad (4.5)
\end{aligned}$$

Then we have

$$\begin{aligned}
&f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l) = \sum_{t=l+1}^{k} (f_{avg}(\pi^r_t) - f_{avg}(\pi^r_{t-1})) \\
&\geq \sum_{t=l+1}^{k} (f_{avg}(\pi^r_t@\pi^o_l) - f_{avg}(\pi^r_{t-1}@\pi^o_l)) \\
&= f^i_{avg}(\pi^o_l@\pi^r) - f^i_{avg}(\pi^o_l@\pi^r_l)
\end{aligned}$$

The inequality is due to (4.5). This finishes the proof of this lemma. □

LEMMA 4.3.

$$\sum_{i\in M}(f^i_{avg}(\pi^r_l@\pi^o_l) - f^i_{avg}(\pi^r_l)) \leq \sum_{i\in M} f^i_{avg}(\pi^r_l)$$

Proof: Define $g(S) = \sum_{i\in M} f^i(S)$. To prove this lemma, it suffice to show that $\mathbb{E}_{S^r}[g(S^r \cup S^o) - g(S^r)] \leq \mathbb{E}_{S^r}[g(S^r)]$. Because f^i is adaptive submodular for all $i \in M$, $f^i(S) = \mathbb{E}_{\Phi\sim p}[f^i(S, \Phi)]$ is submodular in terms of S. Hence, $g(S)$ is also submodular in terms of S due the linear combination of submodular functions is still submodular. Because the first l items are selected non-adaptively by both π^r and π^o, proving this lemma is equivalent to showing that $\mathbb{E}_{S^r}[g(S^r \cup S^o) - g(S^r)] \leq \mathbb{E}_{S^r}[g(S^r)]$. Define $S^{r_0} = \emptyset$, let $\overrightarrow{S^r} = \{S^{r_0}, S^{r_1}, S^{r_2}, \cdots, S^{r_l}\}$ denote the sequence of sets selected by π^r, where S^{r_t} denotes the first t items selected by π^r. Sort S^o in an arbitrary order, for each

$t \in [l]$, let $S^o[t]$ denote the t-th item of S^o. We first provide an upper bound of $\mathbb{E}_{S^r}[g(S^r \cup S^o) - g(S^r)]$.

$$\begin{aligned}
&\mathbb{E}_{S^r}[g(S^r \cup S^o) - g(S^r)] \\
&= \mathbb{E}_{S^r}[\sum_{t\in[l]}(g(S^o[t] \cup S^r) - g(S^o[t-1] \cup S^r))] \\
&\leq \mathbb{E}_{S^r}[\sum_{e\in S^o}(g(\{e\} \cup S^r) - g(S^r))] \\
&\leq \mathbb{E}_{S^r}[\max_{V\subseteq E':|V|=l}\sum_{e\in V}(g(\{e\} \cup S^r) - g(S^r))] \qquad (4.6)
\end{aligned}$$

The first inequality is due to f^i is adaptive submodular. We next provide a lower bound of $\mathbb{E}_{S^r}[g(S^r)]$.

$$\begin{aligned}
&\mathbb{E}_{S^r}[g(S^r)] \\
&= \mathbb{E}_{\overrightarrow{S^r}}[\sum_{t\in[l]}\frac{1}{l}\sum_{e\in U(S^{r_{t-1}})}(g(S^{r_{t-1}} \cup \{e\}) - g(S^{r_{t-1}}))] \\
&= \mathbb{E}_{\overrightarrow{S^r}}[\sum_{t\in[l]}\frac{1}{l}\max_{V\subseteq E':|V|=l}\sum_{e\in V}(g(\{e\} \cup S^{r_{t-1}}) - g(S^{r_{t-1}}))] \\
&\geq \mathbb{E}_{\overrightarrow{S^r}}[\sum_{t\in[l]}\frac{1}{l}\max_{V\subseteq E':|V|=l}\sum_{e\in V}(g(\{e\} \cup S^r) - g(S^r))] \\
&= \mathbb{E}_{\overrightarrow{S^r}}[\max_{V\subseteq E':|V|=l}\sum_{e\in V}(g(\{e\} \cup S^r) - g(S^r))]
\end{aligned}$$

The first equality is due to the design of π^r, i.e., it selects an item uniformly at random from $U(S^{r_{t-1}})$, the inequality is due to g is submodular and $S^{r_{t-1}} \subseteq S^r$ for all $t \in [l]$. The above lower bound of $\mathbb{E}_{S^r}[g(S^r)]$ together with (4.6) imply that $\mathbb{E}_{S^r}[g(S^r \cup S^o) - g(S^r)] \leq \mathbb{E}_{S^r}[g(S^r)]$. This finishes the proof of this lemma. □

Lemma 4.4.

$$f_{avg}(\pi^r@\pi^o) \geq (1 - \frac{1}{l})^l(1 - \frac{1}{k-l})^{k-l}f_{avg}(\pi^o)$$

Proof: Recall that in each of the first l rounds of π^r, it selects an item uniformly at random from a set of l items. According to Lemma 1 in [13], if f^i is adaptive submodular, we have $f^i_{avg}(\pi^r_l@\pi^o) \geq (1 - \frac{1}{l})^l f^i_{avg}(\pi^o)$. Similarly, since in each of the last $k - l$ rounds of π^r, it selects an item randomly from a set of $k - l$ items, then we have $f^i_{avg}(\pi^r@\pi^o) \geq (1-\frac{1}{k-l})^{k-l}f^i_{avg}(\pi^r_l@\pi^o)$ if f^i is adaptive submodular. It follows that $f^i_{avg}(\pi^r@\pi^o) \geq (1-\frac{1}{l})^l(1-\frac{1}{k-l})^{k-l}f^i_{avg}(\pi^o)$. Because $f_{avg}(\pi^r@\pi^o) = \frac{1}{m}\sum_{i\in M} f^i_{avg}(\pi^r@\pi^o)$ and $f_{avg}(\pi^o) = \frac{1}{m}\sum_{i\in M} f^i_{avg}(\pi^o)$, we have $f_{avg}(\pi^r@\pi^o) \geq (1 - \frac{1}{l})^l(1 - \frac{1}{k-l})^{k-l}f_{avg}(\pi^o)$. □

Now we are ready to present the main theorem of this paper.

Theorem 4.1. *Our two-phase randomized greedy policy π^r achieves a $\frac{1}{2}(1 - \frac{1}{l})^l(1 - \frac{1}{k-l})^{k-l}$ approximation ratio, that is, $f_{avg}(\pi^r) \geq \frac{1}{2}(1 - \frac{1}{l})^l(1 - \frac{1}{k-l})^{k-l}f_{avg}(\pi^o)$.*

Proof: Recall that $\pi^r@\pi^o$ runs π^r first, then runs π^o from a fresh start. Hence, the expected utility $f^i_{avg}(\pi^r@\pi^o)$ of $\pi^r@\pi^o$ from task i can be written as:

$$\begin{aligned}
f^i_{avg}(\pi^r@\pi^o) &= f^i_{avg}(\pi^r_l) + (f^i_{avg}(\pi^r_l@\pi^o_l) - f^i_{avg}(\pi^r_l)) \\
&\quad +(f^i_{avg}(\pi^o_l@\pi^r) - f^i_{avg}(\pi^o_l@\pi^r_l)) \\
&\quad +(f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l))
\end{aligned}$$

It follows that

$$\begin{aligned}
&m \times f_{avg}(\pi^r@\pi^o) = \sum_{i\in M} f^i_{avg}(\pi^r@\pi^o) \\
&= \sum_{i\in M} f^i_{avg}(\pi^r_l) + \sum_{i\in M}(f^i_{avg}(\pi^r_l@\pi^o_l) - f^i_{avg}(\pi^r_l)) \\
&\quad + \sum_{i\in M}(f^i_{avg}(\pi^o_l@\pi^r) - f^i_{avg}(\pi^o_l@\pi^r_l)) \\
&\quad + \sum_{i\in M}(f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l)) \\
&\leq 2\sum_{i\in M} f^i_{avg}(\pi^r_l) + \sum_{i\in M}(f^i_{avg}(\pi^o_l@\pi^r) - f^i_{avg}(\pi^o_l@\pi^r_l)) \\
&\quad + \sum_{i\in M}(f^i_{avg}(\pi^r@\pi^o) - f^i_{avg}(\pi^r@\pi^o_l)) \\
&\leq 2(\sum_{i\in M} f^i_{avg}(\pi^r_l) + \sum_{i\in M}(f^i_{avg}(\pi^r) - f^i_{avg}(\pi^r_l))) \\
&= 2\sum_{i\in M} f^i_{avg}(\pi^r) = m \times 2f_{avg}(\pi^r)
\end{aligned}$$

The first inequality is due to Lemma 4.3, the second inequality is due to Lemma 4.1 and Lemma 4.2. It follows that $f_{avg}(\pi^r) \geq \frac{1}{2}f_{avg}(\pi^r@\pi^o)$. Together with $f_{avg}(\pi^r@\pi^o) \geq (1 - \frac{1}{l})^l(1 - \frac{1}{k-l})^{k-l}f_{avg}(\pi^o)$ due to Lemma 4.4, we have $f_{avg}(\pi^r) \geq \frac{1}{2}(1 - \frac{1}{l})^l(1 - \frac{1}{k-l})^{k-l}f_{avg}(\pi^o)$. □

Note that the above approximation ratio can be improved to 1/2 if the utility function f^i is adaptive monotone [8] for all $i \in M$, this is because when f^i is adaptive monotone for all $i \in M$, we have $f_{avg}(\pi^r@\pi^o) \geq f_{avg}(\pi^o)$.

Theorem 4.1, together with the fact that $(1 - \frac{1}{l})^l \geq 1/4$ and $(1 - \frac{1}{k-l})^{k-l} \geq 1/4$ when $l > 1$ and $k - l > 1$, implies the following corollary.

Corollary 4.1. *When $l > 1$ and $k - l > 1$, our two-phase randomized greedy policy π^r achieves a 1/32 approximation ratio, that is, $f_{avg}(\pi^r) \geq \frac{1}{32}f_{avg}(\pi^o)$.*

We next discuss the remaining cases when $l = 1$ or $k - l = 1$.

A $\frac{1}{1+e}$-approximate solution when $k - l = 1$. When $k - l = 1$, the first $k - 1$ items are selected non-adaptively in the training stage, and the last one item is selected after observing the incoming task. Our solution π^a is to randomly pick a policy from π^{a1} and π^{a2} to

follow such that π^{a1} is picked with probability $\frac{1}{1+e}$ and π^{a2} is picked with probability $\frac{e}{1+e}$. We next describe the details of π^{a1} and π^{a2}.

- The first candidate solution π^{a1} is a non-adaptive solution, which selects a fixed set S^a of items of size $k-1$ for all incoming tasks. We compute S^a using the greedy algorithm described in the Initialization phase of π^r.
- The second candidate solution π^{a2} does not select any items at the training stage. After observing the incoming task, say $i \in G$, it picks a singleton $e(i)$ with the largest expected utility, i.e., $e(i) = \arg\max_{e \in E'} f^i(\{e\})$.

THEOREM 4.2. *When $k - l = 1$, π^a achieves a $\frac{1}{1+e}$ approximation ratio, i.e., $f_{avg}(\pi^a) \geq \frac{1}{1+e} f_{avg}(\pi^o)$.*

Proof: According to the design of π^a, it picks π^{a1} (resp. π^{a2}) with probability $\frac{1}{1+e}$ (resp. $\frac{e}{1+e}$). Hence, the expected utility $f^i_{avg}(\pi^a)$ of π^a, for any $i \in M$, can be derived as follows:

$$\begin{aligned} f^i_{avg}(\pi^a) &= \frac{1}{1+e}\sum_{i\in M} f^i_{avg}(\pi^{a1}) + \frac{e}{1+e}\sum_{i\in M} f^i_{avg}(\pi^{a2}) \\ &= \frac{1}{m}\frac{1}{1+e}\sum_{i\in M} f^i(S^a) + \frac{1}{m}\frac{e}{1+e}\sum_{i\in M} f^i(\{e(i)\}) \end{aligned} \tag{4.7}$$

We next derive the expected utility $f^i_{avg}(\pi^o)$ of π^o for any $i \in M$.

$$\begin{aligned} &f^i_{avg}(\pi^o) = f^i(S^o) \\ &+\mathbb{E}_{\Psi^{o_{k-1}},\Pi^o}[\Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \Psi^{o_{k-1}}) \mid \Psi^{o_{k-1}}] \end{aligned} \tag{4.8}$$

where $\Psi^{o_{k-1}}$ denotes a random realization of the states of S^o, and $e(\pi^o, \Psi^{o_{k-1}}, i)$ denotes the (random) item selected by π^o after observing the incoming task $i \in M$ and partial realization $\Psi^{o_{k-1}}$. Because f^i is adaptive submodular, we have $\Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \Psi^{o_{k-1}}) \leq \Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \emptyset)$ due to $\emptyset \subseteq \Psi^{o_{k-1}}$. Moreover, because $e(i) = \arg\max_{e\in E'} f^i(\{e\})$, we have $\Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \emptyset) = f^i(\{e(\pi^o, \Psi^{o_{k-1}})\}) \leq f^i(\{e(i)\})$. It follows that $\Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \Psi^{o_{k-1}}) \leq f^i(\{e(i)\})$ for all $\Psi^{o_{k-1}}$. Hence,

$$\begin{aligned} &\mathbb{E}_{\Psi^{o_{k-1}},\Pi^o}[\Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \Psi^{o_{k-1}}) \mid \Psi^{o_{k-1}}] \\ &\leq f^i(\{e(i)\}) \end{aligned} \tag{4.9}$$

Moreover, because $g(S) = \sum_{i\in M} f^i(S)$ is submodular and S^a is obtained using the greedy algorithm described in the Initialization phase of π^r, according to [2], we have

$$e\sum_{i\in M} f^i(S^a) \geq \sum_{i\in M} f^i(S^o) \tag{4.10}$$

Now we are ready to prove this theorem.

$$\begin{aligned} m \times f_{avg}(\pi^o) &= \sum_{i\in M} f^i_{avg}(\pi^o) \\ &= \sum_{i\in M} f^i(S^o) \\ &\quad + \sum_{i\in M} \mathbb{E}_{\Psi^{o_{k-1}},\Pi^o}[\Delta_i(e(\pi^o, \Psi^{o_{k-1}}, i) \mid \Psi^{o_{k-1}}) \mid \Psi^{o_{k-1}}] \\ &\leq e\sum_{i\in M} f^i(S^a) + \sum_{i\in M} f^i(\{e(i)\}) \\ &= (1+e)(\frac{e}{1+e}\sum_{i\in M} f^i(S^a) + \frac{1}{1+e}\sum_{i\in M} f^i(\{e(i)\})) \\ &= m \times (1+e) f_{avg}(\pi^a) \end{aligned}$$

The second equality is due to (4.8), the inequality is due to (4.9) and (4.10), and the third equality is due to (4.7). Hence, $f_{avg}(\pi^a) \geq \frac{1}{1+e} f_{avg}(\pi^o)$. □

A $\frac{1}{2e}$-approximate solution when $l = 1$. When $l = 1$, we are allowed to select at most $l = 1$ item at the training stage, and the remaining $k-1$ items can be selected adaptively in the test stage. We next propose a randomized policy π^b that achieves a $\frac{1}{2e}$ approximation ratio to this case. π^b does not select any items during the training set, i.e., the initial solution set chosen by π^b is empty. After observing the incoming task, say $i \in G$, π^b samples a policy uniformly at random from π^{b1} and π^{b2} to follow. We next describe the details of π^{b1} and π^{b2}.

- The first candidate solution π^{b1} selects $k-1$ items adaptively in a greedy manner. It starts with an empty set, and at each round $t \in [1, k-1]$ of π^{b1}, it selects an item uniformly at random from the set $U(\psi^{b_{t-1}})$, where $U(\psi^{b_{t-1}}) = \arg\max_{V\subseteq E':|V|=k-1} \sum_{e\in V} \Delta_i(e \mid \psi^{b_{t-1}})$ contains the $k-1$ items with the largest marginal utility on top of the current partial realization $\psi^{b_{t-1}}$.
- The second candidate solution π^{b2} selects a singleton $e(i)$ with the largest expected utility.

We next analyze the performance bound of π^b.

THEOREM 4.3. *When $l = 1$, π^b achieves a $\frac{1}{2e}$ approximation ratio, i.e., $f_{avg}(\pi^b) \geq \frac{1}{2e} f_{avg}(\pi^o)$.*

Proof: Consider a one-step-further version π^{b+} of π^b by allowing it to select k items in the test stage. Clearly, $f^i_{avg}(\pi^{b+}_{k-1}) = f^i_{avg}(\pi^b)$ for any $i \in M$. According to Theorem 1 in [13], we can lower bound the performance of π^{b+} as follows:

$$f^i_{avg}(\pi^{b+}) \geq \frac{1}{e} f^i_{avg}(\pi^o) \tag{4.11}$$

due to f^i is adaptive submodular and π^o is a feasible adaptive policy that selects at most k items. Assume

e_k is the last item added to the solution by π^{b+}, we have $\Delta_i(e_k \mid \psi^{b_{k-1}}) \leq \Delta_i(e_k \mid \emptyset)$ for any $\psi^{b_{k-1}}$ due to f^i is adaptive submodular and $\emptyset \subseteq \psi^{b_{k-1}}$. It follows that $\Delta_i(e_k \mid \psi^{b_{k-1}}) \leq \max_{e \in E'} \Delta_i(e \mid \emptyset) = f^i(\{e(i)\})$. Hence,

$$\begin{aligned} f^i_{avg}(\pi^{b+}) &\leq f^i_{avg}(\pi^{b+}_{k-1}) + f^i(\{e(i)\}) \\ &= f^i_{avg}(\pi^b) + f^i(\{e(i)\}) \end{aligned} \tag{4.12}$$

(4.11) and (4.12) imply that

$$\frac{1}{e} f^i_{avg}(\pi^o) \leq f^i_{avg}(\pi^b) + f^i(\{e(i)\}) \tag{4.13}$$

Because π^b samples a policy uniformly at random from π^{b1} and π^{b2} to follow, we have

$$f^i_{avg}(\pi^b) = (f^i_{avg}(\pi^b) + f^i(\{e(i)\}))/2 \tag{4.14}$$

(4.13) and (4.14) together imply that $\sum_{i \in M} \frac{1}{e} f^i_{avg}(\pi^o) \leq \sum_{i \in M} (f^i_{avg}(\pi^b) + f^i(\{e(i)\})) = 2 \sum_{i \in M} f^i_{avg}(\pi^b)$. Hence, $f_{avg}(\pi^b) \geq \frac{1}{2e} f_{avg}(\pi^o)$ due to $f_{avg}(\pi^b) = \frac{1}{m} \sum_{i \in M} f^i_{avg}(\pi^b)$ and $f_{avg}(\pi^o) = \frac{1}{m} \sum_{i \in M} f^i_{avg}(\pi^o)$. □

5 Conclusion

In this paper, we develop a novel framework of adaptive submodular meta-learning. We extend the notion of submodular meta-learning to the adaptive setting which allows each item to have a random state. Our goal is to find an initial set of items that can quickly adapt to a new task. We proposed a two-phase randomized greedy policy that achieves a 1/32 approximation ratio.

References

[1] A. Adibi, A. Mokhtari, and H. Hassani, *Submodular meta-learning*, Advances in Neural Information Processing Systems, 33 (2020).

[2] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, *Submodular maximization with cardinality constraints*, in Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms, SIAM, 2014, pp. 1433–1452.

[3] Y. Chen and A. Krause, *Near-optimal batch mode active learning and adaptive submodular optimization.*, ICML (1), 28 (2013), pp. 8–1.

[4] S. Dasgupta and D. Hsu, *Hierarchical sampling for active learning*, in Proceedings of the 25th international conference on Machine learning, 2008, pp. 208–215.

[5] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, *Rl ^{2}: Fast reinforcement learning via slow reinforcement learning*, arXiv preprint arXiv:1611.02779, (2016).

[6] A. Fallah, A. Mokhtari, and A. Ozdaglar, *Provably convergent policy gradient methods for model-agnostic meta-reinforcement learning*, arXiv preprint arXiv:2002.05135, (2020).

[7] C. Finn, P. Abbeel, and S. Levine, *Model-agnostic meta-learning for fast adaptation of deep networks*, arXiv preprint arXiv:1703.03400, (2017).

[8] D. Golovin and A. Krause, *Adaptive submodularity: Theory and applications in active learning and stochastic optimization*, Journal of Artificial Intelligence Research, 42 (2011), pp. 427–486.

[9] A. Karbasi, S. Ioannidis, et al., *Comparison-based learning with rank nets*, arXiv preprint arXiv:1206.4674, (2012).

[10] M. Mitrovic, E. Kazemi, M. Feldman, A. Krause, and A. Karbasi, *Adaptive sequence submodularity*, in Advances in Neural Information Processing Systems, 2019, pp. 5352–5363.

[11] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, *An analysis of approximations for maximizing submodular set functions-i*, Mathematical programming, 14 (1978), pp. 265–294.

[12] J. Snell, K. Swersky, and R. Zemel, *Prototypical networks for few-shot learning*, in Advances in neural information processing systems, 2017, pp. 4077–4087.

[13] S. Tang, *Beyond pointwise submodularity: Non-monotone adaptive submodular maximization in linear time*, Theoretical Computer Science, 850 (2021), pp. 249–261.

[14] ———, *Beyond pointwise submodularity: Non-monotone adaptive submodular maximization subject to knapsack and k-system constraints*, arXiv preprint arXiv:2104.04853, (2021).

[15] J. Yuan and S. Tang, *No time to observe: adaptive influence maximization with partial feedback*, in Proceedings of the 26th International Joint Conference on Artificial Intelligence, 2017, pp. 3908–3914.

[16] J. Yuan and S.-J. Tang, *Adaptive discount allocation in social networks*, in Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2017, pp. 1–10.

Improving *Tug-of-War* sketch using Control-Variates method

Rameshwar Pratap* Bhisham Dev Verma† Raghav Kulkarni‡

Abstract

Computing space-efficient summary, or *a.k.a. sketches*, of large data, is a central problem in the streaming algorithm. Such sketches are used to answer *post-hoc* queries in several data analytics tasks. The algorithm for computing sketches typically requires to be fast, accurate, and space-efficient. A fundamental problem in the streaming algorithm framework is that of computing the frequency moments of data streams. The frequency moments of a sequence containing f_i elements of type i, are the numbers $\mathbf{F}_k = \sum_{i=1}^{n} {f_i}^k$, where $i \in [n]$. This is also called as ℓ_k norm of the frequency vector $(f_1, f_2, \ldots f_n)$. Another important problem is to compute the similarity between two data streams by computing the inner product of the corresponding frequency vectors. The seminal work of Alon, Matias, and Szegedy [2], *a.k.a. Tug-of-war* (or AMS) sketch gives a randomized sublinear space (and linear time) algorithm for computing the frequency moments, and the inner product between two frequency vectors corresponding to the data streams. However, the variance of these estimates typically tends to be large. In this work, we focus on minimizing the variance of these estimates. We use the techniques from the classical Control-Variate method [16] which is primarily known for variance reduction in Monte-Carlo simulations, and as a result, we are able to obtain significant variance reduction, at the cost of a little computational overhead. We present a theoretical analysis of our proposal and complement it with supporting experiments on synthetic as well as real-world datasets.

1 Introduction

Several data analytics tasks require analysing massive data stream such as real-time IP traffic analysis [7], metagenomics [6], email/tweets/SMS, time-series data [22], web clicks and crawls, sensor/IoT readings [18]. In many of these applications we may not have enough space/memory available to store the entire data stream. Moreover, these data packets arrive at a rapid rate. Therefore analysing such a large data stream requires sophisticated algorithmic techniques that can not only maintain a small footprint of the datastream but also incorporate the fast dynamic updates. Such a small footprint is commonly known as a *sketch* of the data stream, and is useful in answering important properties about the data stream during post-hoc queries [19].

In this paper, we consider the problems of estimating: i) frequency moments and of the data stream, and ii) inner product of frequency vectors corresponding to a pair of data streams. Let $\sigma_1 = \{a_1, a_2, \ldots, a_{m_1}\}$ and $\sigma_2 = \{b_1, b_2, \ldots, b_{m_2}\}$ be two data streams of lengths m_1 and m_2, respectively, where $\forall i \in [m_1], j \in [m_2]$, we have $a_i, b_j \in [n]$. This also implicitly defines the frequency vector over the elements in the data stream. We denote $\mathbf{f} = \langle f_1, f_2, \ldots, f_n \rangle$, and $\mathbf{g} = \langle g_1, g_2, \ldots, g_n \rangle$, where f_i, g_i denotes the number of occurrences of the element a_i, b_i in the stream σ_1 and σ_2, respectively . We denote k-frequency moments (or ℓ_k norm) of the frequency vector by $\mathbf{F}_k$ and define it as follows:

$$\mathbf{F}_k = \sum_{i=1}^{n} f_i^k.$$

[1] In this work, we focus on the case when $k = 2$. We define the inner product of the frequency vectors $\mathbf{f}$ and $\mathbf{g}$ as follows:

$$\langle \mathbf{f}, \mathbf{g} \rangle = \sum_{i=1}^{n} f_i g_i.$$

The seminal work of Alon, Matias and Szegedy [2] gives a sublinear space algorithm, for estimating the frequency moments, and the inner product between the frequency vectors. However, the variances of their estimates tend to be large when the data streams have many items of large frequencies. In this work, we address this challenge and suggest a technique that can reduce the variances provided by the AMS-sketch, and as a consequence give a more accurate estimation. We use the classical control variate method for the purpose, and we discuss it as follows:

1.1 Our approach – variance reduction using control variate trick:

Our technique exploits the control-variate trick used for reducing variance that occurs while estimating the frequency moments via AMS-sketch [2] for the large data streams. The control-variate is a classical method that is being used for variance reduction in Monte-Carlo simulation, which is done via analyzing the correlated errors [16]. We illustrate

*IIT Mandi, H.P., India (rameshwar.pratap@gmail.com).

†IIT Mandi, H.P., India (bhishamdevverma@gmail.com).

‡Chennai Mathematical Institute, Chennai, India (kulraghav@gmail.com).

[1] Note that $\mathbf{F}_0$ corresponds to the number of distinct elements in the stream n, and $\mathbf{F}_1$ corresponds to the number of elements in the stream m.

this with an example as follows: suppose we have a process generating a random variable X, and we are interested in estimating the quality of $\mathbb{E}[X]$. Let us have another process for generating another random variable Z such that we know exactly the value of its true mean $\mathbb{E}[Z]$. Then for any constant c, the expression $X + c(Z - \mathbb{E}[Z])$ is an unbiased estimator of X, due to the following:

$$\begin{aligned}\mathbb{E}[X + c(Z - \mathbb{E}[Z])] &= \mathbb{E}[X] + c\mathbb{E}[Z - \mathbb{E}[Z]]. \\ &= \mathbb{E}[X] + 0 = \mathbb{E}[X].\end{aligned} \tag{1.1}$$

The variance of the expression $X + c(Z - \mathbb{E}[Z])$ is given by

$$\mathrm{Var}[X + c(Z - \mathbb{E}[Z])] = \mathrm{Var}[X] + c^2\mathrm{Var}[Z] + 2c\mathrm{Cov}[X, Z]. \tag{1.2}$$

By elementary calculus we can find the appropriate value of c which minimise the above expression. Suppose we denote that value by $\hat{c}$, then

$$\hat{c} = -\frac{\mathrm{Cov}[X, Z]}{\mathrm{Var}[Z]}. \tag{1.3}$$

Equations 1.2, 1.3 give us the following

$$\mathrm{Var}[X + c(Z - \mathbb{E}[Z])] = \mathrm{Var}[X] - \frac{\mathrm{Cov}[X, Z]^2}{\mathrm{Var}[Z]}. \tag{1.4}$$

To summarize the above, for a random variable X, we are able to generate another random variable $X + c(Z - \mathbb{E}[Z])$ such that both have the same expected value, i.e., the random variable $X + c(Z - \mathbb{E}[Z])$ is an unbiased estimator of X. Further, the variance of $X + c(Z - \mathbb{E}[Z])$ is smaller than or equal to that of X because the $\mathrm{Cov}[X, Z]^2/\mathrm{Var}[Z]$ is always non-negative – with the equality if there is no correlation between X and Z. The random variable Z is called control variate, and the term $\hat{c}$ is called the control variate coefficient. In order to apply control variate trick for variance reduction, there are some practical considerations that need to be addressed carefully such as choosing an appropriate random variable Z, computing its expected value, and then choosing the coefficient c, etc.

1.2 Our results: As an application of the control variate trick, we provide significant variance reduction in the frequency moments estimators of AMS-sketch. We present our theoretical guarantees on the variance reduction for $\mathbf{F}_2$ as follows.

Theorem 1. *Let X be the random variable denoting the estimate of $\mathbf{F}_2$ in AMS-Sketch [2] (see Algorithm 1, Theorem 7). Then there exists a control variate random variable Z, and the corresponding control variate coefficient $\hat{c}$ such that*

$$\mathrm{Var}(X + \hat{c}(Z - \mathbb{E}[Z])) = \mathrm{Var}(X) - \frac{\left(\mathbf{F}_1^2 - \mathbf{F}_2\right)^2}{\mathbf{F}_0(\mathbf{F}_0 - 1)},$$

where, $\mathbb{E}[X] = \mathbf{F}_2$ and, $\mathrm{Var}(X) = 2(\mathbf{F}_2^2 - \mathbf{F}_4)$, as noted in [2] (Theorem 7).

We further extend our results which is used for variance reduction in estimating the inner product of two frequency vectors corresponding to a pair of data streams. We present our results as follows:

Theorem 2. *Let $\tilde{f}$ and $\tilde{g}$ be the sketches of $\mathbf{f}$ and $\mathbf{g}$ obtained via Tug-of-War sketch, respectively. Let $X^{(2)}$ be the random variable denoting the estimate of $\langle \mathbf{f}, \mathbf{g} \rangle$ in AMS-Sketch [2] (see Algorithm 1, Theorem 8). Then there exists a control variate random variable $Z^{(2)}$, and the corresponding control variate coefficient $\hat{c}$ such that*

$$\begin{aligned}&\mathrm{Var}(X^{(2)} + \hat{c}(Z^{(2)} - \mathbb{E}[Z^{(2)}])) \\ &= \mathrm{Var}(X^{(2)}) - \frac{2\left(\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2)\right)^2}{2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2},\end{aligned}$$

where, $\mathbb{E}[X^{(2)}] = \langle \mathbf{f}, \mathbf{g} \rangle$, and, $\mathrm{Var}(X^{(2)}) = \sum_{i \neq j} {f_i}^2 {g_j}^2 + \sum_{i \neq j, i, j \in [n]} f_i g_i f_j g_j$, as noted in [2] (Theorem 8).

1.2.1 Comment on the overhead of our estimate:

For estimation of $\mathbf{F}_2$ – Theorem 1: Our control variate random variable Z is independent of the actual values of the stream and we only need to know the number of distinct elements, n, in the stream in order to construct Z. Therefore, our new random variable $X + \hat{c}(Z - \mathbb{E}[Z])$ can be estimated with a small computational overhead, that is, $O(\log n)$ space and $O(n)$ time, which is additive to the AMS algorithm.

The mean of the control variate random variable Z (see Equations (3.6),(3.8)) is zero. As we can exactly compute the true mean of Z, we have $\mathbb{E}[X + c(Z - \mathbb{E}[Z])] = \mathbb{E}[X]$. Therefore, our new estimate does not introduce any additional bias to the respective estimate of AMS sketch.

The optimum value of c, given by $\hat{c}$ in Equation (3.12) turns out to be

$$\hat{c} = -\frac{\mathrm{Cov}[X, Z]}{\mathrm{Var}[Z]} = -\frac{\mathbf{F}_1^2 - \mathbf{F}_2}{\mathbf{F}_0(\mathbf{F}_0 - 1)}.$$

In practice, we choose an approximation for $\hat{c}$. The $\mathbf{F}_1$ is the length of the stream σ and its value is m, which we assume is known to us. We also assume that we know the value of $\mathbf{F}_0$.[2] We take the approximation of $\mathbf{F}_2$ obtained using the AMS sketch as a proxy for $\mathbf{F}_2$.

For estimation of inner product – Theorem 2: Our control variate random variable $Z^{(2)}$ is the estimate of sum of ℓ_2 norm of frequency vectors $\mathbf{f}$ and $\mathbf{g}$ (see Equation (3.16)). The mean of the control variate random variable $Z^{(2)}$ (see

[2] We do not need to know the exact value of $\mathbf{F}_0$, knowing an approximate upper bound would suffice.

Equation (3.17)) is $\mathbf{F}_2 + \mathbf{G}_2$. The optimum value of c, given by $\hat{c}$ in Equation (3.12) is

$$\hat{c} = -\frac{\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2)}{(2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2)}.$$

In practice, we choose approximations for $\hat{c}$ and $\mathbb{E}[Z^{(2)}]$. We take the approximation of $\mathbf{F}_2, \mathbf{G}_2$ and $\langle \mathbf{f}, \mathbf{g} \rangle$ obtained using the AMS sketch as proxies of the corresponding values. Therefore, our new random variable $X^{(2)} + \hat{c}(Z^{(2)} - \mathbb{E}[Z^{(2)}])$ can be estimated with a small computational overhead, that is, $O(\log n)$ space and $O(n)$ time, to the AMS algorithm.

1.2.2 Comment on the variance reduction: While estimating $\mathbf{F}_2$ note that when the original variance in AMS-sketch is large, i.e., when $(\mathbf{F}_2^2 - \mathbf{F}_4)$ is large (see Theorem 1), then many of the f_is are also expected to be large.[3] In this case $\mathbf{F}_1^2 - \mathbf{F}_2 = \sum_{i \neq j} f_i f_j$ would also be large. Hence we would expect bigger variance reduction in absolute terms given by $\left(\mathbf{F}_1^2 - \mathbf{F}_2\right)^2 / \mathbf{F}_0(\mathbf{F}_0 - 1)$ due to Theorem 1.

We also wish to get a visual understanding of the variance reduction in $\mathbf{F}_2$ estimation. To do so, we generate a random data stream having 1000 distinct items *s.t.* each item has a random frequency between 1 and 10. Then, using Theorem 1, we calculate the variance of our proposal (CV-AMS) and AMS sketch in the estimation of $\mathbf{F}_2$, and compute their ratio – a smaller value is an indication of better performance. We plot this ratio by varying $\mathbf{F}_1/n$ – keeping the number of distinct items constant but increasing their respective frequencies for randomly sampled items. We summarise our observation in Figure 1. We notice that the ratio of these two variances is small which indicates that CV-AMS has a smaller variance than that of vanilla AMS-sketch.

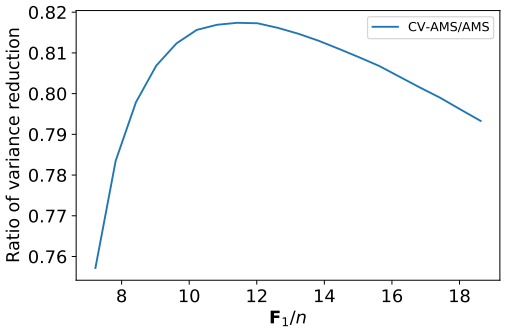


Figure 1: The ratio of the variance between our proposal (CV-AMS) with that of AMS sketch in the estimation of $\mathbf{F}_2$ of the data stream – a smaller value indicates that our proposal has a smaller variance than that of AMS-sketch. X-axis corresponds to $\mathbf{F}_1/n$.

[3]Otherwise, for instance in an extreme case, when there is only single large f_i then $(\mathbf{F}_2^2 - \mathbf{F}_4) = 0$

Similarly, in the estimation of the inner product of the frequency vector when the original variance in AMS-sketch is large, i.e., when $\sum_{i \neq j} {f_i}^2 {g_j}^2$ is large (see Theorem 2), then many of the f_is and g_is are also expected to be large. Therefore we get bigger variance reduction in absolute terms given by $2\left(\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2)\right)^2 / 2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2$.

We also wish to visually analyze the variance reduction in the inner product estimation of a pair of the data streams using our proposal (CV method) and AMS sketch. To do so we generate a random pair of streams having 1000 distinct items and the frequency of each item is randomly sampled between 1 and 100. We also vary the angle (denoted by θ) between data stream and their respective ℓ_2 norms (denoted by $\mathbf{F}_2$ and $\mathbf{G}_2$ of data streams σ_1 and σ_2 respectively). We choose the angle $\theta \in \{10^o, 30^o, 60^o, 90^o\}$, the ratio $\mathbf{F}_2/\mathbf{G}_2 \in \{0.1, 0.4, 0.7, 1\}$. We compute the corresponding variances of our proposal (CV method) and AMS sketch and record their ratio, and summarise it in Figure 2. Note that a smaller value of this ratio is an indication of smaller variance by our proposal. We notice that at smaller values of θ we obtained a much higher variance reduction because the inner product of the corresponding frequency vector is higher, whereas when $\theta = 90^o$, our proposal doesn't provide any variance reduction. Further, we obtain a higher variance reduction when the ratio of $\mathbf{F}_2$ and $\mathbf{G}_2$ is close to 1.

We note that there are some results obtaining further improvement of AMS-sketch for $\mathbf{F}_2$ [12, 13], and for $\mathbf{F}_k$, with $k > 2$ [11, 4, 9, 10]. However, the focus of this work is to demonstrate variance reduction for frequency estimation via control variate method. We believe that our technique can also be applied to the improvements of the AMS-sketch to obtain similar variance reductions. We state it as an open question of the work.

Known applications of control variates: Control variate technique has been used recently for reducing the variances of the estimates obtained in several Monte-Carlo simulations. Kang *et. al.* [15, 14] used it for improving the estimates for inner product and Euclidean distance obtained from random projection. However, to the best of our knowledge this approach has not been tried for reducing the variance of the streaming algorithm. In this work, we initiate this study.

Organization of the paper: The rest of the paper is organized as follows: in Section 2, we state some definitions and known facts which are used in the paper, also for the sake of completeness of the paper we briefly revisit the algorithms of AMS-sketch for $\mathbf{F}_2$ [2] and estimating inner product of frequency vectors of a pair of data streams, and their analysis. In Section 3, we give proofs of our results stated in Theorem 1 and Theorem 2. In Section 4, we complement our theoretical results with experiments on synthetic and real-world datasets. Finally in Section 5, we conclude our discussion and state some potential open questions of the work.

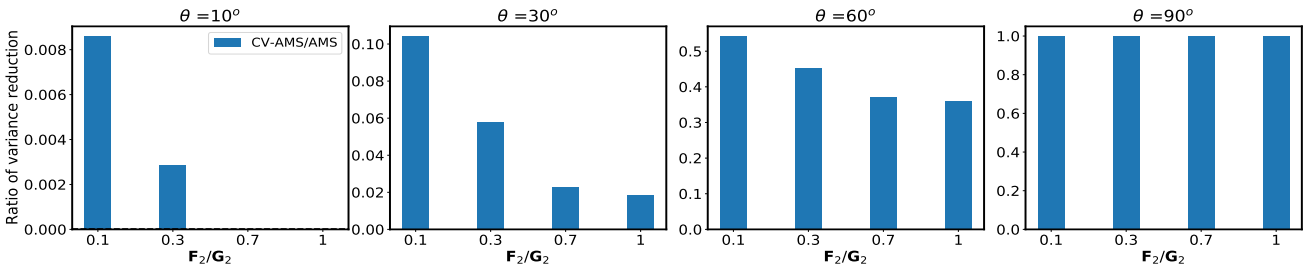


Figure 2: The ratio of the variance of our proposal (CV method) with that of AMS sketch in the inner product estimation of a pair of data streams σ_1 and σ_2 – a lower value indicates that the CV method has a smaller variance. θ denote the angle between the corresponding frequency vectors $\mathbf{f}$ and $\mathbf{g}$ of the data streams σ_1 and σ_2, respectively.

2 Background

Definition 3 (k-Universal Hashing)**.** *A randomized function $h : [n] \mapsto [l]$ is k-universal if $\forall i_1 \neq i_2 \ldots, \neq i_k \in [n]$, the following holds true for any $z_1, z_2, \ldots z_k \in [k]$:*

$$\Pr[h(i_1) = z_1, \textit{and } h(i_2) = z_2, \textit{ and } \ldots h(i_k) = z_k] = \frac{1}{l^k}.$$

A simple k-universal hash function example is the following [21]:

$$h(x) = \left(\left[\sum_{i=0}^{k-1} a_i x^i \right] \mod p \right) \mod l,$$

where p is a large prime number, and a_i's are some randomly sample positive integers smaller than p.

Lemma 4 (The Median-of-Means Improvement Lemma 4.4.1 of [5])**.** *There is a universal constant c such that the following holds. Let random variable X be an unbiased estimator of a real quantity Q. Let $\{X_{ij}\}_{i\in[t],j\in[k]}$ be collection of random variables with each X_{ij} distributed identically to X, where*

$$t = c \log\left(\frac{1}{\delta}\right) \textit{ and } k = \frac{3\mathrm{Var}[X]}{\varepsilon^2 \mathbb{E}[X]^2}.$$

Let $Z = median_{i\in[t]} \left(\frac{1}{k} \sum_{j=1}^{k} X_{ij} \right)$. Then we have $\Pr[|Z - Q| \geq \varepsilon Q] \leq \delta$. That is, Z is an (ε, δ) estimator for Q. This if an algorithm can produce X with s bits of space then there is an (ε, δ)-estimation algorithm using

$$O\left(s \cdot \frac{\mathrm{Var}[X]}{\varepsilon^2 \mathbb{E}[X]^2} \cdot \frac{1}{\varepsilon^2} \log \frac{1}{\delta} \right).$$

bit of space.

We require the following results from probability theory in order to proof some of our results.

Discrimilar: We adapt a similar writing style as of [5] for describing Algorithm 1 and its proof of correctness stated in Theorem 7.

Lemma 5. *[20] Let $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$, and $\mathbf{A}, \mathbf{B}$ are symmetric matrices, then*

$$\mathrm{Var}[\mathbf{w^T A w}] = 2\mathrm{Tr}[\mathbf{A\Sigma A\Sigma}],$$
$$\mathrm{Cov}[\mathbf{w^T A w}, \mathbf{w^T B w}] = 2\mathrm{Tr}[\mathbf{A\Sigma B\Sigma}],$$

where Tr *is the trace operator of the matrix.*

Theorem 6 (Multivariate Lyapunov CLT [8])**.** *Let $\{\mathbf{X}_1, \ldots \mathbf{X}_n\}$ be a sequence of independent random vectors such that each entry of the both (i) the expected value of the random vector $\{\mathbf{X}_i\}_{i=1}^n$, (ii) and the corresponding covariance matrix $\mathbf{\Sigma}_i$, is finite. We define*

$$\mathbf{V}_n = \sum_{i=1}^{n} \mathbf{\Sigma}_i.$$

If for some $\delta > 0$ the following condition holds true

$$\lim_{n\to\infty} ||\mathbf{V}_n^{\frac{1}{2}}||^{2+\delta} \sum_{i=1}^{n} \mathbb{E}\left[||\mathbf{X}_i - \mathbb{E}[\mathbf{X}_i]||^{2+\delta} \right] = 0, \textit{ then}$$
$$\mathbf{V}_n^{-\frac{1}{2}} \sum_{i=1}^{n} (\mathbf{X}_i - \mathbb{E}[\mathbf{X}_i]) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

as n tends to infinity. Where $\xrightarrow{d}$ denotes the convergence in distribution; $\mathbf{0}$ and $\mathbf{I}$ denote vector with each entry equal to zero and the identity matrix, respectively.

2.1 Revisiting AMS-Sketch [2] for $\mathbf{F}_2$ – Tug-of-war Sketch.

Alon, Matias and Szegedy [2] give a sketching algorithm for computing $\mathbf{F}_2$ norm of frequency vector of data stream. This algorithm is also popularly known as *"Tug-of-war"* sketch. We discuss it in Algorithm 1, and Theorem 7 discuss its theoretical analysis.

We now give an analysis on the guarantee offered by AMS-Sketch. Let $\tilde{X}$ denote the value of x, when the algorithm has finished processing the stream σ. Then we have

$$\tilde{X} = \sum_{j\in[n]} f_j h(j). \tag{2.5}$$

Table 1: Notations

Set $\{1, 2, \ldots n\}$	$[n]$	Data stream $\{a_1, a_2, \ldots, a_{m_1}\}$	σ_1
Stream elements	$\forall i \in [m_1], j \in [m_2], a_i, b_j \in [n]$	Data stream $\{b_1, b_2, \ldots, b_{m_2}\}$	σ_2
Freq. vector of σ_1	$\mathbf{f} = \langle f_1, \ldots f_i \ldots f_n \rangle$ *s.t.* $f_i :=$ freq. of i-th item	Sketch of data stream σ_1	$\tilde{f}$
Freq. vector of σ_2	$\mathbf{g} = \langle g_1, \ldots g_i \ldots, g_n \rangle$ *s.t.* $g_i :=$ freq. of i-th item	Sketch of data stream σ_2	$\tilde{g}$
Hash function	$h : [n] \mapsto \{-1, +1\}$	$h(i) : i \in [n]$	Y_i
$\sum_{i=1}^{n} {f_i}^k$	$\mathbf{F}_k$	$\sum_{i=1}^{k} f_i g_i$	$\langle \mathbf{f}, \mathbf{g} \rangle$

Algorithm 1 AMS-Sketch [2] – Tug-of-war Sketch

Initialize: Choose a random hash function $h : [n] \mapsto \{-1, +1\}$, from a 4-universal family
$x \leftarrow 0$
Process(j, c):
$x \leftarrow x + c \cdot h(j)$
Output:
x

Theorem 7 (Adapted from the results of [2]). *Let $\tilde{X}$ be the random variable denoting the value of x when the algorithm finishes processing the stream σ, and let us denote $X = \tilde{X}^2$. Then we have the following*

- $\mathbb{E}[X] = \mathbf{F}_2$,
- $\mathrm{Var}[X] = 2(\mathbf{F}_2^2 - \mathbf{F}_4)$.

2.2 Computing inner product of two frequency vectors

As an application of the control variate trick, we provide significant variance reduction in the frequency moments estimators of AMS-sketch. Algorithm 1 can also be used to compute the inner product of frequency vectors corresponding to two data streams. Let σ_1, σ_2 are two data streams of lengths m_1 and m_2, respectively, where each element of both the streams belong to the universe $[n] := \{1, 2, \ldots, n\}$. Let $\mathbf{f} = \langle f_1, f_2, \ldots, f_n \rangle$ and $\mathbf{g} = \langle g_1, g_2, \ldots, g_n \rangle$ denote the frequency vectors corresponding to the streams σ_1 and σ_2, respectively. Let us denote $\tilde{f}$ and $\tilde{g}$ the sketch of $\mathbf{f}$ and $\mathbf{g}$ obtained via Tug-of-War sketch (Algorithm 1), respectively, where

$$\tilde{f} = \sum_{i=1}^{n} f_i h(i),$$
$$\tilde{g} = \sum_{i=1}^{n} g_i h(i).$$

Theorem 8. *Suppose $\tilde{f}$ and $\tilde{g}$ are two random variables that are output of Algorithm 1 after processing the streams σ_1 and σ_2, and let us denote $X^{(2)} := \tilde{f}\tilde{g}$. Then we have the following*

- $\mathbb{E}[X^{(2)}] = \langle \mathbf{f}, \mathbf{g} \rangle$,
- $\mathrm{Var}[X^{(2)}] = \sum_{i \neq j} {f_i}^2 {g_j}^2 + \sum_{i \neq j, i, j \in [n]} f_i g_i f_j g_j$.

Concentration analysis and the space complexity for Tug-of-war Sketch: While estimating $\mathbf{F}_2$, in Algorithm 1, the sketch is the final value outputted by the variable x. The absolute value of x is at most $f_1 + f_2 + \ldots + f_n = m$. Therefore the space required for the sketch x is $O(\log m)$. The space required to store the hash function is $O(\log n)$. Thus, the overall space requirement is $O(\log m + \log n)$. Further, the space requirement for (ε, δ)-estimator using Lemma 4 is $O\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} (\log m + \log n)\right)$.

Further, in case of inner product estimation of a pair of data streams σ_1 and σ_2, the space required to store their sketches obtained from Algorithm 1 are $O(\log m_1)$ and $O(\log m_2)$, respectively. Therefore the overall space requirement is $O(\log m_1 + \log m_2 + \log n) = O(\log m_1 m_2 + \log n)$. Thus, the space requirement for (ε, δ)-estimator using Lemma 4 is $O\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta} (\log m_1 m_2 + \log n)\right)$.

3 Analysis

The frequency moment estimation computed via AMS-sketches can be made to offer (ε, δ)-guarantee by taking several independent copies of the random variable and computing the median of the means of the estimates by Lemma 4. Further, a reduction in the variance leads to a reduction in the number of independent copies of the random variables required for providing the same guarantee. In what follows we give proofs of Theorems 1, where we perform the variance reduction analysis for one estimate only.

3.1 Improving the variance bounds of AMS-Sketch [2] for $\mathbf{F}_2$ using control variate trick – Proof of Theorem 1

Proof. Recall that in AMS-Sketch, we denote the hash function $h(j)$ with the random variable Y_j. We now define our control variate random variable as follows:

$$Z = \sum_{i \neq j, i, j \in [n]} Y_i Y_j. \tag{3.6}$$

$$= \left(\sum_{i \in [n]} Y_i \right)^2 - \sum_{i \in [n]} Y_i^2 = \left(\sum_{i \in [n]} Y_i \right)^2 - n. \tag{3.7}$$

In the following analysis we will repeatedly use the following equalities: $\mathbb{E}[Y_j] = 0$, $\mathbb{E}[Y_i Y_j] = 0$ and $\mathbb{E}[Y_j^2] = \mathbb{E}[1] = 1$. We first calculate the expected value of the random variable Z.

(3.8)
$$\mathbb{E}[Z] = \mathbb{E}\left[\sum_{i\neq j, i,j\in[n]} Y_i Y_j\right] = \sum_{i\neq j, i,j\in[n]} \mathbb{E}[Y_i Y_j] = 0.$$

We now calculate the variance of the random variable Z.

$$\begin{aligned}
&\mathrm{Var}[Z] = \mathrm{Var}\left[\sum_{i\neq j, i,j\in[n]} Y_i Y_j\right].\\
&= \sum_{i\neq j, i,j\in[n]} \mathrm{Var}[Y_i Y_j] + \sum_{\substack{i\neq j\neq l\neq m,\\ i,j,l,m\in[n]}} \mathrm{Cov}(Y_i Y_j, Y_l Y_m).\\
&= \sum_{i\neq j, i,j\in[n]} \left(\mathbb{E}[Y_i^2 Y_j^2] - \mathbb{E}[Y_i Y_j]^2\right) +\\
&+ \sum_{i\neq j\neq l\neq m, i,j,l,m\in[n]} \left(\mathbb{E}[Y_i Y_j Y_l Y_m] - \mathbb{E}[Y_i Y_j]\cdot\mathbb{E}[Y_l Y_m]\right).\\
&= \sum_{i\neq j, i,j\in[n]} \left(\mathbb{E}[1] - \mathbb{E}[Y_i Y_j]^2\right) +\\
&+ \sum_{i\neq j\neq l\neq m, i,j,l,m\in[n]} \left(\mathbb{E}[Y_i Y_j Y_l Y_m] - \mathbb{E}[Y_i Y_j]\cdot\mathbb{E}[Y_l Y_m]\right).\\
&= \sum_{i\neq j, i,j\in[n]} 1 - 0 + (0-0).
\end{aligned}$$

(3.9)
$$= n(n-1) = \mathbf{F}_0(\mathbf{F}_0 - 1).$$

We now calculate the covariance between random variable X and control variate random variable Z.

$$\mathrm{Cov}(X, Z)$$

(3.10)
$$\begin{aligned}
&= \mathrm{Cov}\left(\sum_{j\in[n]} {f_j}^2 + \sum_{i\neq j, i,j\in[n]} f_i f_j Y_i Y_j, \sum_{\substack{l\neq m,\\ l,m\in[n]}} Y_l Y_m\right).\\
&= \mathrm{Cov}\left(\sum_{i\neq j, i,j\in[n]} f_i f_j Y_i Y_j, \sum_{l\neq m, l,m\in[n]} Y_l Y_m\right).\\
&= \mathbb{E}\left[\left(\sum_{i\neq j, i,j\in[n]} f_i f_j Y_i Y_j\right)\cdot\left(\sum_{l\neq m, l,m\in[n]} Y_l Y_m\right)\right]\cdots\\
&\quad - \mathbb{E}\left[\sum_{i\neq j, i,j\in[n]} f_i f_j Y_i Y_j\right]\cdot\mathbb{E}\left[\sum_{l\neq m, l,m\in[n]} Y_l Y_m\right].\\
&= \mathbb{E}\left[\left(\sum_{i\neq j, i,j\in[n]} f_i f_j Y_i Y_j\right)\cdot\left(\sum_{l\neq m, l,m\in[n]} Y_l Y_m\right)\right].\\
&= \mathbb{E}\left[\sum_{i\neq j, i,j\in[n]} f_i f_j {Y_i}^2 {Y_j}^2\right]\cdots\\
&\quad + \mathbb{E}\left[\sum_{i\neq j\neq l\neq m, i,j,l,m\in[n]} f_i f_j Y_i Y_j Y_l Y_m\right].\\
&= \sum_{i\neq j, i,j\in[n]} f_i f_j \mathbb{E}\left[{Y_i}^2 {Y_j}^2\right]\cdots\\
&\quad + \sum_{i\neq j\neq l\neq m, i,j,l,m\in[n]} f_i f_j \mathbb{E}[Y_i Y_j Y_l Y_m].\\
&= \sum_{i\neq j, i,j\in[n]} f_i f_j \mathbb{E}[1] + \sum_{i\neq j\neq l\neq m, i,j,l,m\in[n]} f_i f_j \times 0.\\
&= \sum_{i\neq j, i,j\in[n]} f_i f_j.
\end{aligned}$$

(3.11)
$$= \left(\sum_{i\in[n]} f_i\right)^2 - \sum_{i\in[n]} f_i^2 = \mathbf{F}_1^2 - \mathbf{F}_2.$$

Equations 3.11 along with Equation 3.9 give the control variate coefficient $\hat{c}$ and variance reduction as follows. This concludes a proof of the theorem.

(3.12)
$$\hat{c} = -\frac{\mathrm{Cov}[X,Z]}{\mathrm{Var}[Z]} = -\frac{\mathbf{F}_1^2 - \mathbf{F}_2}{\mathbf{F}_0(\mathbf{F}_0 - 1)}.$$

(3.13)
$$\text{Variance Reduction} = \frac{\mathrm{Cov}[X,Z]^2}{\mathrm{Var}[Z]} = \frac{\left(\mathbf{F}_1^2 - \mathbf{F}_2\right)^2}{\mathbf{F}_0(\mathbf{F}_0 - 1)}.$$

□

How to compute Z: Recall that (from Equation 3.7) our control variate random variable $Z = \left(\sum_{i\in[n]} Y_i\right)^2 - n$, where $Y_j := h(j)$. Note that Z only depends on the number of distinct elements, and is independent of the data stream σ. It can be easily computed (via Equation 3.7) by examining the hash function $h : [n] \mapsto \{-1, +1\}$ and maintaining a counter for the summation $\sum_{i\in[n]} Y_i$.

Remark 9. *In this work, we use AMS sketch for estimating the ℓ_2 norm of the frequency vector, where each f_i takes a non-negative value. However, we note that AMS sketch also works for any real valued vector, say $\mathbf{u} = \langle u_1, u_2, \ldots, u_n\rangle$, where $\mathbf{u} \in \mathbb{R}^d$ (u_i can take negative values as well). We remark that our algorithm also works in this scenario as well. However, in this case, we assume that we know the*

value of $\sum_{i=1}^{n} u_i$ *and* $\sum_{i=1}^{n} u_i^2$ *to compute the value of* $\hat{c}$ *(see Equations* (3.11),(3.12)*).*

3.2 Variance reduction in Tug-of-War estimator for dot product using Control variate – Proof of Theorem 2: Let $\mathbf{f} = \langle f_1, \ldots, f_n \rangle$ and $\mathbf{g} = \langle g_1, \ldots, g_n \rangle$ be two frequency vectors corresponding to two data streams σ_1 and σ_2, and $\tilde{f}, \tilde{g}$ are their sketches obtained from tug-of-war sketch algorithm (Algorithm 1). Recall that our random variable for estimating the inner product between $\mathbf{f}$ and $\mathbf{g}$ is:

$$X^{(2)} = \tilde{f}\tilde{g}. \tag{3.14}$$

$$\mathbb{E}[X^{(2)}] = \langle \mathbf{f}, \mathbf{g} \rangle. \tag{3.15}$$

We define our control variate random variable as follows:

$$Z^{(2)} = \tilde{f}^2 + \tilde{g}^2. \tag{3.16}$$

$$\mathbb{E}[Z^{(2)}] = \mathbf{F}_2 + \mathbf{G}_2. \tag{3.17}$$

Equation (3.17) hold due to Theorem 8 and linearity of expectation.

We now compute the covariance between control variate random variables $Z^{(2)}$ and our estimator $X^{(2)}$, and the variance of $Z^{(2)}$. We aim to compute it using Lemma 5. However, in order to use this lemma we need to prove that the joint distribution of $\tilde{f}$ and $\tilde{g}$ is bi-variate normal. We show this under the convergence in distribution and as $n \to \infty$. We use the multivariate lyapunov central limit theorem (stated in Theorem 6) for the same. We show it in the following theorem, and the subsequent corollary. We defer their proofs due to the space constraints.

Theorem 10. *If* $\forall i, 1 \leq i \leq n$, $\mathbb{E}\left[(f_i^2 + g_i^2)^{\frac{2+\delta}{2}}\right]$ *is finite for some* $\delta > 0$*, then as* $n \to \infty$*, we have*

$$\mathbf{V_n}^{-\frac{1}{2}} \begin{bmatrix} \tilde{f} \\ \tilde{g} \end{bmatrix} \xrightarrow{d} \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

where $\xrightarrow{d}$ *denotes convergence in distribution,* $\mathbf{V_n} = \begin{bmatrix} \mathbf{F}_2 & \langle \mathbf{f}, \mathbf{g} \rangle \\ \langle \mathbf{f}, \mathbf{g} \rangle & \mathbf{G}_2 \end{bmatrix}_{2\times 2}$*,* $\mathbf{0}$ *is a* (2×1) *dimensional vector with each entry as zero,* $\mathbf{I}$ *is an* (2×2) *identity matrix.*

Corollary 11. *Following the assumption stated in Theorem 10,*

$$\mathbf{V}_n^{-1/2} \begin{bmatrix} \tilde{f} \\ \tilde{g} \end{bmatrix} \xrightarrow{d} \mathcal{N}(\mathbf{0}, \mathbf{I}) \implies \begin{bmatrix} \tilde{f} \\ \tilde{g} \end{bmatrix} \xrightarrow{d} \mathcal{N}(\mathbf{0}, \mathbf{V}_n). \tag{3.18}$$

In the following, we conclude a proof of Theorem 2 using the results stated in Theorem 10, Corollary 11 and Lemma 5.

Proof of Theorem 2:

Proof. Let $X^{(2)}$ be the random variable denoting the estimate of our interest, and $Z^{(2)}$ be our control variate estimate.

$$X^{(2)} := \tilde{f}\tilde{g} = \begin{bmatrix} \tilde{f} & \tilde{g} \end{bmatrix} \begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} \tilde{f} \\ \tilde{g} \end{bmatrix}. \tag{3.19}$$

$$Z^{(2)} := \tilde{f}^2 + \tilde{g}^2 = \begin{bmatrix} \tilde{f} & \tilde{g} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{f} \\ \tilde{g} \end{bmatrix}. \tag{3.20}$$

From Corollary 11, we have

$$\begin{bmatrix} \tilde{f} \\ \tilde{g} \end{bmatrix} \xrightarrow{d} \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} \mathbf{F}_2 & \langle \mathbf{f}, \mathbf{g} \rangle \\ \langle \mathbf{f}, \mathbf{g} \rangle & \mathbf{G}_2 \end{bmatrix} \right). \tag{3.21}$$

Using Lemma 5, we calculate the co-variance between $X^{(2)}$ and $Z^{(2)}$, and the variance of $Z^{(2)}$ as follows:

$$\begin{aligned} &\mathrm{Cov}[X^{(2)}, Z^{(2)}] \\ &= 2\mathrm{Tr}\left[\begin{bmatrix} 0 & \frac{1}{2} \\ \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{F}_2 & \langle \mathbf{f}, \mathbf{g} \rangle \\ \langle \mathbf{f}, \mathbf{g} \rangle & \mathbf{G}_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{F}_2 & \langle \mathbf{f}, \mathbf{g} \rangle \\ \langle \mathbf{f}, \mathbf{g} \rangle & \mathbf{G}_2 \end{bmatrix} \right] \\ &= 2\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2). \end{aligned} \tag{3.22}$$

$$\begin{aligned} &\mathrm{Var}[Z^{(2)}] \\ &= 2\mathrm{Tr}\left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{F}_2 & \langle \mathbf{f}, \mathbf{g} \rangle \\ \langle \mathbf{f}, \mathbf{g} \rangle & \mathbf{G}_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{F}_2 & \langle \mathbf{f}, \mathbf{g} \rangle \\ \langle \mathbf{f}, \mathbf{g} \rangle & \mathbf{G}_2 \end{bmatrix} \right] \\ &= 2(2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2). \end{aligned} \tag{3.23}$$

Equations (3.22) and (3.23) gives us control variate coefficient and variance reduction as follows:

$$\hat{c} = -\frac{\mathrm{Cov}[X^{(2)}, Z^{(2)}]}{\mathrm{Var}[Z^{(2)}]} = -\frac{\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2)}{(2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2)}.$$

$$\begin{aligned} \text{Variance reduction} &= \frac{\mathrm{Cov}[X^{(2)}, Z^{(2)}]^2}{\mathrm{Var}[Z^{(2)}]} \\ &= \frac{2\left(\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2)\right)^2}{2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2}. \end{aligned}$$

□

How to compute $Z^{(2)}$: Recall that (from Equation 3.16) our control variate random variable $Z^{(2)} = \tilde{f}^2 + \tilde{g}^2$. We can compute its value by computing the sketches of data streams σ_1 and σ_2 (using the Tug-of-war sketch, Algorithm 1), computing their squares and adding them up.

How to choose a good control-variate function: To obtain higher variance reduction, one should choose a control variate random variable such that it shares high covariance with a random variable corresponding to estimate, and simultaneously has low variance. Also in order to be practically useful the control variate, its expected value,

variance, covariance with the original random variable should be easily computable from the dataset. We don't claim that the variance reduction obtained by the control variate used in this paper is the best possible. It might be possible to choose a better control variate random variable and achieve higher variance reduction. We leave this as an interesting open question of the work.

4 From theory to practice – experiments

We use the following datasets for our experiments:

- `Synthetic dataset`: we generate a stream of 100000 distinct items such that frequency of each item is randomly sampled between 1 and 5000.
- `Bag-of-word (BoW) dataset [17]`: This dataset consists of a corpus of documents. The raw documents were preprocessed by tokenization and removal of stopwords, then a vocabulary of unique words was generated (by keeping only those words that occurred more than ten times). A document is represented by the frequency vector of its words, i.e., for each word in vocabulary we count the number of its occurrences in the document. For our purpose, we consider each word as an item in the stream and consider its number of occurrences in the entire corpus as its frequency. We considered the KOS dataset for our experiments which has 6906 distinct words and 8472 words in total. Link of the dataset is available here [*].
- `Transaction datasets [1]`: These are well-known transaction datasets discussed in [1]. We considered two datasets from [1] – `T10I4D100K` and `T40I10D100K`. The former has 870 distinct items and 1010228 items in total, whereas the latter has 942 distinct items, and 3960507 items in total. Link of the datasets is available here [**].

4.1 For estimating $\mathbf{F}_2$ using Tug-of-war sketch:

Methodology: Let X be the random variable denoting the estimate of $\mathbf{F}_2$ obtained from the AMS-sketch. Then the updated estimate proposed by our algorithm is $X + c(Z - \mathbb{E}[Z])$, where c is the control variate coefficient, and Z is the random variable denoting the control variate. The optimum value of c, for our proposal, is given by $\hat{c}$ turns out to be

$$\hat{c} = -\frac{\mathrm{Cov}[X, Z]}{\mathrm{Var}[Z]} = -\frac{\mathbf{F}_1^2 - \mathbf{F}_2}{\mathbf{F}_0(\mathbf{F}_0 - 1)} \qquad \text{(see Equation 3.12).}$$

We assume that we know the values of $\mathbf{F}_1$ – the length of the stream and $\mathbf{F}_0$ – the number of distinct elements in the stream. We do not know the exact value of $\mathbf{F}_2$. However we know its estimated value obtained form the AMS-sketch, which we use as a proxy. Thus, we can compute an estimate of $\hat{c}$, which turns out to give good variance reduction as demonstrated by our experiments. Recall that our control variate random variable is

$$Z = \left(\sum_{i \in [n]} Y_i\right)^2 - n \qquad \text{(see Equation 3.7).}$$

The value of Z can be easily computed by summing the hash value of each distinct item in the stream, and subtracting it with the number of distinct element in the stream. Further, the expected value of Z is 0 from Equation 3.8. Therefore, we can compute the value of our estimate $= X + \hat{c}(Z - \mathbb{E}[Z])$. using the estimates of X, $\hat{c}$ and Z. We generate the 4-universal hash functions used in the AMS algorithm following the approach stated in Definition 3.

Evaluation Metric: We evaluate the performance of our approach with the AMS-Sketch algorithm on the following three measures – (i) variance analysis *via* box-plot, (ii) mean absolute error, (iii) medians-of-means estimation (see Lemma 4). We discuss our experimental procedure as follows: For each dataset, we run both AMS-sketch and our proposal 1000 times. This gives us 1000 different estimates for $\mathbf{F}_2$ both from AMS-sketch and from our method. We use these estimates to generate box-plots for variance analysis. To calculate the mean-absolute-error, we compute the absolute difference of each of these estimates with the ground truth $\mathbf{F}_2$, and then compute the mean of these 1000 absolute values. A smaller value of the mean-absolute-error is an indication of better performance. In order to compute the medians-of-means estimation, we randomly make 20 groups each with 50 estimates. We then compute the mean of each group and consider the median of all the 20 means. This gives us a median-of-means estimate. We summarise our results in Figures 3, 4, and 5, respectively.

Insight: In Figure 3, we observe the interquartile range of our proposal is smaller than that of AMS-sketch which implies that the variance of our proposal is smaller. In Figure 4, we notice that the mean absolute error of our proposal is always smaller than that of AMS-sketch. This indicates that the error occurred in our estimate is small. Finally, in 5, the median-of-mean estimate of our method tends to be closer to the ground truth $\mathbf{F}_2$. All these observations indicate that the variance of our estimate as well as the error in approximating the ground truth $\mathbf{F}_2$ are smaller than that of AMS-sketch.

4.2 For estimating inner product between a pair of data streams using Tug-of-war sketch:

[*] `https://archive.ics.uci.edu/ml/datasets/Bag+of+Words`

[**] `http://fimi.uantwerpen.be/data/`

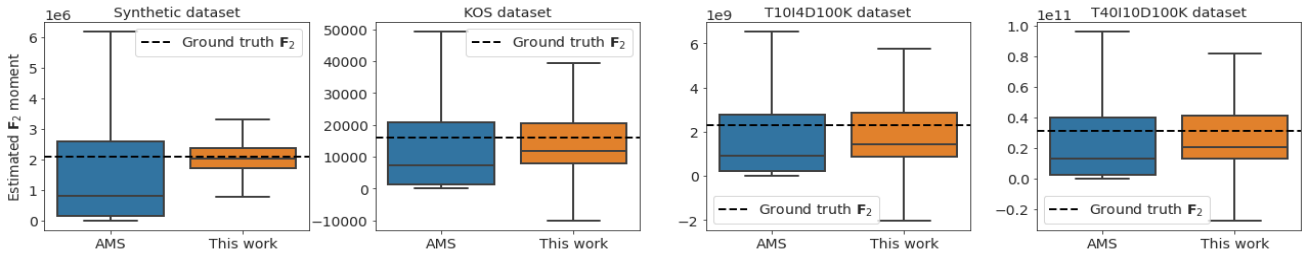


Figure 3: Comparison between AMS-sketch and our estimate on the variance analysis *via* box-plot.

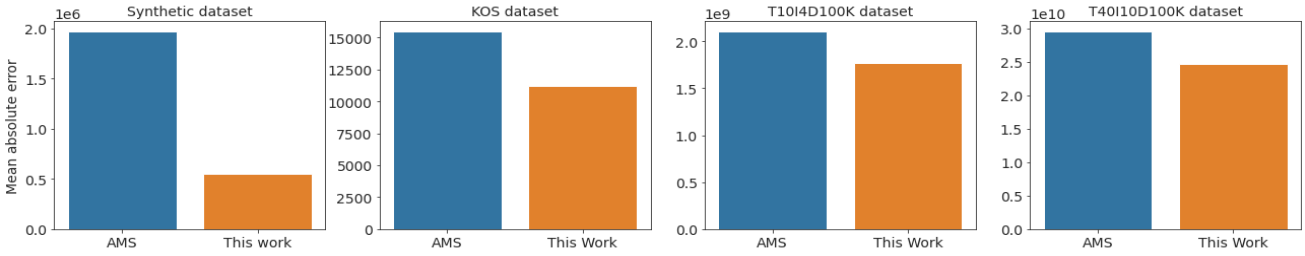


Figure 4: Comparison between AMS-sketch and our estimate on the mean-absolute-error. A smaller value of mean-absolute-error is an indication of better performance.

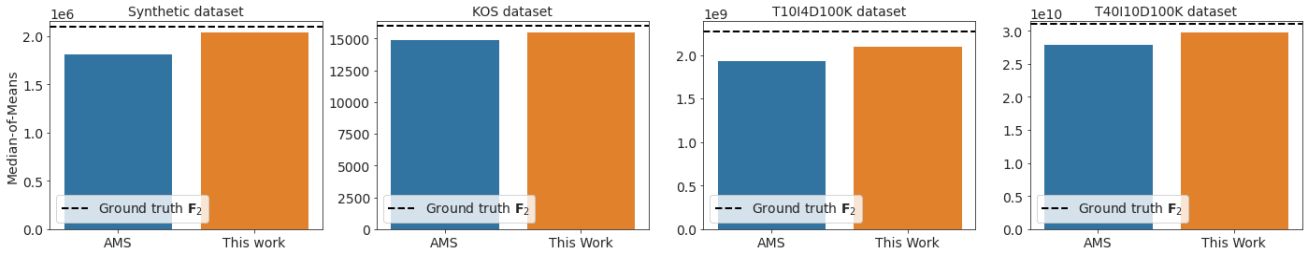


Figure 5: Comparison between AMS-sketch and our algorithm on the median-of-means estimate. The dotted line corresponds to the ground truth $\mathbf{F}_2$. The height of the bar chart closer to the dotted line is an indication of better performance.

Methodology: Let $\tilde{f}$ and $\tilde{g}$ denote the sketches of a pair data streams σ_1 and σ_2 obtained via AMS-sketch. Let $X^{(2)}$ be the random variable denoting the estimate of the inner product of the corresponding frequency vectors of σ_1 and σ_2, respectively. Our control variate estimator is $X^{(2)} + \hat{c}(Z^{(2)} - \mathbb{E}[X^{(2)}]$, where

$$Z^{(2)} = \tilde{f}^2 + \tilde{g}^2, \qquad \text{(see Equation (3.20))}, \text{and}$$
$$\hat{c} = -\frac{\langle \mathbf{f}, \mathbf{g} \rangle (\mathbf{F}_2 + \mathbf{G}_2)}{(2\langle \mathbf{f}, \mathbf{g} \rangle^2 + {\mathbf{F}_2}^2 + {\mathbf{G}_2}^2)}.$$

We compute the value of $Z^{(2)}$ by computing the sum of squares of the sketches of σ_1 and σ_2 obtained via Algorithm 1. To compute the value of $\hat{c}$, we assume that we know the values of $\mathbf{F}_2$ and $\mathbf{G}_2$ in advance. However, we don't know the value of $\langle \mathbf{f}, \mathbf{g} \rangle$ – the very quantity which we want to estimate. For our experiments, we take the estimate obtained via AMS sketch as its proxy.

Evaluation Metric: We require a pair of streams to perform our experiments. We generate it as follows: for synthetic datasets, we generate a pair of streams using the similar procedure mentioned above. For BOW datasets [17], recall that it is a corpus of a set of documents. We split the corpus into two equal halves consisting of the same number of documents, and we consider each half as a separate data stream. For transaction datasets, we split the streams in two equal halves and consider each half as a separate data stream.

We compute the estimate of the inner product of a pair of input data streams 1000 times using both AMS-sketch and our CV method. We use the same three metrics — (i) variance analysis *via* box-plot, (ii) mean absolute error, (iii) medians-of-means estimation , and summarise the corresponding plots in Figures 6, 7, and 8, respectively.

Insight: Here again in Figure 6, we notice that our CV method has smaller variance than that of AMS-sketch. In

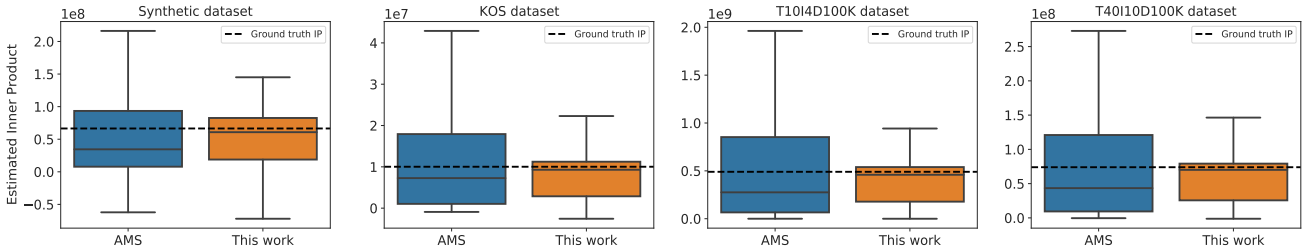


Figure 6: Comparison between AMS-sketch and our estimate on the variance analysis on the task of inner product estimation *via* box-plot.

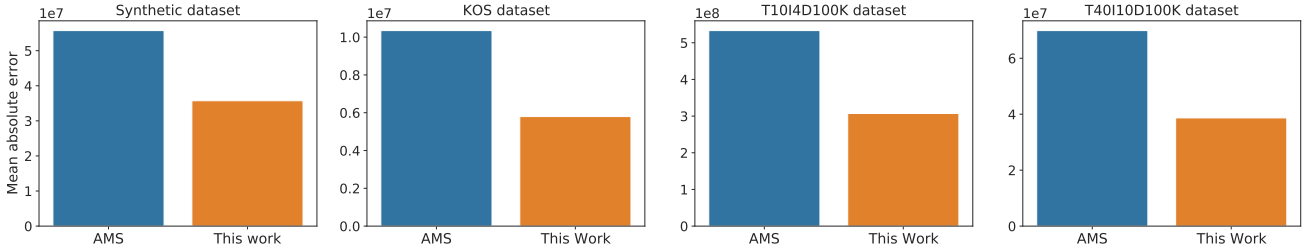


Figure 7: Comparison between AMS-sketch and our estimate on the mean-absolute-error on the task of inner product estimation. A smaller value of mean-absolute-error is an indication of better performance.

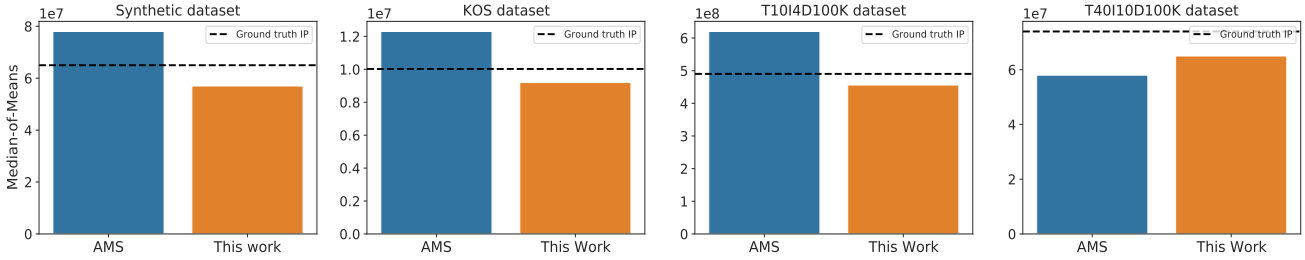


Figure 8: Comparison between AMS-sketch and our algorithm on the median-of-means estimate. The dotted line corresponds to the ground truth inner product. The height of the bar chart closer to the dotted line is an indication of better performance.

Figure 7, we observe that the mean absolute error of our proposal is always smaller than that of AMS-sketch which implies that our proposal has smaller errors in the inner product estimation. Finally, in Figure 8, the median-of-mean estimate of our method tends to be closer to the ground truth inner product. These results indicate that on the task of inner product estimation our CV proposal has smaller variance than that of AMS sketch which leads to a more accurate inner product estimation.

5 Conclusion

In this work, we consider the problem of estimating the frequency moments of a large data stream, and the problem of estimating inner product between a pair of data streams . The breakthrough result due to Alon, Matias, Szegedy [2] gives a sublinear space algorithm for these problems. However, the variances of their estimators tend to be large when frequencies of items are large. We address this challenge and suggest a method for variance reduction at the expense of a small computational overhead. Our proposal relies on the classical control-variates [16] method which is typically used for variance reduction in Monte-Carlo simulations.

Our work leaves several open questions and research directions: *a)* extending our result from $\mathbf{F}_2$ to $\mathbf{F}_k$, for $k > 2$, and for $\mathbf{F}_0$, *b)* improving the variance reduction by choosing better control variate, *c)* variance reduction for other streaming algorithms. To conclude we note that our method is simple and effective. Hence we hope that our method can be adopted in practice. Moreover, we believe that the control-variate trick can benefit large class streaming algorithms [5], and randomized algorithms [3] in general. Illustrating variance reduction for such algorithms using the control variate trick would be interesting future work.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994. URL: http://www.vldb.org/conf/1994/P487.PDF.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

[3] Mikhail J. Atallah, editor. *Algorithms and Theory of Computation Handbook*. Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC Press, 1999. doi:10.1201/9781420049503.

[4] Lakshminath Bhuvanagiri, Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Simpler algorithm for estimating frequency moments of data streams. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 708–713, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109634.

[5] Amit Chakrabarti. Data stream algorithms lecture notes. *July*, 2020. URL: https://www.cs.dartmouth.edu/~ac/Teach/data-streams-lecnotes.pdf.

[6] R A Leo Elworth, Qi Wang, Pavan K Kota, C J Barberan, Benjamin Coleman, Advait Balaji, Gaurav Gupta, Richard G Baraniuk, Anshumali Shrivastava, and Todd J Treangen. To Petabytes and beyond: recent advances in probabilistic and signal processing algorithms and their application to metagenomics. *Nucleic Acids Research*, 48(10):5217–5234, 04 2020. arXiv:https://academic.oup.com/nar/article-pdf/48/10/5217/33326373/gkaa265.pdf, doi:10.1093/nar/gkaa265.

[7] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 19-23, 2002, Pittsburgh, PA, USA*, pages 323–336, 2002. doi:10.1145/633025.633056.

[8] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, January 1968. URL: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{&}path=ASIN/0471257087.

[9] Sumit Ganguly. A note on estimating hybrid frequency moment of data streams. In *Algorithmic Aspects in Information and Management, 5th International Conference, AAIM 2009, San Francisco, CA, USA, June 15-17, 2009. Proceedings*, pages 202–211, 2009. doi:10.1007/978-3-642-02158-9_18.

[10] Sumit Ganguly and David P. Woodruff. High probability frequency moment sketches. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 58:1–58:15, 2018. doi:10.4230/LIPIcs.ICALP.2018.58.

[11] Piotr Indyk and David P. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 202–208, 2005. doi:10.1145/1060590.1060621.

[12] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 745–754, 2011. doi:10.1145/1993636.1993735.

[13] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. On the exact space complexity of sketching and streaming small norms. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1161–1178, 2010. doi:10.1137/1.9781611973075.93.

[14] Keegan Kang. Using the multivariate normal to improve random projections. In *Intelligent Data Engineering and Automated Learning - IDEAL 2017 - 18th International Conference, Guilin, China, October 30 - November 1, 2017, Proceedings*, pages 397–405, 2017. doi:10.1007/978-3-319-68935-7_43.

[15] Keegan Kang and Giles Hooker. Random projections with control variates. In *Proceedings of the 6th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2017, Porto, Portugal, February 24-26, 2017*, pages 138–147, 2017. doi:10.5220/0006188801380147.

[16] S. Lavenberg and P. Welch. A perspective on the use of control variables to increase the efficiency of monte carlo simulations. *Management Science*, 27:322–335, 03 1981. doi:10.1287/mnsc.27.3.322.

[17] M. Lichman. UCI machine learning repository, 2013. URL: http://archive.ics.uci.edu/ml.

[18] Samuel Madden and Michael Franklin. Fjording the stream: An architecture for queries over streaming sensor data. pages 555 – 566, 02 2002. doi:10.1109/ICDE.2002.994774.

[19] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005. doi:10.1561/0400000002.

[20] S.B. Provost and A.M. Mathai. *Quadratic Forms in Random Variables: Theory and Applications/ A.M. Mathai, Serge B. Provost*. Statistics : textbooks and monographs. Marcel Dek ker, 1992. URL: https://books.google.co.in/books?id=-SafoAEACAAJ.

[21] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981. URL: http://dblp.uni-trier.de/db/journals/jcss/jcss22.html#WegmanC81.

[22] Yunyue Zhu and Dennis E. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369. Morgan Kaufmann, 2002. URL: http://dblp.uni-trier.de/db/conf/vldb/vldb2002.html#ZhuS02.

On the Difference between Search Space Size and Query Complexity in Contraction Hierarchies

Claudius Proissl* Tobias Rupp†

Abstract

In this paper we present results that underline the significant difference between the size of node search space size and the query complexity in Contraction Hierarchies. Under different input models, node search spaces were proven to be sublinear [3, 4]. In this paper we prove for the same settings that edge search spaces are *not* sublinear. Our results make clear, that only analyzing the number of processed nodes during a CH query does not suffice for a thorough understanding of the CH query complexity. A broader definition of the search space that takes edges into account appears to be necessary.

1 Introduction

Shortest path computations in graphs are essential in our daily life. While Dijkstra's algorithm [6] can compute shortest paths in near-linear time, this is still not efficient enough for most real-world applications [2]. That motivated the invention of speed-up techniques of which Contraction Hierarchies (CH) [8] is one of the most prominent examples. The general idea of CH is to construct an overlay graph in a preprocessing phase. This overlay graph often allows to answer shortest path queries more efficiently by running a modified version of Dijkstra's algorithm.

CH show a remarkable speed-up in real-world transportation networks. The theoretical explanation for that behavior, however, still suffers from significant gaps. Several graph characterizations have been proposed, which allow to prove bounds on the CH search procedure. The *highway dimension* [1] and the treewidth are two important examples.

Abraham et al. [1] show that for a graph G with highway dimension h one can construct an overlay graph such that a shortest path query using a slight variation of CH takes $O((\Delta + h \log D)h \log D)$ time, where D is the diameter of G and Δ is the maximum degree of a node in G. Moreover, the space requirements to store the overlay graph are in $O(nh \log D)$. While this result yields interesting insights for many graph families, it does not fully resolve the problem. There are realistic graph instances for which in practice CH leads to a considerable speed-up that cannot be explained with the highway dimension. The probably most popular example are grid graphs [10].

Bauer et al. [3] show upper bounds on the query complexity of CH that solely depend on the structure of the graph G (not on the edge weights). Their considerations focus on the notion of the *search space*, which we formally define in Section 4.3. Informally, the search space of a node v in G is the number of nodes that are settled in a CH query with v as source. Bauer et al. [3] then show that the size of the search space of nodes in a graph G with treewidth k is in $O(k \log n)$.

Funke and Storandt [7] follow the idea of investigating search space sizes in CH instead of the query complexity itself. They construct the overlay graph in a random fashion and propose that *bounded growth* is a possible characteristic to explain the success of CH. We recapitulate bounded growth in Section 6.2. Based on these results, Blum et al. [4] show that if a graph G exhibits bounded growth then the random construction produces expected search space sizes in $O(\sqrt{n} \log n)$ and expected *direct* search space sizes in $O(\sqrt{n})$. We explain the difference between search space and direct search space in Section 4.4.

The size of the search space, as typically defined [2], is not an upper bound on the query complexity, though, as it only depends on the number of nodes. The number of edges that need to be considered could be much larger and in the worst case quadratic in the number of nodes. However, the common assumption was that for network-like graphs this worst case upper bound would turn out to be far too pessimistic. In fact, there has been the tendency to neglect the difference between search space sizes and the query complexity of CH. In this paper we show that, unfortunately, both the results of Bauer et al. [3] and those of Blum et al. [4] are good examples that the difference can be significant. In both cases, we show this by constructing lower bounds of $\Omega(n)$ for the (expected) number of processed edges in the respective settings.

*Universität Stuttgart, Germany
†Universität Stuttgart, Germany

2 Related Work

While there has been a lot of work about upper bounds on the query complexity of CH, there are only few publications about lower bounds. Rupp and Funke [9] show that one can construct grid graphs with average search space sizes of $\Omega(\sqrt{n})$.

Blum and Storandt [5] study bounds on the search space size with the focus on lower bounds. For instance, they show that the average search space size in a weak CH G is in $\Omega(b_\alpha)$ for $\alpha \geq 2/3$, where b_α is the smallest α-balanced node separator in G. We do not introduce the notion of weak CH here. Informally, a weak CH consists of more shortcuts such that it preserves shortest paths under any edge cost function.

White [11] shows that for any highway dimension h and any diameter D one can construct a graph G such that queries of the CH of G take $\Omega((h \log D)^2)$ time. This means that the upper bound on the query complexity of Abraham et al. [1] is tight. However, the theorem of White [11] does not upper bound the size of G.[1]

3 Contribution

In this paper we give examples that the difference between the search space sizes with respect to nodes and edges can be significant. Firstly, we address a separator-based CH-construction scheme described by [3] which yields node search spaces of $O(\sqrt{n})$ in the case of grids. We show that this contraction scheme can lead to edge search spaces of size $\Omega(n)$ for a specific family of grid graphs. We also show that for the same graphs significantly better CH-constructions exist which yield edge search spaces of size $O(\sqrt{n} \log n)$.

Secondly, Blum et al. [4] proved that for any graph family with bounded growth the expected node search space size of randomly constructed CH is in $O(\sqrt{n} \log n)$. We give an example that satisfies the requirements in [4] and where the expected search space sizes with respect to edges is in $\Omega(n)$.

Hence, we show that the gap between the average search space size with respect to nodes and the average query complexity can be surprisingly large, even for grid graphs.

We achieve these results by counting edges that need to be processed during queries. This is in contrast to most of the previous work, where edges are neglected and only nodes are considered. Thus, with our examples we are able to emphasize how important it is to take edges into account when analyzing the query complexity in CH.

[1] Private communication with C.White.

4 Preliminaries

In this section we summarize the previous work necessary for the remainder of the paper. Furthermore, we introduce some notation.

4.1 Contraction Hierarchies CH is a speed-up technique for shortest path computations [8]. The basic idea is to construct an overlay graph that consists of shortcut edges (or: shortcuts). These shortcuts reduce the hop length of shortest paths and allow a modified version of bidirectional Dijkstra's algorithm to explore the graph more efficiently.

First, the overlay graph is constructed in a preprocessing phase. It starts with the original graph $G = (V, E)$ and performs the so-called contraction operation until all nodes in V are contracted. During the contraction of node v, node v as well as its adjacent edges are removed from the graph. Afterwards, shortcut edges are added between neighbors of v to ensure that their distances are preserved. A shortcut edge is inserted between the neighbors u and w of v if and only if uvw was a shortest path. The cost of the new shortcut edge is set to the cost of the path uvw.

The order of contraction determines the rank of the nodes. A node v has rank k if it is the k-th node to be contracted. The result of the preprocessing phase is an overlay graph $G^+(V, E^+)$. The set of edges E^+ is the union of the original edges and the shortcuts. An edge $(u, v) \in E^+$ is called upwards if the rank of u is lower than the rank of v. Otherwise, the edge is called downwards.

One can show that for each node pair $u, v \in V$ (and v is reachable from u in G) there is a shortest path from u to v in G^+ which can be split into two parts. The first part (starting from u) exclusively consists of upward edges, while the second part only consists of downward edges. That means a bidirectional search with Dijkstra's algorithm does only need to consider upward edges in the forward direction and downward edges in the backward direction. This modification has shown to reduce the search space of road-network graphs significantly in practice.

4.2 Notation We use the same notation for nodes and ranks. For instance, for two nodes v, $u \in V$ the term $v < u$ means that node v has smaller rank than node u.

Furthermore, for a graph $G(V, E)$ with weighted edges and two nodes v, u in V we write $\pi(v, u)$ for the set of nodes on the shortest path from v to u (including v and u). Let $l(v, u)$ be the length of $\pi(v, u)$, which is the sum of the weights of the edges along $\pi(v, u)$.

4.3 Search Space Given the overlay graph $G^+(V, E^+)$ of a CH. The search space of a node $v \in V$, as typically defined [2] [3], is given by the set $SS(v) \subseteq V$ that consist of all nodes reachable from v while using upward edges only. Remember that an edge $(u, w) \in E^+$ is an upward edge if and only if the rank of node w is greater than the rank of node u.

4.4 Direct Search Space We define the notion of Direct Search Space (DSS) according to Funke and Storandt [7].

DEFINITION 1. *Given a graph $G = (V, E)$ with a node order. The Direct Search Space $DSS(u)$ of a node $u \in V$ is defined as follows.*

$$DSS(u) := \{v \in V : \ \forall\, w \in \pi(u, v)\ w \leq v\},$$

where $\pi(u, v)$ is the shortest path from node u to node v.

In words, a node v is in $DSS(u)$ if the shortest path $\pi(u, v)$ from u to v does not contain any node (including u) with a higher rank than v. Note that $DSS(u) \subseteq SS(u)$ for any node u. That is due to the fact that any node v with the highest rank in the shortest path $\pi(u, v)$ can be reached from node u via upward edges. We refer to [8] for a formal proof of this fact. However, $DSS(u)$ may be smaller than $SS(u)$ as it additionally requires the upward edges to be part of shortest paths from node u.

4.5 Query Complexity As mentioned above, for CH a modified version of bidirectional Dijkstra's algorithm is used to compute shortest paths, which only follows upward edges. With the term *query complexity* we refer to the number of processing steps in a full upward search. While sometimes the bidirectional search can be aborted earlier, random source-target queries tend to be long-range and hence require a full upward search.

4.6 Edge Search Spaces Search spaces, as typically defined, only consider nodes. We now define analogous search spaces that consider edges.

DEFINITION 2. *Given a graph $G = (V, E)$ with a node order. The Edge Search Space $SS_e(u)$ of a node $u \in V$ is defined as follows.*

$$SS_e(u) := \{(v_1, v_2) \in E : \ v_1 \in SS(u),\, v_1 < v_2\}$$

Hence, $SS_e(u)$ contains all upward edges in the search space of node u. The size of $SS_e(u)$ is an upper bound of edges that need to be processed during queries from node u.

We introduce the Minimal Edge Search Space to obtain a lower bound for the query complexity.

DEFINITION 3. *Given a graph $G = (V, E)$ with a node order. The Minimal Edge Search Space $MSS_e(u)$ of a node $u \in V$ is defined as follows.*

$$MSS_e(u) := \{(v_1, v_2) \in E : \ v_1 \in DSS(u),\, v_1 < v_2\}$$

Hence, $MSS_e(u)$ contains all upward edges whose source is in the DSS of node u. We intentionally do not call this set *Direct* Edge Search Space because this name would imply that $MSS_e(u)$ only contained edges that are part of shortest paths from node u. This is in general not the case. Nonetheless, all elements in $MSS_e(u)$ must be considered in queries from node u. This remains true even if the common pruning scheme *stall-on-demand* is performed.

5 Analysis of Separator-based Contraction Schemes

In this section we look at differences between the node and edge search space sizes when CHs are constructed on basis of separators.

5.1 Divide and conquer strategy with separators For planar graphs, there is a straightforward divide-and-conquer CH-construction strategy to upper bound the node search spaces: Compute a balanced separator, then assign the highest ranks to nodes in the separator. Recursively separating the remaining graph in parts A and B has the effect that for nodes $a \in A$ and $b \in B$ no shortcut edge (a, b) needs to be constructed. Having few shortcuts makes the analysis to show that the node search spaces have size $O(\sqrt{n})$ for a grid graph quite simple. For a more generic result and a formal proof, see Bauer et al. [3] (Theorem 5). While their result clearly upper bounds the number of *nodes* to be considered during a query, this might be considerably less than the number of edges to look at. We apply a balanced separator strategy upon lightheaded grids, a family of graphs which was introduced by Rupp and Funke [9] for the specific purpose of showing lower bounds on CH constructions. For this construction, we show that the average minimal edge search space $MSS_{e,avg}$ is in $\Omega(n)$. We also show that it is not the CH framework to blame by providing an alternative construction scheme for these specific graphs which yields sublinear $SS_{e,avg}$ of size $\mathrm{O}(\sqrt{n} \log n)$.

5.2 Lightheaded Grid An example of a lightheaded grid according to Rupp and Funke [9] can be seen in Figure 1. The intention of the grid is that shortest paths have a generic structure and are unambiguous, an illustration of a shortest path tree can be seen in Figure 2. It is important to keep the generic structure of the shortest paths in mind for the following analysis

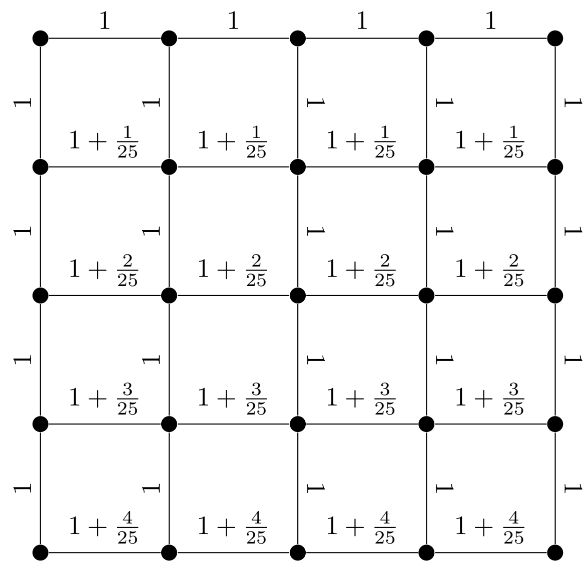


Figure 1: Lightheaded grid for $n = 25$.

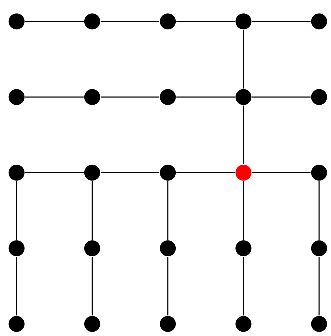


Figure 2: Shortest path tree from red source.

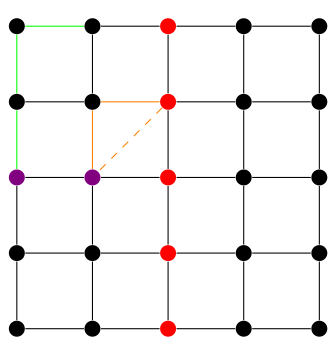


Figure 3: Red and purple separators with example shortest paths.

to keep track where shortcuts have to be added or not.

5.3 Separator Strategy Now we show that for lightheaded grids, the seemingly reasonable separator strategy can yield linear edge search spaces.

THEOREM 5.1. *Applying the divide-and-conquer separator strategy as described in Section 5.1 on a lightheaded grid can yield $MSS_{e,avg}$ of size $\Omega(n)$.*

Proof. We assign the highest ranks according to the separators as indicated in Figure 3: the red nodes form the separator of the whole grid and are assigned to the highest ranks. The purple nodes form the separator of the left part and therefore get smaller ranks than the red nodes but they are ranked higher than the black nodes. The rank order within the red, purple and black nodes can be arbitrary.

Let B be the set of all black nodes in the upper left quadrant. We show that for the individual edge search spaces holds: $\forall b \in B : |MSS_e(b)| \in \Omega(n)$. As $|B| = \Theta(n)$ this will suffice for our claim.

Let now P be the set of all purple nodes and R all the red nodes in the upper half without the center node. Note that $|P| = |R| \in \Omega(\sqrt{n})$.

Due to shortest paths between nodes $p \in P$ and $r \in R$ as illustrated in orange, there have to be shortcuts (p, r) as indicated by the tilted line. So, for any $p \in P, \forall r \in R : (p, r) \in MSS_e(p)$.

Also, all black nodes reach all purple nodes as indicated with the green path, so $\forall b \in B, \forall r \in R : r \in DSS(b)$. Note that shortcut edges (b, r) do not have to exist, though.

Putting our last observations together we notice that $\forall b \in B : \dot{\bigcup}_{p \in DSS(b) \cap P} \{(p, r) | r \in R\} \subseteq MSS_e(b)$. As we have $|P| \cdot |R| \in \theta(n)$ shortcuts which are contained in each search space from a node $b \in B$, we have $\forall b \in B : |MSS_e(b)| \in \Omega(n)$. ◻

5.4 Improved Construction Having just seen linear average edge search spaces for a seemingly reasonable constructed CH may invoke doubts about the theoretical efficiency of the CH framework itself. However, we show that sublinear edge search spaces are still possible here.

THEOREM 5.2. *Lightheaded grids admit CH constructions for which $|SS_{e,avg}| \in O(\sqrt{n} \log n)$.*

Proof. For the construction we still use separators. The scheme can be best understood by looking at the example in Figure 4, where each individual node is labeled with its rank. For simplicity, we refer to a node by its rank label in the following.

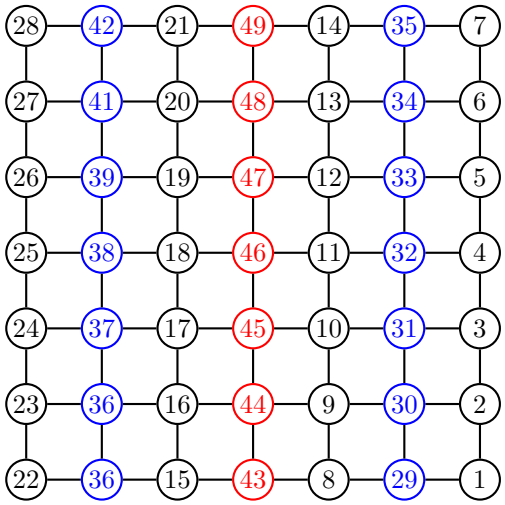


Figure 4: Vertical Red and blue separators.

To make the analysis easy, the ranks were assigned in such a way that no vertical or diagonal shortcuts have to be added during the contraction process. Only horizontal shortcuts may appear, e.g. when node 8 is contracted, there will be the shortcut $(29, 43)$.

The black nodes have the lowest ranks and the nodes at the bottom have lower ranks compared to other nodes in the same column. Therefore, the lowest black nodes have the largest edge search spaces and w.l.o.g we can focus on node 1. For $SS_e(1)$, all the vertical edges from 1 to 7 are included, which are $O(\sqrt{n})$ in the general case. Then all the horizontal edges to the next vertical blue separator 29-35 are included, which are $(1, 29), (2, 30), \cdots, (7, 35)$, also $O(\sqrt{n})$ many. The edges within the blue separator 29-35 are also in the search space, but those are only $O(\sqrt{n})$ again.

From the separator 29-35, only 7 (vertical) upward shortcuts go directly to the next highest separator and are therefore part of $SS_e(1)$ while the columns of nodes inbetween are irrelevant. For the general case of a bigger grid we continue this scheme of analyzing the search space by proceeding with the next highest separator successively.

As we encounter $O(\log n)$ separators in general and each is contributing $O(\sqrt{n})$ edges, we get $|SS_{e,avg}| \in O(\sqrt{n} \log n)$. □

6 Expected Query Complexity in Graphs with Bounded Growth

In this section we revisit the analysis of the query complexity in graphs with bounded growth. In contrast to the work of Blum et al. [4] we consider edge search spaces instead of node search spaces. We start with some definitions and remarks about our notation that are specifically relevant for this section. The actual analysis starts in Section 6.5.

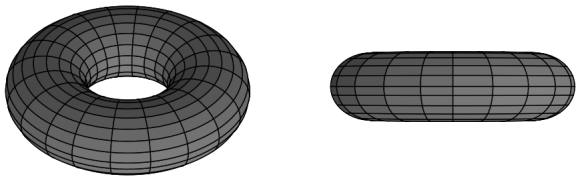


Figure 5: Typical visualizations of a torus graph

6.1 Notation We define the distance between two nodes u and v as $d_{u,v} := \lceil l(u, v) \rceil$. The reason for this unconventional definition is that for the following analysis it is convenient to have an integral upper bound of the length of $\pi(u, v)$. In Section 6.3 we introduce graphs where the value $d_{u,v}$ is equal to the number of hops on the shortest path of any two nodes u and v.

6.2 Bounded Growth To the best of our knowledge bounded growth has so far been defined for graphs with integral edge weights only [7], [4]. As we introduce small perturbations in the edge weights to conveniently ensure unique shortest paths we extend the notion of bounded growth to graphs with any non-negative edge weights.

A graph family $G(V, E)$ has bounded growth if there is a constant c such that for any radius r and any node u

$$|\{v \in V : \ d_{u,v} = r\}| \leq cr.$$

Note that $d_{u,v}$ is defined to be the smallest integer that is greater or equal to the distance between nodes u and v. In words, the number of nodes that have a distance (almost) r from node u is upper bounded by $c \cdot r$.

6.3 Torus Graph We introduce a graph family which we call *torus graph* as it has the structure of a torus. The graph family is subject to the analysis in the following section. A torus graph can be considered as a grid without boundaries. For the sake of simplicity, the edges of this graph are undirected.

DEFINITION 4. *Let $G = (V, E)$ be an undirected graph with $n = k^2$ nodes, $k \in \mathbb{N}$. Let the nodes be labeled with the numbers from 1 to n such that each node has a unique number. G is called a torus graph if there exists*

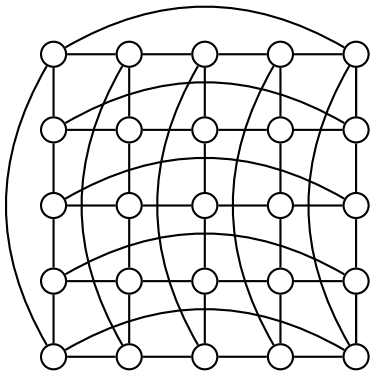


Figure 6: Example torus graph with $k = 5$

a labeling such that

$$E = \{(u,v) : |u - v| \equiv 1 \mod k\} \cup \{(u,v) : |u - v| \equiv k \mod n\}.$$

We can construct a torus graph of k^2 nodes by starting with a grid graph of that size. Then, we add edges connecting opposite boundaries of the grid. If the boundary is a column, then each node in that column is connected to the node in the same row of the opposite boundary. We proceed accordingly with boundaries that are rows. Figure 6 shows an example. A possible labeling in this example would be to count the nodes from left to right, top to bottom. In that case the set $\{(u,v) : |u - v| \equiv 1 \mod k\}$ contains the horizontal edges and the set $\{(u,v) : |u - v| \equiv k \mod n\}$ contains the vertical edges.

The torus graph is similar to grid graphs, which appear in real world networks. However, the torus graph shows symmetries grids do not have. In fact, from each node the adjacency structure of the torus graph looks identical.

To make the following analysis as easy as possible, we would like to have unit edge length and unique shortest paths. However, both is not possible. Therefore, we choose the edge lengths in the interval $(1 - \frac{1}{n}, 1)$ such that no two different optimal paths have the same length. If the edge lengths are selected uniformly at random, this requirement is almost certainly satisfied (probability equals one). Hence, the existence of edge lengths that lead to unique shortest paths is guaranteed.

Note that since the edge lengths are between $1 - \frac{1}{n}$ and 1, each shortest path would also be a shortest path if each edge had the length 1. Moreover, the distance $d_{u,v}$ (as defined in Section 4.2) between any two nodes u and v is equal to the number of hops on the shortest path $\pi(u,v)$.

The torus graph is a graph family with bounded growth. The bounded growth constant c is 4. Furthermore, for distances $r < \frac{k}{2}$ and unit edge length there are exactly $4r$ neighbors with that distance. This is true for each node due to the symmetry of the torus graph.

6.4 Random Contraction Based on the work of Blum et al. [4] we study CH with random contraction order. Given a graph G with n nodes. First, a permutation of the numbers from 1 to n is chosen uniformly at random. Second, G is contracted in this order. We call CH of this kind Random Contraction Hierarchies (random CH). We say that a graph has a node order if the nodes have ranks. If the ranks are randomly chosen we say that the graph has a random node order.

6.5 Analysis of Edge Search Spaces in Torus Graphs In this section we show that for the torus graph the expected number of edges that need to be considered during a query of a random CH is in $\Omega(n)$. The graphs in this section always have a node order. Therefore, each element $u \in V$ does not only denote a node in G but also a rank.

DEFINITION 5. *Let $G = (V, E)$ be an undirected graph with node order. We define*

$$X_u := \{v \in V : \forall w \in \pi(u,v) \backslash \{u,v\}\ w < u\},$$

where $\pi(u,v)$ denotes the shortest path from u to v.

See Figure 7 for an example. In words, a node v is in X_u if the shortest path between u and v (excluding u and v) does exclusively consist of nodes with smaller rank than u. Hence, if $v \in X_u$ then all nodes in $\pi(u,v)$ are in X_u. The following lemma shows the meaning of X_u.

LEMMA 6.1. *Given undirected Contraction Hierarchies $G^+ = (V, E^+)$ and two nodes $u, v \in V$ with $v \in X_u$.*

1. *If $v < u$, then $u \in DSS(v)$.*
2. *If $v > u$, then $(u,v) \in E^+$.*

Proof. First, we assume that $v < u$. Since $v \in X_u$, we know that the shortest path $\pi(u,v)$ does not contain any node with a higher rank than u. Since we only consider undirected graphs, $\pi(u,v) = \pi(v,u)$. It directly follows that $u \in DSS(v)$. Note that $DSS(v)$ denotes the Direct Search Space of v as defined in Section 4.4.

Now let $v > u$. From the definition of X_u we know that v is the only node in the shortest path $\pi(u,v)$ with a higher rank than u. If $\pi(u,v) = \{u,v\}$ we know that $(u,v) \in E^+$. Therefore, w.l.o.g. we assume that $\pi(u,v)$ consists of more than two nodes. During preprocessing,

all nodes between u and v in $\pi(u,v)$ are contracted before u and v. Therefore, when the last node between u and v in $\pi(u,v)$ is contracted, a shortcut (u,v) is inserted. □

Our goal is to count edges that need to be processed in a CH query. Surprisingly, with the help of the sets X_u we can do so by considering triples of nodes. Say we have three nodes u_1, u_2 and u_3 with the ranks $u_1 < u_2 < u_3$ and $u_1, u_3 \in X_{u_2}$. From Lemma 6.1 we know that u_2 is in $DSS(u_1)$. Hence, u_2 is settled during queries from u_1. Moreover, there is an edge $(u_2, u_3) \in E^+$ (Lemma 6.1). Hence, even though it may happen that u_3 is not further processed, the edge (u_2, u_3) definitely needs to be considered during queries from u_1 (because it is an upward edge and u_2 is in $DSS(u_1)$).

Before we continue, we give a short sketch of the remainder of this section. First, we define the Total Minimal Edge Search Space (TMESS) to be the set of all triples that have the form of the triple mentioned above. Afterwards, we prove that the cardinality of this set divided by the number of nodes is a lower bound of the average query complexity. Finally, we show that the expected cardinality of the TMESS is in $\Omega(n^2)$, which completes the proof of our claim.

DEFINITION 6. *Let $G = (V, E)$ be an undirected graph with node order. The Total Minimal Edge Search Space (TMESS) of G is given by*

$$T := \{(u_1, u_2, u_3) \in V^3 : \; u_1 < u_2 < u_3 \text{ and } u_1, u_3 \in X_{u_2}\}.$$

Note that the size of the TMESS T can be expressed as follows

$$|T| = \sum_{v \in V} |MSS_e(v)|,$$

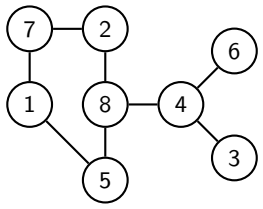
where $MSS_e(v)$ is the Minimal Edge Search Space of node v as defined in Section 4.6. This holds because each triple (u_1, u_2, u_3) on the left hand side corresponds to an edge (u_2, u_3) in $MSS_e(u_1)$ on the right hand side and vice versa.

For the graph shown in Figure 7

$$T = \{(1,5,7), (1,5,8), (1,7,8), (2,7,8), (3,4,6), (3,4,8), (3,6,8), (4,6,8), (5,7,8)\}.$$

Think of an element $(u_1, u_2, u_3) \in T$ as an edge (u_2, u_3) in the DSS of u_1 (as motivated above).

LEMMA 6.2. *Let $G^+ = (V, E^+)$ be an undirected CH. The value $\frac{1}{n}|T|$ is a lower bound of the average query complexity in G^+, where T is the TMESS of G^+.*



X_1	{5,7}
X_2	{7,8}
X_3	{4}
X_4	{3,6,8}
X_5	{1,7,8}
X_6	{3,4,8}
X_7	{1,2,5,8}
X_8	{1,2,3,4,5,6,7}

Figure 7: Example graph with node order and unit edge length

Proof. Let T be the TMESS of G^+. We fix a node $u_1 \in V$ and take all elements of the form $(u_1, u_2, u_3) \in T$ (remember that $u_1 < u_2 < u_3$). By definition of T, node u_2 is in $DSS(u_1)$. Moreover, as G^+ is a CH and by definition of T, there must be an edge $(u_2, u_3) \in E^+$. Since $u_2 < u_3$, this edge must be considered during queries from node u_1. Hence, the number of elements of the form $(u_1, u_2, u_3) \in T$ is a lower bound of the query complexity if the source node is u_1. It directly follows that $\frac{1}{n}|T|$ is a lower bound of the average query complexity. □

In the following we show that the expected size of T is in $\Omega(n^2)$. This result in combination with Lemma 6.2 proves our claim that the expected query complexity is in $\Omega(n)$.

In fact, it turns out that for our purposes it is sufficient to consider a subset of T. We divide the nodes into three disjoint sets L, M and H according to their ranks (low, medium and high ranks). The sets are defined as follows.

$$L := \{u \in V \mid u \leq n - \sqrt{n}\}$$

$$M := \{u \in V \mid n - \sqrt{n} < u \leq n - \frac{\sqrt{n}}{2}\}$$

$$H := \{u \in V \mid n - \frac{\sqrt{n}}{2} < u\}$$

It is easy to see that there are $\Omega(n^2)$ different triples (u_1, u_2, u_3) of the form $u_1 \in L$, $u_2 \in M$ and $u_3 \in H$. Let $\tilde{T}$ be the set of these triples:

$$\tilde{T} := \{(u_1, u_2, u_3) \mid u_1 \in L, u_2 \in M, u_3 \in H\}$$

Note that for each of these triples $(u_1, u_2, u_3) \in \tilde{T}$ it holds $u_1 < u_2 < u_3$. However, not every triple in $\tilde{T}$ is also contained in the TMESS T. Recall that each triple $(u_1, u_2, u_3) \in T$ satisfies two constraints. First, $u_1 < u_2 < u_3$ and second $u_1, u_3 \in X_{u_2}$. Some of the triples in $\tilde{T}$ may violate the second constraint. Hence, we can not directly lower bound the size of the TMESS T with the size of the set $\tilde{T}$.

We address this issue with the following lemma.

LEMMA 6.3. *One can find a constant $p > 0$ such that for any torus graph $G = (V, E)$ with random node order the following holds. For any $(u_1, u_2, u_3) \in \tilde{T}$ the probability $P((u_1, u_2, u_3) \in T) \geq p$, where T is the TMESS of G.*

Since the proof of Lemma 6.3 is rather lengthy, we give it at the end of this section. The lemma states that for any torus graph with random node order the expected size of the cut $T \cap \tilde{T}$ is lower bounded by $p|\tilde{T}|$, where p is a constant greater zero.

With the help of Lemma 6.3 we can now easily prove the following theorem.

THEOREM 6.1. *Let $G = (V, E)$ be a torus graph with random node order. The expected size of the TMESS $E[|T|] \in \Omega(n^2)$.*

Proof.

$$\begin{aligned} E[|T|] &= \sum_{u_2=1}^{n} \sum_{u_1=1}^{u_2-1} \sum_{u_3=u_2+1}^{n} P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\ &\geq \sum_{(u_1,\, u_2,\, u_3) \in \tilde{T}} P((u_1,\, u_2,\, u_3) \in T) \end{aligned}$$

We can lower bound the probability $P((u_1, u_2, u_3) \in T)$ with the constant $p > 0$ from Lemma 6.3.

$$\begin{aligned} E[|T|] &\geq \sum_{(u_1,\, u_2,\, u_3) \in \tilde{T}} p \\ &= \sum_{u_1=1}^{n-\sqrt{n}} \sum_{u_2=n-\sqrt{n}+1}^{n-\frac{\sqrt{n}}{2}} \sum_{u_3=n-\frac{\sqrt{n}}{2}+1}^{n} p \\ &= p(n-\sqrt{n})\frac{\sqrt{n}}{2}\frac{\sqrt{n}}{2} \in \Omega(n^2) \end{aligned}$$

□

The following corollary combines Theorem 6.1 and Lemma 6.2 to prove our previously stated claim.

COROLLARY 6.1. *The expected average query complexity of a random CH of a torus graph takes $\Omega(n)$ time.*

All that remains is to prove Lemma 6.3.

Proof. In this proof, we consider elements in V as ranks that are assigned to nodes by the random node order.

If the set $\tilde{T}$ is empty, there is nothing to show. Let (u_1, u_2, u_3) be any element of $\tilde{T}$. With a fixed element (u_1, u_2, u_3) we know the ranks of these nodes but not their positions within the torus graph. These depend on the random node order.

We introduce the random variables d_1 and d_3, where d_1 denotes the distance between the nodes with the ranks u_1 and u_2 as defined in Section 4.2. Analogously, the random variable d_3 denotes the distance between the nodes with the ranks u_3 and u_2. The term $P(d_1 = r)$, thus, denotes the probability that the distance between u_1 and u_2 is r.

Furthermore, let $\delta_\pi \in \{0, 1\}$ be the random variable that is one if $u_3 \in \pi(u_1, u_2)$ and zero otherwise. With the law of total probability we get

$$\begin{aligned} &P((u_1,\, u_2,\, u_3) \in T) \\ &= P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\ &= \sum_{i=0}^{1} P(\delta_\pi = i) P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = i). \end{aligned}$$

We know that u_3 cannot be part of $\pi(u_1, u_2)$ if $u_1 \in X_{u_2}$ (because $u_3 > u_2$). Therefore, the term simplifies to

$$\begin{aligned} &P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\ &= P(\delta_\pi = 0) P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0). \end{aligned}$$

We further transform the equation by applying the law of total probability two more times.

$$\begin{aligned} &P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\ &= P(\delta_\pi = 0) P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0) \\ &= \sum_{r_3=1}^{D} \Big(P(d_3 = r_3) P(\delta_\pi = 0 \mid d_3 = r_3) \\ &\qquad P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0 \cap d_3 = r_3) \Big) \end{aligned} \tag{6.1}$$

$$\begin{aligned} &= \sum_{r_1=1}^{D} P(d_1 = r_1) \sum_{r_3=1}^{D} \Big(P(d_3 = r_3 \mid d_1 = r_1) \\ &\quad P(\delta_\pi = 0 \mid d_1 = r_1 \cap d_3 = r_3) \\ &\quad P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0 \cap d_1 = r_1 \cap d_3 = r_3) \Big) \end{aligned}$$

where D is the diameter of G. In the following, we analyze this lengthy equation term by term.

We start with $P(d_1 = r_1)$. Let $N(v, r)$ be the number of nodes from node v with distance r. Due to the symmetric structure of the torus graph the function N does not depend on the node v. Hence, we can write $N(r)$ instead. Since u_1 can be any node other than u_2 with the same likelihood we have

$$P(d_1 = r_1) = \frac{N(r_1)}{n-1}.$$

Our following considerations only require a rough estimate of this term. It is easy to see that for any torus

graph we have

$$N(r) \geq r \text{ if } r \leq \frac{\sqrt{n}}{2}.$$

Note that for $r < \frac{\sqrt{n}}{2}$ we have $N(r) = 4r$. Hence, we obtain the lower bound

$$P(d_1 = r_1) \geq \frac{r_1}{n} \text{ if } r_1 \leq \frac{\sqrt{n}}{2}.$$

Next, we estimate the term $P(d_3 = r_3 \mid d_1 = r_1)P(\delta_\pi = 0 \mid d_1 = r_1 \cap d_3 = r_3)$ in Equation 6.1. We distinguish between the two cases $r_1 = r_3$ and $r_1 \neq r_3$.

If $r_1 = r_3$, we know that $P(d_3 = r_3 \mid d_1 = r_1) = \frac{N(r_3)-1}{n-2}$ because there is one "free seat" less with distance r_3 to node u_2 and there are $n-2$ possible candidates (any node except for u_1 and u_2). Moreover, u_3 cannot be part of $\pi(u_1, u_2)$ if it has the same distance to u_2 as u_1. Hence, $P(\delta_\pi = 0 \mid d_1 = r_1 \cap d_3 = r_3) = 1$.

If $r_1 \neq r_3$, $P(d_3 = r_3 \mid d_1 = r_1) = \frac{N(r_3)}{n-2}$ and $P(\delta_\pi = 0 \mid d_1 = r_1 \cap d_3 = r_3) \geq \frac{N(r_3)-1}{N(r_3)}$ because at each distance r_3 there is at most one node that is part of $\pi(u_1, u_2)$ (there could be none if $r_3 > r_1$).

In both cases $P(d_3 = r_3 \mid d_1 = r_1)P(\delta_\pi = 0 \mid d_1 = r_1 \cap d_3 = r_3) \geq \frac{N(r_3)-1}{n-2}$ holds. For distances $r \leq \frac{\sqrt{n}}{2}$ the rough lower bound $N(r) \geq r+1$ holds for any torus graph. Hence, we obtain the lower bound

$$P(d_3 = r_3 \mid d_1 = r_1)P(\delta_\pi = 0 \mid d_1 = r_1 \cap d_3 = r_3) \geq \frac{r_3}{n} \quad \text{for } r_3 \leq \frac{\sqrt{n}}{2}.$$

We address the remaining term $P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0 \cap d_1 = r_1 \cap d_3 = r_3)$ in Equation 6.1 from a combinatorial point of view. But first, let us put in words what the term means. It is the conditional probability that u_1 and u_3 are in X_{u_2} if the distances of these nodes to u_2 are known and if u_3 is not part of the shortest path $\pi(u_1, u_2)$.

Let $\Pi := \pi(u_1, u_2) \cup \pi(u_2, u_3)$ be the union of the two shortest paths from u_1 and u_3 to u_2. The only thing we know about the set Π for sure is that it contains u_1, u_2 and u_3. The remaining $|\Pi| - 3$ nodes are unknown to us. However, in order to make the term $u_1 \in X_{u_2} \cap u_3 \in X_{u_2}$ true, these nodes must have a smaller rank than u_2. Hence, there are $u_2 - 2$ nodes we can choose from (u_1 is already taken). On the other hand, there are $n-3$ possible candidates that could be part of Π. Due to the randomness of the node order, each node among these $n-3$ candidates is part of Π with the same likelihood. That leads to

$$P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0 \cap d_1 = r_1 \cap d_3 = r_3) = \frac{\binom{u_2-2}{|\Pi|-3}}{\binom{n-3}{|\Pi|-3}}.$$

The problem remains that we do not know the size of Π. However, we can easily see that the probability decreases with increasing size of Π. That means, an upper bound of the size of Π leads to a lower bound of the considered probability. Taking the distances $d_1 = r_1$ and $d_3 = r_3$ into account it is clear that

$$|\Pi| \leq r_1 + r_3 + 1.$$

Hence, we get

$$P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2} \mid \delta_\pi = 0 \cap d_1 = r_1 \cap d_3 = r_3) \geq \frac{\binom{u_2-2}{r_1+r_3-2}}{\binom{n-3}{r_1+r_3-2}}.$$

We are able to get rid of the binomial coefficients with the following considerations.

$$\begin{aligned} \frac{\binom{u_2-2}{r_1+r_3-2}}{\binom{n-3}{r_1+r_3-2}} &= \frac{\prod_{i=1}^{r_1+r_3-2} u_2 - i - 1}{\prod_{i=1}^{r_1+r_3-2} n - i - 2} \\ &\geq \left(\frac{u_2 - r_1 - r_3}{n}\right)^{r_1+r_3-2} \\ &\geq \left(\frac{u_2 - \sqrt{n}}{n}\right)^{r_1+r_3-2} \quad \text{if } r_1, r_3 \leq \frac{\sqrt{n}}{2}. \end{aligned} \tag{6.2}$$

We obtain Equation (6.2) by replacing each factor $u_2 - i - 1$ of the numerator with the lower bound $u_2 - r_1 - r_3$ and each factor $n - i - 2$ of the denominator with the upper bound n.

We now come back to Equation (6.1). We substitute the diameter D with $\frac{\sqrt{n}}{2}$. In this way, we only consider distances d_1 and d_2 that are less or equal to $\frac{\sqrt{n}}{2}$. Hence, we can apply the previously shown lower bounds.

$$\begin{aligned} &P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\ &\quad \geq \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} \frac{r_1}{n} \sum_{r_3=1}^{\frac{\sqrt{n}}{2}} \frac{r_3}{n} \left(\frac{u_2 - \sqrt{n}}{n}\right)^{r_1+r_3-2} \\ &\quad \geq \frac{1}{n^2} \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} r_1 \sum_{r_3=1}^{\frac{\sqrt{n}}{2}} r_3 \left(1 - \frac{2}{\sqrt{n}}\right)^{r_1+r_3-2} \end{aligned} \tag{6.3}$$

In Equation (6.3) we move the variable n to the front and replace u_2 by its lower bound $n - \sqrt{n}$, which is a prerequisite of the lemma.

We continue with $x := 1 - \frac{2}{\sqrt{n}}$.

$$\begin{aligned}
&P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\
&\geq \frac{1}{n^2} \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} r_1 \sum_{r_3=1}^{\frac{\sqrt{n}}{2}} r_3 x^{r_1+r_3-2} \\
&= \frac{1}{n^2} \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} r_1 x^{r_1-1} \sum_{r_3=1}^{\frac{\sqrt{n}}{2}} r_3 x^{r_3-1}
\end{aligned}$$

Writing the lower bound in this way makes it clear that we are facing two nested arithmetico-geometric sums. Remember that for any $x \geq 0$ and $m \in \mathbb{N}$ it holds

$$\begin{aligned}
\sum_{i=1}^{m} i x^{i-1} &= \frac{1 - (m+1)\, x^m + m x^{m+1}}{(1-x)^2} \\
&= \frac{1 - (m + 1 - mx)\, x^m}{(1-x)^2}.
\end{aligned}$$

With $m = \frac{\sqrt{n}}{2}$ we get

$$\begin{aligned}
&P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) \\
&\geq \frac{1}{n^2} \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} r_1 x^{r_1-1} \frac{1 - \left(\frac{\sqrt{n}}{2} + 1 - \frac{\sqrt{n}}{2} x\right) x^{\frac{\sqrt{n}}{2}}}{(1-x)^2} \\
&= \frac{1}{n^2} \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} r_1 x^{r_1-1} \frac{1 - \left(\frac{\sqrt{n}}{2} + 1 - \frac{\sqrt{n}}{2} \left(1 - \frac{2}{\sqrt{n}}\right)\right) x^{\frac{\sqrt{n}}{2}}}{(1-x)^2} \\
&= \frac{1 - 2x^{\frac{\sqrt{n}}{2}}}{n^2 (1-x)^2} \sum_{r_1=1}^{\frac{\sqrt{n}}{2}} r_1 x^{r_1-1}.
\end{aligned}$$

Again, we apply the formula for arithmetico-geometric sums. With a similar calculation as above we get

$$\begin{aligned}
P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) &\geq \frac{\left(1 - 2x^{\frac{\sqrt{n}}{2}}\right)^2}{n^2 (1-x)^4} \\
&= \frac{1}{16} \left(1 - 2x^{\frac{\sqrt{n}}{2}}\right)^2 \\
&= \frac{1}{16} \left(1 - 2\left(1 - \frac{2}{\sqrt{n}}\right)^{\frac{\sqrt{n}}{2}}\right)^2.
\end{aligned}$$

Let

$$f(n) := \left(1 - \frac{2}{\sqrt{n}}\right)^{\frac{\sqrt{n}}{2}}.$$

One can show that the function f is monotonically increasing for $n \geq 4$ and that

$$\lim_{n \to \infty} f(n) = \frac{1}{e},$$

where e is Euler's number. In particular, $f(n) < \frac{1}{2}$ for $n \geq 4$. That means, for $n \geq 4$

$$\begin{aligned}
P(u_1 \in X_{u_2} \cap u_3 \in X_{u_2}) &\geq \frac{1}{16} \left(1 - \frac{2}{e}\right)^2 \\
&> 0.
\end{aligned}$$

Since the lemma presumes the existence of at least three nodes and since the smallest torus graph with at least three nodes has four nodes, the lemma is true for any torus graph. □

7 Conclusion

Bauer et al. [3] and Blum et al. [4] showed that for certain graph families, (expected) node search spaces were rather small, i.e. sublinear. It is clear that upper bounds on search space sizes are not necessarily upper bounds on the query complexity as well (even though the name *search space* suggests it). However, the common assumption was that for transportation network-like graphs the difference between these two quantities would be minor.

On the basis of lightheaded grids from Rupp and Funke [9] we showed that the separator-based construction scheme of Bauer et al. [3] can indeed lead to linear edge search spaces. We also showed that the CH framework was not the problem as these specific graphs admit significantly better CH-constructions with respect to the edge search space.

Concerning the result of Blum et al. [4], we introduced a graph family with bounded growth that is very similar to grid graphs and showed that the expected query complexity of its random CH is in $\Omega(n)$. We achieved this result by considering the edges that are processed during queries instead of the nodes. Thus, we presented another relevant example where the difference between the search space sizes and the query complexity is significant.

Note that linear edge search spaces mean that the number of processed edges in CH queries are asymptotically equal to the number of edges processed by Dijkstra's algorithm. A speed-up compared to Dijkstra's algorithm is therefore uncertain in these cases.

Our results motivate two things. First, the query complexity of CH mainly depends on the number of processed edges, not on the nodes. Second, our examples show that the difference between the query complexity and the search space sizes can be surprisingly large.

In summary, there is still no really satisfying theoretical explanation for the tremendous efficiency of speedup techniques like CH in practice. We hope that this paper stimulates further research in that direction.

References

[1] Ittai Abraham, Amos Fiat, Andrew V Goldberg, and Renato F Werneck. 2010. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 782–793.

[2] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. 2016. Route planning in transportation networks. In *Algorithm engineering*. Springer, 19–80.

[3] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. 2016. Search-space size in contraction hierarchies. *Theoretical Computer Science 645* (2016).

[4] Johannes Blum, Stefan Funke, and Sabine Storandt. 2018. Sublinear search spaces for shortest path planning in grid and road networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[5] Johannes Blum and Sabine Storandt. 2020. Lower Bounds and Approximation Algorithms for Search Space Sizes in Contraction Hierarchies. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[6] Edsger W Dijkstra et al. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[7] Stefan Funke and Sabine Storandt. 2015. Provable efficiency of contraction hierarchies with randomized preprocessing. In *International Symposium on Algorithms and Computation*. Springer, 479–490.

[8] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *International Workshop on Experimental and Efficient Algorithms*. Springer, 319–333.

[9] Tobias Rupp and Stefan Funke. 2020. A lower bound for the query phase of contraction hierarchies and hub labels. In *International Computer Science Symposium in Russia*. Springer, 354–366.

[10] Sabine Storandt. 2013. Contraction hierarchies on grid graphs. In *Annual Conference on Artificial Intelligence*. Springer, 236–247.

[11] Colin White. 2015. Lower bounds in the preprocessing and query phases of routing algorithms. In *Algorithms-ESA 2015*. Springer, 1013–1024.

Fairmandering: A column generation heuristic for fairness-optimized political districting*

Wes Gurnee† David B. Shmoys‡

Abstract

The American winner-take-all congressional district system empowers politicians to engineer electoral outcomes by manipulating district boundaries. Existing computational solutions mostly focus on drawing unbiased maps by ignoring political and demographic input, and instead simply optimize for compactness. We claim that this is a flawed approach because compactness and fairness are orthogonal qualities, and introduce a scalable two-stage method to explicitly optimize for arbitrary piecewise-linear definitions of fairness. The first stage is a randomized divide-and-conquer column generation heuristic which produces an exponential number of distinct district plans by exploiting the compositional structure of graph partitioning problems. This district ensemble forms the input to a master selection problem to choose the districts to include in the final plan. Our decoupled design allows for unprecedented flexibility in defining fairness-aligned objective functions. The pipeline is arbitrarily parallelizable, is flexible to support additional redistricting constraints, and can be applied to a wide array of other regionalization problems. In the largest ever ensemble study of congressional districts, we use our method to understand the range of possible expected outcomes and the implications of this range on potential definitions of fairness.

1 Introduction

Section 4, Article 1: "The Times, Places and Manner of holding Elections for Senators and Representatives, shall be prescribed in each State by the Legislature thereof; but the Congress may at any time by Law make or alter such Regulations." Other than this brief statement regarding regulatory authority, the Constitution of the United States offers little guidance on the process by which the members of the House of Representatives are elected. The current district-based system, where districts are most often drawn by state legislatures, enable self-interested and self-preserving politicians to secure a partisan advantage, suppress the vote of minority groups, and protect incumbents from competition. Such practices, broadly known as gerrymandering, enable large discrepancies between the popular vote and political representation, and ensure that electoral outcomes are unresponsive to swings in public opinion. In 2021, 428 congressional, 1938 state senate, and 4826 state house districts will be redrawn, cementing the partisan power balance in America for the next decade.

Given the mathematical richness and societal importance of redistricting, there is a large literature bridging computational and political sciences on the causes [13, 38], consequences [16, 26], and potential remedies of gerrymandering [30, 3, 42]. Until recently, nearly all proposed computational solutions to create more equitable maps focused on creating maximally compact maps drawn without any political data as input [42, 19, 15, 29, 28, 24, 34, 33, 8, 22]. This is in part due to the computational hardness of optimizing for fairness at scale; it is also due to the difficulty of acquiring granular and historical political data, and because some states explicitly ban the use of political data in the redistricting process [23]. While it is true that compactness-optimized maps are blind to partisan bias, it is not true these maps are free from partisan advantage. The two most successful lines of work that begin to address this gap in the literature are multi-level optimization algorithms [40] and ensemble methods, most notably recombination Markov chains [9].

Exact optimization methods are intractable for all but the smallest problems because optimal political districting is NP-hard for any useful objective function [31, 21, 5]. Optimization approaches for fairness therefore either rely on local search techniques [35, 20] or embed a heuristic in some stage of the optimization. The most successful formulation thus far does both by iteratively coarsening the problem input, solving to optimality for multiple objectives, and then uncoarsening with a local search routine at each level of granularity [40]. However, the number of districts in the problem instance limits the achievable coarsening and therefore such a scheme is unable to scale to large states or state legislative boundaries where there are a much greater number of district seats.

Ensemble methods [25, 6, 9] generate a large en-

*Supported in part by NSF grants CCF-1522054, CCF-1526067, and CCF-1740822, DMS-1839346, and CNS-1952063.

†Cornell University - rwg97@cornell.edu

‡Cornell University - david.shmoys@cornell.edu

semble of random maps to infer a property about the distribution of all feasible district plans. These methods are ideally suited for detecting gerrymandering [18, 10, 7] via outlier analysis with respect to the random ensemble of partisan blind plans and evaluating the effect of proposed rules [4] on the space of legal plans. However, because these techniques generate one map at a time (and are not optimizing for any property), they are not efficient at sampling from the tails of any distribution. This is problematic because extreme solutions, such as the most unfair solution under a new set of rules or the fairest possible map which maximizes other desirable qualities, are often the solutions we are looking for.

Our work is novel in that it combines these strands of literature and inherits the strengths of both. We introduce a new algorithm which creates an exponential ensemble of distinct district plans by exploiting the natural combinatorial compositionality of redistricting by generating an expressive set of districts that is also efficient to optimize over. The ensemble is exponential in k, the number of seats (districts) in a plan, making our approach ideally suited for larger states and state assembly districts. While these district plans are not appropriate for assessing certain statistical relationships given their lack of independence, they are nonetheless useful for understanding the range of possibilities that a state's geography and rules allow for. Additionally, by decoupling the generation of districts from the optimization of districts, our formulation enables a more flexible range of objective functions that we can use to optimize for arbitrary mappings from votes to seats, supporting a large family of fairness objectives.

Our core claim is that mapmakers should explicitly optimize for fairness rather than using questionable proxies like compactness. The main contribution of this paper is to introduce a new methodology to achieve this, while being extensible to a wide variety of other regionalization problems [12], particularly those where more complicated objectives outside of compactness are important. Our approach has a number of useful properties including parallelizability and scalability, making huge problem instances tractable, reusability of generated districts and flexibility of objectives, enabling rapid iteration of many definitions of fairness, as well as a modular and transparent execution that facilitates certain human-in-the-loop interactions. These properties allow us to solve problems at unprecedented scale and immediately provide value to independent commissions and other redistricting authorities to quickly and cheaply generate a large volume of high quality plans. In addition to detailed empirical results of different parameter configurations of our algorithm, we use its ensembling capabilities to perform the largest ever ensemble study to understand the range of possible expected electoral outcomes of all multi-district states and the implications of this range on potential definitions of fairness.

2 Methodology

2.1 Preliminaries

The Political Districting Problem (PDP) The basic input to the PDP is a set of geographic polygons and their respective populations. Each polygon is represented as a node in a planar graph $G = (B, E)$ where B is the set of atomic geographic blocks used to construct the districts and $(i, j) \in E$ whenever b_i and b_j are geographically adjacent. We let p_i represent the population of b_i and let d_{ij} be the Euclidean distance between the geographic centroids of b_i and b_j. In this setting, the atomic blocks are typically precincts, census tracts, census blocks, or an aggregation thereof. See Figure 1 for an example.

A feasible solution to the PDP for a state with k districts is a $k-$partition of $B = D_1 \cup \cdots \cup D_k$ (and $D_i \cap D_j = \emptyset$, for each pair $i, j \in \{1, \ldots, k\}, i \neq j$) such that each district D_j is population balanced, geographically contiguous, and compact. That is, each district D_i must have roughly equal population, the subgraph of G with nodes in D_i must be one fully connected component, and the district should have a geographic shape resembling a fairly simple convex polygon. A solution is called a districting or a district plan. Part of the challenge of this application domain is that these constraints are usually under-specified, with each state offering different levels of specificity and few national standards to fill the gaps. For Congressional districts, the Supreme Court ruled that the districts must be balanced "as nearly as is practicable" [44] but state legislature districts only require "substantial equality of population"[32]. In practice this means adding the constraint that for each district D_j

$$\hat{p}(1 - \epsilon_p) \leq \sum_{i \in D_j} p_i \leq \hat{p}(1 + \epsilon_p), \qquad j = 1, \ldots, k$$

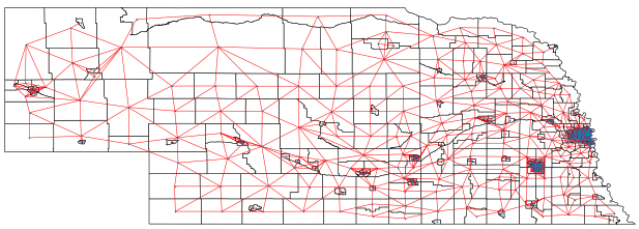


Figure 1: Adjacency graph of Nebraska's 532 census tracts.

where $\hat{p} = \sum p_i/k$ is the ideal district population, with $\epsilon_p < .01$ for Congressional districts, and $\epsilon_p < .05$ for state legislatures.

There are many proposed mathematical measures of compactness in the redistricting literature [45], but each has particular shortcomings, and none are universally agreed upon. Contiguity is also not entirely unambiguous, because of water features and states that have multiple connected components (e.g., Michigan). Any ambiguity must eventually be resolved in the adjacency graph G by the practitioner.

Column Generation Most integer programming formulations for solving the PDP, and regionalization methods more broadly, choose a set of blocks that act as centers of a region and assign every block to exactly one center, all in one step. They associate decision variables for each pair of nodes [33] for the assignment of blocks to centers and an additional variable for each block to indicate whether or not it is a center. In this design paradigm, the generation of districts and the optimization are performed jointly.

Our approach is based on the idea that we can decouple the district generation from optimization by enumerating feasible districts as an input to the final optimization problem [15, 29, 28]. An optimal solution could theoretically be found by enumerating every legal district $D_1, \ldots, D_N$ and solving a set partitioning problem to select the optimal k districts using binary decision variables [15]. Formally, if $A = (a_{ij})$ is the block-district matrix where

$$a_{ij} = \begin{cases} 1 & \text{if } b_i \in D_j \quad (\text{block } i \text{ is in district } j) \\ 0 & \text{else} \end{cases}$$

and x is a binary decision vector where x_j indicates whether or not D_j is in the final plan, the set of feasible solutions is

$$F = \{x : Ax = 1, \quad ||x||_1 = k, \quad x \in \mathbb{Z}_2^N\}.$$

An optimal plan corresponds to the $\hat{x} \in F$ that minimizes (or maximizes) some linear objective cx. While viable for small problem sizes ($|B| < 100$), the number of feasible districts grows exponentially and quickly becomes intractable to enumerate. However, every feasible district plan must have exactly k nonzero variables, implying that the vast majority of the column space is not useful, and therefore unnecessary to generate.

Implicit column generation is a well-studied technique to efficiently solve huge set partitioning problems [2] such as the PDP. These algorithms involve iteratively solving a relaxed master problem and transferring dual information to a column generation subproblem to determine if there are columns with negative reduced costs that should be included in the master problem [28]. However, we found when we scaled this approach to larger problem instances, the degeneracy of the master problem rendered the dual values meaningless, resulting in the generation of low quality columns. The core issue with traditional one-at-a-time column generation is that a single arbitrary district is unlikely to have $k-1$ other districts with which it can be composed to exactly cover the whole state. This observation motivates our heuristic that generates explicitly complimentary columns [17] that can be composed to express a huge number of legal plans.

In addition to the degeneracy of the master problem, there are a number of features of fairness-optimized political redistricting that make "optimal" column generation the wrong approach. When optimizing for expected electoral outcomes, there is significant uncertainty in the objective function. Furthermore, there are many desirable and undesirable properties of feasible maps that should be balanced and analyzed in aggregate. Lastly, the difficulty of solving the master set partitioning problem scales with the number of districts (columns); however, depending on the construction of the districts, the feasible set of plans F, can scale much faster than the set of districts D. For these reasons, instead of focusing on certifying that our columns are "optimal," a better goal is to generate *efficient* columns, those that maximize $|F|/|D|$.

2.2 Stochastic Hierarchical Partitioning (SHP)

Sample Tree What is the fastest way to generate one trillion unique district plans for a state with 16 districts? One way is to first partition the state into two and generate one million district plans for each half each with 8 districts. The fastest way to generate two sets of one million district plans? Split each half into quarters and generate one thousand district plans for each quarter. And to generate a thousand district plans for each quarter? The attentive reader guesses generating 32 plans for each eighth; partition each eighth 32 times and return.

By construction, the above procedure admits over one trillion district plans while generating only 512 districts by solving only 263 partitioning problems. In practice, none of the trillion plans might be any good because they are all similar, in that they respect the structure imposed by nodes higher in the tree. Therefore, in our method, rather than select a single split, we sample each level repeatedly; this exponential leverage of composing hierarchical partitions is the key property we exploit to generate efficient columns.

To formalize this notion, we introduce the concept of a SHP sample tree, a tree of compact and contiguous

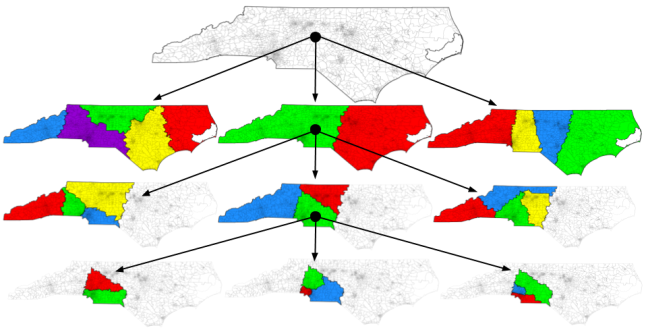


Figure 2: One branch of a sample tree with fan-out width $w = 3$ for North Carolina.

regions that maintain a compositional compatibility invariant (see Figure 2 for an example branch of a sample tree). A sample tree node (R, s) is a collection of contiguous blocks $R \subseteq B$ such that the population of region R is sufficient for the construction of s districts that respect the population constraint. We call s the *capacity* of the region. The root node of the sample tree is (B, k). The compatibility invariant requires that for any node (R, s), a valid district plan with s districts can be constructed by combining any valid district plan from each of its children from a single sampled partition. Leaf nodes have $s = 1$; they constitute the set of generated districts and correspond to the columns of the block-district matrix A.

The shape of the tree dictates the properties of the column space. The two main parameters that we use to control the shape are the sample fan-out *w*idth w, and the split si*z*e z. The former is the number of partitions we sample at each node, and the latter is the size of the partition (binary, ternary, etc.). Parameters w and z control the inherent trade-off in the efficiency and diversity of the columns generated. Sampling more at higher nodes in the tree increases diversity but decreases efficiency. Sampling more at lower nodes has the opposite effect. Increasing the average split size z leads to a shorter tree which yields less exponential leverage but more diversity.

We build the tree by sampling multiple random partitions of each node starting with the root. For computational reasons, we implement this iteratively, rather than recursively, using a queue initialized with the root (B, k). We continue to partition regions into smaller regions and add those regions to the queue unless $s = 1$, in which case we add this region to the set of generated districts. The *root partition* imposes significant structure on the remaining partitions, so in practice, for sake of diversity, we typically sample hundreds or thousands of root partitions; in contrast, each nonleaf child node is sampled only a few times (between 2 and 10). The problem then becomes creating an expressive enough column space by sampling a diverse enough set of region partitions.

Partition As depicted in Figure 3, each random partition of a node follows multiple steps: sample the split size, sample the capacity of each center, sample the position of the centers, and finally solve the partitioning integer program to perform the final assignment of blocks to regions.

In more detail, for a node (R, s), encoding a region $R \subseteq B$ with capacity for s districts, we first sample the split size $z \in [z_{min}, \min(s, z_{max})]$ corresponding to the target number of children and then uniformly at random sample the capacity of each child node $s_1, \ldots, s_z$ such that $\sum_i s_i = s$. To prevent significant imbalance in the tree, we usually add a constraint on the maximum difference between the maximum and minimum subregion capacity. One could imagine also tuning z_{min} and z_{max} depending on the region capacity s and the overall size of the state k, but for simplicity, in our experiments we simply set $z_{min} = 2$ and $z_{max} = 5$; this works well in practice.

The next step is to sample the set of centers $C \subset R$ that will act as the seeds for each region of the partition. We detail a number of ways we do this in Appendix B, but the goal is to let the randomization enable a selection of diverse centers that are well-spaced enough to not sacrifice feasibility or create contorted and noncompact regions. Each center is then matched to a size s_i based on an idealized capacity given by the region's Voronoi diagram with respect to the set of centers. See Appendices B and D for explanations of different center selection and capacity matching methods with detailed empirical results for each proposed method.

Finally, using these centers and capacities, we solve an integer program to assign blocks to the z subregions, each centered at a center c_i with capacity s_i that maximize compactness and satisfy the population balance and contiguity constraints. Enforcing contiguity is one of the reasons political districting problems are so difficult, since it requires an exponential number of constraints to do exactly. Instead of explicitly enforcing contiguity, we use an approach first presented in [28] that enforces each partition be a sub-tree of the shortest path tree rooted at the district center. This requires computing the shortest path distance from each center to all nodes in the subgraph of G with nodes in region R, denoted G_R with edgeset E_R. Let d_{ij}^R be the shortest path distance between b_i and b_j in the subgraph G_R. For a center i and block j, let

$$S_{ij} = \{\ell : \ (\ell, j) \in E_R, \ d_{i\ell}^R < d_{ij}^R\};$$

for a block j to be assigned to a center i, then one element of S_{ij}, the set of neighbors of j closer to i, must also be assigned to center i.

Formally, the partition integer program (PIP) with inputs R and C is as follows:

$$\text{minimize} \quad \sum_{i \in C} \sum_{j \in R} (d_{ij})^{\alpha} p_j x_{ij} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{i \in C} x_{ij} = 1, \qquad \forall j \in R \tag{2.2}$$

$$\sum_{j \in R} p_j x_{ij} \leq \hat{p}(s_i + \epsilon_p) \qquad \forall i \in C \tag{2.3}$$

$$\sum_{j \in R} p_j x_{ij} \geq \hat{p}(s_i - \epsilon_p) \qquad \forall i \in C \tag{2.4}$$

$$\sum_{\ell \in S_{ij}} x_{i\ell} \geq x_{ij}, \qquad \forall i \in C, \quad \forall j \in R \tag{2.5}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in C, \quad \forall j \in R \tag{2.6}$$

where constraint (2.2) enforces that each block is assigned to exactly one subregion, constraints (2.3) and (2.4) enforce population balance, and constraint (2.5) enforces contiguity. The objective function is based on the dispersion, or moment of inertia measure of compactness [45]. We apply a random exponent to the distance cost term drawn uniformly at $\alpha \in [1, 2]$ to promote more diverse region shapes. For states with abnormal geometric features (e.g. those with multiple connected components or with highly nonconvex shapes such as NY, MA, and MI), it is preferable to optimize for network distance rather than Euclidean distance. We choose this method of enforcing contiguity because it is efficiently computable, but it is not without its drawbacks. While it prevents the formation of noncompact districts, it also discounts many districts that would pass an eyeball test and generally reduces the space of feasible solutions. One item of future work is to profile different methods of enforcing contiguity [41].

The capacitated partition program shares many similarities with the capacitated transportation problem, which is solvable in polynomial time with flow networks [14]. However, instead of assigning units of supply to units of demand, we are assigning units of population to districts with population requirements. We include the additional contiguity constraint [5], but given that most compact solutions are contiguous, this does not drastically increase the difficulty. For a region with n nodes and z centers, the partition IP has nz variables and $n+2z+nz$ constraints. These problems rarely take more than a few seconds to construct and solve.

Tree Complexity The first key property of our approach is the leverage we gain from our tree formulation: districts with shared ancestors are always compatible. With a deeper tree, the number of feasible district plans increases rapidly. Importantly, this makes our method more effective when k is larger, whereas most optimization approaches become intractable with large k.

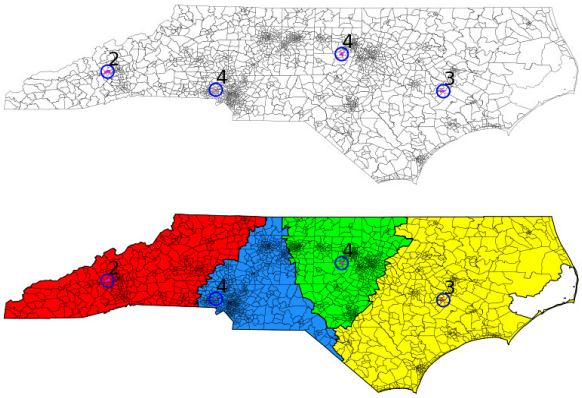


Figure 3: An example partition step of the sample tree root node with split size $z = 4$. First, we sample center population capacities and their geographic locations. Second, we solve the partition integer program to partition the region into contiguous subregions of appropriate population.

Let $P(R, s)$ denote the number of feasible s-district plans of region R given by the sample tree. If $w(R)$ gives the sample width for region R, and $z(i)$ gives the number of subregions for sample i, the number of plans has a recursive formula

$$P(R, s) = \begin{cases} \sum_{i=1}^{w(R)} \prod_{j=1}^{z(i)} P(R_{ij}, s_j) & \text{for } s > 1 \\ 1 & \text{for } s = 1 \end{cases}$$

where R_{ij} is subregion j of sample i. Intuitively this means summing over sibling samples and multiplying over children.

Theorem 1. *Consider a set of blocks B to be partitioned into k districts. For a sample tree with root node (B, k) and with nodes corresponding to distinct partitions, with constant sample width w, and arbitrary split sizes $z' \in [2, z]$, the tree admits $P(B, k)$ total distinct partitions where*

$$w^{\frac{k-1}{z-1}} \leq P(B, k) \leq w^{k-1}.$$

The theorem follows from an inductive counting argument (see Appendix C). Note that the upper bound is tight only when we sample binary partitions (regardless of balance) and the lower bound is tight when we only sample z-partitions and $k = z^n$ for some integer n.

In practice, we have nodes that do not successfully find w samples and generate duplicate regions. Duplicate leaves decrease the number of distinct district plans, whereas duplicate internal nodes can increase the number because they effectively increase the sample width.

To successfully achieve the full w^{k-1} distinct plans for $k = 2^n$, one must solve $w\sum_{d=0}^{n-1}(2w)^d$ partition problems involving center selection and the PIP. This is less work than it appears because as d increases, the size of the PIP decreases exponentially. In particular, where the root node has to partition $|B|$ blocks, a node at depth d will only have to partition $|B|/2^d$ blocks. Again using the binary case, this process will generate $(2w)^{log_2(k)}/2$ districts which admit $w^{(k-1)}$ unique district plans. This exponential ratio between the number of districts and plans is the targeted column efficiency property.

We call the log of the ratio between the number of generated plans and districts, $\log(\frac{|P|}{|D|})$, the *leverage* of the ensemble. To our knowledge, all existing ensemble approaches actually have negative leverage; that is, they generate more districts than plans. For instance, techniques that generate a fully new plan each iteration (such as greedy agglomerative methods [6]) have leverage $\ell = \log(\frac{1}{k})$ and Markov chain techniques [7, 9] generate two districts per plan, $\ell = \log(\frac{1}{2})$. Especially for large k, having low leverage implies that a technique spends significant computational effort generating districts but that effort does not translate into an efficient exploration of the feasible space.

2.3 Optimization

Objective Function The second key property of our decoupled design is the flexibility it grants in specifying the objective function. Whereas standard approaches for one-step optimization require the objective be a piecewise-linear function of blocks, our two-step method enables piecewise-linear functions of district-level metrics that can be arbitrary functions of blocks. This property enables our fairness objective. A fair district plan, in our view, is one that, *on average*, calibrates electoral outcomes with the partisan affiliation of a state to minimize the number of *wasted* votes.

A wasted vote is a vote cast for a losing candidate or a surplus vote cast for the winner over what was needed to win. The efficiency gap (EG) [39], a popular fairness metric, measures the difference between wasted votes for the two parties. The *responsiveness* of a plan is the number r such that a 1% increase in a party's vote-share is accompanied by a r% increase in a party's seat-share. Assuming uniform turnout, minimizing the EG coincides with finding a plan with a constant responsiveness of 2 [43] (i.e., a 1% increase in vote-share should always yield a 2% average increase in seat-share). This is not always possible and scholars debate what the ideal responsiveness should be [27]. However, our method is flexible enough to support optimizing any function of votes to seats so we can accommodate alternative definitions of fairness.

To make such probabilistic statements, we require a probabilistic model of partisan affiliation as an input. We use historical precinct returns from statewide elections to estimate the mean and standard deviation of the two-party vote-share for an arbitrary district (see Appendix A for the details of our data sources and preparation). Let $\hat{\nu}$ be the average statewide partisan vote-share, ν_i be a random variable for the partisan vote-share of district i, and ψ_i be a Bernoulli random variable indicating an electoral win or loss for district i. We assume

$$\nu_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \qquad \psi_i \sim \mathcal{B}(P(\nu_i > .5))$$

where μ_i and σ_i^2 are given by the historical statewide returns of blocks in district i. By linearity of expectation, the expected difference between the statewide seat-share and statewide vote-share is the sum of the differences between the expected district-level seat-share and statewide vote-share:

$$E\left[\frac{1}{k}\sum_{i=1}^{k}\hat{\nu} - \psi_i\right] = \frac{1}{k}\sum_{i=1}^{k}\hat{\nu} - \left(1 - \Phi\left(\frac{\mu_i - .5}{\sigma_i}\right)\right),$$

where Φ is the unit Gaussian cumulative distribution function (CDF). Given the infrequency of elections, in our experiments we use a t random variable CDF, but the model is flexible to support other affiliation models with richer features. Importantly, we can optimize for $h(\hat{\nu}) - \psi_i$ where h gives a desired mapping from votes to seats enabling optimizing to the shape of arbitrary seat-vote curves. For the efficiency gap, $h(\hat{\nu}) = 2\hat{\nu} - .5$, assuming uniform turnout.

Our model is simple by design, making estimation and optimization tractable for millions of arbitrary districts. We do not claim that this is the right model, and in many cases it is the wrong model when incumbency or other specific information is known for a district. However, given the application domain, we believe simplicity is important for adoption, since a more opaque or harder to reason about model would likely be politically more controversial. Regardless, a promising direction for future work is developing more robust affiliation models suitable for use in optimization settings that can generalize to multiple parties or different voting paradigms.

Tree-based Dynamic Programming When the following conditions hold, the sample tree admits an efficient dynamic programming solution to find the optimal objective.

C1. There are no additional constraints on the composition of districts (e.g., maintaining a specified number of competitive or majority-minority districts) other than those implicitly encoded by the structure of the tree.

C2. The objective function is linear (e.g., compactness, competitiveness, but not seat-vote difference)

The intuition underlying the dynamic program is that for a particular partition of a region, the optimal objective (solution) is the sum (composition) of the optimal objectives (solutions) for each of the subregions in the partition. Written recursively, the optimal objective value of region R with capacity s is

$$f^*(R,s) = \begin{cases} \underset{i \in w(R)}{\operatorname{argmin}} \sum_{j=1}^{z(i)} f^*(R_{ij}, s_{ij}) & \text{for } \; s > 1 \\ c_{ij} & \text{for } \; s = 1 \end{cases}$$

where c_{ij} is the cost coefficient of the district R_{ij}, $z(i)$ is the split size of region partition i, and $w(R)$ is the number of partitions sampled.

While this approach is not typically helpful for finding a specific *fair* plan, it is convenient for quickly evaluating the range of the most extreme solutions generated for arbitrary linear metrics. This is especially useful in ensemble studies to evaluate how different inputs or constraints effect the range of possible partisan outcomes.

Master Selection Problem When either condition C1 or C2 above does not hold (most notably when some definition of fairness is the objective), we use the machinery of integer programming to find a family of solutions. This is accomplished by solving a set partitioning master selection problem (MSP) to choose the final set of districts. We solve a separate MSP for each set of leaf nodes that share a parent root partition. In this standard formulation [28], we have a binary decision variable for every generated district x_j (indexed by D) and wish to

$$\text{minimize} \quad \Big| \sum_{j \in D} c_j x_j \Big| \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j \in D} a_{ij} x_j = 1, \qquad \forall i \in B; \tag{2.2}$$

$$\sum_{j \in D} x_j = k; \tag{2.3}$$

$$x_j \in \{0,1\}, \qquad \forall j \in D, \tag{2.4}$$

where $c_j = \mathbf{E}(h(\hat{\nu}) - \psi_j)$ as estimated by our affiliation model and A is the binary block-district matrix with $a_{ij} = 1$ if block b_i is in district d_j. Constraint (2.2) enforces that each block is assigned to exactly one district and constraint (2.3) enforces that there are exactly k districts. Constraint (2.3) is strictly necessary only when $k\epsilon_p > 1$, by virtue of the generation process. Any additional linear constraints placed on the composition of districts can be added seamlessly to the formulation (e.g., requiring a minimum number of districts meet a threshold level of competitiveness). Finally, we make the objective linear by introducing and minimizing a new variable w, such that $\sum_{j \in D} c_j x_j \leq w$ and $\sum_{j \in D} c_j x_j \geq -w$.

We solve this integer program for the columns generated from each root partition separately for three reasons. By sharding the tree by root partition, the whole pipeline becomes arbitrarily parallelizable. One could set up an arbitrary number of computational workers where each worker runs the SHP generation routine and then solves the MSP for each generated root partition. One line of future work is to use optimal transport [1] to characterize the convergence behavior of our method to understand when it is unlikely to improve upon the existing solution pool so we can employ more principled terminating conditions.

One concern that might be raised is that by separating subproblems by the root partition, we limit the possible district plans; however, unless an internal node (R, s) has an identical copy in another subtree, it is unlikely that two columns generated from different partitions will be compatible. That is, with high probability there does not exist $k-2$ other generated districts that can be included to satisfy constraint (2) of the MSP. One could search for duplicates across the entire tree and augment the column set to include all descendants of all duplicates. However, duplicates are fairly rare and limits the inherent parallelism of our approach.

Finally, in a state where a fair solution exists, there are typically a large number of alternative fair solutions. Since there are many other desirable proprieties of districts plans to trade off, one can filter the range of candidate plans to those that meet a requisite level of fairness, and then consider other properties such as competitiveness, maintaining political boundaries, or not separating communities of interest. Solving one MSP per root partition is a very simple way to get a large number of fair solutions, which are also substantially different from each other. One could use a more sophisticated method [37] to find many near optimal solutions for each root partition, but these are more likely to be very similar given the structure imposed by the shared root partition.

3 Results

We provide two types of experimental results. The first set of experiments is designed to tune the algorithm's parameters and understand its behavior. We study the performance of different configurations of the SHP generation routine and the impact of varying the PDP parameters. We further provide empirical results of convergence behavior and runtime performance. These results are included in Appendices D and F[1]

In our main experiment, we generate a large number of districts for every multi-district state to understand the range of possible outcomes using only reasonably shaped districts and compare this to a recombination ensemble baseline. Given the exponential nature of our column space, this constitutes the largest district plan ensemble ever (implicitly) constructed. However, this also makes computing some ensemble-level metrics intractable. Therefore, for some results, we use a pruned sample tree to make enumeration feasible. See Appendix E for all experiment details.

Compactness Compactness has long been used as the objective function in political redistricting research, rationalized by the claim that a compact map drawn with no political input is inherently fair. Although highly noncompact human drawn districts can signal manipulation, there is no *a priori* reason to assume that a randomly drawn compact map will be more fair than a randomly drawn noncompact map. To verify this, we compute the Spearman correlation coefficient [36] for each state between the magnitude of the expected efficiency gap and three different measures of compactness: centralization, Roeck, and cut-edges. Centralization is the average distance a constituent has to walk to the center of their district, averaged over all districts (similar to the dispersion measure of compactness [45]); Roeck is the district area divided by the area of the smallest circumscribing circle averaged over all districts; and cut-edges measures the average number of edges that need to be cut in the adjacency graph to form the districts of a plan.

Figure 4 and Table 1 show that compactness and fairness are orthogonal attributes of a district plan. Any correlation between compactness and fairness is noise stemming from the political geography of a state. For instance in Nebraska, where the correlation is the strongest, the only redistricting decision that affects the expected efficiency gap is whether Omaha is cracked and diluted into two reliably Republican districts (not compact and not fair) or packed into one toss-up district (compact and fair). In many swing states where redistricting decisions can have the most impact, such as Pennsylvania, Wisconsin, and Georgia, the correlation is actually negative. This is because the most compact plans pack the major cities into landslide Democratic districts, while the surrounding suburbs and rural areas have a more efficient placement of Republican voters. Lastly, in the largest states, California and Texas, where no single geographic feature can dominate the correlation, the relationship is almost perfectly independent.

	mean ρ	k-weighted mean ρ
centralization	0.074756	0.038859
roeck	-0.000355	-0.008970
cut_edges	0.105347	0.066784

Table 1: Average (and k-weighted average) Spearman correlation coefficient between the magnitude of expected efficiency gap and three different compactness metrics over all multi-district states.

We do not claim that compactness is not important or desirable; compact plans are more likely to represent more coherent communities and make administering elections marginally easier. However, fairness is a far more important consideration and therefore we want to encourage practitioners to treat compactness as a constraint rather than an objective.

Fairness Unfortunately, fairness, like compactness, does not have a universally agreed-upon characterization; there are different and often competing aspects of fairness, which are difficult to reconcile with the distribution of partisan voters. The political geography of a state, typically defined by a few urban concentrations of more liberal voters punctuating larger and sparser regions of more conservative voters, does not guarantee a fair solution for any reasonable definition of fair. This greatly complicates creating a unified definition of fairness when the number of districts and distribution of voters greatly constraints the space of expected outcomes [11]. We use the efficiency gap [39] as our primary metric because it has gained traction as a standard metric in redistricting lawsuits, has an intuitive and compelling interpretation, and is simple to embed in an integer program as we have shown in Section 2.3.

Figure 5 shows the range of expected outcomes found by our generation algorithm and includes the point which would minimize the efficiency gap as well as the actual average seat-share over the last redistricting round ordered by partisan affiliation (see Appendix E for experiment details). By construction, our approach generates only relatively compact districts which we call *natural* districts, so this range can be thought of as

[1]The full paper with appendices is available at https://arxiv.org/abs/2103.11469; all references to appendices within the text are in reference to this full version.

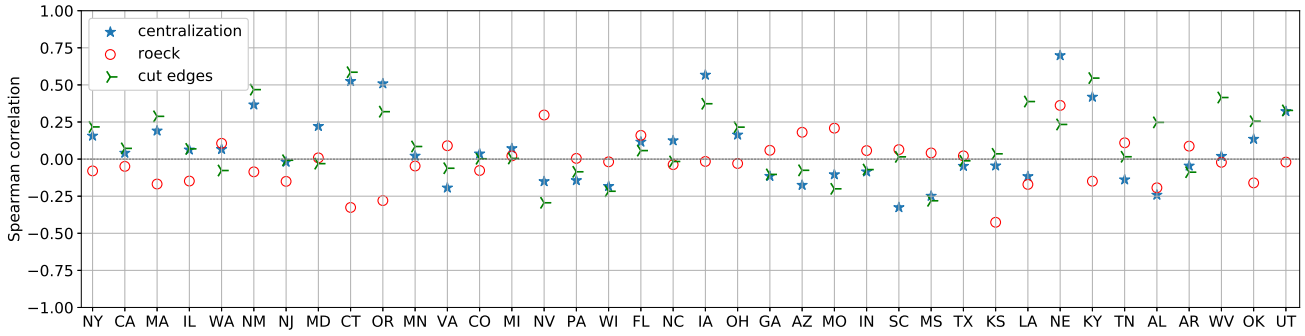


Figure 4: Spearman correlation coefficient between the magnitude of the expected efficiency gap and three different measures of compactness for the 38 states with three or more congressional districts ordered by partisan affiliation.

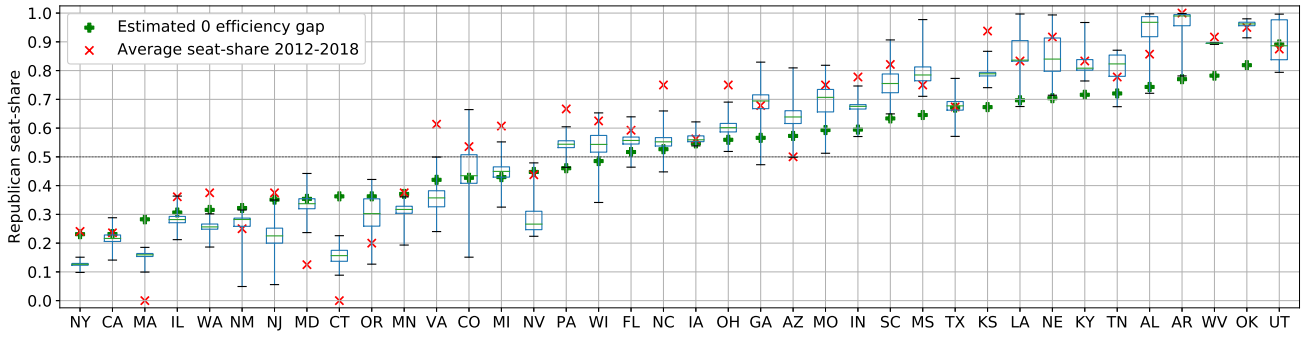


Figure 5: Distribution of expected Republican seat-share of generated plan ensemble for each state with three or more congressional districts ordered by partisan affiliation. The box center indicates the median, with the top and bottom of the box indicating the 75th and 25th percentiles respectively of the down-sampled ensemble. The whiskers denote the minimum and maximum value of any plan found using the tree based dynamic programming optimization algorithm.

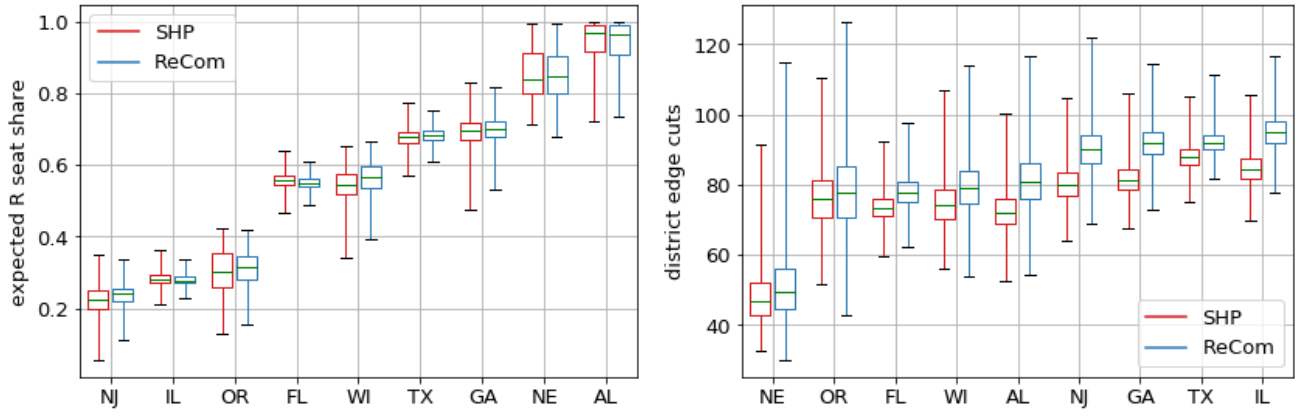


Figure 6: Comparison of expected seat-share (left) and average district edge cuts (right) on ensembles generated with Stochastic Hierarchical Partitioning (SHP) and recombination Markov chains.

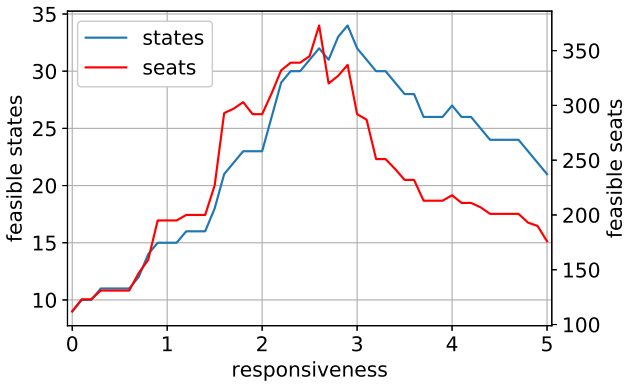


Figure 7: The number of states and total seats that admit a district plan in our ensemble for an expected level of responsiveness.

the geographic tolerance for natural or unintentional gerrymandering [6]: reasonably shaped plans that pass the eyeball test but are actually unrepresentative. In every state, we find a plan that is more compact than the current one; in just 7 states the enacted plan is more compact than our median plan, and in 8 states (most of which are well known for gerrymandering) the enacted plan is less compact than any plan we generate (see Appendix E).

The headline result is that using only natural district boundaries, we can change the expected composition of Democrats in the House of Representatives from 43% to 62%. A 20% swing in the partisan composition of the House is exceptionally consequential, since a body with full control of all congressional boundaries could theoretically create a permanent majority or permanent minority for either party, barring any extreme shifts in partisan affiliation along geographic lines. We run the same experiment with competitiveness (see Appendix E) and in aggregate achieve plans with natural districts that have between 32 and 106 expected total seat swaps per election.

However, the expected partisan seat range is highly variable. Some states like Colorado admit reasonable district plans that can swing by up to 50% of total seats, whereas others like West Virginia or Iowa offer very little room to alter the outcome using only natural districts. Consequently, any definition of fairness that relies on an ideal level of responsiveness (e.g., the efficiency gap) will not be feasible in all states without relaxing the compactness or contiguity constraints. Figure 7 shows the number of feasible states (and the corresponding number of seats) that have a solution that achieves a target level of responsiveness. In particular, only 15 states admit a proportional plan where the expected seat-share equals the expected vote-share. Additionally, we found plans with 0 expected efficiency gap for only half of the multi-district states.

Recombination Baseline To test our claims about more effective sampling of extreme solutions, we compare ensembles generated with our stochastic hierarchical partitioning (SHP) algorithm and recombination Markov chains [9], the predominant tool in quantitative redistricting analyses. Recombination chains generate ensembles by iteratively merging two random adjacent districts, sampling a random spanning tree, and choosing a tree cut to balance the populations of the two connected components to create two new districts.

In Figure 6, we compare the expected seat-share and the edge cuts between ensembles generated with the two methods for a representative sample of states. To make the comparison as meaningful as possible, we generate the same number of distinct districts with each method for each state. In all states but Nebraska, SHP finds a larger range of expected seat-shares (while also being significantly more compact on average). This is expected since small states have very short sample trees and therefore do not benefit as much from more efficient columns as larger states do. Furthermore, by using random spanning trees instead of an inner partition integer program that optimizes for region compactness, recombination chains can sample less compact partitions, which may achieve a more extreme outcome than a natural district would.

4 Conclusion

Part methodological exposition, part civic plea, this paper introduces a scalable new method for creating fair political districts. Using the ensembling capability of our algorithm, we performed the largest ever district ensemble study to show the spurious relationship between fairness and compactness. We also illustrated the range of possible outcomes with reasonably shaped districts to better frame the constraints of fairness. We want to emphasize the overall framework more than any specific design decision, as we see the main contribution of this work as offering an alternate paradigm for solving supervised regionalization problems, most notably congressional redistricting, and potentially a more general heuristic for solving other hard computational problems.

Most importantly, we want our work to actually move the needle on redistricting reform. Our core capability is quickly, cheaply, and transparently generating a huge number of legal and fair district plans. We extend an open invitation to collaborate with states in the upcoming redistricting cycle and hope that the maps of the next decade are more representative than those of the last.

References

[1] Tara Abrishami et al. "Geometry of graph partitions via optimal transport". In: *SIAM Journal on Scientific Computing* 42.5 (2020), A3340–A3366.

[2] Cynthia Barnhart et al. "Branch-and-price: Column generation for solving huge integer programs". In: *Operations research* 46.3 (1998), pp. 316–329.

[3] Mitchell N Berman. "Managing Gerrymandering". In: *Tex L. Rev.* 83 (2004), p. 781.

[4] Bruce E Cain et al. "A reasonable bias approach to gerrymandering: Using automated plan generation to evaluate redistricting proposals". In: *Wm. & Mary L. Rev.* 59 (2017), p. 1521.

[5] Tanima Chatterjee and Bhaskar DasGupta. "On partisan bias in redistricting: computational complexity meets the science of gerrymandering". In: *arXiv preprint arXiv:1910.01565* (2019).

[6] Jowei Chen and Jonathan Rodden. "Unintentional gerrymandering: Political geography and electoral bias in legislatures". In: *Quarterly Journal of Political Science* 8.3 (2013), pp. 239–269.

[7] Maria Chikina, Alan Frieze, and Wesley Pegden. "Assessing significance in a Markov chain without mixing". In: *Proceedings of the National Academy of Sciences* 114.11 (2017), pp. 2860–2864.

[8] Vincent Cohen-Addad, Philip N Klein, and Neal E Young. "Balanced centroidal power diagrams for redistricting". In: *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 2018, pp. 389–396.

[9] Daryl DeFord, Moon Duchin, and Justin Solomon. "Recombination: A family of Markov chains for redistricting". In: *arXiv preprint arXiv:1911.05725* (2019).

[10] Moon Duchin. *Outlier analysis for Pennsylvania congressional redistricting*. 2018.

[11] Moon Duchin et al. "Locating the representational baseline: Republicans in Massachusetts". In: *Election Law Journal: Rules, Politics, and Policy* 18.4 (2019), pp. 388–401.

[12] Juan Carlos Duque, Raúl Ramos, and Jordi Suriñach. "Supervised regionalization methods: A survey". In: *International Regional Science Review* 30.3 (2007), pp. 195–220.

[13] Robert S Erikson. "Malapportionment, gerrymandering, and party fortunes in congressional elections". In: *The American Political Science Review* (1972), pp. 1234–1245.

[14] Lester Randolph Ford Jr and Delbert Ray Fulkerson. "Solving the transportation problem". In: *Management Science* 3.1 (1956), pp. 24–32.

[15] Robert S Garfinkel and George L Nemhauser. "Optimal political districting by implicit enumeration techniques". In: *Management Science* 16.8 (1970), B–495.

[16] Andrew Gelman and Gary King. "Estimating the electoral consequences of legislative redistricting". In: *Journal of the American Statistical Association* 85.410 (1990), pp. 274–282.

[17] Ahmed Ghoniem and Hanif D Sherali. "Complementary column generation and bounding approaches for set partitioning formulations". In: *Optimization Letters* 3.1 (2009), p. 123.

[18] Gregory Herschlag, Robert Ravier, and Jonathan C Mattingly. "Evaluating partisan gerrymandering in Wisconsin". In: *arXiv preprint arXiv:1709.01596* (2017).

[19] Sidney Wayne Hess et al. "Nonpartisan political redistricting by computer". In: *Operations Research* 13.6 (1965), pp. 998–1006.

[20] Douglas M King, Sheldon H Jacobson, and Edward C Sewell. "Efficient geo-graph contiguity and hole algorithms for geographic zoning and dynamic plane graph partitioning". In: *Mathematical Programming* 149.1-2 (2015), pp. 425–457.

[21] Richard Kueng, Dustin G Mixon, and Soledad Villar. "Fair redistricting is hard". In: *Theoretical Computer Science* 791 (2019), pp. 28–35.

[22] Harry A Levin and Sorelle A Friedler. "Automated congressional redistricting". In: *Journal of Experimental Algorithmics (JEA)* 24.1 (2019), pp. 1–24.

[23] Justin Levitt. "A citizen's guide to redistricting". In: *Available at SSRN 1647221* (2008).

[24] Zhenping Li, Rui-Sheng Wang, and Yong Wang. "A quadratic programming model for political districting problem". In: *Proceedings of the first international symposium on optimization and system biology (OSB). Bejing, China*. 2007.

[25] Yan Y Liu, Wendy K Tam Cho, and Shaowen Wang. "PEAR: a massively parallel evolutionary computation approach for political redistricting optimization and analysis". In: *Swarm and Evolutionary Computation* 30 (2016), pp. 78–92.

[26] Nolan McCarty, Keith T Poole, and Howard Rosenthal. "Does gerrymandering cause polarization?" In: *American Journal of Political Science* 53.3 (2009), pp. 666–680.

[27] Anthony J McGann et al. *Gerrymandering in America: The House of Representatives, the Supreme Court, and the future of popular sovereignty*. Cambridge University Press, 2016.

[28] Anuj Mehrotra, Ellis L Johnson, and George L Nemhauser. "An optimization based heuristic for political districting". In: *Management Science* 44.8 (1998), pp. 1100–1114.

[29] Bjørn Nygreen. "European assembly constituencies for Wales-comparing of methods for solving a political districting problem". In: *Mathematical Programming* 42.1-3 (1988), pp. 159–169.

[30] Daniel D Polsby and Robert D Popper. "The third criterion: Compactness as a procedural safeguard against partisan gerrymandering". In: *Yale Law & Policy Review* 9.2 (1991), pp. 301–353.

[31] Clemens Puppe and Attila Tasnádi. "A computational approach to unbiased districting". In: *Mathematical and Computer Modelling* 48.9-10 (2008), pp. 1455–1460.

[32] *Reynolds v. Sims*. 1964.

[33] Federica Ricca, Andrea Scozzari, and Bruno Simeone. "Political districting: from classical models to recent approaches". In: *Annals of Operations Research* 204.1 (2013), pp. 271–299.

[34] Federica Ricca, Andrea Scozzari, and Bruno Simeone. "Weighted Voronoi region algorithms for political districting". In: *Mathematical and Computer Modelling* 48.9-10 (2008), pp. 1468–1477.

[35] Federica Ricca and Bruno Simeone. "Local search algorithms for political districting". In: *European Journal of Operational Research* 189.3 (2008), pp. 1409–1426.

[36] Patrick Schober, Christa Boer, and Lothar A Schwarte. "Correlation coefficients: appropriate use and interpretation". In: *Anesthesia & Analgesia* 126.5 (2018), pp. 1763–1768.

[37] Thiago Serra and JN Hooker. "Compact representation of near-optimal integer programming solutions". In: *Mathematical Programming* (2019), pp. 1–34.

[38] Nicholas O Stephanopoulos. "The causes and consequences of gerrymandering". In: *Wm. & Mary L. Rev.* 59 (2017), p. 2115.

[39] Nicholas O Stephanopoulos and Eric M McGhee. "Partisan gerrymandering and the efficiency gap". In: *U. Chi. L. Rev.* 82 (2015), p. 831.

[40] Rahul Swamy, Douglas King, and Sheldon Jacobson. "Multi-Objective Optimization for Political Districting: A Scalable Multilevel Approach". In: (2019).

[41] Hamidreza Validi, Austin Buchanan, and Eugene Lykhovyd. "Imposing contiguity constraints in political districting models". In: (2020).

[42] William Vickrey. "On the prevention of gerrymandering". In: *Political Science Quarterly* 76.1 (1961), pp. 105–110.

[43] Gregory S Warrington. "Quantifying gerrymandering using the vote distribution". In: *Election Law Journal* 17.1 (2018), pp. 39–57.

[44] *Wesberry v. Sanders*. 1964.

[45] H Peyton Young. "Measuring the compactness of legislative districts". In: *Legislative Studies Quarterly* 13.1 (1988), pp. 105–115.

Faster Parallel Multiterminal Cuts *

Monika Henzinger† Alexander Noe ‡ Christian Schulz §

Abstract

We give an improved branch-and-bound solver for the multiterminal cut problem, based on the recent work of Henzinger et al. [12]. We contribute new, highly effective data reduction rules to transform the graph into a smaller equivalent instance. In addition, we present a local search algorithm that can significantly improve a given solution to the multiterminal cut problem. Our exact algorithm is able to give exact solutions to more and harder problems compared to the state-of-the-art algorithm by Henzinger et al. [12]; and give better solutions for more than two third of the problems that are too large to be solved to optimality. Additionally, we give an inexact heuristic algorithm that computes high-quality solutions for very hard instances in reasonable time.

1 Introduction

The multiterminal cut problem is a fundamental combinatorial optimization problem which was first formulated by Dahlhaus et al. [8] and Cunningham [7]. Given an undirected edge-weighted graph $G = (V, E, w)$ with edge weights $w : E \mapsto \mathbb{N}_{>0}$ and a set T, $|T| = k$, of terminals, the *multiterminal cut* problem is to divide its set of nodes into k blocks such that each block contains exactly one terminal and the weight sum of the edges running between the blocks is minimized. There are many applications of the problem: for example multiprocessor scheduling [22], clustering [19] and bioinformatics [13, 16, 25].

The problem is known to be NP-hard for $k \geq 3$ [8]. For $k = 2$ the problem reduces to the well known minimum s-t-cut problem, which is in P. The minimum s-t-cut problem aims to find the minimum cut in which the vertices s and t are in different blocks. Most algorithms for the minimum multiterminal cut problem use minimum s-t-cuts as a subroutine. Dahlhaus et al. [8] give a $2(1 - 1/k)$ approximation algorithm with polynomial running time based on the notion of *minimum isolating cuts*, i.e. the minimum cut separating a terminal from all other terminals. The currently best known approximation algorithm due to Buchbinder et al. [4] uses a linear program relaxation to achieve an approximation ratio of 1.323. Recently, Henzinger et al. [12] introduced a branch-and-reduce framework for the problem that is multiple orders of magnitudes faster than classic ILP formulations for the problem. This allows researchers to solve instances to optimality that are significantly larger than was previously possible and hence enables the use of multiterminal cut algorithms in practical applications.

Contribution. We give an improved solver for the multiterminal cut problem, based on the recent work of Henzinger et al. [12]. We contribute new, highly effective reductions to transform the graph into a smaller equivalent instance. In addition, we present a local search algorithm that can significantly improve a given solution to the multiterminal cut problem. Additionally, we combine the branch-and-reduce solver with an integer linear program solver to more efficiently solve subproblems emerging over the course of the algorithm. With our newly introduced reductions, the state-of-the-art algorithm by Henzinger et al. [12] is able to solve significantly harder instances to optimality and give better solutions to instances that are too large to solve to optimality. Additionally, we give an inexact algorithm that gives high-quality solutions to hard problems in reasonable time.

2 Preliminaries

2.1 Basic Concepts Let $G = (V, E, w)$ be a weighted undirected graph with vertex set V, edge set $E \subseteq V \times V$ and positive edge weights $w : E \to \mathbb{N}_{>0}$. We extend w to a set of edges $E' \subseteq E$ by summing the weights of the edges; that is, $w(E') := \sum_{e=(u,v)\in E'} w(u, v)$ and sets of nodes where $w(V_1, V_2)$ is the sum of edge weights connecting sets V_1 and V_2. Let $n = |V|$ be the number of vertices and $m = |E|$

*Research supported by the Austrian Science Fund (FWF) and netIDEE SCIENCE project P 33775-N
Partially supported by DFG grant SCHU 2567/1-2.
We further thank the Vienna Scientific Cluster (VSC) for providing high performance computing resources.

†University of Vienna, Faculty of Computer Science, Vienna, Austria, monika.henzinger@univie.ac.at
‡University of Vienna, Faculty of Computer Science, Vienna, Austria, alexander.noe@univie.ac.at
§Heidelberg University, Heidelberg, Germany, christian.schulz@informatik.uni-heidelberg.de

be the number of edges in G. The *neighborhood* $N(v)$ of a vertex v is the set of vertices adjacent to v. The *weighted degree* of a vertex is the sum of the weights of its incident edges. For a set of vertices $A \subseteq V$, we denote by $E[A] := \{(u,v) \in E \mid u \in A, v \in V \setminus A\}$; that is, the set of edges in E that start in A and end in its complement. A *k-cut*, or *multicut*, is a partitioning of V into k disjoint non-empty blocks, i.e. $V_1 \cup \cdots \cup V_k = V$. The weight of a k-cut is defined as the weight sum of all edges crossing block boundaries, i.e. $w(E \cap \bigcup_{i<j} V_i \times V_j)$.

2.2 Multiterminal Cuts A *multiterminal cut* for k terminals $T = \{t_1, ..., t_k\}$ is a multicut with $t_1 \in V_1, ..., t_k \in V_k$. Thus, a multiterminal cut pairwisely separates all terminals from each other. The edge set of the multiterminal cut with minimum weight of G is called $\mathcal{C}(G)$ and the associated optimal partitioning of vertices is denoted as $\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$. For a vertex $v \in V$, $\mathcal{V}_v$ denotes the block affiliation of v in the optimal partitioning $\mathcal{V}$. $\mathcal{C}$ can be seen as the set of all edges that cross block boundaries in $\mathcal{V}$, i.e. $\mathcal{C}(G) = \bigcup\{e = (u,v) \mid \mathcal{V}_u \neq \mathcal{V}_v\}$. The weight of the minimum multiterminal cut is denoted as $\mathcal{W}(G) = w(\mathcal{C}(G))$. At any point in time, the best currently known upper bound for $\mathcal{W}(G)$ is denoted as $\widehat{\mathcal{W}}(G)$ and the best currently known multiterminal cut is denoted as $\widehat{\mathcal{C}}(G)$. If graph G is clear from the context, we omit it in the notation. Note that the optimal partitioning $\mathcal{V}$ and the corresponding cut $\mathcal{C}$ are not necessarily unique, our aim is to find *one* minimum multiterminal cut, even if multiple cuts of equal value exist.

In this paper we use *minimum s-T-cuts*. For a vertex s (*source*) and a non-empty vertex set T (*sinks*), the minimum s-T-cut is the smallest cut in which s is one side of the cut and all vertices in T are on the other side. This is a generalization of minimum s-t-cuts that allows multiple vertices in T and can be easily replaced by a minimum s-t-cut by connecting every vertex in T with a new super-sink by infinite-capacity edges. We denote the capacity of a minimum-s-T-cut, i.e. the sum of weights in the smallest cut separating s from T, by $\lambda(G, s, T)$. This cut is also called the *minimum isolating cut* [8] for vertex s and vertex set T and the minimum isolating cut where the source side is the largest is called the *largest minimum isolating cut* for s and T.

In our algorithm we use *graph contraction* and *edge deletions*. Given an edge $e = (u,v) \in E$, we define G/e to be the graph after *contracting* e. In the contracted graph, we delete vertex v and all incident edges. For each edge $(v,x) \in E$, we add an edge (u,x) with $w(u,x) = w(v,x)$ to G or, if the edge already exists, we give it the edge weight $w(u,x) + w(v,x)$. For the *edge deletion* of an edge e, we define $G - e$ as the graph G in which e has been removed. Other vertices and edges remain the same. An *articulation point* is a vertex whose removal disconnects the graph G into multiple disconnected components. For a given multiterminal cut S, the graph $G \backslash S$ splits G into k connected components, as defined by the cut edges in S, each containing exactly one terminal.

While the multiterminal cut problem is NP-hard, it is *fixed-parameter tractable* (FPT), parameterized by the multiterminal cut weight $\mathcal{W}(G)$. A problem is fixed-parameter tractable with respect to some parameters σ so that there is an algorithm with runtime $f(\sigma) \cdot n^{\mathcal{O}(1)}$ and f is a computable function. Marx [15] proved that the multiterminal cut problem is FPT and Chen et al. [6] gave the first FPT algorithm with a running time of $4^{\mathcal{W}(G)} \cdot n^{\mathcal{O}(1)}$, later improved by Xiao [26] to $2^{\mathcal{W}(G)} \cdot n^{\mathcal{O}(1)}$ and by Cao et al. [5] to $1.84^{\mathcal{W}(G)} \cdot n^{\mathcal{O}(1)}$.

2.3 `VieCut-MTC` We present an improved solver for the multiterminal cut problem. Our work is based on a recent result by Henzinger et al. [12], in the following named `VieCut-MTC`. In this section we give a short summary of their results, for further details we refer the reader to their original work [12]. The `VieCut-MTC` multiterminal cut solver is a shared-memory parallel solver for the multiterminal cut problem. `VieCut-MTC` is a branch-and-reduce algorithm that performs a set of local contraction routines to transform the graph G into an instance of smaller size H, where the minimum multiterminal cut $\mathcal{W}(G) = \mathcal{W}(H)$, i.e. the minimum multiterminal cut $\mathcal{C}(G)$ can still be found on the smaller instance H. For this purpose, they use the following lemmas:

LEMMA 2.1. *[5][12] If an edge $e = (u,v) \in G$ is guaranteed not to be in at least one multiterminal cut $\mathcal{C}(G)$ (i.e. $\mathcal{V}_u = \mathcal{V}_v$), we can contract e and $\mathcal{W}(G/e) = \mathcal{W}(G)$.*

LEMMA 2.2. *[5][12] If an edge $e = (u,v) \in E$ is guaranteed to be in a minimum multiterminal cut, i.e. there is a minimum multiterminal cut $\mathcal{C}(G)$ in which $\mathcal{V}_u \neq \mathcal{V}_v$, we can delete e from G and $\mathcal{C}(G-e)$ is still a valid minimum multiterminal cut.*

Lemma 2.1 allows the contraction of edges that are guaranteed not to be in at least on multiterminal cut and Lemma 2.2 allows the deletion of edges that are guaranteed to be in a multiterminal cut. An example for such an edge is an edge that connects two terminal vertices.

Largest Minimum Isolating Cut Dahlhaus et al. [8] show that there exists a minimum multiterminal cut $\mathcal{C}(G)$ for a graph G such

that for every terminal $t \in T$ all vertices on the source side of the largest minimum isolating cut are in block t. Thus, according to Lemma 2.1, the source sides can be contracted into their respective terminals. The cut value of this problem is equal to the sum of all isolating cuts minus the heaviest, as any set of $t-1$ isolating cuts pairwisely separates all terminals form each other. A lower bound for the optimal solution is the sum of all isolating cuts divided by two [8, 12].

Reductions A variety of reductions in the work of Henzinger et al. [12] use Lemma 2.1 to contract edges and effectively reduce the size of the input graph. For every *low degree vertex* v with $|N(v)| \leq 2$, one can contract the heaviest edge incident to v as there is at least one multiterminal cut that does not contain it. Every *heavy edge* $e = (v, u)$ with $w(e) \cdot 2 \geq w(E[v])$ can also be contracted. This condition can be relaxed to *heavy triangles*, where an edge $e = (v_1, v_2)$ that is part of a triangle (v_1, v_2, v_3) can be contracted if $w(e) + w(v_1, v_3) \cdot 2 \geq w(E[v_1])$ and $w(e) + w(v_2, v_3) \cdot 2 \geq w(E[v_2])$. A more global reduction uses the CAPFOREST algorithm of Nagamochi et al. [17, 18] to find a connectivity lower bound of every edge in G in almost linear time. If an edge $e = (u, v)$ has *high connectivity*, i.e. there is no small cut that separates u and v, and no multiterminal cut that separates its incident vertices can be better than $\widehat{\mathcal{W}}(G)$, the edge can be contracted according to Lemma 2.1. For full descriptions and proofs of these reductions we refer the reader to Section 4 of [12].

Branching When it is not possible to find any more edges to contract or delete, `VieCut-MTC` selects an edge e incident to a terminal and creates two subproblems: G/e represents the problem in which e is not part of the multiterminal cut $\mathcal{C}(G)$ and $G - e$ represents the problem in which it is. Both subproblems are added to a shared-memory parallel problem queue $\mathcal{Q}$ and solved independently from each other.

3 Improved Reductions and Branching

We now introduce a set of new reductions to further decrease the problem size. Additionally, we give an alternative branching rule that allows for faster branching.

3.1 New Reductions

`VieCut-MTC` contracts edges incident to low degree vertices, edges with high weight and edges whose incident vertices have a high connectivity. Additionally, `VieCut-MTC` contracts the largest minimum isolating cut for each terminal to the remainder of the terminal set. We now introduce additional reductions that are able to further shrink the graph and thus speed up the algorithm.

3.1.1 Articulation Points

Let $\phi \in V$ be an articulation point in G whose removal disconnects the graph into multiple connected components. For any of these components that does not contain any terminals, we show that all vertices in the component can be contracted into ϕ.

LEMMA 3.1. *For an articulation point ϕ whose removal disconnects the graph G into multiple connected components $(G_1, \ldots, G_p)$ and a component G_i with $i \in \{1, \ldots, p\}$ that does not contain any terminals, no edge in G_i or connecting G_i with ϕ can be part of $\mathcal{C}(G)$.*

Proof. Let e be an edge that connects two vertices in $\{V_i \cup \phi\}$. Assume $e \in \mathcal{C}(G)$, i.e. e is part of the minimum multiterminal cut of G. This means that vertices in $\{V_i \cup \phi\}$ are not all in the same block. By changing the block affiliation of all vertices in $\{V_i \cup \phi\}$ to $\mathcal{V}_\phi$ we can remove all edges connecting vertices in $\{V_i \cup \phi\}$ from the multiterminal cut, thus decrease the weight of the multiterminal cut by at least $w(e)$. As ϕ is an articulation point, G_i is only connected to the rest of G through ϕ and thus no new edges are introduced to the multiterminal cut. This is a contradiction to the minimality of $\mathcal{C}(G)$, thus no edge e that connects two vertices in $\{V_i \cup \phi\}$ is in the minimum multiterminal cut $\mathcal{C}(G)$. ∎

Using Lemmas 2.1 and 3.1 we can contract all components that contain no terminals into the articulation point ϕ. All articulation points of a graph can be found in linear time using an algorithm by Tarjan and Vishkin [23] based on depth-first search. The algorithm performs a depth-first search and checks in the backtracking step whether for a vertex v there exists an alternative path from the parent of v to every of descendant of v. If there is no alternative path, v is an articulation point in G.

3.1.2 Equal Neighborhoods

In many cases, the resulting graph of the reductions contains groups of vertices that are connected to the same neighbors. If the neighborhood and respective edge weights of two vertices are equal, we can use Lemmas 2.1 and 3.2 to contract them into a single vertex.

LEMMA 3.2. *For two non-terminal vertices v_1 and v_2 with $\{N(v_1) \backslash v_2\} = \{N(v_2) \backslash v_1\}$ where for all $v \in \{N(v_1) \backslash v_2\}$, $w(v_1, v) = w(v_2, v)$, there is at least one minimum multiterminal cut where $\mathcal{V}_{v_1} = \mathcal{V}_{v_2}$.*

Proof. Let C be a partitioning of the vertices in G with $C(v_1) \neq C(v_2)$, let ζ be the corresponding cut, where $e = (u, v) \in \zeta$, if $C(u) \neq C(v)$ and let $cw(v)$ be the

total weight of edges in ζ incident to a vertex $v \in V$. W.l.o.g. let v_2 be the vertex with $cw(v_2) \geq cw(v_1)$. We analyze this in two steps: We assume that when moving v_2 to $C(v_1)$ that all edges incident to v_2 in its old location are removed from ζ, which drops the weight of ζ by $cw(v_2)$ and then all edges incident to v_2 in its new location are added to ζ, which is exactly $cw(v_1)$ by the conditions of the lemma. Thus the weight of ζ changes by $cw(v_1) - cw(v_2) \leq 0$. If the edge $e_{12} = (v_1, v_2)$ exists, both $cw(v_1)$ and $cw(v_2)$ are furthermore decreased by $w(e_{12})$, as the edge connecting them is not a cut edge anymore. As we only moved the block affiliation of v_2, the only edges newly introduced to ζ are edges incident to v_2. Thus, the total weight of the multiterminal cut was not increased by moving v_1 and v_2 into the same block and we showed that for each cut ζ, in which $C(v_1) \neq C(v_2)$ there exists a cut of equal or better value in which v_1 and v_2 are in the same block. Thus, there exists at least one multiterminal cut where $\mathcal{V}_{v_1} = \mathcal{V}_{v_2}$. □

We detect equal neighborhoods for all vertices with neighborhood size smaller or equal to a constant c_N using two linear time routines. To detect neighboring vertices v_1 and v_2 with equal neighborhood, we sort the neighborhood vertex IDs including edge weights by vertex IDs (excluding the respective other vertex) for both v_1 and v_2 and check for equality. To detect non-neighboring vertices v_1 and v_2 with equal neighborhood, we create a hash of the neighborhood sorted by vertex ID for each vertex with neighborhood size smaller or equal to c_N. If hashes are equal, we check whether the condition for contraction is actually fulfilled. As the neighborhoods to sort only have constant size, they can be sorted in constant time and thus the procedures can be performed in linear time. We perform both tests, as the neighborhoods of neighboring vertices contain each other and therefore do not result in the same hash value; and non-neighboring vertices are not in each others neighborhood and therefore finding them requires checking the neighborhood of every neighbor, which results in a large search space. We set $c_N = 5$, as in most cases where we encountered equal neighborhoods they are in vertices with neighborhood size ≤ 5.

3.1.3 Maximum Flow from Non-terminal Vertices Let v be an arbitrary vertex in $V \backslash T$, i.e. a non-terminal vertex of G. Let $(V_v, V \backslash V_v)$ be the largest minimum isolating cut of v and the set of terminal vertices T. Lemma 3.3 shows that there is at least one minimum multiterminal cut $\mathcal{C}(G)$ so that $\forall x \in V_v : \mathcal{V}_x = \mathcal{V}_v$ and thus V_v can be contracted into a single vertex.

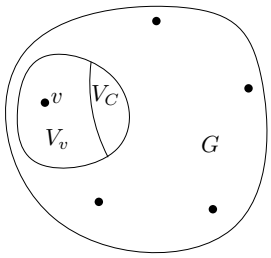


Figure 1: Illustration of vertex sets in Lemma 3.3

LEMMA 3.3. *Let v be a vertex in $V \backslash T$. Let $(V_v, V \backslash V_v)$ be the largest minimum isolating cut of v and the set of terminal vertices T and let $\lambda(G, v, T)$ be the weight of the minimum isolating cut $(V_v, V \backslash V_v)$. There exists at least one minimum multiterminal cut $\mathcal{C}(G)$ in which $\forall x \in V_v : \mathcal{V}_x = \mathcal{V}_v$.*

Proof. As $(V_v, V \backslash V_v)$ is a minimum isolating cut with the terminal set as sinks, we know that no terminal vertex is in V_v. Assume that $\mathcal{C}(G)$ cuts V_v, i.e. there is a non empty vertex set $V_C \in V_v$ so that $\forall x \in V_C : \mathcal{V}_x \notin \mathcal{V}_v$. We will show that the existence of such a vertex set contradicts the minimality of $\mathcal{C}(G)$. Figure 1 gives an illustration of the vertex sets defined here.

Due to the minimality of the minimum isolating cut, we know that $w(V_C, V_v \backslash V_C) \geq w(V_C, V \backslash V_v)$ (i.e. the connection of V_C to the rest of V_v is at least as strong as the connection of V_C to $(V \backslash V_v)$), as otherwise we could remove V_C from V_v and find an isolating cut of smaller size.

We now show that by changing the block affiliation of all vertices in V_C to $\mathcal{V}_v$, i.e. removing all vertices from the set V_C, we can construct a multiterminal cut of equal or better cut value. By changing the block affiliation of all vertices in V_C to $\mathcal{V}_v$, we remove all edges connecting V_C to $(V_v \backslash V_C)$ from $\mathcal{C}(G)$ and potentially more, if there were edges in $\mathcal{C}(G)$ that connect two vertices both in V_C. At most, the edges connecting V_C and $(V \backslash V_v)$ are newly added to $\mathcal{C}(G)$. As $w(V_C, V_v \backslash V_C) \geq w(V_C, V \backslash V_v)$, the cut value of $\mathcal{C}(G)$ will be equal or better than previously. Thus, there is at least one multiterminal cut in which V_C is empty and therefore $\forall x \in V_v : \mathcal{V}_x = \mathcal{V}_v$. □

We can therefore run a maximum s-T-flow from a non-terminal vertex to the set of all terminals T and contract the source side of the largest minimum isolating cut into a single vertex. These flow problems can be solved embarrassingly parallel, in which every processor solves an independent maximum s-T-flow problem for a different non-terminal vertex v.

While it is possible to run a flow problem from every vertex in V, this is obviously not feasible as it would entail excessive running time overheads. Promising vertices to use for maximum flow computations are

either high degree vertices or vertices with a high distance from every terminal. High degree vertices are promising, as due to their high degree it is more likely that we can find a minimum isolating cut of size less than their degree. Vertices that have a high distance to all terminals are on 'the edge of the graph', potentially in a subgraph only weakly connected to the rest of the graph. Running a maximum flow then allows us to contract this subgraph. In every iteration, we run 5 flow problems starting from high-distance vertices and 5 flow problems starting from high-degree vertices.

3.2 Vertex Branching When the `VieCut-MTC`-algorithm is initialized, it only has a single problem containing the whole graph G. While independent minimum isolating cuts are computed in parallel, most of the shared-memory parallelism in `VieCut-MTC` comes from the embarrassingly parallel solving of different problems on separate threads. When branching, `VieCut-MTC` selects the highest degree vertex that is adjacent to a terminal and branches on the heaviest edge connecting it to one of the terminals. The algorithm thus creates only up to two subproblems and is still not able to use the whole machine.

We propose a new branching rule that overcomes these limitations by selecting the highest degree vertex incident to at least one terminal and use it to create multiple subproblems to allow for faster startup. Let x be the vertex used for branching, $\{t_1, \dots, t_i\}$ for some $i \geq 1$ be the adjacent terminals of x and w_M be the weight of the heaviest edge connecting x to a terminal. We now create up to $i+1$ subproblems as follows:

For each terminal t_j with $j \in \{1, \dots, i\}$ with $w(x, t_j) + w(x, V \backslash T) > w_M$ create a new problem P_j where edge (x, t_j) is contracted and all other edges connecting x to terminals are deleted. Thus in problem P_j, vertex x belongs to block $\mathcal{V}_{t_j}$. If $w(x, t_j) + w(x, V \backslash T) \leq w_M$, i.e. the weight sum of the edges connecting x with t_j and all non-terminal vertices is not heavier than w_M, the assignment to block $\mathcal{V}_{t_j}$ cannot be optimal and thus we do not need to create the problem P_j, also called *pruning* of the problem. The following Lemma 3.4 proves the correctness of this pruning step.

LEMMA 3.4. *Let $G = (V, E)$ be a graph, $T \subseteq V$ be the set of terminal vertices in G, and $x \in V$ be a vertex that is adjacent to at least one terminal and for an $i \in \{1, \dots, |T|\}$ be the index of the terminal for which $e_i = (x, t_i)$ is the heaviest edge connecting x with any terminal. Let w_M be the weight of e_i. If there exists a terminal t_j adjacent to x with $j \in \{1, \dots, |T|\}$ with $w(x, t_j) + w(x, V \backslash T) \geq w_M$, there is at least one minimum multiterminal cut $\mathcal{C}(G)$ so that $\mathcal{V}_x \neq j$, i.e. x is not in block j.*

Proof. If $\mathcal{V}_x = i$, i.e. x is in the block of the terminal it has the heaviest edge to, the sum of cut edge weights incident to x is $\leq E(x) - w_M$, as edge e_i of weight w_M is not a cut edge in that case. If $\mathcal{V}_x = j$, i.e. x is in the block of terminal j, the sum of cut edge weights incident to x is $\geq E(x) - (w(x, V \backslash T) + w(x, t_j))$, as all edges connecting x with other terminals than t_j are guaranteed to be cut edges. As $w(x, t_j) + w(x, V \backslash T) \geq w_M$, even if all non-terminal neighbors of x are in block j, the weight sum of incident cut edges is not lower than when x is placed in block i. As the block affiliation of x can only affect its incident edges, the cut value of every solution that sets $\mathcal{V}_x = j$ would be improved or remain the same by setting $\mathcal{V}_x = i$. ∎

If $w(x, V \backslash T) > w_M$ and $i < |T|$, we also create problem P_{i+1}, in which all edges connecting x to a terminal are deleted. This problem represents the assignment of x to a terminal that is not adjacent to it. We add each subproblem whose lower bound is lower than the currently best found solution $\widehat{\mathcal{W}}$ to the problem queue $\mathcal{Q}$. As we create up to $|T|$ subproblems, this allows for significantly faster startup of the algorithm and allows us to use the whole parallel machine after less time than before.

3.3 Integer Linear Programming Integer Linear Programming can be used as an alternative to branch-and-reduce [12] and for some problems this is faster than branch-and-reduce. We integrate the ILP formulation from the work of [12] and include it directly into `VieCut-MTC` as an alternative to branching. We give the ILP solver a time limit and if it is unable to find an optimal solution within the time limit, we instead perform a branch operation. In Section 5.2 we study which subproblems to solve with an ILP first.

3.4 Improving Bounds with Greedy Optimization The `VieCut-MTC` algorithm prunes problems which cannot result in a solution which is better than the best solution found so far. Therefore, even though it is a deterministic algorithm that will output the optimal result when it terminates, performing greedy optimization on intermediate solutions allows for more aggressive pruning of problems that cannot be optimal. Additionally, `VieCut-MTC` has reductions that depend on the value of $\widehat{\mathcal{W}}(G)$ and can thus contract more vertices if the cut value $\widehat{\mathcal{W}}(G)$ is lower.

For a subproblem $H = (V_H, E_H)$ with solution ρ, the original graph $G = (V_G, E_G)$ and a mapping $\pi : V_G \to V_H$ that maps each vertex in V_G to the vertex in V_H that encompasses it, we can transfer the solution ρ to a solution γ of G by setting the block affiliation of

every vertex $v \in V_G$ to $\gamma(v) := \pi(\rho(v))$. The cut value of the solution $w(\gamma)$ is defined as the sum of weights of the edges crossing block boundaries, i.e. the sum of edge weights where the incident vertices are in different blocks. Let $\xi_i(V_G)$ be the set of all vertices $v \in V_G$ where $\gamma(v) = i$.

We introduce the following greedy optimization operators that can transform γ into a better multiterminal cut solution γ_{IMP} with $w(\gamma_{\text{IMP}}) < w(\gamma)$.

3.4.1 Kernighan-Lin Local Search Kernighan and Lin [14] give a heuristic for the traveling-salesman problem that has been adapted to many hard optimization problems [20, 24, 27, 11], where each vertex $v \in V_G$ is assigned a gain $g(v) = \max_{i \in \{i,\ldots,|T|\}, i \neq \gamma(v)} \sum w(v, \xi_i(V_G)) - w(v, \xi_{\gamma(v)}(V_G))$, i.e. the improvement in cut value to be gained by moving v to another block, the best connected other block. We perform runs where we compute the gain of every vertex that has at least another neighbor in a different block and move all vertices with non-negative gain. Additionally, if a vertex v has a negative gain, we store its gain and associated best connected other block. For any neighbor u of v that also has the same best connected other block, we check whether $g(w)+g(v)+2 \cdot w(v,u) > 0$, i.e. moving both u and v at the same time is a positive gain move. If it is, we perform the move.

3.4.2 Pairwise Maximum Flow For any pair of blocks $1 \leq i < j \leq |T|$ where $w(\xi_i(V_G), \xi_j(V_G)) > 0$, i.e. there is at least one edge from block i to block j, we can create a maximum s-t flow problem between them: we create a graph F_{ij} that contains all vertices in $\xi_i(V_G)$ and $\xi_j(V_G)$ and all edges that connect these vertices.

Let H be the current problem graph created by performing reductions and branching on the original graph G. All vertices that are encompassed in the same vertex in problem graph H as the terminals i and j are hereby contracted into the corresponding terminal vertex. We perform a maximum s-t-flow between the two terminal vertices and re-assign vertex assignments in γ according to the minimum s-t-cut between them. As we only model blocks $\xi_i(V_G)$ and $\xi_j(V_G)$, this does not affect other blocks in γ. In the first run we perform a pairwise maximum flow between every pair of blocks i and j where $w(\xi_i(V_G), \xi_j(V_G)) > 0$ in random order. We continue on all pairs of blocks where $w(\xi_i(V_G), \xi_j(V_G))$ was changed since the end of the previous maximum flow iteration between them.

We first perform Kernigham-Lin local search until there is no more improvement, then pairwise maximum flow until there is no more improvement, followed by another run of Kernigham-Lin local search. As pairwise maximum flow has significantly higher running time, we spawn a new thread to perform the optimization if there is a CPU core that is not currently utilized.

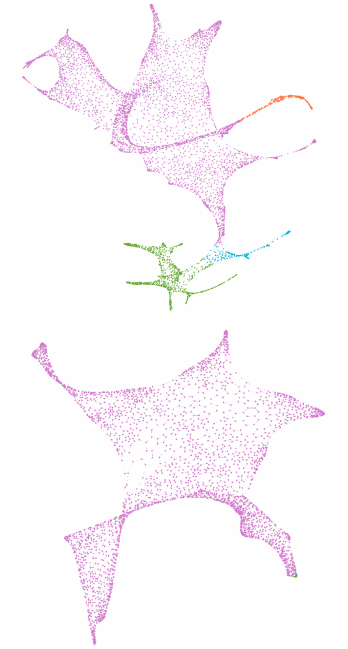


Figure 2: Minimum multiterminal cut for graph `uk` [21] and four terminals - on original graph (top) and remaining graph at time of first branch operation (bottom), visualized using Gephi-0.9.2 [2]

4 Fast Inexact Solving

`VieCut-MTC` in an exact algorithm, i.e. when it terminates the output is guaranteed to be optimal. As the multiterminal cut problem is NP-complete [8], it is not feasible to expect termination in difficult instances of the problem. Henzinger et al. [12] report that their algorithm often does not terminate with an optimal result but runs out of time or memory and returns the best result found up to that point. Thus, it makes sense to relax the optimality constraint and aim to find a high-quality (but not guaranteed to be optimal) solution faster.

A key observation herefor is that in many problems, most, if not all vertices that are not already contracted into a terminal at the time of the first branch, will be assigned to a few terminals whose weighted degree at that point is highest. See Figure 2 for an example with 4 terminals (selected with high distance to each other) on graph `uk` from the Walshaw Graph Partitioning Archive [21]. As we can see, at the time of the first branch (right figure), most vertices that are not assigned to the pink terminal in the optimal solution are already contracted into their respective terminals.

The remainder is mostly assigned to a single terminal. As we can observe similar behavior in many problems, we propose the following heuristic speedup operations:

Let $\delta \in (0, 1)$ be a contraction factor and T_H be the set of all terminals that are not yet isolated in graph H. In each branching operation on an intermediate graph H, we delete all edges around the $\lceil \delta \cdot |T_H| \rceil$ terminals with lowest degree. Additionally, we contract all vertices adjacent to the highest degree terminal that are not adjacent to any other terminal into the highest degree terminal. This still allows us to find all solutions in which no more vertices were added to the lowest degree terminals and the adjacent vertices are in the same block as the highest degree terminals.

Additionally, in a branch operation on vertex v, we set a maximum branching factor β and only create problems where v is contracted into the β adjacent terminals it has the heaviest edges to and one problem in which it is not contracted into either adjacent terminal. This is based on the fact that all other edges connecting v to other terminals will be part of the multiterminal cut and the greedy assumption that it is likely that the optimal solution does not contain at least one of these heavy edges. By default, we set $\delta = 0.1$ and $\beta = 5$.

5 Experiments and Results

We now perform an experimental evaluation of the proposed work. This is done in the following order: first we analyze the impact of different reductions introduced in the work of Henzinger et al. [12] and in this work. We then analyze which subproblems to solve using integer linear programming and then compare the results of `VieCut-MTC` with our exact and inexact algorithms on a variety of graphs from different sources. Here, `VieCut-MTC` denotes the algorithm of Henzinger et al. [12], `Exact-MTC` denotes the exact version of our algorithm and `Inexact-MTC` denotes the heuristic algorithm proposed in Section 4

We implemented the algorithms using C++-17 and compiled all code using g++ version 7.3.0 with full optimization (`-O3`). Our experiments are conducted on two machine types. Machine A is a machine with two Intel Xeon E5-2643v4 with 3.4 GHz with 6 CPU cores each and 1.5 TB RAM in total. Machine B is a machine in the Vienna Scientific Cluster with two Intel Xeon E5-2650v2 with 2.6GHz with 8 CPU cores each and 64 GB RAM in total. We limit the maximum amount of memory used for each problem to 32 GB. ILP problems are solved using Gurobi 8.1.0. When we report a mean result we give the geometric mean as problems differ significantly in cut size and time. Our code is freely available under the permissive MIT license[1].

To evaluate the performance of different multiterminal cut algorithms, we use a wide variety of graphs from different sources. We re-use a large subset of the map, social and web graphs graphs used by Henzinger et al. [12]. Additionally, we add numerical graphs from the Walshaw Graph Partitioning Benchmark [21] and a set of graphs from the 10^{th} DIMACS implementation challenge [1] and the SuiteSparse Matrix Collection (formerly UFSparse Matrix Collection) [9]. Table 1 gives an overview over the graphs used in this work.

Table 1: Large Real-world Benchmark Instances

Graph	Source	n	m
Map Graphs			
ak2010	[1]	45 292	$109K$
ca2010	[1]	$710K$	$1.74M$
ct2010	[1]	67 578	$168K$
de2010	[1]	24 115	58 028
hi2010	[1]	25 016	62 063
luxembourg.osm	[9]	$115K$	$120K$
me2010	[1]	69 518	$168K$
netherlands.osm	[9]	$2.22M$	$2.44M$
nh2010	[1]	48 837	$117K$
nv2010	[1]	84 538	$208K$
ny2010	[1]	$350K$	$855K$
ri2010	[1]	25 181	62 875
sd2010	[1]	88 360	$205K$
vt2010	[1]	32 580	77 799
Social, Web and Numerical Graphs			
598a	[21]	$111K$	$742K$
astro-ph	[9]	16 706	$121K$
bcsstk30	[21]	28 924	$1.01M$
ca-CondMat	[9]	23 133	93 439
caidaRouterLevel	[9]	$192K$	$609K$
citationCiteseer	[9]	$268K$	$1.16K$
cit-HepPh	[9]	34 546	$422K$
cnr-2000	[9]	$326K$	$2.74M$
coAuthorsCiteseer	[9]	$227K$	$814K$
cond-mat-2005	[9]	40 421	$176K$
coPapersCiteseer	[9]	$434K$	$16.0M$
cs4	[21]	22 499	43 858
eu-2005	[3]	$862K$	$16.1M$
fe_body	[21]	45 087	$164K$
higgs-twitter	[9]	$457K$	$14.9M$
in-2004	[3]	$1.38M$	$13.6M$
NACA0015	[9]	$1.04M$	$3.11M$
uk-2002	[3]	$18.5M$	$261M$
venturiLevel3	[9]	$4.03M$	$8.05M$
vibrobox	[21]	12 328	$165K$

[1] `https://github.com/VieCut/VieCut`

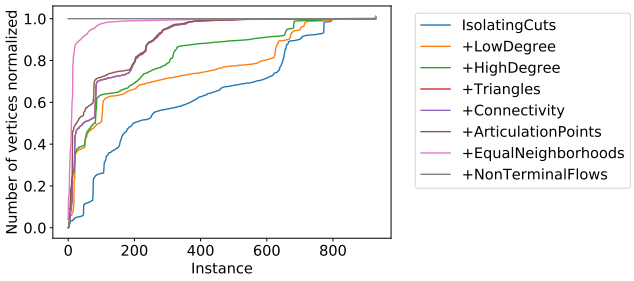


Figure 3: Number of vertices in graph after reductions are finished, normalization by (# vtx with all reductions / # vtx in variant), sorted by normalized value.

As the instances generally do not have any terminals, we find random vertices that have a high distance from each other in the following way: we start with a random vertex r, run a breadth-first search starting at r and select the vertex v encountered last as first terminal. While the number of terminals is smaller than desired, we add another terminal by running a breadth-first search from all current terminals and adding the vertex encountered last to the list of terminals. We then run a bounded-size breadth-first search around each terminal to create instances where the minimum multiterminal cut does not have $k-1$ blocks consisting of just a single vertex each. This results in problems in which well separated clusters of vertices are partitioned and the task consists of finding a partitioning of the remaining vertices in the boundary regions between already partitioned blocks. This relates to clustering tasks, in which well separated clusters are labelled and the task consists of labelling the remaining vertices in-between. Additionally, we use a subset of the generated instances of Henzinger et al. [12] to compare our work to `VieCut-MTC`. These graphs have unit-weight edges, however contracted subproblems often have weighted edges.

5.1 Reductions We first analyze the impact of the different reductions on the size of the graph at the time of first branch. For this, we run experiments on all graphs in Table 1 with $k = \{4, 8, 10\}$ terminals and 10% of all vertices added to the terminals on machine B. On these instances, we run subsets of all contractions exhaustively and check which factor of vertices remain in the graph. A value of 1 thus indicates that the reductions were unable to find any edges to contract, a value close to 0 shows that almost no vertices remain and the resulting problem is far smaller than the original problem. Figure 3 gives the result with 8 different variants, starting with a version that only runs isolating cuts and adding one reduction family per version. For this, we sorted the reductions by their impact on the total running time. In Figure 3, we can see that using all reductions allows us to reduce the number of vertices by more than a half in about half of all instances and can find a sizable number of reductions on almost any instance.

We can see that running the local reductions in `VieCut-MTC` are very effective on almost all instances. In average, `IsolatingCuts` reduce the number of vertices by 33%, `LowDegree` reduces the number of vertices in the remaining graph by 17%, `HighDegree` by 7% and `Triangles` by 8%. In contrast, `Connectivity` only has a negligible effect, as it contracts edges whose connectivity is larger than a value related to the difference of upper bound to total weight of deleted edges. As there are almost no deleted edges in the beginning, this value is very high and almost no edge has high enough connectivity.

In average, `ArticulationPoints` reduces the number of vertices on the already contracted graphs by 1.9%, `EqualNeighborhoods` reduces the number of vertices by 7.8% - mostly in sparse regions of the graph, and `NonTerminalFlows` reduces the number of vertices by 2.0%. However, there are some instances in which the newly introduced reductions reduce the number of vertices remaining by more than 99%.

5.2 Integer Linear Programming In order to get a wide variety of ILP problems, we run the `Inexact-MTC` algorithm on all instances in Table 1 with $k = 10$ terminals and 10% of vertices added to the terminals on machine B. As `Inexact-MTC` removes low-degree terminals and contracts edges, we have subproblems with very different sizes and numbers of terminals. In this experiment, whenever `Inexact-MTC` chooses between branching and ILP on graph G, we select a random integer $r \in (1{,}200\,000)$. We use a random integer as we want to have problems of varying sizes. We select 200000 as the maximum, as we did not encounter any larger instances that were solved in the allotted time. If $|E| < r$, the problem is solved with ILP, otherwise the algorithm branches on a vertex incident to a terminal. The timeout is set to 60 seconds.

Figure 4 shows the time needed to solve the ILP problems in relation to the number of edges in the graph. We can see that there is a strong correlation between problem size and total running time, but there are still a large number of outliers that cannot be solved in the allotted time even though the instances are rather small. In the following, we set the limit to 50 000 edges and solve all instances with fewer than 50 000 edges with an integer linear program. If the instance has at least 50 000 edges, we branch on a vertex incident to a terminal and create more subproblems.

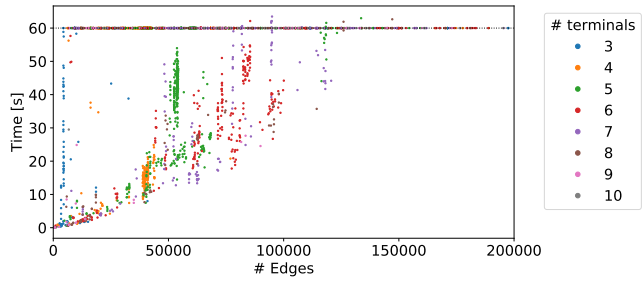


Figure 4: Running time of ILP subproblems in relation to $|E|$.

5.3 Comparison to VieCut-MTC We use the experiment of Section 8.7 in the work of Henzinger et al. [12] to compare Exact-MTC to VieCut-MTC on the instances used in their work. The experiment uses a set of large social and web graphs with pre-defined clusters and $k = \{3, 4, 5, 8\}$ terminals, where $10 - 25\%$ of vertices are marked as terminal vertices initially, a total of 160 instances. We run the experiments on machine A using all 12 cores and set the time limit to 600 seconds.

Out of 160 instances, VieCut-MTC terminates with an optimal result in 32 instances, while Exact-MTC terminates with an optimal result in 46 instances. Out of the 115 instances that were not solved to optimality by both algorithms, Exact-MTC gives a better result on 75 instances and the same result on the other 38 instances. The geometric mean values of results given by Exact-MTC and Inexact-MTC are both about 1.5% lower than VieCut-MTC. Note that in the experiments performed in [12], which uses a larger machine (32 cores) and has a timeout of 3600 seconds, VieCut-MTC has a geometric mean of about 0.1% better than VieCut-MTC in this work. The largest part of the improvement of Exact-MTC and Inexact-MTC over VieCut-MTC is gained by the greedy optimization detailed in Section 3.4.

Figure 5a shows the performance profile [10] of this experiment. We can see that both Exact-MTC and Inexact-MTC are almost always optimal or very close to it. In contrast, VieCut-MTC gives noticeably worse results on about 20% of instances and more than 5% worse results on 10% of instances.

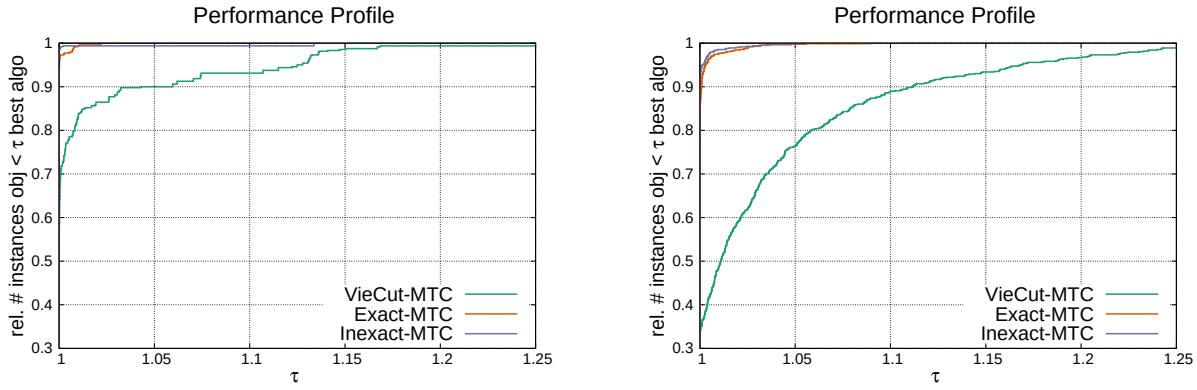
5.4 Large Multiterminal Cut Problems We compare VieCut-MTC, Exact-MTC and Inexact-MTC on all graphs with $k = \{4, 5, 8, 10\}$ terminals and 10% and 20% of vertices added to the terminal. For each combination of graph, number of terminals and factor of vertices in terminal, we create three problems with random seeds $s = \{0, 1, 2\}$. Thus, we have a total of 816 problems. We set the time limit per algorithm and problem to 600 seconds. We run the experiment on machine A using all 12 CPU cores. If the algorithm does not terminate in the allotted time or memory limit, we report the best intermediate result. Note that is a soft limit, in which the algorithm finishes the current operation and exits afterwards if the time or memory limit is reached. As many of these are very large instances, most instances in this section are not solved to optimality.

Table 2: Result overview for large multiterminal cut problems on graphs from Table 1.

# Terminals		VieCut-MTC	Exact-MTC	Inexact-MTC
4	Best Solution	109	**183**	175
	Mean Solution	161 799	**159 402**	159 499
	Better Exact	6	**94**	—
5	Best Solution	81	**173**	158
	Mean Solution	216 191	**210 928**	211 090
	Better Exact	6	**121**	—
8	Best Solution	42	139	**175**
	Mean Solution	346 509	331 112	**330 856**
	Better Exact	2	**162**	—
10	Best Solution	37	129	**173**
	Mean Solution	412 138	392 561	**391 822**
	Better Exact	1	**165**	—

Table 2 gives an overview of the results. For each algorithm, we give the number of times, where it gives the best (or shared best) solution over all algorithms; the geometric mean of the cut value; and for VieCut-MTC and Exact-MTC the number of instances in which they have a better result than the respective other. In all instances, in which VieCut-MTC and Exact-MTC terminate with the optimal result, Inexact-MTC also gives the optimal result. We can see that in the problems with 4 and 5 terminals, Exact-MTC slightly outperforms Inexact-MTC both in number of best results and mean solution value. In the problems with 8 and 10 terminals, Inexact-MTC has slightly better results in average. Thus, disregarding the optimality constraint can allow the algorithm to give better solutions faster especially in hard problems with a large amount of terminals.

However, both new algorithms outperform VieCut-MTC on almost all instances where not all algorithms give the same result. Here, Exact-MTC gives a better result than VieCut-MTC in 66% of all instances, while VieCut-MTC gives the better result in only 2% of all instances. As most problems do not terminate with an optimal result, we are unable to say how far the solutions are from the globally optimal solution. Note that Inexact-MTC gives an optimal result in all instances in which all algorithms terminate. Figure 6 shows the progress of the best solution for the algorithms in a set of problems. For both Exact-MTC and Inexact-MTC we can see large improvements to the cut value when the local search algorithm is finished on the first subproblem. In



(a) Instances of Section 5.3

(b) Instances of Section 5.4

Figure 5: Performance profiles for multiterminal cut algorithms

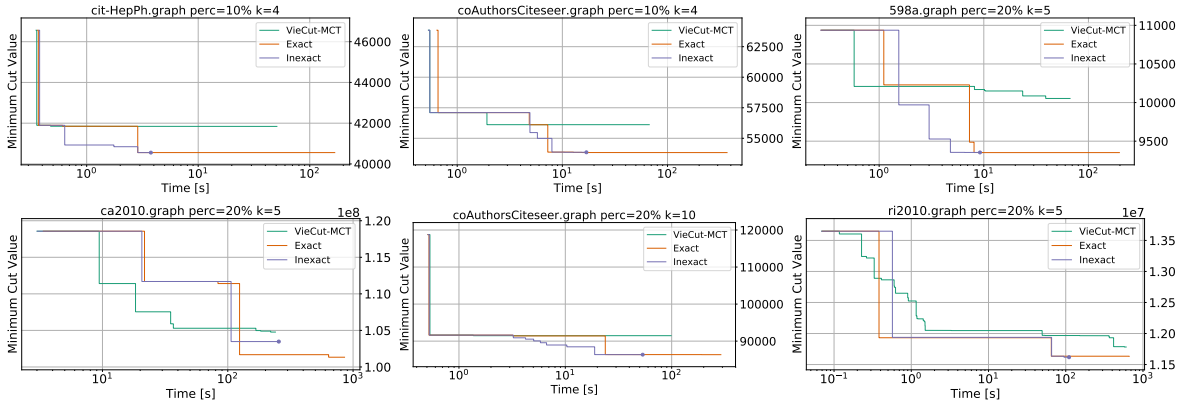


Figure 6: Progression of best result over time. Dot at end marks termination of algorithm.

contrast, `VieCut-MTC` has more small step-by-step improvements and generally gives worse results.

Figure 5b shows the performance profile for the instances in this section. Here we can see that `VieCut-MTC` has significantly worse results on a large subset of the instances, with more than 10% of instances where the result is worse by more than 10%. Also, on a few instances, the results given by `Exact-MTC` and `Inexact-MTC` differ significantly. In general, both of them outperform `VieCut-MTC` on most instances that are not solved to optimality by every algorithm.

6 Conclusion

In this paper, we give a fast parallel solver that gives high-quality solutions for large multiterminal cut problems. We give a set of highly-effective reduction rules that transform an instance into a smaller equivalent one. Additionally, we directly integrate an ILP solver into the algorithm to solve subproblems well suited to be solved using an ILP; and develop a flow-based local search algorithm to improve a given optimal solution. These optimizations significantly increase the number of instances that can be solved to optimality and improve the cut value of multiterminal cuts in instances that can not be solved to optimality. Our algorithm gives better solutions in more than two thirds of these instances, often improving the result by more than 5% on hard instances. Additionally, we give an inexact algorithm for the multiterminal cut problem that aggressively shrinks the graph instances and is able to even outperform the exact algorithm on many of the hardest instances that are too large to be solved to optimality while still giving the exact solution for most easier instances. Important future work consists of improving the scalability of the algorithm by giving a distributed memory version.

References

[1] David A Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. *Encyclopedia of Social Network Analysis and Mining*, pages 73–82, 2014.

[2] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Third international AAAI conference on weblogs and social media*, 2009.

[3] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proceedings of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[4] Niv Buchbinder, Joseph Seffi Naor, and Roy Schwartz. Simplex partitioning via exponential clocks and the multiway cut problem. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 535–544. ACM, 2013.

[5] Yixin Cao, Jianer Chen, and J-H Fan. An o∗(1.84 k) parameterized algorithm for the multiterminal cut problem. *Information Processing Letters*, 114(4):167–173, 2014.

[6] Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.

[7] William H Cunningham. The optimal multiterminal cut problem. In *Reliability of computer and communication networks*, pages 105–120, 1989.

[8] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[9] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[10] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[11] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[12] Monika Henzinger, Alexander Noe, and Christian Schulz. Shared-memory branch-and-reduce for multiterminal cuts. In *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 42–55. SIAM, 2020.

[13] Ulas Karaoz, TM Murali, Stan Letovsky, Yu Zheng, Chunming Ding, Charles R Cantor, and Simon Kasif. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of the National Academy of Sciences*, 101(9):2888–2893, 2004.

[14] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[15] Dániel Marx. Parameterized graph separation problems. *Theoretical Computer Science*, 351(3):394–406, 2006.

[16] Elena Nabieva, Kam Jim, Amit Agarwal, Bernard Chazelle, and Mona Singh. Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 21(suppl_1):i302–i310, 2005.

[17] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[18] Hiroshi Nagamochi, Tadashi Ono, and Toshihide Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming*, 67(1):325–341, 1994.

[19] Ulrich Pferschy, Rüdiger Rudolf, and Gerhard J. Woeginger. Some geometric clustering problems. *Nord. J. Comput.*, 1(2):246–263, 1994.

[20] Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA 2013)*, volume 7933 of *LNCS*, pages 164–175. Springer, 2013.

[21] Alan J Soper, Chris Walshaw, and Mark Cross. A combined evolutionary search and multilevel optimisation approach to graph-partitioning. *Journal of Global Optimization*, 29(2):225–241, 2004.

[22] Harold S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Eng.*, 3(1):85–93, 1977.

[23] Robert E Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.

[24] Jesper Larsson Träff. Direct graph k-partitioning with a kernighan–lin like heuristic. *Operations Research Letters*, 34(6):621–629, 2006.

[25] Alexei Vazquez, Alessandro Flammini, Amos Maritan, and Alessandro Vespignani. Global protein function prediction from protein-protein interaction networks. *Nature biotechnology*, 21(6):697, 2003.

[26] Mingyu Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory of Computing Systems*, 46(4):723–736, 2010.

[27] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

Parameterized algorithms for identifying gene co-expression modules via weighted clique decomposition*

Madison Cooley† Casey S. Greene‡ Davis Issac§ Milton Pividori¶
Blair D. Sullivan‖

June 1, 2021

Abstract

We present a new combinatorial model for identifying regulatory modules in gene co-expression data using a decomposition into weighted cliques. To capture complex interaction effects, we generalize the previously-studied weighted edge clique partition problem. As a first step, we restrict ourselves to the noise-free setting, and show that the problem is fixed parameter tractable when parameterized by the number of modules (cliques). We present two new algorithms for finding these decompositions, using linear programming and integer partitioning to determine the clique weights. Further, we implement these algorithms in Python and test them on a biologically-inspired synthetic corpus generated using real-world data from transcription factors and a latent variable analysis of co-expression in varying cell types.

1 Introduction

Biomedical research has recently seen a burgeoning of methods that incorporate network analysis to improve understanding and prediction of complex phenotypes [17]. These approaches leverage information encoded in the interactions of proteins or genes, which are naturally modeled as graphs. Further, there has been an explosion of available data including large gene expression compendia [5, 20] and protein-protein interaction maps [35].

A core problem in this area has always been identifying groups of co-acting genes/proteins, which often manifest as a clique or dense subgraph in the resulting network. In this work, we consider the specific setting of identifying gene co-expression modules (or pathways) from large datasets, with a downstream objective of aiding the development of new therapies for human disease.

There is substantial evidence that drugs with genetic support are more likely to progress through the drug development pipeline [30]. Prior work has shown that approaches that consider genes roles in biological networks can be robust to gene mapping noise [10], which might suggest alternative treatment avenues when a directly associated gene cannot be targeted.

Unfortunately, the membership of genes in modules and the relative strength of effect a module has on co-expression of its constituents are not directly observable. In gene co-expression analysis, what we are able to obtain is pairwise correlations for all genes in the organism [24]. Existing approaches rely on machine-learning to identify clusters in these data sets [10, 22]; here, we propose a new combinatorial model for the problem.

By modeling the observed gene expression data as a projection of a weighted bipartite graph representing gene-module membership and strength of expression for each module, we can represent the problem as a decomposition of the co-expression network into a collection of (potentially overlapping) weighted cliques (we call this Weighted Clique Decomposition).

While the resulting problem is naturally NP-hard, we demonstrate that techniques from parameterized algorithms enable efficient approaches when the number of modules is small. We present two parameterized algorithms for solving this problem[1]; both run in polynomial time in the network size, but have exponential dependence on the number of modules. As a first step towards practicality, we implement these methods [6] and provide preliminary experimental results on biologically-inspired synthetic networks with ground-truth modules derived from data on gene transcription factors and gene co-expression modules identified using a machine-learning approach.

*This work was supported by the NIH R01 HG010067 and the Gordon & Betty Moore Foundation under awards GBMF4552 and GBMF4560.

†University of Utah, mcooley@cs.utah.edu
‡University of Colorado School of Medicine, greenescientist@gmail.com
§Hasso Plattner Institute, davis.issac@hpi.de
¶University of Pennsylvania, milton.pividori@pennmedicine.upenn.edu
‖University of Utah, sullivan@cs.utah.edu

[1] one of which restricts to integral edge weights

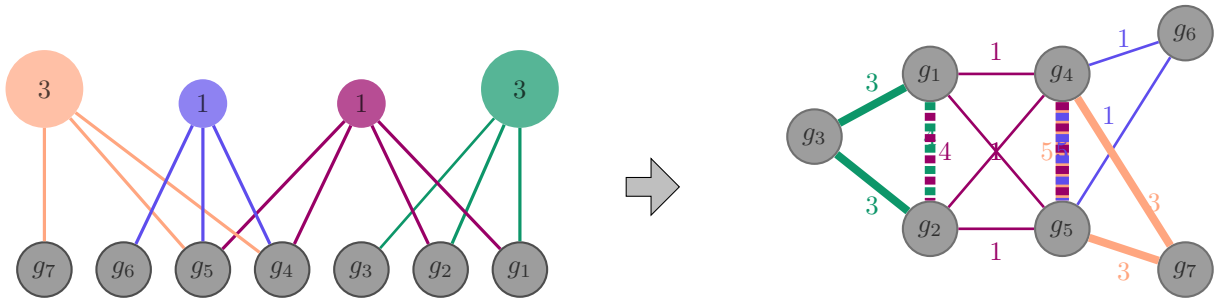


Figure 1: A bipartite graph (left) of genes $g_1, \ldots g_7$ and modules (top, labelled with strength of expression) gives rise to a gene-gene interaction network (right) with edges weighted by the sum of the strengths of all modules that contain both endpoints (indicated by color coding).

The appendices for this paper may be found in the full version, available at `http://arxiv.org/abs/2106.00657`. Appendices A, B, C, and G provide algorithm details, correctness proofs, and runtime analysis. Appendix D provides details on data generation, Appendix E contains supplemental experimental results, and NP-hardness of our primary problem (EWCD) is proven in Appendix F.

2 Motivating Biological Problem

Complex human traits and diseases are caused by an intricated molecular machinery that interacts with environmental factors. For example, although asthma has some common features such as wheeze and shortness of breath, research suggests that this highly heterogeneous disease is comprised of several conditions [38], such as childhood-onset asthma and adult-onset asthma, which present different prognosis and response to treatment, and also differ in their genetic risk factors [31]. Genome-wide association studies (GWAS) are designed to improve our understanding of how genetic variation leads to phenotype by detecting genetic variants correlated with disease. GWAS have prioritized causal molecular mechanisms that, when disturbed, confer disease susceptibility, and these findings were later translated to new treatments [36]. Drug targets backed by the support of genetic associations are more likely to succeed through the process of clinical development [30]. However, understanding the influence of genetic variation on disease pathophysiology towards the development of effective therapeutics is complex. GWAS often reveal variants with small effect sizes that do not account for much of the risk of a disease [32]. On the one hand, widespread gene pleiotropy (a gene affecting several unrelated phenotypes) and polygenic traits (a single trait affected by several genes) reveal the highly interconnected nature of biomolecular networks [26, 7].

Instead of looking at single gene-disease associations, methods that consider groups of genes that are functionally related (i.e., that belong to the same pathways) can be more robust to identify putative mechanisms that influence disease, and also provide alternative treatment avenues when directly associated genes are not druggable [23, 11]. Large gene expression compendia such as recount2 [5] or ARCHS4 [20] provide unified resources with publicly available RNA-seq data on tens of thousands of samples. Leveraging this massive amount of data, unsupervised network-based learning approaches [22, 33, 10] can detect meaningful gene co-expression patterns: sets of genes whose expression is consistently modulated across the same tissues or cell types. However, this is particularly challenging because the observed data is an aggregated and noisy projection of a highly complex transcriptional machinery: co-expressed genes can be controlled by the same regulatory program or module, but single genes can also play different roles in different modules expressed in distinct tissues or cell differentiation stages [2, 37]. For example, Marfan syndrome (MFS) is a rare genetic disorder caused by a mutation in gene *FBN1*, which encodes a protein that forms elastic and nonelastic connective tissue [29]. However, MFS is characterized by abnormalities in bones, joints, eyes, heart, and blood vessels, suggesting that *FBN1* is implicated in independent pathways across different tissues or cell types. In other words, the membership of genes in modules and the relative strength of effect a module has on the co-expression of its constituents are not directly observable from gene expression data.

3 Problem Modeling

We begin by observing that gene-module membership is naturally represented by a bipartite graph B, where each gene has an edge to all modules it participates in. Further, in order to capture the notion of varied effect-strength among modules, we associate a non-

negative real-valued weight w_i to each module c_i, since we are interested in sets of co-expressed genes. In other words, we assume that all pairs of genes that are common to module i will be co-expressed with strength w_i; thus, the genes in each module will form a clique in the co-expression network. Further, we assume that modules interact with one another in a linear, additive manner. That is, the co-expression between genes u and v is the sum of the weights of all modules containing both u and v. In a noise-free setting, this means that the gene-gene co-expression network is exactly a union of (potentially overlapping) cliques $m_1, \dots m_k$ with associated weights $w_1, \dots w_k$ so that the weight on uv is exactly $\sum_{\{u,v\} \subseteq m_i} w_i$. It is important to note that not all valid solutions are interesting; specifically, one can always assign each pair of genes to its own clique of size 2, and get a valid solution. We rely on the principal of parsimony, and try to find an assignment which minimizes the number of modules in a valid solution. Realistically, the edge-weights will not satisfy exact equality, and we will need to consider an optimization variant of our problem which minimizes an objective function incorporating penalties for over/under-estimating the observed co-expressions.

To this end, we introduce a penalty function ϕ on the edges based on the discrepancy between the weight predicted by clique (module) membership and the original weight (observed co-expression value), then minimize ϕ to determine an optimal solution. For example, a natural choice for ϕ might be the sum of the absolute value of the discrepancies on each edge. Formally, this leads to the following problem:

WEIGHTED CLIQUE DECOMPOSITION

Input: a graph $G = (V, E)$, a non-negative weight function w_e on E, a penalty function ϕ, and a positive integer k.

Output: a set of at most k cliques $C_1, \dots C_k$ with weights $\gamma_1, \dots \gamma_k \in \mathbb{R}^+$ that define $\gamma_{uv} = \sum_{i:uv \in C_i} \gamma_i$ for all $uv \in E$, such that $\phi(\{(w_e, \gamma_e) : e \in E\})$ is minimized.

In the remainder of this paper, we restrict our attention to the setting where equality can be achieved (as one might expect in synthetic data); further discussion of ideas for addressing the optimization variant is deferred to the future work section. For convenience, we define a decision version of WCD for this setting (this is equivalent to having a penalty function which is zero for matching the weight on an edge and infinite for any discrepancy):

EXACT WEIGHTED CLIQUE DECOMPOSITION

Input: a graph $G = (V, E)$, a non-negative weight function w_e for $e \in E$, and a positive integer k.

Output: a set of at most k cliques $C_1, \dots C_k$ with weights $\gamma_1, \dots \gamma_k \in \mathbb{R}^+$ such that $w_{uv} = \sum_{i:uv \in C_i} \gamma_i$ for all $uv \in E$ (if one exists, otherwise output NO).

If the clique weights are constrained to be integers then the problem becomes a generalization of the NP-hard problem EDGE CLIQUE PARTITION [21, 14]. The NP-hardness of the fractional-clique-weight version also follows easily from the reduction in [21]. For completeness, we give the proof in Appendix F.

3.1 Annotated and Matrix Formulations We will work with the following more general version of EWCD in our algorithms, where some of the vertices are annotated with vertex weights.

ANNOTATED EWCD

Input: a graph $G = (V, E)$, a non-negative weight function w_e for $e \in E$, a special set of vertices $S \subseteq V$, a non-negative weight function w_v for $v \in S$, and a positive integer k.

Output: a set of at most k cliques $C_1, \dots C_k$ with weights $\gamma_1, \dots \gamma_k \in \mathbb{R}^+$ such that $w_{uv} = \sum_{i:uv \in C_i} \gamma_i$ for all $uv \in E$ and $w_v = \sum_{i:v \in C_i} \gamma_i$ for all $v \in S$ (if one exists, otherwise output NO).

Note that EWCD is the special case of AEWCD when the set $S = \varnothing$.

We also introduce an equivalent matrix formulation of AEWCD, as our techniques are heavily based on linear algebraic properties. For this we use matrices that allow wildcard entries denoted by $\star$. For $a, b \in \mathbb{R} \cup \{\star\}$, we say $a \overset{\star}{=} b$ if either $a = b$ or $a = \star$ or $b = \star$. For matrices A and B, we say $A \overset{\star}{=} B$ if $A_{ij} \overset{\star}{=} B_{ij}$ for each i, j. We call the matrix problem as BINARY SYMMETRIC WEIGHTED DECOMPOSITION WITH DIAGONAL WILDCARDS. Note that EWCD is the special case where all the diagonal entries are wildcards.

BSWD-DW

Input: a symmetric matrix $A \in \left(\mathbb{R}_0^+ \cup \{\star\}\right)^{n \times n}$ with wildcards appearing on a subset of diagonal entries, and a positive integer k

Output: a matrix $B \in \{0,1\}^{n \times k}$ and a diagonal matrix $W \in (\mathbb{R}_0^+)^{k \times k}$ such that $A \stackrel{\star}{=} BWB^T$. (if such (B,W) exist, otherwise output NO).

4 Parameterized Algorithms

Parameterized algorithms are a method used to tackle NP-hard problems where, besides the input size n, we are given an additional parameter k, most often representing the solution size. E.g., in our problem WCD, the parameter k is the number of cliques. An algorithm is said to be *fixed parameter tractable* if the runtime is polynomial in the input size and exponential only in the parameter—often resulting in tractable algorithms when the parameter is much smaller compared to the input size. One of the most effective tools in parameterized algorithms is *kernelization*, which is essentially a preprocessing framework that reduces the input to an equivalent instance of the same problem whose size depends only on the parameter k. The reduced instance is called a *kernel*. Sometimes, the reduction is not to the same problem itself but to a different related problem, in which case it is called a *compression*. For an extensive introduction to the topic, we refer to the book by Cygan et al. [9].

5 Prior Work

The WCD problem with integer clique weights is a generalization of the Weighted Edge Clique Partition problem which in turn generalizes Edge Clique Partition [21]):

Weighted Edge Clique Partition

Input: a graph $G = (V, E)$, a weight function $w_e : E \to Z^+$ and a positive integer k.

Output: a set of at most k cliques such that each edge appears in exactly as many cliques as its weight (if it exists, otherwise output NO).

Weighted Edge Clique Partition (WECP) was introduced by Feldmann et al. [13] last year. They gave a 4^k-compression and a $2^{\mathcal{O}(k^{3/2}w^{1/2}\log(k/w))} + \mathcal{O}(n^2 \log n)$ time algorithm for WECP, where w is the maximum edge weight. The compression is into a more general problem called Annotated Weighted Edge Clique Partition (AWECP) where some vertices also have input weights and these vertices are constrained to be in as many cliques as its weight in the output. The authors worked with an equivalent matrix formulation for AWECP called Binary Symmetric Decomposition with Diagonal Wildcards (BSD-DW) where given a $n \times n$ symmetric matrix A with wildcards (denoted by $\star$) in the diagonal, the task is to find a $n \times k$ binary matrix B such that $BB^T \stackrel{\star}{=} A$ where $\stackrel{\star}{=}$ denotes that the wildcards are considered equal to any number. The algorithm of Feldmann et al. [13] builds upon the linear algebraic techniques used by Chandran et al. [3] for solving the Biclique Partition problem. Our algorithms further build upon the techniques of [13]. Note that one could encode the clique weights (in the integer weight case) into the WECP problem by thinking of a clique of weight w as w identical unweighted cliques. This makes the parameter k equal to the sum of clique weights, and hence the algorithms of Feldmann et al. [13] are not sufficient for our application.

The unit-weighted case of WECP called Edge Clique Partition (ECP) has been more well studied, especially from the parameterized point of view. It is known that ECP admits a k^2-kernel in polynomial time [28]. The fastest FPT algorithm for ECP is the algorithm by Feldmann et al. [13] which runs in $2^{\mathcal{O}(k^{3/2}\log k)} + \mathcal{O}(n^2 \log n)$ for ECP. There are faster algorithms for ECP in special graph classes, for instance a $2^{\mathcal{O}(\sqrt{k})}n^{\mathcal{O}(1)}$ time algorithm for planar graphs, $2^{dk}n^{\mathcal{O}(1)}$ time algorithm for graphs with degeneracy d, and a $2^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ time algorithm for K_4-free graphs [14]. A closely related problem to ECP is the Edge Clique Cover problem. Here, each edge should be present in at least one clique but can be present in any number of cliques. This unrestricted covering version is much harder and is known to *not* admit algorithms running faster than $2^{2^{o(k)}}n^{\mathcal{O}(1)}$ [8].

There are a few papers that study symmetric matrix factorization problems that are similar to the Binary Symmetric Decomposition with diagonal Wildcards (BSD-DW) problem, defined by Feldmann et al. [13]. Recall that BSD-DW is equivalent to the AWECP problem. Zhang et al. [39] studied the objective of minimizing $\|A - BB^T\|_2^2$. Their matrix model does not translate into the clique model as they do not have wildcards in the diagonal. Chen et al. [4] studied the objective of minimizing $\|A - BB^T\|_0$, but also without wildcards. A matrix model that has diagonal wildcards was considered by Moutier et al. [27] under the name Off-Diagonal Symmetric Non-negative Matrix Factorization, but they allow B to be any

non-negative matrix and not just binary.

The non-symmetric variants of these matrix problems known as Binary Matrix Factorization, have been receiving a lot of attention recently [25, 15, 16, 1, 19, 3]. Here the objective is to minimize $\|A - BC\|$, where A is an $m \times n$ input matrix, B is an $m \times k$ output binary matrix and C is a $k \times n$ output binary matrix. For example, a constant approximation algorithm running in $2^{\mathcal{O}(k^2 \log k)}(mn)^{\mathcal{O}(1)}$ is known [19]. In the graph-world the non-symmetric problems correspond to finding a partition of the edges of a bipartite graph into bicliques (complete bipartite graphs) instead of cliques [3].

6 Algorithms

We give two algorithms for BSWD-DW and hence also for the equivalent AEWCD and the special case EWCD. Both algorithms will have a common framework similar to that of Feldmann et al. [13]. The algorithms will differ in the method in which the clique weights (represented by the diagonal matrix W) are inferred. One uses an LP based method while the other uses an integer partition dynamic program.

The first step in our pipeline is to preprocess disjoint cliques and cliques that overlap only on single vertices (and thus have no overlapping edges) out of each graph. The specifics of this process are outlined in Appendix G, but it essentially runs a modified breadth-first search algorithm. Similar to Feldmann et al. [13], the second step in our algorithms is to give a kernel. The kernel follows the same reduction rules as in Feldmann et al. [13] i.e. by reducing blocks of twin vertices. The proof of correctness follows analogously, and we omit it here due to space constraints. After the kernelization, we can assume that the number of vertices of G (equivalently the number of rows of matrix A) is at most 4^k.

Theorem 6.1. *AEWCD* (BSWD-DW *resp.*) *has a kernel with at most* 4^k *vertices* (4^k *rows resp.*) *that can be found in* $\mathcal{O}(n^3)$ *time.*

The third step is to run a clique decomposition algorithm on the kernelized AEWCD instance to obtain the clique assignments for each vertex and clique weights. Let A be the input instance for BSWD-DW and let G be the corresponding input instance to AEWCD. Both our algorithms use the basis-guessing principle used by Feldmann et al. [13], first introduced by Chandran et al. [3]. The principle is that once we correctly guess the entries of a row-basis of B, then the remaining rows of B can be filled iteratively without backtracking. However, the technique does not carry over directly to the clique-weighted problem we have here. We additionally need to infer the clique-weight matrix W, which poses some additional challenges. Note that it is not feasible to guess the entries of the diagonals of W as each entry could be as large as the largest element in A. So once we have a guess for the basis, we also need to infer compatible values of W. Since there could be multiple choices for compatible W, we are not guaranteed to hit the correct solution for W, likely producing some backtracking while filling the non-basis rows. We tackle this by showing that if we guess a row-basis plus an additional k rows (thus at most $2k$ rows) then the choice of W does not matter. If our guess for this $2k$ rows (we call it the *pseudo-basis* of B) is correct, then we show that we can fill the other rows iteratively without any backtracking. The intuition of why we need the additional k rows is as follows: in the version without the W matrix, once we fix the basis $\widetilde{B}$ fo B, the matrix $\widetilde{A}$ given by the corresponding rows of A is fixed. In particular the diagonal values $\widetilde{A}_{ii}$ (that could have been wildcards and hence not fixed apriori) are now fixed. But with the matrix W, for different choices of W, we get different diagonal entries in $\widetilde{A}$. We only need to add at most one more row to the pseudo-basis in order to fix one of these diagonal entries. Thus we need at most k additional rows in the pseudo-basis.

We guess the rows of the pseudo-basis on-demand i.e., we add a row as basis-row only if a compatible row cannot be found for it under the current inferred clique weights W from the current basis matrix. Every time we add a new row to the basis, we recompute the clique weights W. The two algorithms that we present, differ in how they infer the clique weights for the current pseudo-basis. The first algorithm uses a linear programming method while the second uses an integer partitioning dynamic programming method. In our algorithms, we will often use partially filled matrices, i.e. some of the entries are allowed to have *null* values. If a row or matrix has all null values we call them null row and null matrix respectively.

6.1 Clique Weight Recovery by Linear Programming We use a linear program to infer the clique weights for the current pseudo-basis of the algorithm. The pseudocode is given in `InferCliqWts-LP` (Algorithm 2). Suppose $\widetilde{B}$ is the current pseudo-basis matrix i.e. $\widetilde{B}$ is an $n \times k$ matrix where the current pseudo-basis rows (at most $2k$) are filled by 0's and 1′s, and the other rows are null rows. For each pair of distinct non-null rows $\widetilde{B}_i$ and $\widetilde{B}_j$ we add the constraint $\widetilde{B}_i^T W \widetilde{B}_j = A_{ij}$ to the LP. Also, for each A_{ii} that is not a $\star$, we add the constraint $\widetilde{B}_i^T W \widetilde{B}_i = A_{ii}$. Note that the variables of the LP are the diagonal entries $W_{11}, W_{22}, \cdots, W_{kk}$. We also have non-negativity constraints $W_{11} \geq 0, \cdots, W_{kk} \geq 0$. Any feasible solution to this LP gives us a set of clique weights compatible with the current pseudo-basis. If

the LP is infeasible, then we conclude that the current pseudo-basis guess is infeasible and proceed to the next guess. Note that since we are only concerned about a feasible solution satisfying the constraints, we do not have an objective function for the LP. We point out that solving this LP is rather efficient as the number of variables are k and number of constraints are at most $4k^2$ and can be solved incrementally as we add constraints everytime a basis row is added.

ALGORITHM 1. `CliqueDecomp-LP`

1: **for** $P \in \{0,1\}^{2k\times k}$ **do**
2: initialize $\widetilde{B}$ to a $n \times k$ null matrix
3: $b, i \leftarrow 1$
4: **while** $b \leq 2k$ **do**
5: $\widetilde{B}_i \leftarrow P_b$
6: $b \leftarrow b+1$
7: $W \leftarrow$ `InferCliqWts-LP` $(A, \widetilde{B})$
8: **if** W is not null matrix **then**
9: $(B, i) \leftarrow$ `FillNonBasis` $(A, \widetilde{B}, W)$
10: **if** $i = n+1$ **then return** (B, W)
11: **else** $b \leftarrow 2k+1$ ▷ *null W; break out of while*
12: **return** No

ALGORITHM 2. `InferCliqWts-LP` $(A, \widetilde{B})$

1: **let** $\gamma_1, \cdots, \gamma_k \geq 0$ be variables of the LP
2: **for** all pairs of non-null rows $\widetilde{B}_i, \widetilde{B}_j$ s.t. $A_{ij} \neq \star$ **do**
3: Add LP constraint $\sum_{1\leq q\leq k} \widetilde{B}_{iq}\widetilde{B}_{jq}\gamma_q = A_{ij}$
4: **if** the LP is infeasible **then return** the null matrix
5: **else return** the diagonal matrix given by $\gamma_1, \ldots, \gamma_k$

ALGORITHM 3. `FillNonBasis` $(A, \widetilde{B}, W)$

1: $B \leftarrow \widetilde{B}$
2: **while** B has a null row **do**
3: **let** B_i be the first null row
4: **for** $v \in \{0,1\}^k$ **do**
5: **if** `iWCompatible` (A, B, W, i, v) **then**
6: $B_i \leftarrow v$
7: **goto** line 2
8: **return** (B, i) ▷ *there is no (i,W)-compatible v*
9: **return** $(B, n+1)$ ▷ *B has no null row*

Once we have inferred a W compatible with the current pseudo-basis, we then try to fill the remaining rows (we call them non-basis rows) one by one in `FillNonBasis`. We say that a vector $v \in \{0,1\}^k$ is (i,W) compatible with row B_j if $v^T W B_j = A_{ij}$. We say that v is (i,W) compatible with matrix B if it

ALGORITHM 4. `iWCompatible` $(A, \widetilde{B}, W, i, v)$

1: **for** each non-null row B_j **do**
2: **if** $v^T W B_j \neq A_{ij}$ **then return** false
3: **if** $v^T W v \not\stackrel{\star}{=} A_{ii}$ **then return** false
4: **return** true

is (i,W)-compatible with each non-null row B_j, and $v^T W v \stackrel{\star}{=} A_{ii}$. We say B and W are compatible with each other if for each pair of non-null rows B_i and B_j, we have $B_i^T W B_j \stackrel{\star}{=} A_{ij}$. We keep filling the rows B_i of B one-by-one with (i,W)-compatible rows until either B is completely filled or there is an i such that there is no (i,W)-compatible vector in $\{0,1\}^k$. In the former case we show that (B,W) gives a solution, and in the latter case we proceed on to take row i into the pseudo-basis row. Note that when we take a new row into the pseudo-basis row we throw away all the non-basis rows and make them null rows again. We will show that we only need to take up to $2k$ rows into the pseudo-basis for the algorithm to correctly find a solution.

6.1.1 Algorithm Correctness

THEOREM 6.2. *`CliqueDecomp-LP` (Algorithm 1) correctly solves the BSWD-DW problem, and hence also correctly solves AEWCD and EWCD, in time $\mathcal{O}(4^{k^2}k^2(32^k + k^3L))$, where L is the number of bits required for input representation.*

First we prove in the following lemma that if `CliqueDecomp-LP` outputs Yes, i.e. if it outputs through line 10, then the matrices B and W output indeed satisfy that $A \stackrel{\star}{=} BWB^T$. The proof follows because we checked for (i,W)-compatibility whenever we filled B_i. The full proof of the Lemma can be found in Appendix B.1.

LEMMA 6.1. *If `CliqueDecomp-LP` returns through line 10, then the matrices B and W output satisfy that $A \stackrel{\star}{=} BWB^T$.*

Lemma 6.1 immediately implies that if the instance is a No-instance then the algorithm does not output through line 10. Since the only other possibility for output is through Line 12, which outputs No, we can conclude that for a No-instance we correctly output No. The following arguments are therefore related to the correctness of Yes instances.

For arguing the correctness in the Yes case, we fix a valid solution (B^*, W^*) of the instance. If the output occurs through Line 10, then by Lemma 6.1, we are done. So for the sake of contradiction assume that the output does not occur through Line 10. For $I \subseteq [n]$, we

define B_I^* as the $n \times k$ matrix whose i-th row is equal to B_i^* for all $i \in I$ and the other rows are null rows. The following lemma follows because we iterate over all possible values of pattern matrix P.

LEMMA 6.2. *Let $I \subseteq [n]$ be such that $|I| \leq 2k-1$. If $\widetilde{B}$ is equal to B_I^* at some point in the algorithm, and if* `FillNonBasis` *$(A, \widetilde{B} = B_I^*, W)$ called in Line 9 returns $i \leq n$, then $\widetilde{B}$ is equal to $B_{I\cup\{i\}}^*$ at some point in the algorithm.*

So, if we start with $I = \varnothing$, and repeatedly apply Lemma 6.2, then at some point in the algorithm, we have $\widetilde{B} = B_I^*$ such that $|I| = 2k$. Towards this, we define the matrix $E(\widetilde{B})$ formed by the rows that are element-wise products of pairs of non-null rows in $\widetilde{B}$. More precisely:

DEFINITION 6.1. *$E(\widetilde{B})$ is the matrix containing rows $\widetilde{B}_i \odot \widetilde{B}_j$ for each pair i, j (not necessarily distinct) such that $A_{ij} \neq \star$. Here $\odot$ denotes element-wise product.*

The above definition means that $E(\widetilde{B})$ is the coefficient matrix of the LP that the algorithm would construct in the call `InferCliqWts-LP` $(A, \widetilde{B})$.

DEFINITION 6.2. (PSEUDO-RANK) *The pseudo-rank of $\widetilde{B}$ is defined as the sum of ranks of $\widetilde{B}$ and $E(\widetilde{B})$, where by rank of $\widetilde{B}$ we mean the rank of the matrix formed by the non-null rows of $\widetilde{B}$.*

Since the number of columns in $\widetilde{B}$ and $E(\widetilde{B})$ are each k, we have the following lemma.

LEMMA 6.3. *The pseudo-rank of $\widetilde{B}$ is at most $2k$.*

We say that a vector $v \in \{0,1\}^k$ *i-extends* $\widetilde{B}$ if $\widetilde{B}_i$ is currently a null row, and adding v as $\widetilde{B}_i$ increases the pseudo-rank of $\widetilde{B}$.

LEMMA 6.4. *If* `FillNonBasis` *$(A, \widetilde{B} = B_I^*, W)$ called on Line 9 returns $i \leq n$, then B_i^* i-extends B_I^*.*

Proof. Suppose for the sake of contradiction that B_i^* does not i-extend B_I^*. This means that B_i^* is linearly dependent on the non-null rows of B_I^* and each $B_i^* \odot B_j^*$ for $j \in I$ is linearly dependent on the rows of $E(\widetilde{B})$. Also, if $A_{ii} \neq \star$ then $B_i^* \odot B_i^*$ is linearly dependent on the rows of $E(\widetilde{B})$. Now, consider each non-null row B_j of the matrix B when `iWCompatible` (A, B, W, i, v) was called in Line 5 in `FillNonBasis`. We prove that $B_i^{*T} W B_j = A_{ij}$ and that $B_i^{*T} W B_i^* \stackrel{\star}{=} A_{ii}$. This then implies that B_i^* is (i, W)-compatible with B and hence `FillNonBasis` could not have returned i, giving a contradiction.

First consider the case when B_j is a pseudo-basis row, i.e. $j \in I$. Since B_i^* does not i-extend B_I^*, we know $B_i^* \odot B_j^*$ is linearly dependent on the rows of $E(\widetilde{B})$. This means that adding B_i^* as $\widetilde{B}_i$ would not add any linearly independent equality constraints to the LP system that solves for W. So, either the LP becomes infeasible or all the solutions to the LP still remain solutions. But the LP is not infeasible as the diagonal elements of W^* gives a feasible solution to the LP. Thus, the current W remains a feasible solution even after the addition of B_i^* the row $\widetilde{B}_i$. Hence, $B_i^{*T} W B_j^* = A_{ij}$.

Now, consider the case when B_j is not a pseudo-basis row, i.e. $j \notin I$. In other words $\widetilde{B}_j$ is a null row and B_j was added in `FillNonBasis`. Since B_i^* does not i-extend B_I^*, we know that B_i^* is linearly dependent on the non-null rows of B_I^*. In other words, $B_i^* = \sum_{\ell \in I} \lambda_\ell B_\ell^*$ where each $\lambda_\ell \in \mathbb{R}$. Then,

$$B_i^{*T} W B_j = \sum_{\ell \in I} \lambda_\ell B_\ell^{*T} W B_j \tag{6.1}$$

$$= \sum_{\ell \in I} \lambda_\ell A_{\ell j} \tag{6.2}$$

$$= A_{ij} \tag{6.3}$$

where Eq. (6.2) is because B_j could have been selected for row j only if it was (j, W)-compatible with B_I^*, and Eq. (6.3) follows by using that $W^* B^*$ is a linear map from B^* to A and hence the linear dependencies in B^* are preserved in A. More precisely,

$$\begin{aligned} \sum_{\ell \in I} \lambda_\ell A_{\ell j} &= \sum_{\ell \in I} \lambda_\ell B_\ell^{*T} W^* B_j^* \\ &= B_i^* W^* B_j^* \\ &= A_{ij} \end{aligned}$$

Note that we have here crucially used $\ell \neq j$ (as $j \notin I$) and $j \neq i$ to say $=$ and not just $\stackrel{\star}{=}$. This is the reason we required a separate argument for $j \in I$. The argument for $B_i^{*T} W B_i^* \stackrel{\star}{=} A_{ii}$ follows the same argument as in the case of $j \in I$ by observing that if $A_{ii} \neq \star$ then $B_i^* \odot B_i^*$ is linearly dependent on the rows of $E(\widetilde{B})$. ∎

Now starting with $I = \varnothing$, and applying Lemmas 6.2 and 6.4 repeatedly, we have that at some point in the algorithm, $\widetilde{B}$ is equal to some B_I^* such that the pseudo-rank of B_I^* is $2k$. At this point, the `FillNonBasis` call at Line 9 should return $i = n+1$ because if it returned $i \leq n$, then adding B_i^* to $\widetilde{B}$ would make the pseudo-rank of $\widetilde{B}$ equal to $2k+1$, a contradiction to Lemma 6.3. Hence, the algorithm outputs through Line 10. This concludes the correctness of the algorithm. We defer the runtime analysis to Appendix C.1.

6.2 Clique Weight Recovery by Integer Partitioning We give an algorithm for inferring the current pseudo-basis's clique weights by solving an integer partitioning dynamic program. The pseudocode is given in `InferCliqWts-IP` (Algorithm 6). Similar to 6.1, consider $\widetilde{B}$, the current pseudo-basis matrix. Additionally, a list $\mathbf{W}$ is maintained, containing partially filled diagonal weight matrices each of which are *compatible* with the current pseudo-basis matrix $\widetilde{B}$. Here compatibility is defined as follows. For a matrix B, first define its *relevant indices*, denoted by $R(B)$, as the set of all $r \in [k]$ such that there exist non-null rows B_i, B_j such that $B_{ir}B_{jr} = 1$ and $A_{ij} \neq \star$. For a diagonal matrix W we define $F(W)$ as the set of all $i \in [k]$ such that W_{ii} is not null. We say that $\widetilde{B}$ and W are compatible if $F(W) = R(\widetilde{B})$ and for each pair of non-null rows $\widetilde{B}_i, \widetilde{B}_j$ such that $A_{ij} \neq \star$, it is true that $\sum_{r \in R(\widetilde{B})} \widetilde{B}_{ir} W_{rr} \widetilde{B}_{jr} = A_{ij}$. We maintain in $\mathbf{W}$, all the possible fillings of indices $R(B)$ of the diagonal vector of clique-weight matrix W such that W is compatible with $\widetilde{B}$.

`InferCliqWts-IP` is given as input the current $\widetilde{B}$ after initially inserting P_b at Line 6 in `CliqueDecomp-IP`. Thus, $\widetilde{B}_i$ is the potential basis row we are considering. At the start of the function call, we know that each non-null rows $\widetilde{B}_j$ is (j, W)-compatible with each non-null row $\widetilde{B}_{j'}$ for $j, j' \neq i$. If $\widetilde{B}_i$ is not (i, W)-compatible, this implies that either there are null weights in W in relevant positions, or the current basis P being considered is not the correct one. Let $\widetilde{B}_j$ be a non-null row such that $\widetilde{B}_i$ is not (i, W)-compatible with $\widetilde{B}_j$. We define $X \subseteq [k]$ as the indices of null positions in the diagonal of W, $\overline{\mathbf{X}} = [k] \setminus X$, and $P \subseteq [k]$ as the set of positions in $\widetilde{B}_i \odot \widetilde{B}_j$ having 1 values. The sum $t = \sum_{l \in P \cap \overline{\mathbf{X}}}^{k} \widetilde{B}_{il} \widetilde{B}_{jl} W_{ll}$ is the sum of all previously fixed clique-weights that contribute to A_{ij}. The difference $s = t - A_{ij}$ has to be contributed by $P \cap X$. The `UpdateWs` function finds all possible ways to sum to s using $|P \cap X|$ number of non-negative integers via a dynamic program. This is an integer partitioning problem and is a simple variant of the common change-making problem. Then for each such combination a new W-matrix is created by inserting the combination in the indices $P \cap X$. `UpdateWs` returns the list of all such W-matrices created. Note that s could be negative in which case `UpdateWs` returns an empty list.

6.2.1 Algorithm Correctness

THEOREM 6.3. *`CliqueDecomp-IP` (Algorithm 5) correctly solves the BSWD-DW problem, and hence also correctly solves AEWCD and EWCD, in time $\mathcal{O}(4^{k^2} 32^k w^k k)$ where w is the maximum entry of A.*

ALGORITHM 5. `CliqueDecomp-IP`

1: **for** $P \in \{0,1\}^{2k \times k}$ **do**
2: $\quad \widetilde{B} \leftarrow n \times k$ null matrix
3: $\quad b, i \leftarrow 1$
4: $\quad \mathbf{W} \leftarrow \{\text{null matrix}\}$
5: $\quad$ **while** $b \leq 2k$ **do**
6: $\quad\quad \widetilde{B}_i \leftarrow P_b$
7: $\quad\quad b \leftarrow b + 1$
8: $\quad\quad \mathbf{S} \leftarrow$ `InferCliqWts-IP` $(A, \widetilde{B}, \mathbf{W}, i)$
9: $\quad\quad$ **if** $\mathbf{S}$ is not empty **then**
10: $\quad\quad\quad \mathbf{W} \leftarrow \mathbf{S}$
11: $\quad\quad\quad (B, i) \leftarrow$ `FillNonBasis` $(\text{A}, \widetilde{B}, W[0])$
12: $\quad\quad\quad$ **if** $i = n + 1$ **then return** $(B, W[0])$
13: $\quad\quad$ **else**
14: $\quad\quad\quad b \leftarrow 2k + 1$
15: **return** No

ALGORITHM 6. `InferCliqWts-IP` $(A, \widetilde{B}, \mathbf{W}, i)$

1: $\mathbf{S} \leftarrow []$
2: **for** $l \leftarrow 1$ to $|\mathbf{W}|$ **do**
3: $\quad \mathbf{X} \leftarrow \{x \in [k] \mid W[l]_{xx} \text{ is null}\}$
4: $\quad \overline{\mathbf{X}} \leftarrow [k] \setminus X$
5: $\quad \mathbf{temp} \leftarrow$ empty queue
6: $\quad \mathbf{temp}$.push($\mathbf{W}[l]$)
7: $\quad$ **for** each non-null row $\widetilde{B}_j$ s.t. $A_{ij} \neq \star$ **do**
8: $\quad\quad iters \leftarrow |\mathbf{temp}|$
9: $\quad\quad \mathbf{P} \leftarrow \{p \in [k] \mid \widetilde{B}_{ip} \widetilde{B}_{jp} = 1\}$
10: $\quad\quad$ **for** $q \leftarrow 1$ to $iters$ **do**
11: $\quad\quad\quad S' \leftarrow \mathbf{temp}$.pop()
12: $\quad\quad\quad s \leftarrow A_{ij} - \sum_{f \in \mathbf{P} \cap \overline{\mathbf{X}}} S'_{ff}$
13: $\quad\quad\quad \mathbf{T} \leftarrow$ `UpdateWs` $(S', \mathbf{P} \cap \mathbf{X}, s)$
14: $\quad\quad\quad \mathbf{temp}$.push($\mathbf{T}$)
15: $\quad\quad\quad$ **if temp** is empty **then** goto Line 16
16: $\quad \mathbf{S}$.push($\mathbf{temp}$)
17: **return S**

ALGORITHM 7. `UpdateWs` $(W, \mathbf{I}, s)$

1: $\mathbf{V} \leftarrow [\,]$
2: $\mathbf{Y} \leftarrow$ all partitions of s into $|I|$ non-negative integers
3: **for** each *parts* in $\mathbf{Y}$ **do**
4: $\quad y \leftarrow 0$
5: $\quad C \leftarrow W$
6: $\quad$ **for** each $i \in \mathbf{I}$ **do**
7: $\quad\quad C_{ii} \leftarrow parts[y]$
8: $\quad\quad y \leftarrow y + 1$
9: $\quad \mathbf{V}$.push(C)
10: **return V**

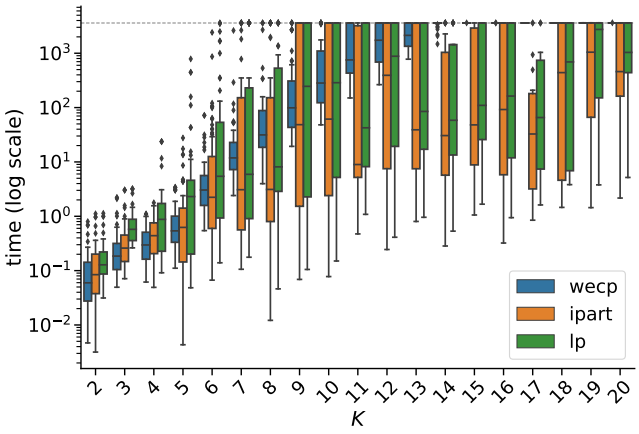


Figure 2: Log-scale plot showing distribution of total algorithm runtimes when binned by K (the sum of the clique weights). All K values shown in Figure 8 in Appendix E.

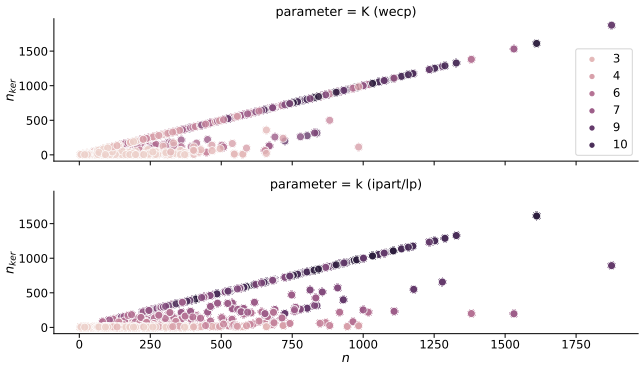


Figure 3: Instance size reduction due to kernelization (from n to n_{ker}); points along the diagonal experienced no reduction from kernel rules. Top shows reduction using parameter K, bottom shows reduction when using parameter k.

The proof of `CliqueDecomp-IP`'s correctness is similar to the proof of `CliqueDecomp-LP` in Section 6.1.1. We describe the differences in Appendix B.2 and defer the runtime analysis to Appendix C.2.

7 Experimental Setup

This section describes the synthetic corpora; hardware descriptions can be found in Appendix E.1. We generate two sets of biologically-inspired synthetic graphs. The first dataset defines modules (cliques) using known relationships between transcription factors and genes; the second uses latent variables from a machine learning approach for analyzing co-expression data.

7.1 TF-Dataset Our first dataset emulates the bipartite gene-module network by using known relationships between transcription factors (TFs) and genes [12]. To generate a network with a ground truth of k cliques, we randomly select k TFs and form the network which is the union of all associated genes with edges between those that share at least one selected TF.

Since the relative strengths of effect on expression are unknown, we specify a desired maximum edge weight (see Appendix D.1), and generate integral clique weights as described in Appendix D.2. A heavy-tailed distribution is chosen to mimic the view that modules have widely varying effects on gene co-expression, and a small minority likely have drastically higher impact than all others [33].

7.2 LV-Dataset A similar approach is taken when generating the set of the latent variable-associated synthetic graphs. In this data [18, 33], each latent variable (LV) represents a set of genes that are co-expressed in the same cell types. A score for every gene in each LV indicates the strength of its association to the module. Further, some latent variables have been shown to align with prior knowledge of pathway associations [33]. Our generator randomly selects k latent variables, with 80% drawn from those known to be aligned with pathways, and the remaining 20% chosen uniformly from all LVs. For each LV, we only include genes with association scores above a threshold, determined as described in Appendix D.3.

In contrast to the TF data, here the clique weights have a basis in the underlying data. For each LV, we compute the average associate score over all included genes then linearly transform this to control the maximum edge weight in the network (see Appendix D.1).

8 Results

This section highlights the key outcomes of our preliminary experimental evaluation. We begin by highlighting the effects of reparameterization, comparing the algorithm of [13] (referred to as `wecp`) to `CliqueDecomp-IP` and `CliqueDecomp-LP` (shortened to `ipart` and `lp` for consistency with figure legends).

8.1 Effects of Reparameterization To compare the effects on the runtime of reparameterizing from the sum of the clique weights K to the number of distinct cliques k, we tested `wecp`, `ipart`, and `lp` on all instances with $k \leq 11$ and small/medium weight scalings. As seen in Figure 2, and Figures 5, 6, 8 in Appendix E, both algorithms parameterized by k are faster across the entire corpus when $k >= 6$. It should be noted that the slower `ipart` and `lp` runtimes for $k < 6$ are partly

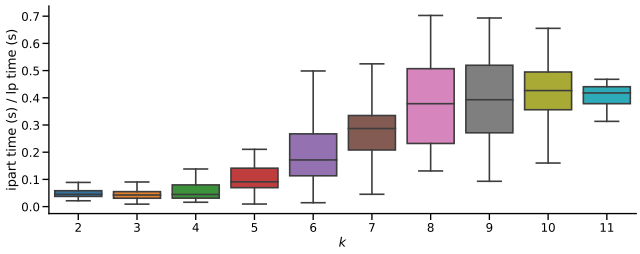


Figure 4: Relative runtimes of `ipart` and `lp` on full corpus with $2 \leq k \leq 11$ and all weight scalings; times exclude kernel (which is shared). Outliers not shown.

due to the preprocessing often removing highly weighted cliques, effectively setting $k = K$. When $k = K$, it is slightly faster to run `wecp` due to not having to compute the clique weights. Figure 2 shows the runtimes for $K \in [2, 20]$, and Figure 8 shows all $K \in [2, 49]$. After $K = 13$ `wecp` timed out for every instance, whereas `ipart` and `lp` are still able to compute solutions in under an hour.

Some performance increase may be attributed to the kernelization, as shown in Figure 3. Since one of the reduction rules relies on the parameter, the instance size after kernelization (denoted n_{ker}) is different between `wecp` (which uses K) shown in the top figure, and `ipart`/`lp` (which use k) shown in the bottom figure. Figure 3 shows that the kernel gives great reduction when parameterized by k when k is small and has less of an impact when k is large. Figure 6 in Appendix E shows a comparison of the runtimes when instances are sorted by the size of the kernelized instance.

8.2 Integer Partitioning vs LP From Figure 2, we observe that `ipart` runs slightly faster on average than `lp`, which was unexpected. To further compare the two, we ran both algorithms on a larger corpus including all instances with $k \leq 11$ regardless of weight scaling. Figure 4 shows the runtime ratio of the two algorithms[2]. We observe that despite a consistent advantage for `ipart` on small k, the methods' runtimes seem to converge as we approach $k = 9$ and we hypothesize that `lp` will become the dominant approach for larger k when testing with a longer timeout than one hour. To test whether the specific clique-weight assignment mattered, we evaluated runtimes on 6 random weight assignment permutations for each instance. Table 4 in Appendix E.2 shows that both algorithms are virtually unaffected by the assignments.

Finally, since `ipart`'s complexity depends on the maximum weight w, we evaluated this effect by comparing performance across the small, medium, and large weight-scale variants of each instance. Figure 10 in Appendix E shows that the relative increase in runtime between weight scalings is fairly consistent across algorithms, indicating minimal effect. However, it is noteworthy that `ipart` experiences much higher variance.

8.3 Ground Truth While these algorithms are guaranteed to find a decomposition using at most k weighted cliques if one exists, a unique solution is not guaranteed. For the synthetic corpus, we verified that our output recovered the LVs/TFs selected by the generators, but we do not know whether this will generalize to real data (where weight distributions may be quite different) or much larger k. Additionally, Appendix E.3 analyzes the impact on incorrect k input on the recovered solution.

9 Conclusions & Future Work

This paper offers a new combinatorial framing of the problem of module identification in gene co-expression data, Weighted Clique Decomposition. Further, we present two new parameterized algorithms for the noise-free setting (EWCD), removing the dependence of a prior approach on the magnitude of the clique weights. To address concerns of practicality, we implement both approaches and evaluate them on a corpus of biologically-inspired genetic association data. The empirical results show that both new approaches significantly outperform the Weighted Edge Clique Partition algorithm of [13], and that worst-case asymptotic runtime bounds are not realized on typical inputs.

While these algorithms provide a nice first step towards a polynomial-time algorithm for module-identification, there remain many unaddressed challenges before use on real data. While our exponential dependence on the number of modules is to be expected from an FPT algorithm, in many real-world datasets there are hundreds of underlying functional groups. One way to overcome this limitation is to incorporate a hierarchical approach, which would provide an extra biologically meaningful outcome: the relationships between cliques could be mapped to representations such as the Gene Ontology, where descendents represent more specialized function.

We would like to extend our approach to the non-exact setting, targeting approximation schemes for the optimization variant of the problem. Understanding the correct penalty function for under- and over-estimating edges, and whether this should be amplified for edges below the threshold in the original coexpression data will be critical in informing a useful technique.

[2] these times exclude the shared kernel to emphasize the difference in the two approaches

References

[1] F. Ban, V. Bhattiprolu, K. Bringmann, P. Kolev, E. Lee, and D. P. Woodruff. A PTAS for ℓ_p-low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 747–766. Society for Industrial and Applied Mathematics, 2019.

[2] William S. Bush, Matthew T. Oetjens, and Dana C. Crawford. Unravelling the human genome-phenome relationship using phenome-wide association studies. *Nature Reviews Genetics*, 17(3):129–145, 2016. URL: https://doi.org/10.1038/nrg.2015.36.

[3] S. Chandran, D. Issac, and A. Karrenbauer. On the parameterized complexity of biclique cover and partition. In *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[4] S. Chen, Z. Song, R. Tao, and R. Zhang. Symmetric boolean factor analysis with applications to instahide. arxiv. preprint, 2021. arXiv:2102.01570.

[5] Leonardo Collado-Torres, Abhinav Nellore, Kai Kammers, Shannon E Ellis, Margaret A Taub, Kasper D Hansen, Andrew E Jaffe, Ben Langmead, and Jeffrey T Leek. Reproducible RNA-seq analysis using recount2. *Nat. Biotechnol.*, 35(4):319–321, 2017.

[6] Madison Cooley, Casey S. Greene, Davis Issac, Milton Pividor, and Blair D. Sullivan. Cricca. https://github.com/TheoryInPractice/cricca, 2021.

[7] Heather J. Cordell. Detecting gene-gene interactions that underlie human diseases. *Nature Reviews Genetics*, 10(6):392–404, 2009. URL: https://doi.org/10.1038/nrg2579.

[8] M. Cygan, M. Pilipczuk, and M. Pilipczuk. Known algorithms for edge clique cover are probably optimal. *SIAM Journal on Computing*, 45(1):67–83, 2016.

[9] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5. Springer, 2015.

[10] Christiaan A. de Leeuw, Joris M. Mooij, Tom Heskes, and Danielle Posthuma. Magma: Generalized gene-set analysis of gwas data. *PLOS Computational Biology*, 11(4):e1004219, 2015. URL: https://doi.org/10.1371/journal.pcbi.1004219, doi:10.1371/journal.pcbi.1004219.

[11] Mikhail G. Dozmorov, Kellen G. Cresswell, Silviu-Alin Bacanu, Carl Craver, Mark Reimers, and Kenneth S. Kendler. A method for estimating coherence of molecular mechanisms in major human disease and traits. *BMC Bioinformatics*, 21(1):473, 2020. URL: https://doi.org/10.1186/s12859-020-03821-x, doi:10.1186/s12859-020-03821-x.

[12] Ahmed Essaghir, Federica Toffalini, Laurent Knoops, Anders Kallin, Jacques van Helden, and Jean-Baptiste Demoulin. Transcription factor regulation can be accurately predicted from the presence of target gene signatures in microarray gene expression data. *Nucleic Acids Research*, 38(11):e120–e120, 03 2010. URL: https://doi.org/10.1093/nar/gkq149, arXiv:https://academic.oup.com/nar/article-pdf/38/11/e120/16764928/gkq149.pdf, doi:10.1093/nar/gkq149.

[13] A. E. Feldmann, D. Isaac, and A. Rai. Fixed-parameter tractability of the weighted edge clique partition problem. In *15th International Symposium on Parameterized and Exact Computation (IPEC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[14] R. Fleischer and X. Wu. Edge clique partition of k 4-free and planar graphs. In *International Conference on Computational Geometry, Graphs and Applications . , , Heidelberg*, pages 84–95, Berlin, 2010. Springer.

[15] F. V. Fomin, P. A. Golovach, D. Lokshtanov, F. Panolan, and S. Saurabh. Approximation schemes for low-rank binary matrix approximation problems. *ACM Transactions on Algorithms (TALG)*, 16(1):1–39, 2019.

[16] F. V. Fomin, P. A. Golovach, and F. Panolan. Parameterized low-rank binary matrix approximation. *Data Mining and Knowledge Discovery*, 34(2):478–532, 2020.

[17] Casey S. Greene, Arjun Krishnan, Aaron K. Wong, Emanuela Ricciotti, Rene A. Zelaya, Daniel S. Himmelstein, Ran Zhang, Boris M. Hartmann, Elena Zaslavsky, Stuart C. Sealfon, Daniel I. Chasman, Garret A. FitzGerald, Kara Dolinski, Tilo Grosser, and Olga G. Troyanskaya. Understanding multicellular function and disease with human tissue-specific networks. *Nature Genetics*, 47(6):569–576, 2015. URL: https://doi.org/10.1038/ng.3259.

[18] Qiwen Hu and Jaclyn Taroni. Multiplier - latent variables extracted from recount2. https://figshare.com/articles/dataset/recount_rpkm_RData/5716033/4, 2018.

[19] R. Kumar, R. Panigrahy, A. Rahimi, and D. Woodruff. Faster algorithms for binary matrix factorization. In *International Conference on Machine Learning*, pages 3551–3559. PMLR, 2019.

[20] Alexander Lachmann, Denis Torre, Alexandra B. Keenan, Kathleen M. Jagodnik, Hoyjin J. Lee, Lily Wang, Moshe C. Silverstein, and Avi Maayan. Massive mining of publicly available rna-seq data from human and mouse. *Nature Communications*, 9(1):1366, 2018. URL: https://doi.org/10.1038/s41467-018-03751-6.

[21] SH Ma, WD Wallis, and JL Wu. The complexity of the clique partition number problem. *Congr. Numer*, 67:59–66, 1988.

[22] Weiguang Mao, Elena Zaslavsky, Boris M. Hartmann, Stuart C. Sealfon, and Maria Chikina. Pathway-level information extractor (plier) for gene expression data. *Nature Methods*, 16(7):607–610, 2019. URL: https://doi.org/10.1038/s41592-019-0456-1.

[23] Jrg Menche, Amitabh Sharma, Maksim Kitsak, Susan Dina Ghiassian, Marc Vidal, Joseph Loscalzo, and Albert-Lszl Barabsi. Uncovering disease-

disease relationships through the incomplete interactome. *Science*, 347(6224):1257601, February 2015. URL: http://science.sciencemag.org/content/347/6224/1257601.abstract, doi:10.1126/science.1257601.

[24] Daniele Mercatelli, Laura Scalambra, Luca Triboli, Forest Ray, and Federico M. Giorgi. Gene regulatory network inference resources: A practical overview. *Biochimica et Biophysica Acta (BBA) - Gene Regulatory Mechanisms*, 1863(6):194430, 2020. URL: https://www.sciencedirect.com/science/article/pii/S1874939919300410, doi:10.1016/j.bbagrm.2019.194430.

[25] P. Miettinen and S. Neumann. Recent developments in boolean matrix factorization. preprint, 2020. arXiv:2012.03127.

[26] Jason H. Moore, Folkert W. Asselbergs, and Scott M. Williams. Bioinformatics challenges for genome-wide association studies. *Bioinformatics*, 26(4):445–455, 2010. URL: https://doi.org/10.1093/bioinformatics/btp713.

[27] F. Moutier, A. Vandaele, and N. Gillis. Off-diagonal symmetric nonnegative matrix factorization. *Numerical Algorithms, pp*, pages 1–25, 2021.

[28] E. Mujuni and F. Rosamond. Parameterized complexity of the clique partition problem. In *Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77*, pages 75–78, 2008.

[29] National Heart, Lung, and Blood Institute (NHLBI. Marfan syndrome. https://www.nhlbi.nih.gov/health-topics/marfan-syndrome, (Accessed in March 4, 2021).

[30] Matthew R Nelson, Hannah Tipney, Jeffery L Painter, Judong Shen, Paola Nicoletti, Yufeng Shen, Aris Floratos, Pak Chung Sham, Mulin Jun Li, Junwen Wang, Lon R Cardon, John C Whittaker, and Philippe Sanseau. The support of human genetic evidence for approved drug indications. *Nat. Genet.*, 47(8):856–860, 2015.

[31] Milton Pividori, Nathan Schoettler, Dan L Nicolae, Carole Ober, and Hae Kyung Im. Shared and distinct genetic risk factors for childhood-onset and adult-onset asthma: genome-wide and transcriptome-wide studies. *Lancet Respir Med*, 7(6):509–522, 2019. PMCID: PMC6534440. doi:10.1016/S2213-2600(19)30055-4.

[32] Vivian Tam, Nikunj Patel, Michelle Turcotte, Yohan Boss, Guillaume Par, and David Meyre. Benefits and limitations of genome-wide association studies. *Nature Reviews Genetics*, 20(8):467–484, 2019. doi:10.1038/s41576-019-0127-1.

[33] Jaclyn N. Taroni, Peter C. Grayson, Qiwen Hu, Sean Eddy, Matthias Kretzler, Peter A. Merkel, and Casey S. Greene. Multiplier: A transfer learning framework for transcriptomics reveals systemic features of rare disease. *Cell Systems*, 8(5):380–394.e4, 2019. URL: http://www.sciencedirect.com/science/article/pii/S240547121930119X.

[34] P. M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337, 1989. doi:10.1109/SFCS.1989.63499.

[35] Kavitha Venkatesan, Jean-Franois Rual, Alexei Vazquez, Ulrich Stelzl, Irma Lemmens, Tomoko Hirozane-Kishikawa, Tong Hao, Martina Zenkner, Xiaofeng Xin, Kwang-Il Goh, Muhammed A. Yildirim, Nicolas Simonis, Kathrin Heinzmann, Fana Gebreab, Julie M. Sahalie, Sebiha Cevik, Christophe Simon, Anne-Sophie de Smet, Elizabeth Dann, Alex Smolyar, Arunachalam Vinayagam, Haiyuan Yu, David Szeto, Heather Borick, Amlie Dricot, Niels Klitgord, Ryan R. Murray, Chenwei Lin, Maciej Lalowski, Jan Timm, Kirstin Rau, Charles Boone, Pascal Braun, Michael E. Cusick, Frederick P. Roth, David E. Hill, Jan Tavernier, Erich E. Wanker, Albert-Lszl Barabsi, and Marc Vidal. An empirical framework for binary interactome mapping. *Nature Methods*, 6(1):83–90, 2009. URL: https://doi.org/10.1038/nmeth.1280, doi:10.1038/nmeth.1280.

[36] Peter M Visscher, Naomi R Wray, Qian Zhang, Pamela Sklar, Mark I McCarthy, Matthew A Brown, and Jian Yang. 10 years of GWAS discovery: Biology, function, and translation. *Am. J. Hum. Genet.*, 101(1):5–22, 2017.

[37] Kyoko Watanabe, Sven Stringer, Oleksandr Frei, Maša Umićević Mirkov, Christiaan de Leeuw, Tinca J C Polderman, Sophie van der Sluis, Ole A Andreassen, Benjamin M Neale, and Danielle Posthuma. A global overview of pleiotropy and genetic architecture in complex traits. *Nat. Genet.*, 51(9):1339–1348, 2019.

[38] Sally E. Wenzel. Asthma phenotypes: the evolution from clinical to molecular approaches. *Nature Medicine*, 18(5):716–725, 2012. URL: https://doi.org/10.1038/nm.2678.

[39] Z. Y. Zhang, Y. Wang, and Y. Y. Ahn. Overlapping community detection in complex networks using symmetric binary matrix factorization. *Physical Review E*, 87:6, 2013.

BELLA: Berkeley Efficient Long-Read to Long-Read Aligner and Overlapper

Giulia Guidi[1,3] Marquita Ellis[1,3] Daniel Rokhsar[2,4]
Katherine Yelick[1,3] Aydın Buluç[1,3]

[1]Department of Electrical Engineering and Computer Sciences, University of California at Berkeley
[2]Department of Molecular and Cell Biology, University of California at Berkeley
[3]Computational Research Division, Lawrence Berkeley National Laboratory
[4]DOE Joint Genome Institute

Abstract

Recent advances in long-read sequencing allow characterization of genome structure and its variation within and between species at a resolution not previously possible. Detection of overlap between reads is an essential component of many long read genome pipelines, such as *de novo* genome assembly. Longer reads simplify genome assembly and improve reconstruction contiguity, but current long read technologies are associated with moderate to high error rates.

In this work, we present Berkeley Efficient Long-Read to Long-Read Aligner and Overlapper (BELLA), a novel overlap detection and alignment algorithm using sparse matrix-matrix multiplication. In addition, we present a probabilistic model that demonstrates the feasibility of using k-mers for overlap candidate detection and shows its flexibility when applied to different k-mer selection strategies. Based on such a model, we introduce a notion of *reliable k-mers*. Combining reliable k-mers with our *binning* mechanism increases the computational efficiency and accuracy of our algorithm. Finally, we present a new method based on Chernoff bounds to separate true overlaps from false positives by combining alignment techniques and probabilistic modeling. Our goal is to maximize the balance of precision and recall.

For both real and synthetic data, BELLA is among the best F1 scores, showing a stability of performance that is often lacking in competing software. BELLA's F1 score is consistently within 1.7% of the top performer. In particular, we show improved *de novo* assembly quality on synthetic data when BELLA is coupled with the miniasm assembler.

1 Introduction

Recent advances in long-read sequencing technologies have enabled characterization of genome structure and its variation between and within species that was not previously possible. However, post-sequencing data analysis remains a challenging task. One of the most challenging aspects of analyzing DNA fragments from high-throughput sequencing, namely *reads*, is whole-genome assembly [32], i.e., the process of aligning and assembling DNA fragments to reconstruct the original sequence. More specifically, *de novo* genome assembly reconstructs a genome from redundantly sampled reads without prior knowledge of the genome, enabling the study of previously uncharacterized genomes [30].

Long-read technologies [11, 14] generate long reads with an average length often exceeding 10,000 base pairs (bp). These allow resolution of complex genomic repeats and enable more accurate ensemble views that were not possible with previous short-read technologies [28, 25]. However, the improved read length of these technologies is accompanied by lower accuracy, with error rates from 0.5% to 15%. Nevertheless, the error distribution for Pacific Biosciences long reads [13] is more random and uniform compared to short-read technologies.

A long read assembler typically uses the Overlap-Layout-Consensus (OLC) assembly paradigm [2]. The first step in OLC assembly is to detect overlaps between reads to create an overlap (or string) graph. The OLC paradigm benefits from longer reads, as significantly fewer reads are needed to cover the genome, limiting the size of the overlap graph. Highly accurate overlap detection is a major computational bottleneck in OLC assembly [24], mainly due to the computationally intensive nature of pairwise alignment.

Currently, several algorithms are capable of overlapping error-prone long read data with varying accuracy. The prevailing approach is to use an indexing data structure, such as a k-mer index table (i.e., substrings of fixed length k) or suffix array, to identify a set of initial candidate read pairs, thereby reducing the high cost of computing pairwise alignments in a second step [8].

The process of identifying a set of initial candidate read pairs, sometimes referred to simply as "overlap", affects both the accuracy and runtime of the algorithm. Accurate identification of initial candidate read pairs minimizes the runtime of pairwise alignment, while retaining all pairs that truly overlap in the genome. Computationally efficient and accurate overlapping and alignment algorithms have the potential to improve existing long-read assemblers, enabling *de novo* reference assemblies, detection of genetic variations of higher quality, and accurate metagenome classification.

Our main contributions are:

1. Using a Markov chain model [22], we show that a

k-mer seed-based approach is useful for accurately identifying overlap candidates. Our model is general enough to be applicable to different types of sequencing data and k-mer selection strategies.
2. To increase computational efficiency without loss of accuracy, we develop a simple procedure for pruning k-mers and prove that it preserves almost all true overlaps with high probability.
3. To take advantage of high performance techniques, we reformulate the problem of overlap detection in terms of sparse matrix-matrix multiplication (SpGEMM), which has not been previously applied in the context of long-read overlap and alignment. This approach is flexible and can be implemented independently of the k-mer selection strategy.
4. By coupling our overlap detection with our newly developed seed-and-extend alignment algorithm, we present a novel method to separate true alignments from false positives.

2 Related Work

The increasing popularity of long read sequencing data has led to many efforts in the literature to perform accurate overlap detection. In the context of long read overlap detection, we can distinguish between "base-level alignment" and "overlap-only" strategies below.

Base-Level Alignment: DALIGNER [24] uses k-mers to find overlap candidates and then perform alignment. It parses the sequences in k-mers, sorts them, and finds overlapping sequences with a merge operation. To filter out spurious overlap candidates, a pairwise alignment is performed using a linear expected-time heuristic based on the difference algorithm [23]. BLASR [6], originally developed to align noisy long-read sequencing data to reference genomes, later became popular as a read-to-read aligner. It too first uses k-mers to detect initial overlap candidates and then filters them using alignment. In addition to the aligner itself, Chaisson and Tesler [6] presented a mathematical model that proved the feasibility of using a k-mer seed to find a match between a noisy long-read sequence and a correct reference sequence. In this paper, we make a similar contribution by presenting a different model that proves the feasibility of using a k-mer seed to find a match between two noisy long-read sequences, and that is not restricted to regular k-mer selection strategies.

Overlap Only: Li's minimap2 [19] also uses seeds to find matches. However, it uses a different kind of k-mer, called a minimizer, which reduces the number of seeds because it selects only one minimizer in a window w whose value is the minimum according to a function. It does not perform any alignment. Instead, it computes an approximate alignment score based on the location of the minimizers on the sequences and excludes those whose quality is below a defined threshold. MECAT [31] identifies overlap candidates using k-mers and introduces a pseudo-linear alignment scoring algorithm to filter excessive candidates using a distance difference factor to score k-mer matches. MHAP [2] is a probabilistic algorithm for sequence overlap detection. It estimates Jaccard similarity by compressing sequences to their representative identity composed of min-mers, filtering out false candidates.

SpGEMM is a relatively unknown primitive in genomics. Most notably, Besta et al. [3] used SpGEMM to compute similarity between genomes in distributed memory, after the appearance of our preprint [15].

3 Proposed Algorithm

In this paper, a computationally efficient and accurate overlap detection and alignment algorithm for long read genomic pipelines is developed and implemented in a software package called BELLA.

BELLA uses a seed-based approach to detect overlaps in the context of long read applications. Such an approach parses reads into k-mers (i.e., substrings of fixed length k), which are then used as feature vectors to identify overlaps between all reads. Using a Markov chain model, we demonstrate the feasibility of a k-mer seed-based approach for long read overlap detection. The descriptiveness of our model allows us to model the probability of finding a correct common seed between two sequences even when k-mer strategies other than ours are used, such as minimizers [19] and syncmers [10].

Importantly, not all k-mers are equal in terms of their usefulness for overlap detection. For example, the vast majority of k-mers that occur only once in the dataset are errors (and are also not useful for detecting overlaps between read pairs). Similarly, k-mers that occur more frequently than one would expect given the sequencing depth and error rate are likely from repetitive regions. It is a common practice to prune the k-mer space using various methods [18, 21, 5].

BELLA implements a novel method for filtering out k-mers that are likely to either contain errors or originate from a repetitive region. The k-mers retained by BELLA are considered *reliable*, where the reliability of a k-mer is defined as its probability of originating from a unique (non-repetitive) region of the genome. Our reliable k-mer detection maximizes retention of k-mers from unique regions of the genome using probabilistic analysis given error rate and sequencing depth.

BELLA uses a sparse matrix to represent its data internally, where the rows are reads, the columns are reliable k-mers, and a nonzero $\mathbf{A}(i, j) \neq 0$ is the position of the j-th k-mer within the i-th read. The construction

of this sparse matrix requires efficient k-mer counting.

Overlap detection is implemented in BELLA using SpGEMM, which allows our algorithm to achieve fast overlap without using approximate approaches. SpGEMM is a highly flexible and efficient paradigm that allows for better organization of computation and generality, as it can manipulate complex data structures, such as those used to perform overlap detection using common k-mers. Our k-mer selection can be easily replaced by other selection strategies without compromising the SpGEMM-based overlap detection, demonstrating the generality of our approach. Implementing this method in our pipeline enables the use of high-performance techniques that have not been previously applied in the context of long read alignment. It also enables continuous performance improvements in this step due to increasingly optimized implementations of SpGEMM [26, 9].

BELLA's overlap detection has been coupled with our high-performance seed-and-extend algorithm, which means that the alignment between two reads starts from a common seed (identified in the previous overlap detection) and not necessarily from the beginning of the reads. To refine the seed selection, we introduce a procedure called *binning*. The k-mer positions in a read pair are used to estimate the length of the overlap, and the k-mers are "binned" based on their length estimates. We consider for the alignment only k-mers that belong to the most crowded bins, which we call *consensus* k-mers. During the alignment phase, BELLA uses a new method to separate true alignments from false positives as a function of the alignment score. We prove that the probability of false positives decreases exponentially as the length of overlap between reads increases.

Existing tools also implement approximate overlap detection using sketches. A sketch is a space-reduced representation of a sequence. Several randomized hash functions convert k-mers into integer fingerprints, and a subset of them is selected to represent the sketch of a sequence according to some criterion. For example, Berlin et al. [2] keep only the smallest integer for each hash function and use the collection of these minimal-valued fingerprints as the sketch. These methods are fast, but only approximate, since the sketch is a lossy transformation. In contrast, BELLA uses an explicit k-mer representation that allows us to couple our overlap detection with a seed-and-extend alignment to refine the output and improve the precision of our algorithm.

4 Methods

4.1 Overlap Feasibility Chaisson and Tesler [6] proposed a theory of how long reads contain subsequences that can be used to anchor alignments to the reference genome. The sequences are modeled as random processes that generate error-free regions whose length is geometrically distributed, with each such region separated by an error [13]. The result obtained from their theory is the minimum sequence length to have an *anchor* within a confidence interval.

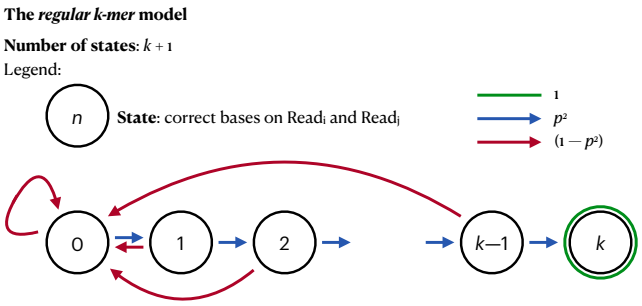


Figure 1: Proposed Markov Chain model demonstrating the feasibility of using k-mers for overlap detection.

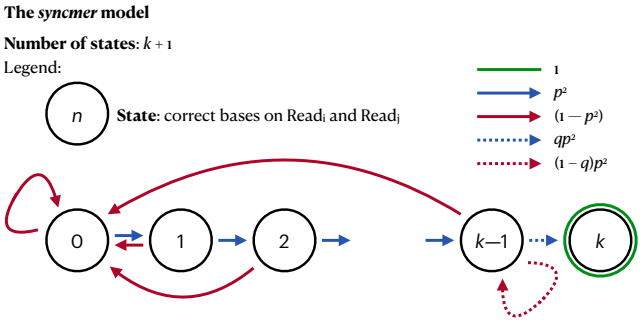


Figure 2: Proposed Markov Chain model using syncmer approach instead of the k-mer one, where $q = 1/c$.

Here we present an alternative model of how these subsequences, also called k-mers, can be used to anchor alignments between two erroneous long read sequences, allowing accurate overlap detection between all reads in a dataset. The initial assumption of our model defines the probability of correctly sequencing a base as equal to $p = (1-e)$, where e is the error rate of the sequencer. From this notion, we model the probability of observing k correct consecutive bases on both $read_1$ and $read_2$ as a Markov chain process [22].

The Markov chain process is characterized by a *transition matrix* $\mathbf{P}$ that contains the probabilities of transitioning from one state to another. Each row index *start* of $\mathbf{P}$ represents the initial state, and each column index *end* of $\mathbf{P}$ represents the final state. Each entry of $\mathbf{P}$ is a non-negative number indicating a *transition probability*. Our transition matrix has $(k+1)$ possible states, resulting in $(k+1)^2$ transition probabilities for the transition from *start* to *end*. The probability of having a correct base in both reads is p^2. For every state except the *absorbing* state k, an error in at least

Algorithm 1 Probability of observing at least one correct k-mer in an overlap region of length $\mathrm{L} > k$.

```
procedure EstimateSharedKmerProbability(k,L,p)
    states ← (k + 1)
    P ← 0                          ▷ Entire matrix initialized to 0
    for i ← 0 to states do
        P[i, 0] ← (1 − p^2)
        P[i, i + 1] ← p^2
    end for
    P[states, states] ← 1
    v ← (1, 0, . . . , 0)          ▷ Initialized to standard unit vector
    for i ← 0 to L do              ▷ Compute vP^L without exponentiation
        v ← vP
    end for
    return v[states]
end procedure
```

one of the two sequences sets the model back to state 0, which happens with probability $1 - p^2$; otherwise, the Markov chain transition from state i to $i + 1$ happens with probability p^2. The absorbing state k cannot be abandoned since both $read_1$ and $read_2$ have already seen k successive correct bases. Therefore, its transition probability is 1. Figure 1 describes the process: each state contains the number of successfully sequenced bases obtained on both reads up to that point, while the arrows represent the transition probabilities.

One can then find the probability of being in one of the states after L steps in the Markov chain by computing the L-th power of the matrix $\mathbf{P}$, where L is the length of overlap between the two sequences. More efficiently, this can be computed iteratively with only L sparse matrix vector products, starting from the unit vector $\mathbf{v} \leftarrow (1, 0, \ldots, 0)$ (Algorithm 1). This approach is sufficient since we are only interested in the probability of being in the absorbing final state. This operation leads to the probability of having a correct k-mer at the same location on both reads, given a certain overlap region. This model is the driving factor for choosing the optimal k-mer length used in overlap detection.

Our Markov chain can be modified to accommodate different k-mer selection strategies, such as syncmers and minimizers. Given a compression factor $c > 1$ that sets a minimal value for the k-mer code, a k-mer κ is a *mincode syncmer* if $code(\kappa) \leq H/c$, where H is the maximal possible code [10]. The probability that a given k-mer is chosen as a mincode syncmer is $1/c$. In this case, we can modify our model to include the probability that a k-mer is correct *and* that it is retained as a mincode syncmer. The transition from the $(k-1)$-th state to the k-th state in the Markov chain will model syncmer selection so that we have a probability of transitioning from $k-1$ to k that is equal to qp^2, where $q = 1/c$, i.e., the k-th is correctly sequenced and the k-mer is a mincode syncmer. Then, we have a probability of $(1-q)p^2$ to stay in the $(k-1)$-th state, which means that we have sequenced a correct base on both sequences, but the k-mer is not selected as a syncmer. The probability of returning to the initial state is unchanged. The Markov chain model that is modified for mincode syncmers is shown in Figure 2.

We used the mincode syncmer as an example but the same probabilistic model applies to other syncmer types, including those with better spacing properties, such as closed syncmers [10]. This is because the selection of a k-mer as a syncmer is by definition a local decision and is not affected by neighboring k-mers.

The case of the minimizer is slightly different. A k-mer is a minimizer if it has the smallest code among w consecutive k-mers, where w is called the window length [29]. For a k-mer κ that is correctly sequenced in both reads, we need to consider the number of competing k-mers in its windows to determine the probability that κ is selected as the minimizer *from both reads*. If there are no sequencing errors, there are w competing k-mers including κ itself. Because errors can change the competing k-mers in each read independently, the maximum number of competing k-mers including κ itself is $2w-1$. It is possible to compute the exact expected number of competing k-mers, but since this range is narrow within a factor of two, we can also use the upper and lower bounds instead when choosing minimizer parameters (w, k) in practice.

4.2 Reliable k-mers Repetitive regions of the genome cause certain k-mers to occur frequently in input reads. K-mers from these regions pose two problems for pairwise overlap and alignment. First, their presence increases the computational cost, both in the overlap and alignment phases, as these k-mers generate numerous and possibly incorrect overlaps. Second, they often do not provide valuable information.

Our argument here is that k-mers that originate from a repetitive region in the genome can be ignored for seed-based overlap. This is because either (a) the read is longer than the repeat, in which case there should be enough sequence data from the non-repetitive section to find overlaps, or (b) the read is shorter than the repeat, in which case their overlaps are inherently ambiguous and uninformative and will not be particularly useful for downstream tasks such as *de novo* assembly. In the case of a nearly identical region, we would expect to find

a k-mer that comes from a unique region of that nearly identical repetition to identify that region.

Following the terminology proposed by Lin et al. [21], we refer to k-mers not present in the genome as *non-genomic* and thus characterize k-mers present in the genome as *genomic*. A genomic k-mer may be *repeated* if it occurs multiple times in the genome, or *unique* if it does not. One can think of the presence of k-mers in each read as the feature vector of that read. Therefore, the feature vector should contain all unique k-mers, as they are often the most informative features.

Since we do not know the genome before assembly, we estimate the genomic uniqueness of k-mers from redundant, error-prone reads. Here we present a mathematically based procedure that selects a frequency range for k-mers that we consider *reliable*. The basic question guiding the procedure for selecting reliable k-mers is the following: "Given that a k-mer is sequenced from a unique (non-repeated) region of the genome, what is the probability that it occurs at least m times in the input data?" For a genome G sequenced at depth d, the conditional modeled probability is:

(4.1)
$$Pr(\text{FREQ}(k\text{-}mer, G, d) \geq m | \text{COUNT}(\text{MAP}(k\text{-}mer, G) = 1)$$

where $\text{MAP}(k\text{-}mer, G)$ is the set of sites in genome G to which k-*mer* can be mapped, the function COUNT() computes the cardinality of a given input set, and $\text{FREQ}(k\text{-}mer, G, d)$ is the expected number of occurrences of k-*mer* within sequenced reads, assuming that each region of G is sequenced d times (*sequencing depth*). In this sense, BELLA's approach to selecting reliable k-mers is very different from the way Lin et al. [21] select their *solid strings*. While solid strings discard rare k-mers, our model discards highly recurrent k-mers because (a) unique k-mers are sufficient to find informative overlaps, and (b) a unique k-mer has a low probability of occurring frequently.

The probability of correctly sequencing a k-mer is approximately $(1-e)^k$, where e is the error rate. The probability of correctly sequencing a k-mer once can be generalized to the probability of seeing it multiple times in the data, provided that each correct sequencing of that k-mer is an independent event. For example, if the sequencing depth is d, the probability of observing a unique k-mer k_i in the input data d times is approximately $(1-e)^{dk}$. More generally, the number of correct sequencing of a unique k-long genome segment at a sequencing depth d follows a binomial distribution:

(4.2)
$$B(n = d, p = (1-e)^k)$$

where n is the number of trials and p is the probability of success. From this, we derive that the probability of observing a k-mer k_i (corresponding to a unique non-repetitive region of the genome) m times within a sequencing input data with depth d is:

(4.3)
$$Pr(m; d, (1-e)^k) = \binom{d}{m}(1-e)^{km}(1-(1-e)^k)^{(d-m)}$$

where m is the multiplicity of the k-mer in the input data, e is the error rate, d is the sequencing depth, and k is the k-mer length. Given the values of d, e and k, the curve $Pr(m; d, (1-e)^k)$ can be calculated.

Equation 4.3 is used to identify the range of reliable k-mers. To choose the lower bound l, we compute $Pr(m; d, (1-e)^k)$ for each multiplicity m and sum these probabilities cumulatively, starting from $m = 2$. The cumulative sum does not start at $m = 1$ because a k-mer that occurs only once in the input data (and therefore appears on a single read) cannot be used to identify the overlap between two reads. The lower bound l is the smaller m value after which the cumulative sum exceeds a user-defined threshold ϵ. The choice of l is significant when the sequencing error rate is relatively low ($\approx 5\%$) or when the sequencing coverage is high ($\approx 50-60\times$), or both. This is because in these cases, a k-mer with small multiplicity has a high probability of being incorrect.

The upper bound u is chosen in a similar way. Here, starting from the largest possible value of m (i.e. d), the probabilities are added up cumulatively. In this case, u is the largest value of m after which the cumulative sum exceeds the user-defined threshold ϵ. The k-mers that are more frequent than u have too low a probability of belonging to a unique region of the genome, and multiple mapped k-mers would lead to an increase in computational cost and possibly misassembly.

K-mers with multiplicity greater than u and multiplicity less than l in the input set are discarded and not used as read features in the downstream algorithm. Our reliable k-mers selection discards at most 2ϵ useful information when the data fits the model, in the form of k-mers that can be used for overlap detection.

When using syncmers instead of k-mers, the reliable range calculations are unchanged. This is because any given k-mer is either selected as a syncmer for all its occurrences in the dataset, or it is never selected as a syncmer. For minimizers, the exact computation of the reliable range is non-trivial, since errors in flanking sequences affect whether a k-mer is retained as a minimizer. Therefore, we leave this as future work.

4.3 Sparse Matrix Construction and Multiplication BELLA uses a sparse matrix format to store its data and a sparse matrix multiplication (SpGEMM) to

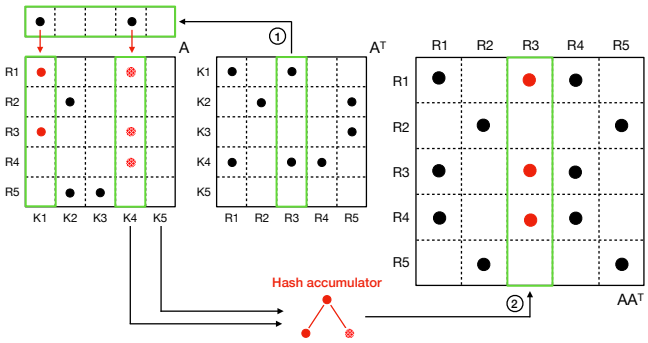


Figure 3: Column-wise sparse matrix multiplication. $\mathbf{A}^\mathsf{T}(:, R_3)$ is chosen as the "active column": its non-zero elements define which columns of $\mathbf{A}$ to consider by looking at their corresponding row indices in $\mathbf{A}^\mathsf{T}$.

identify overlaps. Sparse matrices express data access patterns concisely and clearly, and allow for better organization of computation. The sparse matrix $\mathbf{A}$, also known as the *data matrix*, is a $|reads|$-by-$|k\text{-}mers|$ matrix with reads as rows and the entries of the k-mer dictionary as columns. If the j-th reliable k-mer is present in the i-th read, the (i, j) cell of $\mathbf{A}$ is nonzero. $\mathbf{A}$ is then multiplied by its transpose, $\mathbf{A}^\mathsf{T}$, yielding a sparse *overlap matrix* $\mathbf{A}\mathbf{A}^\mathsf{T}$ of dimensions $|reads|$-by-$|reads|$. Each cell (i, j) of the overlap matrix that is non-zero contains the number of shared k-mers between the i-th read and the j-th read, and the corresponding positions in the corresponding read pair of (at most) two shared k-mers.

Column-wise sparse matrix multiplication is efficiently implemented using the Compressed Sparse Columns (CSC) format for storing sparse matrices. However, other options are certainly possible in the future, which is one of the advantages of our work. Any novel sparse matrix format and multiplication algorithm would apply to the overlap problem and allow for further performance improvements, as several software packages already implement these primitives, including Intel MKL and Sandia's KokkosKernels [9].

The SpGEMM algorithm shown in Figure 3 is functionally equivalent to a k-mer-based seed index table common in other long read alignment codes. However, unlike hash tables, the CSC format allows true, constant-time random access to the columns. More importantly, the computational problem of accumulating the contributions of multiple shared k-mers to each read pair is automatically handled by choosing appropriate data structures within SpGEMM. Figure 3 illustrates the merging operation of BELLA, which uses a hash table data structure indexed by the row indices of $\mathbf{A}$, following the multi-threaded implementation proposed by Nagasaka et al. [26]. This hash table based accumulator is merely one way to compute SpGEMM and many other methods have been proposed in the literature [12]. Finally, the content of the hash table is stored in a column of the final matrix once all the required nonzeros for that column are accumulated.

Since BELLA performs a seed-and-extend alignment from pairs of reads that share at least t (by default $t = 1$) k-mers, the overlap phase must keep track of the positions of the shared k-mers. In cases where multiple k-mers are shared between any pair of reads, it is not economical to store all k-mer matches. Even though the previously described reliable k-mer selection procedure eliminates most matches from repeated regions, it is still possible to observe misleading k-mer matches due to sequencing errors and repetitions.

To ensure optimal seed selection for the alignment step, BELLA applies the following binning method. Whenever a common k-mer is found between a pair of reads, we use its position in these reads to estimate an overlap length and orientation. A new overlap estimate forms its bin with a single element unless it is already within the boundaries of a previous bin (i.e., within an adjustable distance β, which is 500 by default). In this case, the vote count of the adjacent bin is increased by one, as long as the two originating k-mers do not overlap. Only the two bins with the highest vote count are retained, along with a representative k-mer that supports this overlap estimate.

In the resulting sparse overlap matrix $\mathbf{A}\mathbf{A}^\mathsf{T}$, each non-zero cell (i, j) is a structure consisting of an integer value storing the number of shared k-mers and an integer array of size 4 storing the position of (up to two) shared k-mers on read i and read j corresponding to the overlap estimate bins with the highest number of votes. To enable this special multiplication, which performs scalar multiplication and addition differently, we use semiring abstraction [17]. Multiplication on a semiring allows the user to overload scalar multiplication and addition operations while still using the same SpGEMM algorithm. Many existing SpGEMM implementations support custom semirings, including those that implement the GraphBLAS API [4].

As genome size increases, so does the memory required to construct the final overlap matrix. For large genomes, the sparse overlap matrix $\mathbf{A}\mathbf{A}^\mathsf{T}$ may not fit in memory even if the data matrix $\mathbf{A}$ does. BELLA avoids this situation by splitting the multiplication into batches based on the available RAM. In each phase, only one batch of columns of the overlap matrix is created. The set of nonzeros in this batch of the overlap matrix is immediately tested for alignments (as described in Section 4.4). The pairs that pass the alignment test are written to the output file of BELLA, so that the current batch of the overlap matrix can be discarded.

Due to the nature of our problem, the sparse overlap matrix $\mathbf{AA}^\mathsf{T}$ is a symmetric matrix. Therefore, we compute the multiplication using only the lower triangle of $\mathbf{A}$ and avoid computing the pairwise alignment twice for each pair. Currently, there are no known specialized SpGEMM implementations for $\mathbf{AA}^\mathsf{T}$ that store and operate only on $\mathbf{A}$, but we hope to develop one in the future. This would have halved the memory requirement. The obvious solution of computing inner projections of rows of $\mathbf{A}$ is suboptimal, since it must perform $\Omega(|reads|^2)$ inner products even though the majority of inner products are zero. In contrast, our column-wise implementation runs faster than $O(|reads|^2)$ when the overlap matrix $\mathbf{AA}^\mathsf{T}$ is sparse. Since the main purpose of the overlap process is to filter overlap candidates, the overlap matrix tends to be sparse over 99%.

4.4 Pairwise Alignment High precision is desirable to avoid unnecessary work in the subsequent steps of *de novo* assembly. Therefore, BELLA filters overlap candidates by performing fast, near linear-time pairwise seed-and-extend alignments.

Unlike approaches that rely on sketches or minimizers, such as Minimap2 [19] and MHAP [2], seed-and-extend alignment can be performed directly using the k-mers from the overlap phase of BELLA. BELLA's alignment module is based on our high-performance seed-and-extend banded alignment, which uses a narrow adaptive band that significantly improves performance and reduces the search space for optimal alignment.

Our binning mechanism chooses at most two seed k-mers to be used as input for the seed-and-extend x-drop alignment. For each read pair in the overlap matrix, the alignment of one or two seeds is extended until the alignment score falls x points below the previous best score. If the final score is less than a threshold n, the sequence pair is discarded.

To filter out spurious candidates, we use an *adaptive threshold*, calculated based on the estimated overlap between a given pair of reads. The choice of scoring matrix used in the pairwise alignment step can justify that the alignment score threshold is a linear function of the estimated overlap length.

Given an estimated overlap region of length L and probability p^2 of obtaining a correct base on both sequences, we would expect $m = p^2 \cdot L$ correct matches within this overlap region. The alignment score χ is:

$$\chi = \alpha m - \beta(L - m) = \alpha p^2 L - \beta(L - p^2 L) \tag{4.4}$$

where m is the number of matches, L is the overlap length, α is the value associated with a match in the scoring matrix, while β is the penalty for a mismatch or a gap ($\alpha, \beta > 0$). Under these assumptions, we define the ratio φ as χ over the estimated overlap length L as:

$$\varphi = \frac{\chi}{L} = \alpha p^2 - \beta(1 - p^2). \tag{4.5}$$

The expected value of φ is equal to $2 \cdot p^2 - 1$ when an exact alignment algorithm is used. We want to define a cutoff with respect to $(1 - \delta)\varphi$ such that we keep pairs above this cutoff as true alignments and discard the remaining pairs. To this scope, we use a Chernoff bound [7, 16] to define the value of δ, and prove that there is only a small probability of missing a true overlap of length $L \geq 2000$ bp (which is typically the minimum overlap length for a sequence to be considered a true positive) using the cutoff defined above.

Let Z be a sum of independent random variables $\{Z_i\}$, with $E[Z] = \mu_z$; we assume for simplicity that $Z_i \in \{0, 1\}$, for all $i \leq L$. The Chernoff bound defines an upper bound on the probability of Z deviating by a certain amount δ from its expected value. In particular, we use a corollary of the multiplicative Chernoff bound [1] defined for $0 \leq \delta \leq 1$ as:

$$\mathbf{Pr}[Z \leq (1 - \delta)\mu_z] \leq e^{\frac{-\delta^2 \mu_z}{2}} \tag{4.6}$$

To obtain the Chernoff bound for the ratio φ, we consider a random variable $X_i \in \{-\beta, \alpha\}$ such that:

$$X_i = \begin{cases} \alpha, & \text{with probability } p^2 \\ -\beta, & \text{with probability } 1 - p^2 \end{cases} \tag{4.7}$$

where $\alpha, \beta > 0$ are still the values associated with a match and a mismatch and a gap/indel in the scoring matrix, respectively; its expected value $E[X_i]$ is exactly equal to $\varphi = \alpha p^2 - \beta(1 - p^2)$. Since the Chernoff bound is defined for a sum of independent random variables $Z_i \in \{0, 1\}$, we need to move from $X_i \in \{-\beta, \alpha\}$ to $Z_i \in \{0, 1\}$. Thus, we define a new random variable $Y_i = X_i + \beta$ as a linear transformation of X_i that can take values $\{0, \alpha + \beta\}$. Given $E[Y_i] = E[X_i] + \beta = (\alpha + \beta)p$, we can normalize Y_i to obtain the desired random variable Z_i:

$$Z_i = \frac{X_i + \beta}{\alpha + \beta}, \text{ where } Z_i \in \{0, 1\} \tag{4.8}$$

From the linearity of expectation, we obtain:

$$E[Z] = E\left[\frac{X + \beta}{\alpha + \beta}\right] = \frac{E[X] + \beta L}{\alpha + \beta} = \frac{(2p^2 - 1)L + \beta L}{\alpha + \beta} \tag{4.9}$$

Table 1: Datasets used for evaluation. Datasets above the line are real data, while datasets below the line were generated using PBSIM [27]. Download: `portal.nersc.gov/project/m1982/bella/`.

Short Name	Depth	Species and Strain	Fastq Size
E.coli (Sample)	30X	*Escherichia coli* MG1655	266 MB
E.coli	100X	*Escherichia coli* MG1655	929 MB
E. coli (CCS Sample)	29X	*Escherichia coli* MG1655	240 MB
E. coli (CCS)	290X	*Escherichia coli* MG1655	2.60 GB
C.elegans	40X	*Caenorhabditis elegans* Bristol	8.90 GB
P. aeruginosa	30X	*Pseudomonas aeruginosa* PAO1	359 MB
V. vulnificus	30X	*Vibrio vulnificus* YJ016	288 MB
A. baumannii	30X	*Acinetobacter baumannii*	248 MB
C.elegans	20X	*Caenorhabditis elegans*	3.75 GB

Finally, substituting Eq. 4.8 and Eq. 4.9 into Eq. 4.6 and simplifying with our scoring matrix $\alpha, \beta = 1$, we get the final expression:

$$\text{(4.10)}\quad \mathbf{Pr}[X \leq (1-\delta)\mu_x] \leq e^{-\delta^2 p^2 L}, \text{ with } E[X] = \mu_x$$

For two sequences that correctly overlap by $L = 2000$, the probability that their alignment score is more than 10% ($\delta = 0.1$) below the mean is $\leq 5.30 \times 10^{-7}$. BELLA, with an x-drop value of $x = 50$ and an adaptive threshold derived from the scoring matrix, and the cutoff rate set to $\delta = 0.1$, achieves high values for recall and precision among state-of-the-art software tools.

5 Experimental Setting

The datasets used for evaluation are shown in Table 1. The selected genomes vary in size and complexity, as runtime and accuracy depend on these features [20]. In addition, we include a dataset based on PacBio CCS technology that was sampled at varying depths. This technology is more accurate than PacBio CLR, but has higher cost and shorter read length (although it is still classified as a long read technology).

Recall, precision, F1 score, and runtime are used as performance metrics. Recall is defined as the fraction of true-positives from the aligner/overlapper to the total ground truth. Precision is the fraction of true positives from the aligner/overlapper to the total number of elements found by the aligner/overlapper. F1 score is the harmonic mean of precision and recall.

A read pair is considered true-positive if the sequences align for at least 2 kb in the reference genome. The threshold $t = 2$ kb is derived from the procedure proposed by Heng Li [19] and the ground truth is generated using Minimap2. A description of our evaluation procedure and ground truth generation can be found in the supplementary material of our preprint [15].

In addition, we report preliminary assembly results on simulated datasets obtained by coupling the overlappers/aligners with the Miniasm assembler [19]. As metrics for assembler quality, we use the number of contigs, the number of misassemblies, N50, and the total assembly length. A contig is defined as a set of overlapping reads that together represent a consensus region of the genome. A misassembly is a position in a contig whose flanking sequences are more than 1kbps apart. N50 is a measure of assembly contiguity and is defined as the minimum contig length to cover 50% of the genome.

Results were collected on a dual-socket computer with two 20-core Intel Xeon Gold 6148 CPU ("Skylake") processors, each running at 2.4 GHz with 384 GB DDR4 2400 MHz memory, using 2 threads per core (80 threads total). To run MHAP `v2.1.3` on *C. elegans* 40X and *C. elegans* 20X, we increased the Java heap space to our largest machine, as the default setting of 32 GB resulted in out-of-memory failures. DALIGNER results are reported only for real PacBio CLR data because it requires single-cell PacBio sequencing data. For all simulated datasets and real Pacbio CCS datasets, DALIGNER quickly failed and produced the associated error "`Pacbio header line format error`". Our attempt to reformat the data also failed.

6 Results

BELLA is evaluated against several state-of-the-art long read overlap detection and alignment software (see Section 2), using both synthetic and real PacBio data. The synthetic data were generated using PBSIM [27] with an error rate of 15%. The advantage of the synthetic data is that the ground truth is known. Table 2 and Table 3 show the results on synthetic and real data, respectively, in terms of accuracy and runtime. The last column of each table indicates whether the respective overlapper also performs nucleotide-level alignment. The full runtime breakdown for BELLA is shown in Figure 7 in the supplementary material of our preprint [15].

Table 4 shows the assembly results on simulated data. Each overlapper was coupled with the Miniasm assembler [19]. If the tool also computes alignment at the nucleotide level by default, as BELLA does, the post-alignment data is used as input to Miniasm.

Table 2 shows that MECAT trades recall for precision and achieves the highest precision, but misses a large number of the true overlaps. In contrast, BELLA, Minimap2, and BLASR were consistently strong in both precision and recall (generally over 80%), but BLASR had a much higher computational cost (on average 2.6× slower than BELLA). The F1 score of BELLA is consistently higher than that of competing software, with the exception of Minimap2, which had a slight improvement of 1.1% in three of four data sets, while BELLA had an improvement of 1.2% over Miniamp2 in *C. elegans* 20X.

Table 2: Recall, precision, F1 score, and time comparison (**synthetic data**). The last column indicates whether the tool computes alignments. **Bold font** indicates best performance, underlined font indicates second best. DALIGNER was not run on synthetic datasets.

Dataset	Overlapper	Recall	Precision	F1 Score	Time (s)	Alignment
P. aeruginosa 30X	BELLA	97.66	89.68	93.50	140.43	Y
	BLASR	86.86	90.54	88.66	230.97	Y
	MECAT	38.40	**95.20**	54.72	21.76	N
	Minimap2	**99.10**	88.83	**93.69**	**17.76**	N
	MHAP	72.68	63.42	67.74	72.72	N
V. vulnificus 30X	BELLA	**97.27**	84.05	90.18	42.76	Y
	BLASR	87.31	84.74	86.01	179.51	Y
	MECAT	43.53	88.89	58.44	17.20	N
	Minimap2	96.71	**89.33**	**92.87**	**12.93**	N
	MHAP	74.52	45.10	56.19	70.21	N
A. baumannii 30X	BELLA	**97.54**	84.90	90.78	44.69	Y
	BLASR	89.51	84.58	86.98	152.76	Y
	MECAT	46.31	**90.39**	61.25	15.65	N
	Minimap2	96.89	85.79	**91.01**	**10.06**	N
	MHAP	76.88	28.79	41.89	69.02	N
C. elegans 20X	BELLA	91.80	88.02	**89.87**	2,352.24	Y
	BLASR	95.61	78.19	86.02	3,655.12	Y
	MECAT	13.45	**95.09**	23.57	222.10	N
	Minimap2	**95.76**	82.84	88.83	**194.77**	N
	MHAP	82.57	6.41	11.90	5,353.30	N

Table 3: Recall, precision, F1 score, and time comparison (**real data**). BLASR result for *C. elegans* 40X is not reported as BLASR `v5.1` does not accept `fastq` larger than 4 GB.

Dataset	Overlapper	Recall	Precision	F1 Score	Time (s)	Alignment
E. coli (Sample)	BELLA	82.66	**85.69**	**84.15**	60.94	Y
	DALIGNER	89.97	62.62	73.84	**8.70**	Y
	BLASR	77.64	79.64	78.63	160.05	Y
	MECAT	78.41	78.71	78.56	24.45	N
	Minimap2	**91.40**	76.36	83.21	16.57	N
	MHAP	79.71	66.93	72.76	43.67	N
E. coli	BELLA	65.08	71.22	68.01	374.37	Y
	DALIGNER	82.18	54.50	65.54	**58.50**	Y
	BLASR	35.41	72.01	47.48	715.19	Y
	MECAT	54.61	**72.69**	62.37	84.21	N
	Minimap2	**80.68**	62.30	**70.30**	107.76	N
	MHAP	67.84	44.60	53.81	287.66	N
E. coli (CCS Sample)	BELLA	96.32	**99.84**	**98.05**	66.49	Y
	BLASR	92.38	97.30	94.77	491.49	Y
	MECAT	98.21	88.39	93.04	66.12	N
	Minimap2	98.90	58.34	73.39	51.78	N
	MHAP	**99.05**	38.29	55.23	**50.98**	N
E. coli (CCS)	BELLA	97.67	99.81	**98.73**	8,444.64	Y
	BLASR	9.11	**100.00**	16.70	11,058.86	Y
	MECAT	15.71	99.95	27.15	**289.38**	N
	Minimap2	98.83	69.94	81.91	5,828.72	N
	MHAP	**98.99**	38.80	55.75	2,277.66	N
C. elegans 40X	BELLA	75.43	73.81	**74.61**	9,042.36	Y
	DALIGNER	62.81	58.66	60.67	22,797.50	Y
	MECAT	73.05	**75.27**	74.14	**733.79**	N
	Minimap2	**94.13**	34.06	50.02	1,733.26	N
	MHAP	86.63	5.31	10.01	9,102.23	N

Minimap2 was the fastest tool for synthetic data and performed only overlap but no alignment.

Table 3 shows that while BLASR performed reasonably well on synthetic data, it had a lower hit rate than other software on real data. BLASR did not run on *C. elegans* 40X because its latest version (`v5.1`) does not accept `fastq` larger than 4 GB[1]). DALIGNER proved to be the fastest of the tools on *E. coli* 30X and *E. coli* 100X, but its performance drops dramatically when moving to a larger dataset, where DALIGNER has the worst runtime, 2.5× slower than BELLA, which also performs alignment. BELLA's F1 score outperformed competing software except Minimap2 on *E. coli* 100X. It is noteworthy that the precision and F1 score of BELLA is significantly better for PacBio CCS data, the sequencing data with lower error rates and shorter read length, than for competing software. For the CCS data, the δ parameter of BELLA was increased from 0.1 to 0.7.

In Table 4, we show BELLA improved assembly quality compared to competing software. MECAT produced fewer contigs than BELLA on *C. elegans* 20X, but its N50 and assembly length are significantly smaller, implying that MECAT did not retain enough true overlaps for assembly. Miniasm produced no assembly when coupled with MHAP and BLASR.

7 Discussion

BELLA proposes a computationally efficient and accurate approach to overlap and alignment of noisy long reads based on mathematical models that minimize the cost of overlap detection while maximizing the retention of true overlaps. Tables 2 and 3 show the competitive accuracy of BELLA compared to the literature and demonstrate the effectiveness of the methods we introduced and implemented. BELLA's runtime is within the average of competing software, which is remarkable considering that we perform nucleotide-level alignment that is sufficiently accurate to facilitate downstream analysis.

For synthetic data, BELLA achieves both high recall and precision, consistently among the best. For real PacBio CLR data, BELLA's recall and precision are generally lower than for synthetic data, yet BELLA's F1 results are among the best and show performance stability that competing software often does not. Notably, BELLA has a 49.16% higher F1 score than Minimap2 for *C. elegans* 40X. BELLA's precision and F1 score on real CCS data appreciably outperform competing software. Overall, a good performer on one dataset becomes one of the worst on another, whereas BELLA's F1 score is consistently within 1.7% of the top entry.

Tables 2 and 3 show that BELLA achieves higher F1 score values on synthetic data and real CCS data compared to real CLR data. The way ground truth is generated could explain this behavior. On synthetic data, ground truth comes directly with the dataset itself. Thus, we know exactly where a read in the reference genome comes from and which other reads overlap with it. For real data, the positions of the reads in the reference are determined by mapping the

[1]https://github.com/PacificBiosciences/pbbioconda/issues/46

Table 4: Preliminary assembly results of synthetic datasets. The overlappers' outputs were translated into PAF format and paired with Miniasm [19]. DALIGNER was not run with synthetic datasets. Miniasm produced no output when paired with BLASR and MHAP, and neither tool produced misassemblies.

Dataset	Overlapper	Contigs	N50	Total Length
P. aeruginosa 30X 6,264,404 bp	BELLA	**39**	**299,124**	6,539,838
	MECAT	130	30,078	3,123,445
	Minimap2	118	84,638	6,368,698
V. vulnificus 30X 5,126,696 bp	BELLA	**39**	**201,956**	5,319,876
	MECAT	101	30,132	2,585,336
	Minimap2	105	58,711	4,941,291
A. baumannii 30X 4,335,793 bp	BELLA	**35**	**185,443**	4,486,557
	MECAT	88	33,248	2,234,729
	Minimap2	93	61,967	4,092,102
C. elegans 20X 100,286,401 bp	BELLA	2,792	**35,782**	82,763,749
	MECAT	**366**	10,661	2,398,741
	Minimap2	2,875	19,894	49,724,174

reads to the reference using Minimap2 in its "mapping mode". Intuitively, such a procedure is suboptimal, since there is no guarantee that Minimap2 correctly locates each read. BELLA could potentially find a better set of true overlaps than those identified by Minimap2. Given a uniformly covered genome, we observed that Minimap2 and other long read mappers tend to map reads to "hotspots" within a genome rather than mapping them evenly across the genome. This leads to uneven coverage and overestimation of overlaps by a factor of 1.14×. Recall beyond a certain point on real data would mean that the overlapper also overestimates the overlap cardinality. It is possible that the actual accuracy of BELLA is actually higher for real data. As future work, we plan to investigate these issues in more detail. This bias may not be present when mapping CCS data to the reference, as error rates are lower, making it easier for the mapper to find the correct position on the reference genome.

Table 4 provides insight into the beneficial impact of BELLA in a *de novo* assembly pipeline. Our methods, such as the reliable k-mer strategy, noticeably improve assembly contiguity compared to MECAT and Minimap2. Importantly, Miniasm produced no output when using MHAP or BLASR, leading us to emphasize that an assembler is often built with a particular overlapping tool in mind and programmed to exploit that tool's methods. To fully exploit the potential of BELLA, we plan to build our assembler on top of it.

8 Conclusion

Long-read sequencing technologies enable highly accurate reconstruction of complex genomes. Read overlapping is a major computational bottleneck in long read pipelines for genome analysis, such as genome assembly.

In this paper, we introduced BELLA, a computationally efficient and highly accurate long read-to-long read aligner and overlapper. BELLA uses a k-mer-based approach to detect overlaps between noisy long reads. Then we demonstrated the feasibility of the k-mer-based approach using a mathematical model based on Markov chains and showed the generality of such a model. BELLA provides a novel algorithm for pruning k-mers that are unlikely to be useful for overlap detection and whose presence would only add unnecessary computational cost. Our algorithm for reliably detecting k-mers explicitly maximizes the probability of keeping k-mers that belong to unique regions of the genome.

BELLA achieves fast overlap without sketching using sparse matrix multiplication (SpGEMM), implemented using high-performance software and libraries developed for this subroutine. Any novel sparse matrix format and multiplication algorithm would be applicable to overlap detection and would enable further performance improvements. Moreover, our SpGEMM approach is general and flexible enough that it can be coupled with any k-mer selection strategy. Our overlap detection is coupled with our newly developed seed-and-extend banded alignment algorithm. The optimal k-mer seed is chosen by our binning mechanism, which uses k-mer positions within a read pair to estimate the length of the overlap and to "bin" k-mers to form a consensus.

Finally, we developed a new method to separate true alignments from false positives as a function of the alignment score. This method shows that the probability of false positives decreases exponentially as the overlap length between sequences increases.

BELLA achieves consistently high accuracy scores compared to state-of-the-art tools on both synthetic and real-world data, while being performance competitive. BELLA significantly improves assembly results on synthetic data, validating our approach. Future work includes further characterization of real data properties, performance improvements, and the development of a complete *de novo* assembler built on BELLA.

Code: `https://github.com/PASSIONLab/BELLA`.

Acknowledgements

This work is supported by the Advanced Scientific Computing Research (ASCR) program within the Office of Science of the DOE under contract number DE-AC02-05CH11231. Resources from NERSC were used supported by the Office of Science of the DOE under Contract No. DE-AC02-05CH11231. This research was also supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

References

[1] D. Angluin and L. G. Valiant, *Fast probabilistic algorithms for hamiltonian circuits and matchings*, Journal of Computer and system Sciences, 18 (1979), pp. 155–193.

[2] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy, *Assembling large genomes with single-molecule sequencing and locality-sensitive hashing*, Nature biotechnology, 33 (2015), pp. 623–630.

[3] M. Besta, R. Kanakagiri, H. Mustafa, M. Karasikov, G. Rätsch, T. Hoefler, and E. Solomonik, *Communication-efficient jaccard similarity for high-performance distributed genome comparisons*, in 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2020, pp. 1122–1132.

[4] A. Buluç, T. Mattson, S. McMillan, J. Moreira, and C. Yang, *Design of the GraphBLAS API for C*, in IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2017, pp. 643–652.

[5] A. B. Carvalho, E. G. Dupim, and G. Goldstein, *Improved assembly of noisy long reads by k-mer validation*, Genome research, 26 (2016), pp. 1710–1720.

[6] M. J. Chaisson and G. Tesler, *Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory*, BMC bioinformatics, 13 (2012), p. 238.

[7] H. Chernoff et al., *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, The Annals of Mathematical Statistics, 23 (1952), pp. 493–507.

[8] J. Chu, H. Mohamadi, R. L. Warren, C. Yang, and I. Birol, *Innovations and challenges in detecting long read overlaps: an evaluation of the state-of-the-art*, Bioinformatics, 33 (2016), pp. 1261–1270.

[9] M. Deveci, C. Trott, and S. Rajamanickam, *Performance-portable sparse matrix-matrix multiplication for many-core architectures*, in IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2017, pp. 693–702.

[10] R. Edgar, *Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences*, PeerJ, 9 (2021), p. e10805.

[11] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, et al., *Real-time DNA sequencing from single polymerase molecules*, Science, 323 (2009), pp. 133–138.

[12] J. Gao, W. Ji, Z. Tan, and Y. Zhao, *A systematic survey of general sparse matrix-matrix multiplication*, arXiv preprint arXiv:2002.11273, (2020).

[13] F. Giordano, L. Aigrain, M. A. Quail, P. Coupland, J. K. Bonfield, R. M. Davies, G. Tischler, D. K. Jackson, T. M. Keane, J. Li, et al., *De novo yeast genome assemblies from MinION, PacBio and MiSeq platforms*, Scientific reports, 7 (2017), p. 3935.

[14] S. Goodwin, J. Gurtowski, S. Ethe-Sayers, P. Deshpande, M. C. Schatz, and W. R. McCombie, *Oxford nanopore sequencing, hybrid error correction, and de novo assembly of a eukaryotic genome*, Genome research, 25 (2015), pp. 1750–1756.

[15] G. Guidi, M. Ellis, D. Rokhsar, K. Yelick, and A. Buluç, *BELLA: Berkeley efficient long-read to long-read aligner and overlapper*, bioRxiv, (2020), p. 464420.

[16] W. Hoeffding, *Probability inequalities for sums of bounded random variables*, Journal of the American statistical association, 58 (1963), pp. 13–30.

[17] J. Kepner and J. Gilbert, *Graph algorithms in the language of linear algebra*, SIAM, 2011.

[18] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, *Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation*, Genome research, 27 (2017), pp. 722–736.

[19] H. Li, *Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences*, Bioinformatics, 32 (2016), pp. 2103–2110.

[20] Z. Li, Y. Chen, D. Mu, J. Yuan, Y. Shi, H. Zhang, J. Gan, N. Li, X. Hu, B. Liu, et al., *Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph*, Briefings in functional genomics, 11 (2012), pp. 25–37.

[21] Y. Lin, J. Yuan, M. Kolmogorov, M. W. Shen, M. Chaisson, and P. A. Pevzner, *Assembly of long error-prone reads using de bruijn graphs*, Proceedings of the National Academy of Sciences, 113 (2016), pp. E8396–E8405.

[22] A. Markov, *Extension of the limit theorems of probability theory to a sum of variables connected in a chain*, (1971).

[23] E. W. Myers, *An O(ND) difference algorithm and its variations*, Algorithmica, 1 (1986), pp. 251–266.

[24] G. Myers, *Efficient local alignment discovery amongst noisy long reads*, in International Workshop on Algorithms in Bioinformatics, Springer, 2014, pp. 52–67.

[25] N. Nagarajan and M. Pop, *Parametric complexity of sequence assembly: theory and applications to next generation sequencing*, Journal of computational biology, 16 (2009), pp. 897–908.

[26] Y. Nagasaka, S. Matsuoka, A. Azad, and A. Buluç, *Performance optimization, modeling and analysis of sparse matrix-matrix products on multi-core and many-core processors*, Parallel Computing, (2019), p. 102545.

[27] Y. Ono, K. Asai, and M. Hamada, *PBSIM: Pacbio reads simulator—toward accurate genome assembly*, Bioinformatics, 29 (2012), pp. 119–121.

[28] A. M. Phillippy, M. C. Schatz, and M. Pop, *Genome assembly forensics: finding the elusive misassembly*, Genome biology, 9 (2008), p. R55.

[29] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount,

and J. A. Yorke, *Reducing storage requirements for biological sequence comparison*, Bioinformatics, 20 (2004), pp. 3363–3369.
[30] J. T. Simpson and R. Durbin, *Efficient de novo assembly of large genomes using compressed data structures*, Genome research, 22 (2012), pp. 549–556.
[31] C.-L. Xiao, Y. Chen, S.-Q. Xie, K.-N. Chen, Y. Wang, Y. Han, F. Luo, and Z. Xie, *Mecat: fast mapping, error correction, and de novo assembly for single-molecule sequencing reads*, nature methods, 14 (2017), p. 1072.
[32] W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, and B. Shen, *A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies*, PloS one, 6 (2011), p. e17915.

Parallel Clique Counting and Peeling Algorithms *

Jessica Shi† Laxman Dhulipala† Julian Shun†

Abstract

We present a new parallel algorithm for k-clique counting/listing that has polylogarithmic span (parallel time) and is work-efficient (matches the work of the best sequential algorithm) for sparse graphs. Our algorithm is based on computing low out-degree orientations, which we present new linear-work and polylogarithmic-span algorithms for computing in parallel. We also present new parallel algorithms for producing unbiased estimations of clique counts using graph sparsification. Finally, we design two new parallel work-efficient algorithms for approximating the k-clique densest subgraph, the first of which is a $1/k$-approximation and the second of which is a $1/(k(1+\epsilon))$-approximation and has polylogarithmic span. Our first algorithm does not have polylogarithmic span, but we prove that it solves a P-complete problem.

In addition to the theoretical results, we also implement the algorithms and propose various optimizations to improve their practical performance. On a 30-core machine with two-way hyper-threading, our algorithms achieve 13.23–38.99x and 1.19–13.76x self-relative parallel speedup for k-clique counting and k-clique densest subgraph, respectively. Compared to the state-of-the-art parallel k-clique counting algorithms, we achieve up to 9.88x speedup, and compared to existing implementations of k-clique densest subgraph, we achieve up to 11.83x speedup. We are able to compute the 4-clique counts on the largest publicly-available graph with over two hundred billion edges for the first time.

1 Introduction

Finding k-cliques in a graph is a fundamental graph-theoretic problem with a long history of study both in theory and practice. In recent years, k-clique counting and listing have been widely applied in practice due to their many applications, including in learning network embeddings [43], understanding the structure and formation of networks [59, 56], identifying dense subgraphs for community detection [53, 48, 21, 26], and graph partitioning and compression [22].

For sparse graphs, the best known sequential algorithm is by Chiba and Nishizeki [12], and requires $O(m\alpha^{k-2})$ work (number of operations), where α is the arboricity of the graph.[1] The state-of-the-art clique parallel k-clique counting algorithm is KCLIST [15], which achieves the same work bound, but does not have a strong theoretical bound on the span (parallel time). Furthermore, KCLIST as well as existing parallel k-clique counting algorithms have limited scalability for graphs with more than a few hundred million edges, but real-world graphs today frequently contain billions to hundreds of billions of edges [34].

k-clique Counting. In this paper, we design a new parallel k-clique counting algorithm, ARB-COUNT that matches the work of Chiba-Nishezeki, has polylogarithmic span, and has improved space complexity compared to KCLIST. Our algorithm is able to significantly outperform KCLIST and other competitors, and scale to larger graphs than prior work. ARB-COUNT is based on using low out-degree orientations of the graph to reduce the total work. Assuming that we have a low out-degree ranking of the graph, we show that for a constant k we can count or list all k-cliques in $O(m\alpha^{k-2})$ work, and $O(k\log n + \log^2 n)$ span with high probability (*whp*),[2] where m is the number of edges in the graph and α is the arboricity of the graph. Having work bounds parameterized by α is desirable since most real-world graphs have low arboricity [17]. Theoretically, ARB-COUNT requires $O(\alpha)$ extra space per processor; in contrast, the KCLIST algorithm requires $O(\alpha^2)$ extra space per processor. Furthermore, KCLIST does not achieve polylogarithmic span.

We also design an approximate k-clique counting algorithm based on counting on a sparsified graph. We show in the full version of the paper that our approximate algorithm produces unbiased estimates and runs in $O(pm\alpha^{k-2} + m)$ work and $O(k\log n + \log^2 n)$ span *whp* for a sampling probability of p.

Parallel Ranking Algorithms. We present two new parallel algorithms for efficiently ranking the vertices, which we use for k-clique counting. We show that a distributed algorithm by Barenboim and Elkin [5] can be implemented in linear work and polylogarithmic span. We also parallelize an external-memory algorithm by Goodrich and Pszona [25] and obtain the same complexity bounds. We believe that our parallel ranking algorithms may be of independent interest, as many other subgraph finding algorithms use low out-degree

*The full version of this paper is available at https://arxiv.org/abs/2002.10047.

†MIT CSAIL, Cambridge, MA (jeshi@mit.edu, laxman@mit.edu, jshun@mit.edu)

[1] A graph has arboricity α if the minimum number of spanning forests needed to cover the graph is α.

[2] We say $O(f(n))$ ***with high probability (whp)*** to indicate $O(cf(n))$ with probability at least $1 - n^{-c}$ for $c \geq 1$, where n is the input size.

orderings (e.g., [25, 41, 28]).

Peeling and k-Clique Densest Subgraph. We also present new parallel algorithms for the k-clique densest subgraph problem, a generalization of the densest subgraph problem that was first introduced by Tsourakakis [53]. This problem admits a natural $1/k$-approximation by peeling vertices in order of their incident k-clique counts. We present a parallel peeling algorithm, ARB-PEEL, that peels all vertices with the lowest k-clique count on each round and uses ARB-COUNT as a subroutine. The expected amortized work of ARB-PEEL is $O(m\alpha^{k-2} + \rho_k(G)\log n)$ and the span is $O(\rho_k(G)k\log n + \log^2 n)$ *whp*, where $\rho_k(G)$ is the number of rounds needed to completely peel the graph. We also prove in the full version of the paper that the problem of obtaining the hierarchy given by this process is P-complete for $k > 2$, indicating that a polylogarithmic-span solution is unlikely.

Tsourakakis also shows that naturally extending the Bahmani et al. [4] algorithm for approximate densest subgraph gives an $1/(k(1+\epsilon))$-approximation in $O(\log n)$ parallel rounds, although they were not concerned about work. We present an $O(m\alpha^{k-2})$ work and polylogarithmic-span algorithm, ARB-APPROX-PEEL, for obtaining a $1/(k(1+\epsilon))$-approximation to the k-clique densest subgraph problem. We obtain this work bound using our k-clique algorithm as a subroutine. Danisch et al. [15] use their k-clique counting algorithm as a subroutine to implement these two approximation algorithms for k-clique densest subgraph, but their implementations do not have provably-efficient bounds.

Experimental Evaluation. We present implementations of our algorithms that use various optimizations to achieve good practical performance. We perform a thorough experimental study on a 30-core machine with two-way hyper-threading and compare to prior work. We show that on a variety of real-world graphs and different k, our k-clique counting algorithm achieves 1.31–9.88x speedup over the state-of-the-art parallel KCLIST algorithm [15] and self-relative speedups of 13.23–38.99x. We also compared our k-clique counting algorithm to other parallel k-clique counting implementations including Jain and Seshadhri's PIVOTER [28], Mhedhbi and Salihoglu's worst-case optimal join algorithm (WCO) [35], Lai *et al.*'s implementation of a binary join algorithm (BINARYJOIN) [30], and Pinar *et al.*'s ESCAPE [41], and demonstrate speedups of up to several orders of magnitude.

Furthermore, by integrating state-of-the-art parallel graph compression techniques, we can process graphs with tens to hundreds of billions of edges, significantly improving on the capabilities of existing implementations. *As far as we know, we are the first to report* 4*-clique counts for Hyperlink2012, the largest publicly-available graph, with over two hundred billion undirected edges.*

We study the accuracy-time tradeoff of our sampling algorithm, and show that is able to approximate the clique counts with 5.05% error 5.32–6573.63 times more quickly than running our exact counting algorithm on the same graph. We compare our sampling algorithm to Bressan *et al.*'s serial MOTIVO [11], and demonstrate 92.71–177.29x speedups. Finally, we study our two parallel approximation algorithms for k-clique densest subgraph and show that our we are able to outperform KCLIST by up to 29.59x and achieve 1.19–13.76x self-relative speedup. We demonstrate up to 53.53x speedup over Fang *et al.*'s serial COREAPP [21] as well.

The contributions of this paper are as follows:

(**1**) A parallel algorithm with $O(m\alpha^{k-2})$ and polylogarithmic span *whp* for k-clique counting.
(**2**) Parallel algorithms for low out-degree orientations with $O(m)$ work and $O(\log^2 n)$ span *whp*.
(**3**) An $O(m\alpha^{k-2})$ amortized expected work parallel algorithm for computing a $1/k$-approximation to the k-clique densest subgraph problem, and an $O(m\alpha^{k-2})$ work and polylogarithmic-span *whp* algorithm for computing a $1/(k(1+\epsilon))$-approximation.
(**4**) Optimized implementations of our algorithms that achieve significant speedups over existing state-of-the-art methods, and scale to the largest publicly-available graphs.

Our code is publicly available at: https://github.com/ParAlg/gbbs/tree/master/benchmarks/CliqueCounting.

2 Preliminaries

Graph Notation. We consider graphs $G = (V, E)$ to be simple and undirected, and let $n = |V|$ and $m = |E|$. For any vertex v, $N(v)$ denotes the neighborhood of v and $\deg(v)$ denotes the degree of v. If there are multiple graphs, $N_G(v)$ denotes the neighborhood of v in G. For a directed graph DG, $N(v) = N_{DG}(v)$ denotes the out-neighborhood of v in DG. For analysis, we assume that $m = \Omega(n)$. The ***arboricity*** ***(α)*** of a graph is the minimum number of spanning forests needed to cover the graph. α is upper bounded by $O(\sqrt{m})$ and lower bounded by $\Omega(1)$ [12].

A ***k-clique*** is a subgraph $G' \subseteq G$ of size k where all $\binom{k}{2}$ edges are present. The ***k-clique densest subgraph*** is a subgraph $G' \subseteq G$ that maximizes across all subgraphs the ratio between the number of k-cliques induced by vertices in G' and the number of vertices in G' [53]. An ***c-orientation*** of an undirected graph is a total ordering on the vertices, where the oriented out-degree of each vertex (the number of its neighbors higher than it in the ordering) is bounded by c.

Model of Computation. For analysis, we use the work-span model [29, 13]. The ***work*** W of an algorithm is the total number of operations, and the ***span*** S is the longest dependency path. We can execute a parallel computation in $W/P + S$ running time using P processors [9]. We aim for ***work-efficient*** parallel algorithms in this model, that is, an algorithm with work complexity that asymptotically matches the best-known sequential time complexity for the problem. We assume concurrent reads and writes and atomic adds are

supported in the model in $O(1)$ work and span.

Parallel Primitives. We use the following primitives. ***Reduce-Add*** takes as input a sequence A of length n, and returns the sum of the entries in A. ***Prefix sum*** takes as input a sequence A of length n, an identity ε, and an associative binary operator $\oplus$, and returns the sequence B of length n where $B[i] = \bigoplus_{j<i} A[j] \oplus \varepsilon$. ***Filter*** takes as input a sequence A of length n and a predicate function f, and returns the sequence B containing $a \in A$ such that $f(a)$ is true, in the same order that these entries appeared in A. These primitives take $O(n)$ work and $O(\log n)$ span [29].

We also use ***parallel integer sort***, which sorts n integers in the range $[1, n]$ in $O(n)$ work *whp* and $O(\log n)$ span *whp* [42]. We use ***parallel hash tables*** that support n operations (insertions, deletions, and membership queries) in $O(n)$ work and $O(\log n)$ span *whp* [24]. Given hash tables $\mathcal{T}_1$ and $\mathcal{T}_2$ containing n and m elements respectively, the intersection $\mathcal{T}_1 \cap \mathcal{T}_2$ can be computed in $O(\min(n, m))$ work and $O(\log(n + m))$ span *whp*.

Parallel Bucketing. A ***parallel bucketing structure*** maintains a mapping from keys to buckets, which we use to group vertices by their k-clique counts in our k-clique densest subgraph algorithms. The bucket value of keys can change, and the structure updates the bucket containing these keys.

In practice, we use the bucketing structure by Dhulipala et al. [16]. However, for theoretical purposes, we use the batch-parallel Fibonacci heap by Shi and Shun [49], which supports b insertions in $O(b)$ amortized expected work and $O(\log n)$ span *whp*, b updates in $O(b)$ amortized work and $O(\log^2 n)$ span *whp*, and extracts the minimum bucket in $O(\log n)$ amortized expected work and $O(\log n)$ span *whp*.

Graph Storage. In our implementations, we store our graphs in compressed sparse row (CSR) format, which requires $O(m + n)$ space. For large graphs, we compress the edges for each vertex using byte codes that can be decoded in parallel [50]. For our theoretical bounds, we assume that graphs are represented in an adjacency hash table, where each vertex is associated with a parallel hash table of its neighbors.

3 Clique Counting

In this section, we present our main algorithms for counting k-cliques. We describe our parallel algorithm for low out-degree orientations in Section 3.1, our parallel k-clique counting algorithm in Section 3.2, and practical optimizations in Section 3.4. We discuss briefly our parallel approximate counting algorithm in Section 3.3.

3.1 Low Out-degree Orientation (Ranking) Recall that an c-orientation of an undirected graph is a total ordering on the vertices, where the oriented out-degree of each vertex (the number of its neighbors higher than it in the ordering) is bounded by c. Although this problem has been widely studied in other contexts, to the best of our knowledge, we are not aware of any previous work-efficient parallel algorithms for solving this problem. We show that the Barenboim-Elkin and Goodrich-Pszona algorithms, which are efficient in the CONGEST and I/O models of computation respectively, lead to work-efficient low-span algorithms.

Both algorithms take as input a user-defined parameter ϵ. The Barenboim-Elkin algorithm also requires a parameter, α, which is the arboricity of the graph (or an estimate of the arboricity). As an estimate of the arboricity, we use the approximate densest-subgraph algorithm from [17], which yields a $(2+\epsilon)$-approximation and takes $O(m+n)$ work and $O(\log^2 n)$ span. The algorithms peel vertices in rounds until the graph is empty; the peeled vertices are appended to the end of ordering. Both algorithms peel a constant fraction of the vertices per round. For the Goodrich-Pszona algorithm, an $\epsilon/(2+\epsilon)$ fraction of vertices are removed on each round, so the algorithm finishes in $O(\log n)$ rounds. The Barenboim-Elkin algorithm peels vertices with induced degree less than $(2+\epsilon)\alpha$ on each round. By definition of arboricity, there are at most $n\alpha/d$ vertices with degree at least d. Thus, the number of vertices with degree at least $(2+\epsilon)\alpha$ is at most $n/(2+\epsilon)$, and a constant fraction of the vertices have degree at most $(2+\epsilon)\alpha$. Since a subgraph of a graph with arboricity α has arboricity at most α, each round peels at least a constant fraction of remaining vertices, and the algorithm terminates in $O(\log n)$ rounds. We provide pseudocode for the algorithms in the full version of the paper.

For the c-orientation given by the Barenboim-Elkin algorithm, vertices have out-degree less than $(2+\epsilon)\alpha$ by construction. For the c-orientation given by the Goodrich-Pszona algorithm, the number of vertices with degree at least $(2+\epsilon)\alpha$ is at most $n/(2+\epsilon)$, so the $\epsilon/(2+\epsilon)$ fraction of the lowest degree vertices must have degree less than $(2+\epsilon)\alpha$.

We implement each round of the Goodrich-Pszona algorithm using parallel integer sorting to find the $\epsilon/(2+\epsilon)$ fraction of vertices with lowest induced degree. Our parallelization of Barenboim-Elkin uses a parallel filter to find the set of vertices to peel. We can implement a round in both algorithms in linear work in the number of remaining vertices, and $O(\log n)$ span. We obtain the following theorem, which we prove in the full version of the paper.

THEOREM 3.1. *The Goodrich-Pszona and Barenboim-Elkin algorithms compute $O(\alpha)$-orientations in $O(m)$ work (whp for Goodrich-Pszona), $O(\log^2 n)$ span (whp for Goodrich-Pszona), and $O(m)$ space.*

Finally, in the rest of this paper, we direct graphs in CSR format after computing an orientation, which can be done in $O(m)$ work and $O(\log n)$ span using prefix sum and filter.

3.2 Counting algorithm Our algorithm for k-clique counting is shown as ARB-COUNT in Algorithm 1. On Line 12,

Algorithm 1 Parallel k-clique counting algorithm

```
1: procedure REC-COUNT-CLIQUES(DG, I, ℓ)
2:     ▷ I is the set of potential neighbors to complete the clique, and ℓ is the recursive level
3:     if ℓ = 1 then return |I|                                  ▷ Base case
4:     Initialize T to store clique counts per vertex in I
5:     parfor v in I do
6:         I' ← INTERSECT(I, N_DG(v))                            ▷ Intersect I with directed neighbors of v
7:         t' ← REC-COUNT-CLIQUES(DG, I', ℓ − 1)
8:         Store t' in T
9:     t ← REDUCE-ADD(T)                                         ▷ Sum clique counts in T
10:    return t
11: procedure ARB-COUNT(G = (V, E), k, ORIENT)
12:    DG ← ORIENT(G) ▷ Apply a user-specified orientation algorithm
13:    return REC-COUNT-CLIQUES(DG, V, k)
```

ARB-COUNT first directs the edges of G such that every vertex has out-degree $O(\alpha)$, as described in Section 3.1. Then, it calls a recursive subroutine REC-COUNT-CLIQUES that takes as input the directed graph DG, candidate vertices I that can be added to a clique, and the number of vertices ℓ left to complete a k-clique (Line 13). With every recursive call to REC-COUNT-CLIQUES, a new candidate vertex v from I is added to the clique and I is pruned to contain only out-neighbors of v (Line 6). REC-COUNT-CLIQUES terminates when precisely one vertex is needed to complete the k-clique, in which the number of vertices in I represents the number of completed k-cliques (Line 3). The counts obtained from recursive calls are aggregated using a REDUCE-ADD and returned (Lines 9–10).

Finally, by construction, ARB-COUNT and REC-COUNT-CLIQUES can be easily modified to store k-clique counts per vertex. We append -V to indicate the corresponding subroutines that store counts per vertex, which are used in our peeling algorithms. Similarly, ARB-COUNT can be modified to support k-clique listing.

Complexity Bounds. Aside from the initial call to REC-COUNT-CLIQUES which takes $I = V$, in subsequent calls, the size of I is bounded by $O(\alpha)$. This is because at every recursive step, I is intersected with the out-neighbors of some vertex v, which is bounded by $O(\alpha)$. The additional space required by ARB-COUNT per processor is $O(\alpha)$, and since the space is allocated in a stack-allocated fashion, we can bound the total additional space by $O(P\alpha)$ on P processors when using a work-stealing scheduler [8]. Thus, the total space for ARB-COUNT is $O(m + P\alpha)$. In contrast, the KCLIST algorithm requires $O(m + P\alpha^2)$ space.

Moreover, considering the first call to REC-COUNT-CLIQUES, the total work of INTERSECT is given by $O(m)$ *whp*, because the sum of the degrees of each vertex is bounded by $O(m)$. Also, using a parallel adjacency hash table, the work of INTERSECT in each subsequent recursive step is given by the minimum of $|I|$ and $|N_{DG}(v)|$, and thus is bounded by $O(\alpha)$ *whp*. We recursively call REC-COUNT-CLIQUES k times as ℓ ranges from 1 to k, but the first call involves a trivial intersect where we retrieve all directed neighbors of v, and the final recursive call returns immediately with $|I|$. Hence, we have $k - 2$ recursive steps that call INTERSECT non-trivially, and so in total, ARB-COUNT takes $O(m\alpha^{k-2})$ work *whp*.

The span of ARB-COUNT is defined by the span of INTERSECT and REDUCE-ADD in each recursive call. As discussed in Section 2, the span of INTERSECT is $O(\log n)$ *whp*, due to the use of the parallel hash tables, and the span of REDUCE-ADD is $O(\log n)$. Thus, since we have $k - 2$ recursive steps with $O(\log n)$ span, and taking into account the $O(\log^2 n)$ span *whp* in orienting the graph, ARB-COUNT takes $O(k \log n + \log^2 n)$ span *whp*. ARB-COUNT-V obtains the same work and span bounds as ARB-COUNT, since the atomic add operations do not increase the work or span. The total complexity of k-clique counting is as follows.

THEOREM 3.2. ARB-COUNT *takes* $O(m\alpha^{k-2})$ *work and* $O(k \log n + \log^2 n)$ *span whp, using* $O(m + P\alpha)$ *space on* P *processors.*

3.3 Sampling We discuss in the full version of the paper a technique, colorful sparsification, that allows us to produce approximate k-clique counts, based on previous work on approximate triangle and butterfly (biclique) counting [39, 45]. The technique uses our k-clique counting algorithm (Algorithm 1) as a subroutine, and we prove the following theorem in the full version of the paper.

THEOREM 3.3. *Our sampling algorithm with parameter* $p = 1/c$ *gives an unbiased estimate of the global* k*-clique count and takes* $O(pm\alpha^{k-2} + m)$ *work and* $O(k \log n + \log^2 n)$ *span whp, and* $O(m + P\alpha)$ *space on* P *processors.*

3.4 Practical Optimizations We now introduce practical optimizations that offer tradeoffs between performance and space complexity. First, in the initial call to REC-COUNT-CLIQUES, for each v, we construct the induced subgraph on $N_{DG}(v)$ and replace DG with this subgraph in later recursive levels. Thus, later recursive levels can skip edges that have already been pruned in the first level. Because the out-degree of each vertex is bounded above by $O(\alpha)$, we require $O(\alpha^2)$ extra space per processor to store these induced subgraphs.

Moreover, as mentioned in Section 2, we store our graphs (and induced subgraphs) in CSR format. To efficiently intersect the candidate vertices in I with the requisite out-neighbors, we relabel vertices in the induced subgraph constructed in the second level of recursion to be in the range $[0, \ldots, O(\alpha)]$, and then use an array of size $O(\alpha)$ to mark vertices in I. For each vertex I, we check if its out-neighbors are marked in our array to perform INTERSECT.

While this would require $O(k\alpha)$ extra space per processor to maintain a size $O(\alpha)$ array per recursive call, we find

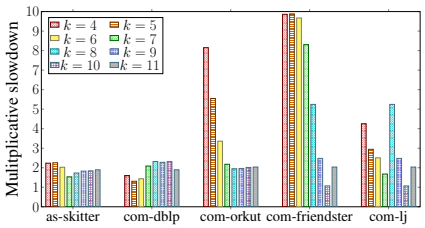


Figure 1: Multiplicative slowdowns of KCLIST's parallel k-clique counting implementation, compared to ARB-COUNT. The best runtimes between node and edge parallelism for KCLIST and ARB-COUNT, and among different orientations for ARB-COUNT are used.

that in practice, parallelizing up to the first two recursive levels is sufficient. Subsequent recursive calls are sequential, so we can reuse the array between recursive calls by using the labeling scheme from Chiba and Nishizeki's serial k-clique counting algorithm [12]. We record the recursive level ℓ in our array for each vertex in I, perform INTERSECT by checking if the out-neighbors have been marked with ℓ in the array, and then reset the marks. This allows us to use only $O(\alpha)$ extra space per processor to perform INTERSECT operations.

In our implementation, ***node parallelism*** refers to parallelizing only the first recursive level and ***edge parallelism*** refers to parallelizing only the first two recursive levels. These correspond with the ideas of node and edge parallelism in Danisch et al.'s KCLIST algorithm [15]. We also implemented dynamic parallelism, where more recursive levels are parallelized, but this was slower in practice—further parallelization did not mitigate the parallel overhead introduced.

Finally, for the intersections on the second recursive level (the first set of non-trivial intersections), it is faster in practice to use an array marking vertices in $N_{DG}(v)$. If we let $I_1 = N_{DG}(v)$ denote the set of neighbors obtained after the first recursive level, then to obtain the vertices in I_2 in the second level, we use a size n array to mark vertices in I_1 and perform a constant-time lookup to determine for $u \in I_1$, which out-neighbors $u' \in N_{DG}(u)$ are also in I_1; these u' form I_2. Past the second level, we relabel vertices in the induced subgraph as mentioned above and only require the $O(\alpha)$ array for intersections. Thus, we use linear space per processor for the second level of recursion only.

In total, the space complexity for intersecting in the second level of recursion and storing the induced subgraph on $N_{DG}(v)$ dominates, and so we use $O(\max(n, \alpha^2))$ extra space per processor.

3.5 Comparison to KCLIST Some of the practical optimizations for ARB-COUNT overlap with those in KCLIST [15]. Specifically, KCLIST also stores the induced subgraph on $N_{DG}(v)$, offers node and edge parallelism options, and uses a size n array to mark vertices to perform intersections. However, ARB-COUNT is fundamentally different due to the low out-degree orientation and because it does not inherently require labels or subgraphs stored between recursive levels.

Notably, the induced subgraph that ARB-COUNT computes at the first level of recursion takes $O(\alpha^2)$ space per processor because of the low out-degree orientation, whereas KCLIST takes $O(n^2)$ space per processor for their induced subgraph. Then, ARB-COUNT further saves on space and computation by maintaining only the subgraph computed from the first level of recursion to intersect with vertices in later recursive levels, which is solely possible due to the low out-degree orientation, whereas KCLIST necessarily recomputes an induced subgraph on every recursive level. As a result, ARB-COUNT is also able to compute intersections using only an array of size $O(\alpha)$ per recursive level, whereas KCLIST requires an array of size $O(n)$ per level.

In total, KCLIST uses $O(n^2)$ extra space per processor, whereas ARB-COUNT uses $O(\max(n, \alpha^2))$ extra space per processor. Compared to KCLIST, ARB-COUNT has lower memory footprint, span, and constant factors in the work, which allow us to achieve speedups between 1.31–9.88x over KCLIST's best parallel runtimes and which allows us to scale to the largest publicly-available graphs, considering the best optimizations, as shown in Figure 1. Note that for large k on large graphs, the multiplicative slowdown decreases because KCLIST incurs a large preprocessing overhead due to the large induced subgraph computed in the first recursive level, which is mitigated by higher counting times as k increases. These results are discussed further in Section 5.1.

4 k-Clique Densest Subgraph

We present our new work-efficient parallel algorithms for approximating the k-clique densest subgraph problem, using the vertex peeling algorithm.

4.1 Vertex Peeling

Algorithm. Algorithm 2 presents ARB-PEEL, our parallel algorithm for vertex peeling, which also gives a $1/k$-approximate to the k-clique densest subgraph problem. An example of this peeling process is shown in Figure 2. The algorithm uses ARB-COUNT to compute the initial per-vertex k-clique counts (C), which are given as an argument to the algorithm. The algorithm first initializes a parallel bucketing structure that stores buckets containing sets of vertices, where all vertices in the same bucket have the same k-clique count (Line 11). Then, while not all of the vertices have been peeled, it repeatedly extracts the vertices with the lowest induced k-clique count (Line 14), updates the count of the number of peeled vertices (Line 15), and updates the k-clique counts of vertices that are not yet finished that participate in k-cliques with the peeled vertices (Line 16). UPDATE also returns the number of k-cliques that were removed as well as the set of vertices whose k-clique counts changed. We then update the buckets of the vertices whose k-clique counts changed (Line 17). Lastly, the algorithm checks if the new induced subgraph has higher density than the current maximum density,

Algorithm 2 Parallel vertex peeling algorithm

1: **procedure** UPDATE($G = (V, E), k, DG, C, A$)
2: Initialize T to store k-clique counts per vertex in A
3: **parfor** v in A **do**
4: $I \leftarrow \{u \mid u \in N_G(v)$ and u has not been previously peeled or $u \in A$ and $u \in N_{DG}(v)\}$ ▷ To avoid double counting
5: $(t', U) \leftarrow$ REC-COUNT-CLIQUES-V($DG, I, k-1, C$)
6: Store t' in T
7: $t \leftarrow$ REDUCE-ADD(T) ▷ Sum k-clique counts in T
8: **return** (t, U)

9: **procedure** ARB-PEEL($G = (V, E), k, DG, C, t$)
10: ▷ C is an array of k-clique counts per vertex and t is the total # of k-cliques
11: Let B be a bucketing structure mapping V to buckets based on # of k-cliques
12: $d^* \leftarrow t/|V|, f \leftarrow 0$
13: **while** $f < |V|$ **do**
14: $A \leftarrow$ vertices in next bucket in B (to be peeled)
15: $f \leftarrow f + |A|$
16: $(t', U) \leftarrow$UPDATE(G, k, DG, C, A) ▷ Update # of k-cliques
17: Update the buckets of vertices in U, peeling A
18: **if** $t'/(|V| - f) > d^*$ **then**
19: $d^* \leftarrow t'/(|V| - f)$ ▷ Update maximum density
20: **return** d^*

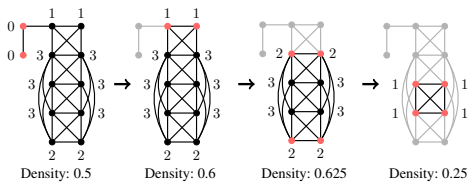


Figure 2: An example of our peeling algorithm ARB-PEEL for $k = 4$. Each vertex is labeled with its current 4-clique count. At each step, we peel the vertices with the minimum 4-clique count, highlighted in red, and then recompute the 4-clique counts on the unpeeled vertices. If there are multiple vertices with the same minimum 4-clique count, we peel them in parallel. Each step is labeled with the k-clique density of the remaining graph.

and if so updates the maximum density (Lines 18–19).

The UPDATE procedure (Line 1–8) performs the bulk of the work in the algorithm. It takes each vertex in A (vertices to be peeled), builds its induced neighborhood, and counts all $(k-1)$-cliques in this neighborhood using ARB-COUNT, as these $(k-1)$-cliques together with a peeled vertex form a k-clique (Line 5). On Line 4, we avoid double counting k-cliques by ignoring vertices already peeled in prior rounds, and for vertices being peeled in the same round, we first mark them in an auxiliary array and break ties based on their rank (i.e., for a k-clique involving multiple vertices being peeled, the highest ranked vertex is responsible for counting it).

This algorithm computes a density that approximates the density of the k-clique densest subgraph. A subgraph with this density can be returned by rerunning the algorithm.

In the full version of the paper, we prove that ARB-PEEL correctly generates a subgraph with the same approximation guarantees of Tsourakakis' sequential k-clique densest subgraph algorithm [53], and the following bounds on the complexity of ARB-PEEL. $\rho_k(G)$ is defined to be the k-clique peeling complexity of G, or the number of rounds needed to peel the graph where in each round, all vertices with the minimum k-clique count are peeled. Note that $\rho_k(G) \leq n$. The proof requires applying bounds from the batch-parallel Fibonacci heap [49] and using the Nash-Williams theorem [36].

THEOREM 4.1. ARB-PEEL *computes a* $1/k$*-approximation to the* k*-clique densest subgraph problem in* $O(m\alpha^{k-2} + \rho_k(G)\log n)$ *expected amortized work,* $O(\rho_k(G)k\log n + \log^2 n)$ *span whp, and* $O(m + P\alpha)$ *space, where* $\rho_k(G)$ *is the* k*-clique peeling complexity of* G*.*

Discussion. To the best of our knowledge, Tsourakakis presents the first sequential algorithm for this problem, although the work bound is worse than ours in most cases. Sariyuce et al. [46] present a sequential algorithm for a more general problem, but in the case that is equivalent to k-clique peeling, their fastest algorithm runs in $O(R(G,k))$ work and $O(C(G,k))$ space, where $R(G,k)$ is the cost of an arbitrary k-clique counting algorithm and $C(G,k)$ is the number of k-cliques in G. They provide another algorithm which runs in $O(m+n)$ space, but requires $O(\sum_v d(v)^k)$ work, which could be as high as $O(n^k)$. Our sequential bounds are asymptotically better than theirs in terms of either work or space, except in the highly degenerate case where $C(G,k) = o(\rho \log n)$. Sariyuce et al. [47] also give a parallel algorithm, which is similarly not work-efficient.

4.2 Approximate Vertex Peeling We present a $1/(k(1+\epsilon))$-approximate algorithm ARB-APPROX-PEEL for the k-clique densest subgraph problem based on approximate peeling. The algorithm is similar to ARB-PEEL, but in each round, it sets a threshold $t = k(1+\epsilon)\tau(S)$ where $\tau(S)$ is the density of the current subgraph S, and removes all vertices with at most τ k-cliques. Tsourakakis [53] describes this procedure and shows that it computes a $1/(k(1+\epsilon))$-approximation of the k-clique densest subgraph in $O(\log n)$ rounds. Although the round complexity in Tsourakakis' implementation is low, no non-trivial bound was known for its work. ARB-APPROX-PEEL is similar to Tsourakakis' algorithm, except we utilize the fast, parallel k-clique counting methods introduced in this paper. We prove the following in the full version of the paper.

THEOREM 4.2. ARB-APPROX-PEEL *computes a* $1/(k(1+\epsilon))$*-approximation to the* k*-clique densest subgraph and runs in* $O(m\alpha^{k-2})$ *work and* $O(k\log^2 n)$ *span whp, and* $O(m + P\alpha)$ *space.*

Note that the span for ARB-APPROX-PEEL matches or improves upon that for ARB-PEEL; notably, when $\rho_k(G) = o(\log n)$, then ARB-APPROX-PEEL takes $O(\rho_k(G)k\log n + \log^2 n)$ span *whp*, which is better than what is stated in Theorem 4.2.

	n	m
com-dblp [31].	317,080	1,049,866
com-orkut [31].	3,072,441	117,185,083
com-friendster [31].	65,608,366	1.806×10^9
com-lj [31].	3,997,962	34,681,189
ClueWeb [14]	978,408,098	7.474×10^{10}
Hyperlink2014 [34]	1.725×10^9	1.241×10^{11}
Hyperlink2012 [34]	3.564×10^9	2.258×10^{11}

Table 1: Sizes of our input graphs. ClueWeb, Hyperlink2012, and Hyperlink2014 are symmetrized to be undirected graphs, and are stored and read in a compressed format from the Graph Based Benchmark Suite (GBBS) [17].

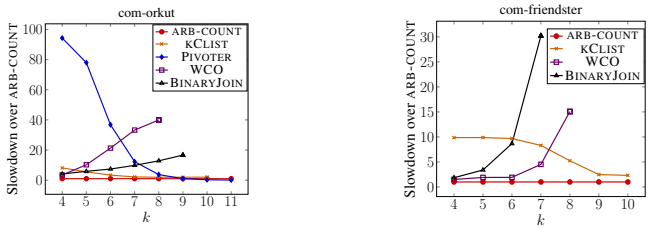


Figure 3: Multiplicative slowdowns of various parallel k-clique counting implementations, compared to ARB-COUNT, on com-orkut and com-friendster. The best runtimes for each implementation were used, and we have excluded any running time over 5 hours for WCO and BINARYJOIN. Note that PIVOTER was unable to perform k-clique counting on com-friendster due to memory limitations, and as such is not included in this figure.

4.3 Practical Optimizations We use the same optimizations described in Section 3.4 for updating k-clique counts. Also, we use the bucketing structure given by Dhulipala et al. [16], which keeps buckets relating k-clique counts to vertices, but only materializes a constant number of the lowest buckets. If large ranges of buckets contain no vertices, this structure skips over such ranges, allowing for fast retrieval of vertices to be peeled in every round using linear space.

5 Experiments

Environment. We run most of our experiments on a machine with 30 cores (with two-way hyper-threading), with 3.8GHz Intel Xeon Scalable (Cascade Lake) processors and 240 GiB of main memory. For our large compressed graphs, we use a machine with 80 cores (with two-way hyper-threading), with 2.6GHz Intel Xeon E7 (Broadwell E7) processors and 3844 GiB of main memory. We compile our programs with g++ (version 7.3.1) using the `-O3` flag. We use OpenMP for our k-clique counting runtimes, and we use a lightweight scheduler called Homemade for our k-clique peeling runtimes [7]. We terminate any experiment that takes over 5 hours, except for experiments on the large compressed graphs.

Graph Inputs. We test our algorithms on real-world graphs from the Stanford Network Analysis Project (SNAP) [31], CMU's Lemur project [14], and the WebDataCommons dataset [34]. The details of the graphs are in Table 1, and we show additional statistics in the full version of the paper.

Algorithm Implementations. We test different orientations for our counting and peeling algorithms, including the Goodrich-Pszona and Barenboim-Elkin orientations from Section 3.1, with $\varepsilon = 1$. We also test other orientations that do not give work-efficient and polylogarithmic-span bounds, but are fast in practice, including the orientation given by ranking vertices by non-decreasing degree, the orientation given by the k-core ordering [33], and the orientation given by the original ordering of vertices in the graph.

Moreover, we compare our algorithms against KCLIST [15], which contains state-of-the-art parallel and sequential k-clique counting algorithms, and parallel k-clique peeling implementations. KCLIST additionally includes a parallel approximate k-clique peeling implementation. We include a simple modification to their k-clique counting code to support faster k-clique counting, where we simply return the number of k-cliques instead of iterating over each k-clique in the final level of recursion. KCLIST also offers the option of node or edge parallelism, but only offers a k-core ordering to orient the input graphs. Note that KCLIST does not offer a choice of orientation.

We additionally compare our counting algorithms to Jain and Seshadhri's PIVOTER algorithm [28], Mhedhbi and Salihoglu's worst-case optimal join algorithm (WCO) [35], Lai *et al.*'s implementation of a binary join algorithm (BINARYJOIN) [30], and Pinar *et al.*'s ESCAPE algorithm [41]. Note that PIVOTER is designed for counting all cliques, and the latter three algorithms are designed for general subgraph counting. Finally, we compare our approximate k-clique counting algorithm to Bressan *et al.*'s MOTIVO algorithm for approximate subgraph counting [11], which is more general. For k-clique peeling, we compare to Fang *et al.*'s COREAPP algorithm [21] and Tsourakakis's [53] triangle densest subgraph implementation.

5.1 Counting Results Table 2 shows the best parallel runtimes for k-clique counting over the SNAP datasets, from ARB-COUNT, KCLIST, PIVOTER, WCO, and BINARYJOIN, considering different orientations for ARB-COUNT, and considering node versus edge parallelism for ARB-COUNT and for KCLIST. We also show the best sequential runtimes from ARB-COUNT. We do not include triangle counting results, because for triangle counting, our k-clique counting algorithm becomes precisely Shun and Tangwongsan's [51] triangle counting algorithm. Furthermore, we performed experiments on ESCAPE by isolating their 4- and 5-clique counting code, but KCLIST consistently outperforms ESCAPE; thus, we have not included ESCAPE in Table 2. Figure 3 shows the slowdowns of the parallel implementations over ARB-COUNT on com-orkut and com-friendster.

We also obtain parallel runtimes for $k = 4$ on large compressed graphs, using degree ordering and node parallelism, on a 80-core machine with hyper-threading; note that

		$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	$k=11$
com-dblp	ARB-COUNT T_{60}	**0.10**	**0.13**	**0.30**	**2.05**^{e}	24.06^{e}	281.39^{e}	2981.74^{*e}	>5 hrs
	ARB-COUNT T_1	1.57	1.71	5.58	64.27	837.82	9913.01	>5 hrs	>5 hrs
	KCLIST T_{60}	0.16	0.17	0.43^{e}	4.28^{e}	55.78^{e}	640.48^{e}	6895.16^{e}	>5 hrs
	PIVOTER T_{60}	2.88	2.88	2.88	2.88	**2.88**	**2.88**	**2.88**	**2.88**
	WCO T_{60}	0.19	0.37	3.84	66.06	1126.69	9738.00	>5 hrs	>5 hrs
	BINARYJOIN T_{60}	0.12	0.42	2.08	39.29	627.48	7282.79	>5 hrs	>5 hrs
com-orkut	ARB-COUNT T_{60}	**3.10**	**4.94**	**12.57**	**42.09**	**150.87**$^{\circ}$	**584.39**$^{\circ}$	2315.89°	8843.51$^{\circ e}$
	ARB-COUNT T_1	79.62	158.74	452.47	1571.49	5882.83	>5 hrs	>5 hrs	>5 hrs
	KCLIST T_{60}	25.27	27.40	42.23	91.67^{e}	293.92^{e}	1147.50^{e}	4666.03^{e}	>5 hrs
	PIVOTER T_{60}	292.35	385.04	462.05	517.29	559.75	598.88	**647.18**	**647.18**
	WCO T_{60}	10.71	50.51	267.47	1398.89	6026.99	>5 hrs	>5 hrs	>5 hrs
	BINARYJOIN T_{60}	12.74	29.09	93.06	413.50	1938.06	9732.86	>5 hrs	>5 hrs
com-friendster	ARB-COUNT T_{60}	**109.46**	**111.75**	**115.52**	**139.98**	**300.62**	**1796.12**^{e}	**16836.41**$^{\circ e}$	>5 hrs
	ARB-COUNT T_1	2127.79	2328.48	2723.53	3815.24	8165.76	>5 hrs	>5 hrs	>5 hrs
	KCLIST T_{60}	1079.22	1104.28	1117.31	1162.84	1576.61^{e}	4449.81^{e}	>5 hrs	>5 hrs
	WCO T_{60}	201.82	379.59	1001.52	4229.20	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	BINARYJOIN T_{60}	163.90	212.53	221.93	632.40	4532.60	>5 hrs	>5 hrs	>5 hrs
com-lj	ARB-COUNT T_{60}	**1.77**	**7.52**	**258.46**	**10733.21**	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	ARB-COUNT T_1	33.04	231.15	8956.53	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	KCLIST T_{60}	7.53	22.13	647.77^{e}	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	PIVOTER T_{60}	268.06	1475.99	7816.13	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	WCO T_{60}	6.62	80.78	3448.70	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	BINARYJOIN T_{60}	4.10	42.32	1816.87	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs

Table 2: Best runtimes in seconds for our parallel (T_{60}) and single-threaded (T_1) k-clique counting algorithm (ARB-COUNT), as well as the best parallel runtimes from KCLIST [15], PIVOTER [28], WCO [35], and BINARYJOIN [30]. Note that we cannot report runtimes from PIVOTER for the com-friendster graph, because for all k, PIVOTER runs out of memory and is unable to complete k-clique counting. The fastest runtimes for each experiment are bold and in green. All runtimes are from tests in the same computing environment, and include time spent preprocessing and counting (but not time spent loading the graph). For our parallel and serial runtimes and KCLIST, we have chosen the fastest orientations and choice between node and edge parallelism per experiment. For the runtimes from ARB-COUNT, we have noted the orientation used; $^{\circ}$ refers to the Goodrich-Pszona orientation, * refers to the orientation given by k-core, and no superscript refers to the orientation given by degree ordering. For the runtimes from ARB-COUNT and KCLIST, we have noted whether node or edge parallelism was used; ^{e} refers to edge parallelism, and no superscript refers to node parallelism.

		$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
com-dblp	ARB-PEEL T_{60}	0.14	**0.21**	**0.23**$^{\circ}$	**1.29**$^{\circ}$	**18.77**	**276.69**$^{\circ}$	**3487.09**$^{\circ}$
	ARB-PEEL T_1	0.27	0.37	1.378	17.99	258.24	3373.05	>5 hrs
	KCLIST T_1	0.19	0.25	1.10	14.98	221.98	2955.87	>5 hrs
	COREAPP T_1	**0.10**	0.23	1.09	12.21	244.81	7674.55	>5 hrs
com-orkut	ARB-PEEL T_{60}	**33.15**$^{\circ}$	**76.91**	**221.28**	**721.73**	**2466.99**$^{\circ}$	**9062.99**$^{\circ}$	>5 hrs
	ARB-PEEL T_1	130.04	184.28	422.20	1032.19	3123.72	>5 hrs	>5 hrs
	KCLIST T_1	87.71	218.94	587.24	2029.43	7414.77	>5 hrs	>5 hrs
	COREAPP T_1	113.27	546.13	2460.65	16320.24	>5 hrs	>5 hrs	>5 hrs
com-friendster	ARB-PEEL T_{60}	**371.52**	**1747.92**	**4144.96**	**6870.06**	>5 hrs	>5 hrs	>5 hrs
	ARB-PEEL T_1	3297.14	11540.73	12932.28	14112.95	>5 hrs	>5 hrs	>5 hrs
	KCLIST T_1	2225.70	3216.92	4325.73	6933.32	>5 hrs	>5 hrs	>5 hrs
	COREAPP T_1	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs	>5 hrs
com-lj	ARB-PEEL T_{60}	**6.46**	**26.36**	**324.77**	**12920.08**	>5 hrs	>5 hrs	>5 hrs
	ARB-PEEL T_1	17.74	70.12	822.10	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	KCLIST T_1	16.64	42.16	839.13	>5 hrs	>5 hrs	>5 hrs	>5 hrs
	COREAPP T_1	7.20	27.53	1595.04	>5 hrs	>5 hrs	>5 hrs	>5 hrs

Table 3: Best runtimes in seconds for our parallel and single-threaded k-clique peeling algorithm (ARB-PEEL), as well as the best sequential runtimes from previous work (KCLIST and COREAPP) [15, 21]. KCLIST and COREAPP do not have parallel implementations of k-clique peeling; they are only serial. The fastest runtimes for each experiment are bolded and in green. All runtimes are from tests in the same computing environment, and include only time spent peeling. For our parallel runtimes, we have chosen the fastest orientations per experiment, while for our serial runtimes, we have fixed the degree orientation. For the parallel runtimes from ARB-PEEL, we have noted the orientation used; $^{\circ}$ refers to the Goodrich-Pszona orientation, and no superscript refers to the orientation given by degree ordering.

KCLIST, PIVOTER, WCO, and BINARYJOIN cannot handle these graphs. The runtimes are: 5824.76 seconds on ClueWeb with 74 billion edges (< 2 hours), 12945.25 seconds on Hyperlink2014 with over one hundred billion edges (< 4 hours), and 161418.89 seconds on Hyperlink2012 with over two hundred billion edges (< 45 hours). As far as we know, these are the first results for 4-clique counting for graphs of this scale.

Overall, on 30 cores, ARB-COUNT obtains speedups between 1.31–9.88x over KCLIST, between 1.02–46.83x over WCO, and between 1.20-28.31x over BINARYJOIN. Our largest speedups are for large graphs (e.g., com-friendster) and for moderate values of k, because we obtain more parallelism relative to the necessary work.

Comparing our parallel runtimes to KCLIST's serial runtimes (which were faster than those of WCO and BINARYJOIN), we obtain between 2.26–79.20x speedups, and considering only parallel runtimes over 0.7 seconds, we obtain between 16.32–79.20x speedups. By virtue of our orientations, our single-threaded runtimes are often faster than the serial runtimes of the other implementations, with up to 23.17x speedups particularly for large graphs and large values of k. Our self-relative parallel speedups are between 13.23–38.99x.

We also compared with PIVOTER [28], which is designed for counting all cliques, but can be truncated for fixed k. Their algorithm is able to count all cliques for com-dblp and com-orkut in under 5 hours. However, their algorithm is not theoretically-efficient for fixed k, taking $O(n\alpha^2 3^{\alpha/3})$ work, and as such their parallel implementation is up to 196.28x slower compared to parallel ARB-COUNT, and their serial implementation is up to 184.76x slower compared to single-threaded ARB-COUNT. These slowdowns are particularly prominent for small k. Also, PIVOTER's truncated algorithm does not give significant speedups over their full algorithm, and PIVOTER requires significant space and runs out of memory for large graphs; it is unable to compute k-clique counts at all for $k \geq 4$ on com-friendster.

Of the different orientations, using degree ordering is generally the fastest for small k because it requires little overhead and gives sufficiently low out-degrees. However, for larger k, this overhead is less significant compared to the time for counting and other orderings result in faster counting. The cutoff for this switch occurs generally at $k = 8$. Note that the Barenboim-Elkin and original orientations are never the fastest orientations. The slowness of the former is because it gives a lower-granularity ordering, since it does not order between vertices deleted in a given round. We found that the self-relative speedups of orienting the graph alone were between 6.69–19.82x across all orientations, the larger of which were found in large graphs. We discuss preprocessing overheads in more detail in the full version of the paper.

Moreover, in both ARB-COUNT and KCLIST, node parallelism is faster on small k, while edge parallelism is faster on large k. This is because parallelizing the first level of recursion is sufficient for small k, and edge parallelism introduces greater parallel overhead. For large k, there is more work, which edge parallelism balances better, and the additional parallel overhead is mitigated by the balancing. The cutoff for when edge parallelism is generally faster than node parallelism occurs around $k = 8$. We provide more detailed analysis in the full version of the paper.

We also evaluated our approximate counting algorithm on com-orkut and com-friendster, and compared to MOTIVO [11]. We defer a detailed discussion to the full version of the paper. Overall, we obtain significant speedups over exact k-clique counting and have low error rates over the exact global counts, with between 5.32–2189.11x speedups over exact counting and between 0.42–5.05% error. We also see 92.71–177.29x speedups over MOTIVO for 4-clique and 5-clique approximate counting on com-orkut.

5.2 Peeling Results Table 3 shows the best parallel and sequential runtimes for k-clique peeling on SNAP datasets for ARB-PEEL, KCLIST, and COREAPP (KCLIST and COREAPP only implement sequential algorithms).

Overall, our parallel implementation obtains between 1.01–11.83x speedups over KCLIST's serial runtimes. The higher speedups occur in graphs that require proportionally fewer parallel peeling rounds ρ_k compared to its size; notably, com-dblp requires few parallel peeling rounds, and we see between 4.78–11.83x speedups over KCLIST on com-dblp for $k \geq 5$. As such, our parallel speedups are constrained by ρ_k. Similarly, we obtain up to 53.53x speedup over COREAPP's serial runtimes. COREAPP outperforms our parallel implementation on triangle peeling for com-dblp, again owing to the proportionally fewer parallel peeling rounds in these cases. ARB-PEEL achieves self-relative parallel speedups between 1.19–13.76x. Our single-threaded runtimes are generally slower than KCLIST's and COREAPP's sequential runtimes owing to the parallel overhead necessary to aggregate k-clique counting updates between rounds. In the full version of the paper, we present a further analysis of the distributions of number of vertices peeled per round.

Moreover, the edge density of the approximate k-clique densest subgraph found by ARB-PEEL converges towards 1 for $k \geq 3$, and as such, ARB-PEEL is able to efficiently find large subgraphs that approach cliques. In particular, the k-clique densest subgraph that ARB-PEEL finds on com-lj contains 386 vertices with an edge density of 0.992. Also, the k-clique densest subgraph that ARB-PEEL finds on com-friendster contains 141 vertices with an edge density of 0.993.

We also tested Tsourakakis's [53] triangle densest subgraph implementation; however, it requires too much memory to run for com-orkut, com-friendster, and com-lj on our machines. It completes 3-clique peeling on com-dblp in 0.86 seconds, while our parallel ARB-PEEL takes 0.27 seconds.

Finally, we compared our parallel approximate ARB-

APPROX-PEEL to KCLIST's parallel approximate algorithm on com-orkut and com-friendster. ARB-APPROX-PEEL is up to 29.59x faster than KCLIST for large k, and we see between 5.95–80.83% error on the maximum k-clique density obtained compared to the density obtained from k-clique peeling.

6 Related Work

Theory. A trivial algorithm can compute all k-cliques in $O(n^k)$ work. Using degree-based thresholding enables clique counting in $O(m^{k/2})$ work, which is asymptotically faster for sparse graphs. Chiba and Nishizeki give an algorithm with improved complexity for sparse graphs, in which all k-cliques can be found in $O(m\alpha^{k-2})$ work [12], where α is the arboricity of the graph.

For arbitrary graphs, the fastest theoretical algorithm uses matrix multiplication, and counts $3l$ cliques in $O(n^{l\omega})$ time where ω is the matrix multiplication exponent [37]. The k-clique problem is a canonical hard problem in the FPT literature, and is known to be $W[1]$-complete when parametrized by k [19]. We refer the reader to [57], which surveys other theoretical algorithms for this problem.

Recent work by Dhulipala et al. [18] studied k-clique counting in the parallel batch-dynamic setting. One of their algorithms calls our ARB-COUNT as a subroutine.

Practice. The special case of counting and listing triangles ($k = 3$) has received a huge amount of attention over the past two decades (e.g., [55, 54, 51, 39], among many others). Finocchi et al. [23] present parallel k-clique counting algorithms for MapReduce. Jain and Seshadri [27] provide algorithms for estimating k-clique counts. The state-of-the-art k-clique counting and listing algorithm is KCLIST by Danisch et al. [15], which is based on the Chiba-Nishizeki algorithm, but uses the k-core ordering (which is not parallel) to rank vertices. It achieves $O(m\alpha^{k-2})$ work, but does not have polylogarithmic span due to the ordering and only parallelizing one or two levels of recursion. Concurrent with our work, Li et al. [32] present an ordering heuristic for k-clique counting based on graph coloring, which they show improves upon KCLIST in practice. It would be interesting in the future to study their heuristic applied to our algorithm.

Additionally, many algorithms have been designed for finding 4- and 5-vertex subgraphs (e.g., [41, 40, 2, 58, 44]) as well as estimating larger subgraph counts (e.g., [10, 11]), and these algorithms can be used for counting exact or approximate k-clique counting as a special case. Worst-case optimal join algorithms from the database literature [1, 38, 35, 30] can also be used for k-clique listing and counting as a special case, and would require $O(m^{k/2})$ work.

Very recently, Jain and Seshadri [28] present a sequential and a vertex parallel PIVOTER algorithm for counting all cliques in a graph. However, their algorithm cannot be used for k-clique listing as they avoid processing all cliques, and requires much more than $O(m\alpha^{k-2})$ work in the worst case.

Low Out-degree Orientations. A canonical technique in the graph algorithms literature on clique counting, listing, and related tasks [20, 28, 41] is the use of a low out-degree orientation. Matula and Beck [33] show that k-core gives an $O(\alpha)$ orientation. However, the problem of computing this ordering is P-complete [3], and thus unlikely to have polylogarithmic span. More recent work in the distributed and external-memory literature has shown that such orderings can be efficiently computed in these settings. Barenboim and Elkin give a distributed algorithm that finds an $O(\alpha)$-orientation in $O(\log n)$ rounds [5]. Goodrich and Pszona give a similar algorithm for external-memory [25]. Concurrent with our work, Besta et al. [6] present a parallel algorithm for generating an $O(\alpha)$-orientation in $O(m)$ work and $O(\log^2 n)$ span, which they use for parallel graph coloring.

Vertex Peeling and k-clique Densest Subgraph. An important application of k-clique counting is its use as a subroutine in computing generalizations of approximate densest subgraph. In this paper, we study parallel algorithms for k-clique densest subgraph, a generalization of the densest subgraph problem introduced by Tsourakakis [53]. Tsourakakis presents a sequential $1/k$-approximation algorithm based on iteratively peeling the vertex with minimum k-clique-count, and a parallel $1/(k(1+\epsilon))$-approximation algorithm based on a parallel densest subgraph algorithm of Bahmani et al. [4]. Sun *et al.* [52] give additional approximation algorithms that converge to produce the exact solution over further iterations; these algorithms are more sophisticated and demonstrate the tradeoff between running times and relative errors. Recently, Fang *et al.* [21] propose algorithms for finding the largest (j, Ψ)-core of a graph, or the largest subgraph such that all vertices have at least j subgraphs Ψ incident on them. They propose an algorithm for Ψ being a k-clique that peels vertices with larger clique counts first and show that their algorithm gives a $1/k$-approximation to the k-clique densest subgraph.

7 Conclusion

We presented new work-efficient parallel algorithms for k-clique counting and peeling with low span. We showed that our implementations achieve good parallel speedups and significantly outperform state-of-the-art. A direction for future work is designing work-efficient parallel algorithms for the more general (r, s)-nucleus decomposition problem [48].

Acknowledgments

This research was supported by NSF Graduate Research Fellowship #1122374, DOE Early Career Award #DE-SC0018947, NSF CAREER Award #CCF-1845763, Google Faculty Research Award, Google Research Scholar Award, DARPA SDH Award #HR0011-18-3-0007, and Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

References

[1] C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré. EmptyHeaded: A relational engine for graph processing. *ACM Trans. Database Syst.*, 42(4), 2017.

[2] N. K. Ahmed, J. Neville, R. A. Rossi, N. G. Duffield, and T. L. Willke. Graphlet decomposition: framework, algorithms, and applications. *Knowl. Inf. Syst.*, 50(3), 2017.

[3] R. Anderson and E. W. Mayr. A P-complete problem and approximations to it. Technical report, 1984.

[4] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and MapReduce. *Proc. VLDB Endow.*, 5(5), Jan. 2012.

[5] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. *Distributed Computing*, 22(5), 2010.

[6] M. Besta, A. Carigiet, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, and T. Hoefler. High-performance parallel graph coloring with strong guarantees on work, depth, and quality. In *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[7] G. E. Blelloch, D. Anderson, and L. Dhulipala. Brief announcement: ParlayLib – a toolkit for parallel algorithms on shared-memory multicore machines. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2020.

[8] R. D. Blumofe and C. E. Leiserson. Space-efficient scheduling of multithreaded computations. *SIAM J. Comput.*, 27(1), 1998.

[9] R. P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2), Apr. 1974.

[10] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi. Motif counting beyond five nodes. *ACM Trans. Knowl. Discov. Data*, 12(4), 2018.

[11] M. Bressan, S. Leucci, and A. Panconesi. Motivo: Fast motif counting via succinct color coding and adaptive sampling. *Proc. VLDB Endow.*, 12(11), July 2019.

[12] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1), Feb. 1985.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[14] B. Croft and J. Callan. The Lemur project. https://www.lemurproject.org/, 2016.

[15] M. Danisch, O. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs*. In *International Conference on World Wide Web*, 2018.

[16] L. Dhulipala, G. Blelloch, and J. Shun. Julienne: A framework for parallel graph algorithms using work-efficient bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2017.

[17] L. Dhulipala, G. E. Blelloch, and J. Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *ACM Symposium on Parallelism in Algorithms and Architectures*, 2018.

[18] L. Dhulipala, Q. C. Liu, J. Shun, and S. Yu. Parallel batch-dynamic k-clique counting. In *SIAM Symposium on Algorithmic Principles of Computer Systems*, 2021.

[19] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4), 1995.

[20] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, 2010.

[21] Y. Fang, K. Yu, R. Cheng, L. V. S. Lakshmanan, and X. Lin. Efficient algorithms for densest subgraph discovery. *Proc. VLDB Endow.*, 12(11), July 2019.

[22] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2), 1995.

[23] I. Finocchi, M. Finocchi, and E. G. Fusco. Clique counting in MapReduce: Algorithms and experiments. *J. Exp. Algorithmics*, 20, Oct. 2015.

[24] J. Gil, Y. Matias, and U. Vishkin. Towards a theory of nearly constant time parallel algorithms. In *IEEE Symposium on Foundations of Computer Science*, 1991.

[25] M. T. Goodrich and P. Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *European Symposium on Algorithms*, 2011.

[26] E. Gregori, L. Lenzini, and S. Mainardi. Parallel k-clique community detection on large-scale networks. *IEEE Trans. Parallel Distrib. Syst.*, 24(8), Aug 2013.

[27] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using Turán's theorem. In *International Conference on World Wide Web*, 2017.

[28] S. Jain and C. Seshadhri. The power of pivoting for exact clique counting. In *ACM International Conference on Web Search and Data Mining*, 2020.

[29] J. Jaja. *Introduction to Parallel Algorithms*. Addison-Wesley Professional, 1992.

[30] L. Lai, Z. Qing, Z. Yang, X. Jin, Z. Lai, R. Wang, K. Hao, X. Lin, L. Qin, W. Zhang, Y. Zhang, Z. Qian, and J. Zhou. Distributed subgraph matching on timely dataflow. *Proc. VLDB Endow.*, 12(10), June 2019.

[31] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2019.

[32] R.-H. Li, S. Gao, L. Qin, G. Wang, W. Yang, and J. X. Yu. Ordering heuristics for k-clique listing. *Proc. VLDB Endow.*, 13(12), July 2020.

[33] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3), July 1983.

[34] R. Meusel, S. Vigna, O. Lehmberg, and C. Bizer. The graph structure in the web–analyzed on different aggregation levels. *J. Web Sci.*, 1(1), 2015.

[35] A. Mhedhbi and S. Salihoglu. Optimizing subgraph queries by combining binary and worst-case optimal joins. *Proc. VLDB Endow.*, 12(11), July 2019.

[36] C. S. J. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1), 1961.

[37] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 026(2), 1985.

[38] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3), Mar. 2018.

[39] R. Pagh and C. E. Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Inf. Process. Lett.*, 112(7), Mar. 2012.
[40] H.-M. Park, F. Silvestri, R. Pagh, C.-W. Chung, S.-H. Myaeng, and U. Kang. Enumerating trillion subgraphs on distributed systems. *ACM Trans. Knowl. Discov. Data*, 12(6), Oct. 2018.
[41] A. Pinar, C. Seshadhri, and V. Vishal. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *International Conference on World Wide Web*, 2017.
[42] S. Rajasekaran and J. H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM J. Comput.*, 18(3), June 1989.
[43] R. A. Rossi, N. K. Ahmed, and E. Koh. Higher-order network representation learning. In *International Conference on World Wide Web*, 2018.
[44] R. A. Rossi, R. Zhou, and N. K. Ahmed. Estimation of graphlet counts in massive networks. *IEEE Trans. Neural Netw. Learning Syst.*, 30(1), 2019.
[45] S.-V. Sanei-Mehri, A. E. Sariyuce, and S. Tirthapura. Butterfly counting in bipartite networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.
[46] A. E. Sariyüce and A. Pinar. Peeling bipartite networks for dense subgraph discovery. In *ACM International Conference on Web Search and Data Mining*, 2018.
[47] A. E. Sariyüce, C. Seshadhri, and A. Pinar. Local algorithms for hierarchical dense subgraph discovery. *Proc. VLDB Endow.*, 12(1), Sept. 2018.
[48] A. E. Sariyüce, C. Seshadhri, A. Pinar, and U. V. Çatalyürek. Nucleus decompositions for identifying hierarchy of dense subgraphs. *ACM Trans. Web*, 11(3), July 2017.
[49] J. Shi and J. Shun. Parallel algorithms for butterfly computations. In *SIAM Symposium on Algorithmic Principles of Computer Systems*, 2020.
[50] J. Shun, L. Dhulipala, and G. E. Blelloch. Smaller and faster: Parallel processing of compressed graphs with Ligra+. In *IEEE Data Compression Conference*, 2015.
[51] J. Shun and K. Tangwongsan. Multicore triangle computations without tuning. In *IEEE International Conference on Data Engineering*, 2015.
[52] B. Sun, M. Danisch, T.-H. H. Chan, and M. Sozio. KClist++: A simple algorithm for finding k-clique densest subgraphs in large graphs. *Proc. VLDB Endow.*, 13(10), June 2020.
[53] C. Tsourakakis. The k-clique densest subgraph problem. In *International Conference on World Wide Web*, 2015.
[54] C. E. Tsourakakis. Counting triangles in real-world networks using projections. *Knowl. Inf. Syst.*, 26(3), 2011.
[55] C. E. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. *Social Network Analysis and Mining*, 1(2), Apr 2011.
[56] C. E. Tsourakakis, J. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. In *International Conference on World Wide Web*, 2017.
[57] V. Vassilevska. Efficient algorithms for clique problems. *Inf. Process. Lett.*, 109(4), 2009.
[58] P. Wang, J. Zhao, X. Zhang, Z. Li, J. Cheng, J. C. S. Lui, D. Towsley, J. Tao, and X. Guan. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Trans. Knowl. Data Eng.*, 30(1), Jan 2018.
[59] H. Yin, A. R. Benson, and J. Leskovec. Higher-order clustering in networks. *Physical Review E*, 97(5), 2018.

A Message-Driven, Multi-GPU Parallel Sparse Triangular Solver *

Nan Ding† Yang Liu ‡ Samuel Williams† Xiaoye S. Li‡

Abstract

Sparse triangular solve is used in conjunction with Sparse LU for solving sparse linear systems, either as a direct solver or as a preconditioner. As GPUs have become a first-class compute citizen, designing an efficient and scalable SpTRSV on multi-GPU HPC systems is imperative. In this paper, we leverage the advantage of GPU-initiated data transfers of NVSHMEM to implement and evaluate a Multi-GPU SpTRSV. We create a novel producer-consumer paradigm to manage the computation and communication in SpTRSV and implement it using two CUDA streams. Our multi-GPU SpTRSV implementation using CUDA streams achieves a 3.7× speedup when using twelve GPUs (two nodes) relative to our implementation on a single GPU, and up to 6.1× compared to *cusparse_csrsv2()* over the range of one to eighteen GPUs. To further explain the observed performance and explore the key features of matrices to estimate the potential performance benefits when using multi-GPU, we extend the critical path model of SpTRSV to GPUs. We demonstrate the ability of our performance model to understand various aspects of performance and performance bottlenecks on multi-GPU and motivate code optimizations.

1 Introduction

Over the last decade, accelerated computing architectures have become more and more popular in modern HPC systems. A total of 147 systems on the 2020 TOP500 list are using accelerators [1], of which, 110 systems use NVIDIA Volta chips [2]. Concurrently, sparse triangular solve (SpTRSV) is considered an indispensable task in a wide range of applications from numerical simulation [3] to machine learning [4]. Designing an efficient and scalable sparse triangular solver (SpTRSV) on modern multi-GPU HPC systems is imperative but challenging. In recent years, substantial efforts have focused on single GPU SpTRSV [5, 6]. However, with more scientific insights derived from computation, the demand for ever finer-resolution problems calls for SpTRSV to exploit ever larger scales of parallelism. Unfortunately, given the slow pace in HBM memory capacity scaling, one cannot guarantee the problem can always fit into a single GPU's memory.

In this paper, we implement and evaluate a multi-GPU SpTRSV. We leverage NVSHMEM [7] to perform direct GPU-GPU communication. NVSHMEM is a parallel programming interface based on OpenSHMEM [8] that provides efficient and scalable (one-sided) communication for NVIDIA GPU clusters. The advantages of using NVSHMEM is that it uses GPU-initiated data transfers. This allows users to perform both computations and communications in one CUDA kernel instead of transferring data between the CPU and the GPU. Unfortunately, NVSHMEM also has a major limitation. It limits the number of thread blocks that can be concurrently scheduled on one V100 GPU to 80 to avoid potential deadlocks when using point-to-point synchronization in the CUDA kernel. Nominally, such a limitation would significantly restrict SpTRSV concurrency.

To overcome the concurrency limitation of NVSHMEM, we propose a coupled producer-consumer parallelism using CUDA streams. CUDA streams are often used to overlap computation and communication for the simple producer-consumer style of parallelism such as stencils where one stream performs all computations and the other stream handles communication [9]. Although a simple CUDA stream synchronize might suffice for stencils, SpTRSV has a more complex producer-consumer relationship — the producer (sender) and the consumer (receiver) can swap roles in turn to dispatch new work (message). We use two CUDA streams to handle this complex coupled producer-consumer parallelism. The advantages of using CUDA streams include not only the mitigation of the NVSHMEM concurrency

*This research is supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) programs under Contract No. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory. This research used resources of the the Oak Ridge Leadership Facility which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. We thank Akhil Langer from NVIDIA Corporation for his willingness to answer our myriad of questions on NVSHMEM.

†Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA, (nanding|swwilliams@lbl.gov)

‡Scalable Solvers Group, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA, (liuyangzhuan|xsli@lbl.gov)

limitation, but also enabling the of overlap of communication and computation.

Aligned with the advances in hardware and communication paradigms, performance modeling of SpTRSV is critical to assess potential performance gains in terms of machine capability. Modeling SpTRSV depends heavily on the structure of a given matrix and the underlying architecture. Whereas the Roofline model [10, 11] can effectively bound performance and identify bottlenecks for well-structured, load-balanced codes, it can only provide a very loose bound on performance for codes like SpTRSV. To that end, a SpTRSV performance model is proposed in work [12] for pure MPI implementations on CPUs. The model is based on the critical path analysis which follows the task dependency graph of the sparse matrix using the well-known level-set method [13, 14] with a breadth-first search [15]. Process decomposition is also considered into the critical path analysis. The computations and communications in each MPI process are serialized. Therefore, it lacks concurrency in each process (GPU), including concurrent messaging and computation from the thread blocks in one GPU (corresponding to one MPI process in [12]). Ultimately, the contributions in this paper include: **(1)** Develop a method of coupled producer-consumer parallelism using CUDA streams which overcomes the concurrency limitation of NVSHMEM, and overlap the communications and computations, **(2)** Implement a multi-GPU (both intra- and inter-node) SpTRSV using coupled CUDA streams which achieves a 3.7× speedup when using twelve GPUs (two nodes) compared to the single GPU implementation on Summit, and **(3)** Extend our SpTRSV performance model for GPUs that enables insights into various aspects of performance and performance bottlenecks on multiple GPUs.

2 Distributed-memory Parallel Sparse Triangular Solve

SpTRSV computes a solution vector x for a $n \times n$ linear system $Lx = b$ [1], where L is a lower triangular matrix, and b is a $n \times k$ right-hand side (RHS) matrix or vector ($k = 1$). For a sparse matrix L, the computation of x_i needs some or all of the previous solution rows x_j, $j < i$, depending on the sparsity pattern of the i^{th} row of L. This computation dependency can be precisely expressed by a DAG. We use a supernodal DAG [16] formulation. For a lower triangular matrix L, a supernode is a set of consecutive columns of L with the triangular block just below the diagonal being full, and the same nonzero structure below the triangular block. After a supernode partition is obtained along the columns of L, we apply the same partition row-wise to obtain a 2D block partitioning. The nonzero block pattern defines the supernodal DAG. We assume $b(K)$ and $x(K)$ represent the subvector associated with supernode K. $L(I, K)$ denotes the nonzero submatrix corresponding to supdernodes I and K. Thus, the solution of subvector $x(K)$ can be computed as Eq. (2.1).

$$x(K) = L(K,K)^{-1}\Big(b(K) - \sum_{I=1}^{K-1} L(K,I) \cdot x(I)\Big) \tag{2.1}$$

The distributed-memory SpTRSV [12] partitions the matrix L among multiple processes using a 2D block cyclic layout. Each process is in charge of a subset of solution subvectors $x(K)$. The solution of these subvectors and partial summation results require communication. Figure 1 describes the data flow of a sparse triangular solve using a 2×2 process decomposition. Processes assigned to the diagonal blocks, called *diagonal processes*, compute the corresponding blocks of x. In the process decomposition of Figure 1, processes {0, 4, 8} are the diagonal processes. Within one block column, the process owning $x(I)$ sends $x(I)$ to the process of $L(K, I)$ as No. ① in Figure 1. After receiving the required $x(I)$ subvector, each process computes its local summation. Within one row, the local sums are sent to the diagonal process which will perform the inversion (No. ② in Figure 1). In this case, process 0 is the producer, and process 6 is the consumer in the first block column. However, process 6 reverts to being the producer in the fifth block row. An asynchronous binary tree [17] is used to perform the column broadcast and row reduction, while a one-sided `MPI_Put` is used to reduce the communication latency. Each message contains the data and a checksum payload. The checksum payload is used by receivers to check completion. The binary tree is built in the setup phase that is executed once for multiple solves. A broadcast tree per supernode column K is built for the processes participating in the column broadcast. Similarly, one reduction tree is built per supernode row I within the processes participating in the row reduction. Note that each process in a tree need only keep track of its parent and children.

3 Benefits and Challenges of NVSHMEM

NVSHMEM is a parallel programming interface based on OpenSHMEM that provides efficient and scalable communication (one-sided) for NVIDIA GPU clusters. The advantages of using NVSHMEM for multi-GPU programming is that it uses GPU-initiated data

[1] We use lower triangular matrix to formulate the problem in the paper. Note that the proposed methodology can be easily ported to solve an upper triangular system $Ux = b$.

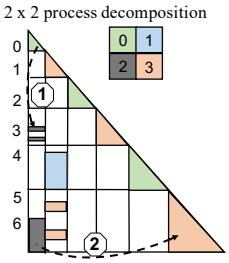


Figure 1: Data flow and process decomposition of the distributed-memory SpTRSV. The dashed arrows represent the data flow. The numbers represent two communication behaviors: ① block column broadcast, and ② block row reduction.

transfers. This feature allows programmers to execute both computations and communications in one CUDA kernel in lieu of initiating communication from the CPU.

Some numerical methods have a relatively simple communication pattern, such as stencils, which adhere to the Bulk Synchronous Parallel (BSP) model [18]. Inter-processor communications follow the discipline of strict barrier synchronization. As such, communication models like MPI and its CUDA-aware variant [19] can satisfy the requirements of those applications. Conversely, DAG-like computations, like SpTRSV, have a more complex communication pattern. Point-to-point communications can happen at any time between any two processes (depending on the sparsity pattern and the process decomposition) with no strict barrier synchronization. Therefore, the GPU-initiated communication nature of NVSHMEM provides a big advantage over other communication paradigms.

One challenge when using one-sided communication is how to notify receivers that the data has completely arrived. NVSHMEM provides signaling operations and point-to-point synchronization operations. These operations relieve users of the burden of implementing their own data synchronization. However, it also brings one limitation. When using synchronizations in the CUDA kernel, the number of thread blocks that can be launched on one V100 GPU is limited to 80 (the number of SMs) to avoid potential deadlocks. This is an inherent limitation of the NVSHMEM+CUDA+NVIDIA GPU environment, and can significantly restrict the concurrency in SpTRSV.

To overcome the limitation of 80 thread blocks, we propose a coupled producer-consumer parallelism using CUDA streams. We use two streams to execute two kernels concurrently. One kernel, named `WAIT` in *stream[0]*, handles NVSHMEM point-to-point synchronizations. It is launched using `nvshmemx_collective_launch()` with a number of thread blocks less than 80 (to ensure the GPU is not fully occupied). The other kernel, named `SOLVE` in *stream[1]*, is responsible for computation and sending data/notification. It is launched after the `WAIT` kernel as a normal CUDA kernel thereby maximizing concurrency. We use a bit scalar ($flag_w$, Algorithm 1 line 4) to control the launch order of the two kernels. The `WAIT` kernel is launched first, and sets the scalar to *True*. The `SOLVE` kernel is not launched until $flag_w == True$.

4 Multi-GPU SpTRSV using CUDA Streams.

Algorithm 1 details our design for our multi-GPU SpTRSV, and the variables are listed in Table 1. We bind one process to one GPU so that each GPU (corresponding to a process in Section 2) is in charge of a subset of solution subvectors $x(K)$.

Let us assume that a lower triangular matrix $L(n, n)$ has N supernodes in total, N_g supernode columns per GPU, and N_r supernode rows per GPU. We launch $N_g + 2$ thread blocks per GPU for the matrix L, where two thread blocks are for the `WAIT` kernel, and N_g thread blocks are used by the `SOLVE` kernel. The `SOLVE` kernel uses those N_g thread blocks to perform the requisite TRSV (Triangular Solve Matrix-Vector, diagonal blocks) and GEMV (General Matrix-Vector Multiplications, off-diagonal blocks) computations, broadcasts x subvector, notifies the consumers.

We perform row reductions in both kernels whenever the data dependency ($fmod(I) == 0$, line 21 and 49) is met. A counter $fmod(I)$ is computed per supernode row I to record the number of local inner-products and non-local messages for their contribution to $lsum(I)$. The number of local updates equals the number of blocks in row I one process owns, while the number of non-local updates is always no more than two (two children in the binary reduction tree).

In the pre-processing phase, we compute two masks M_c and M_r (line 2) for every GPU (process). A mask is a bit vector encoding a bit for each block column (M_c, size of N_g) or two bits of each block row (M_r, size of $2{\cdot}N_r$). According to the communication binary tree, one parent broadcast x subvector to two children at most. That is, one thread block waits for at most one message in each block column broadcast. In each row reduction, two (or one) children send its local summation ($lsum$) to their parent. Each thread block waits for at most two messages in a row reduction. The mask of a block column i, $M_c[i]$ (or row j, $M_r[j * 2]$ and $Mr[j * 2 + 1]$) represents whether block column i (or row j) needs communication or not. Thus, columns (or rows) that do not expect messages are masked. Each thread in the `WAIT` kernel has its own entry of the two masks.

Algorithm 1 Multi-GPU SpTRSV using two streams (variable definitions are listed in Table 1)

```
1: procedure PRE-PROCESSING(on CPU)
2:     Compute M_c, M_r for each GPU
3:     NVSHMEM launch WAIT, dimGrid(2), dimBlock(maxTH), stream[0]
4:     flag_w=0                ▷ the bit scalar to control the launch order
5:     while(flag_w!=1);                                  ▷ spin wait
6:     CUDA launch SOLVE, dimGrid(N_g), dimBlock(16x16), stream[1]
7: end procedure

8: procedure SOLVE(on GPU, stream[1])
9:     K = bid                 ▷ one thread block handles one block column
10:    if I am the diagonal process in charge of K then
11:        while(fmod(K)!=0);          ▷ spin wait, called by thread 0
12:        x(K)+ = lsum(K)
13:        x(K) = L(K,K)^{-1} · x(K)
                                        ▷ parallelize TRSV over threads
14:    else
15:        while(flag_x[K]!=1);        ▷ spin wait, called by thread 0
16:        NVSHMEM SEND ready_x(K) to my children's ready_x buffer
                                                  ▷ called by all threads
17:        for each L(I,K)! = 0, I > K  do
                                  ▷ parallelize I and GEMV over threads
18:            lsum(I) = lsum(I) + L(I,K) · ready_x(K)
19:            fmod(I) = fmod(I) − 1
20:            if fmod(I) == 0 then
21:                NVSHMEM SEND ready_lsum(I) to my parent's ready_lsum buffer
                                                  ▷ called by one thread
22:            end if
23:        end for
24:    end if
25: end procedure

26: procedure WAIT(on GPU,stream[0])
27:    flag_w=1                ▷ the bit scalar to control the launch order
28:    if bid==0 then                  ▷ handle block column broadcast
29:        while expecting more tasks do
30:            idx=nvshmem_int_wait_until_any(flag_x,M_c)
                                        ▷ M_c is distributed across threads
31:            M_c[idx]=1      ▷ message arrived in block column idx
32:        end while
33:    end if

34:    if bid==1 then                  ▷ handle block row reduction
35:        while expecting more tasks do
36:            idxr=nvshmem_int_wait_until_any(flag_lsum,M_r)
                                        ▷ M_r is distributed across threads
37:            M_r[idxr]=1     ▷ message arrived in block row idxr/2
38:            if (cnt[I]! = flag_lsum[I * 2] + flag_lsum[I * 2 + 1] then
                        ▷ cnt[I]: number of required messages in row I
                ▷ cnt[I]: pre-compute when building communication trees
39:                if flag_lsum[I * 2] == 1 then
40:                    lsum(I)+ = ready_lsum(I)
41:                    fmod(I) = fmod(I) − 1
42:                end if
43:                if flag_lsum[I * 2 + 1] == 1 then
44:                    lsum(I)+ = ready_lsum(I * 2)
45:                    fmod(I) = fmod(I) − 1
46:                end if
47:            end if
48:            if fmod(I) == 0 then
49:                NVSHMEM SEND ready_lsum(I) to my parent's ready_lsum buffer
                                                  ▷ called by one thread
50:            end if
51:        end while
52:    end if
53: end procedure
```

The `SOLVE` kernel in *stream[1]* (Algorithm 1 lines 8-25) performs computations (TRSV/GEMV), sends data and notification (lines 21 and 16) to receivers. As we assign one thread block to execute each supernode column, the number of thread blocks launched equals the number of local supernodes N_g on that GPU. Through empirical tuning, we use 256 threads for each thread block. Each GPU (process) performs TRSV first if it is a diagonal process (lines 10-14). Otherwise, it will spin wait until the message arrives ($flag\_x[K] = 1$: message arrived in block column K, line 15). Once it has received the message, that thread block immediately sends the x subvector to its children in the binary broadcast tree, and then performs GEMV computation, and sends $lsum$ if the counter $fmod(I)$ becomes zero. NVSHMEM SEND (line 16 and 21) is a set of NVSHMEM operations as Table 1 shows. The senders participating in column broadcast use three operations to send one message: ① `nvshmem_double_put_nbi_block()` called by thread blocks is used to send the data (x subvectors) to NVSHMEM buffer $ready\_x$. ② `nvshmemx_int_signal()` called by one thread is used to send the notification to NVSHMEM buffer $flag\_x$. ③ `nvshmem_fence()` is used between ① and ② to ensure that the notification is sent after the data. Senders in row reduction follow the same manner. The only difference is that they use `nvshmem_double_put_nbi()` called by one thread to send the $lsum$. Correspondingly, the point-to-point synchronization *nvshmemx_wait_until_any* in `WAIT` kernel guarantees that the message receive order preserves the message send order.

Table 1: Algorithm 1 References

	Descriptions		
N_g	number of thread blocks per GPU	`SOLVE` kernel	
N_r	number of block rows per GPU		
mz	maximum size of an individual message	equals to maximum supernode size	
bid	thread block id	1D grid	
$maxTH$	number of threads per block in `WAIT` kernel	1D, cudaOccupancyMaxPotentialBlockSize	
K	block column K		
I	block row I		
$mybrID$	block row id		
$ready\_x$	receive buffer for x	nvshmem_malloc, size is $mz \cdot N_g$	
$ready\_lsum$	receive buffer for block rows	nvshmem_malloc, size is $2 \cdot mz \cdot N_r$	
$flag\_x$	notification buffer for x	nvshmem_malloc, size is N_g	
$flag\_lsum$	notification buffer for $lsum$	nvshmem_malloc, size is $2 \cdot N_r$	
M_c	mask vector for block column broadcast	cuda_malloc, size is N_g	
M_r	mask vector for block row reduction	cuda_malloc, size is $2 \cdot N_r$	
	NVSHMEM Communication APIs	Descriptions	Callsite Scope
Sender	nvshmem_double_put_nbi_block	send x subvector	thread blocks
	nvshmem_double_put_nbi()	send $lsum$	threads
	nvshmem_fence()	ensure the orders	
	nvshmemx_int_signal()	send notification	threads
Receiver	nvshmem_int_wait_until_any()	receive notification	threads

The `WAIT` kernel in *stream[0]* (lines 26-53) uses *nvshmemx_wait_until_any* to receive the data completion notification from the senders. We launch two thread blocks with $maxTH$ threads in each thread block. Here $maxTH$ is the maximum block size returned by *cudaOccupancyMaxPotentialBlockSize*. The first thread block is used for column broadcasts while the second is used for row reductions. Consider the thread block used for block column broadcasts (line 28-33), recall that we have calculated a mask vector M_c in the pre-processing phase. Here we distribute this mask among the threads in this thread block, and let each thread wait for a sub-

set of all the block columns. Figure 2 describes the mask distribution among threads. Assuming we have a mask of length 17 (i.e., $N_g = 17$), and one thread block is launched for column broadcast, with five threads in it. $mask[i] = 0$ means the GPU expects one message in block column i, and 1 means that no message is needed. There are 10 messages in total, and we let each thread wait for two messages to balance the number of messages to be waited across the threads. $flag\_x$ is a bit vector including a bit for each column, and it is a NVSHMEM buffer which will be updated by other GPUs. While the mask M_c is a local buffer to manage the scope of waiting columns for each thread.

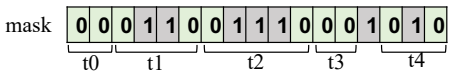


Figure 2: Each thread (ti) in `WAIT` kernel has its own entry in mask.$mask[i] = 0$ means block column i waits for one message. 1 means no message is needed.

In the row reduction (line 34-52), each block row has two bits in M_r and $flag\_lsum$ since one block row receives two messages at most. We initialize the two bits to zeros if one block row expects messages, and then distribute M_r across threads. Once received all required message, the corresponding thread will accumulate the local summation, and then send to its parent according to the binary reduction tree if the counter $fmod(I)$ becomes zero. Recall that each block row receives two messages at most, and we initialize two bits to zeros no matter it expects one message or two messages. Now the question is how receivers know whether they have received all required messages. A counter $cnt[I]$ (line 38) for each row I is computed when building communication trees on CPUs. Thus, block row I receives all required messages when the summation of the two bits in buffer $flag\_lsum$ equals to $cnt[I]$.

In contrast to traditional bulk synchronous parallel GPU implementations, our multi-GPU SpTRSV design can be considered as a message-driven algorithm. If a received message is a subvector of x, the GPU forwards the message according to the binary broadcast tree (line 16) before performing local accumulation. Otherwise if the message is $lsum$, the GPU accumulates it to the local sum. Once the counter $fmod$ becomes zero, the GPU has received all required messages and executed all its assigned GEMVs, then that GPU forwards the $lsum$ according to the binary reduction tree.

Inter-Stream Communication: The two kernels in two streams need to interact with each other so that (1) the `SOLVE` kernel can execute GEMV when the expected x subvector is received, and (2) both kernels need to track the counter $fmod$ in order to send the $lsum$ in time.

Recalled that the `WAIT` kernel uses $flag\_x$ (column broadcast) to receive the notification from the senders. That buffer is located in GPU global memory. Therefore, the `SOLVE` kernel can access $flag\_x$ simultaneously in another stream as Figure 3 shows (corresponding to Algorithm 1 line 15). Thread blocks that need to receive messages in the `SOLVE` kernel keep reading the $flag\_x$ buffer until the corresponding location in that buffer is updated by the `WAIT` kernel.

The counter $fmod$ is used to maintain the data dependency in a row reduction and is also located in GPU global memory so that the two kernels can access $fmod$ concurrently. The `SOLVE` kernel atomically decrements $fmod$ once it finishes the local inner-products. Meanwhile, the `WAIT` kernel also atomically decrements $fmod$ once it receives non-local messages as visualized in Figure 3 (corresponding to Algorithm 1 lines 20 and 48). When $fmod$ becomes zero, the corresponding thread, either the `SOLVE` kernel or the `WAIT` kernel, will send its local summation to its parent according to the binary reduction tree.

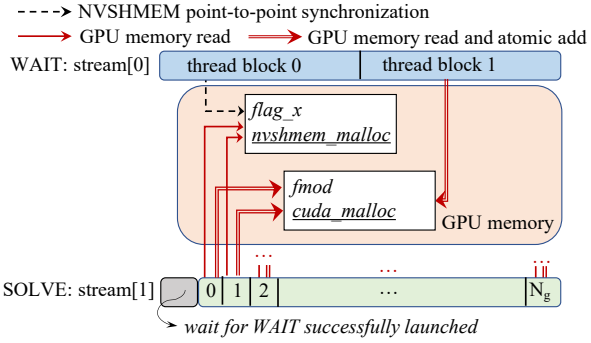


Figure 3: Communication between two streams. The two kernels use $flag\_x$ and $fmod$ to maintain data dependencies.

Extensibility: Our work can be migrated to other GPU-accelerated architectures that support GPU-initiated one-sided communications, e.g., AMD GPUs with ROC_SHMEM [20, 21] which has a similar syntax to NVSHMEM.

5 SpTRSV Performance Model for GPUs

We extend the critical path model [12] to GPUs to explain the observed performance, and explore the key features of matrices to estimate the potential performance benefits when using multiple GPUs.

The model is based on the critical path analysis which follows the task dependency graph of the sparse matrix using the well-known level-set method [13, 14] with a breadth-first search [15]. We further extend this critical path model to GPUs by (1) refining the DAG nodes in each level according to messaging patterns, and

Algorithm 2 SpTRSV performance model for GPUs

Application Inputs:
$Nsuper$: number of supernodes
w_i: width of block i (bytes)
h_i: height of block i (bytes)
Architecture Inputs:
bw_p: peak memory bandwidth per GPU
bw_g: GPU-GPU data transfer bandwidth (bytes/GEMV/second)
L_g: GPU-GPU data transfer latency
P: number of GPUs (processes)
MAX_TB: Number of thread blocks per GPU that can achieve the peak memory bandwidth
Outputs:
T_{comp_p}: TRSV/GEMV time of GPU p
T_{comm_p}: GPU-GPU data transfer time of GPU p
T_{tot}: total SpTRSV time

1: **procedure** MODELING
2: Analyze Critical path: find DAG levels and DAG nodes
3: Count $out_{b\_l\_p}$ (broadcast) of each DAG level l for GPU p
4: Count $out_{r\_l\_p}$ (reduction) of each DAG level l for GPU p
5: $bw_m = \frac{bw_p}{Nsuper/P}$ ▷ memory bandwidth per thread block (bytes/second)
6: **If** (bw_m < 1.3 GB/s) bw_m=1.3 GB/s;
7: **If** (bw_m > 5.2 GB/s) bw_m=5.2 GB/s;
8: **for** each DAG level l **do**
9: $mynodes_{l\_p} = 0$, $mybytes_{l\_p} = 0$
▷ $mynodes_{l\_p}$: number of active DAG nodes in level l of GPU p
10: **for** each DAG nodes i **do**
11: $mynodes_{l\_p} += 1$
12: $mybytes_{l\_p} += w_i \cdot h_i$
13: **if** $mynodes_{l\_p}$ >= MAX_TB **then** ▷ reach memory capacity
14: $T_{comp_{l\_p}} += \frac{mybytes_{l\_p}}{bw_p}$
▷ $T_{comp_{l\_p}}$: compute time of level l in GPU p
15: $mybytes_{l\_p} += 0$
16: $mynodes_{l\_p} += 0$
17: **end if**
18: **if** $mynodes_{l\_p}$ < MAX_TB && ends level l **then**
19: **if** $tune == 1$ **then**
20: $bw_m = \frac{bw_p}{mynodes_{l\_p}}$
21: **If** (bw_m < 1.3 GB/s) bw_m=1.3 GB/s;
22: **If** (bw_m > 5.2 GB/s) bw_m=5.2 GB/s;
23: **end if**
24: $T_{comp_{l\_p}} += \frac{mybytes_{l\_p}}{bw_m * mynodes_{l\_p}}$
25: **end if**
26: **end for**
27: **end for**
28: **for** each DAG level l **do**
29: **for** each supernode column c in DAG level l **do**
30: **if** $out_{b\_l\_p\_c}! = 0$ **then** ▷ broadcast
31: $out_{nb\_l\_p\_c} = out_{b\_l\_p\_c} > 2? \log_2(out_{b\_l\_p\_c}) : out_{b\_l\_p\_c}$
▷ $out_{nb\_l\_p\_c}$: non-overlapped messages in column c on level l in GPU p
32: $T_{comm_{l\_p}} += L_g + out_{nb\_l\_p\_c} \cdot \frac{w_i}{bw_g}$
33: **end if**
34: **end for**
35: **for** each supernode row c in DAG level l **do**
36: **if** $out_{r\_l\_p\_c}! = 0$ **then** ▷ broadcast
37: $out_{nr\_l\_p\_c} = out_{r\_l\_p\_c} > 2? \log_2(out_{r\_l\_p\_c}) : out_{r\_l\_p\_c}$
▷ $out_{nr\_l\_p\_c}$: non-overlapped messages of row c on level l in GPU p
38: $T_{comm_{l\_p}} += L_g + out_{nr\_l\_p\_c} \cdot \frac{h_i}{bw_g}$
39: **end if**
40: **end for**
41: **end for**
42: Find GPU p_{max} ($\sum_{l=0}^{L} T_{comm_{l\_p}} + T_{comp_{l\_p}}$) who has the longest time
43: $T_{tot} = T_{pmax}$
44: **If** ($\frac{T_{single\ GPU}}{T_{tot}} > P$) re-model with $tune = 1$
45: **end procedure**

(2) taking memory scaling bandwidth into consideration when modeling GEMV and TRSV time.

The computation dependency of SpTRSV can be precisely expressed by a DAG. Let us consider a L matrix which is factorized via `SuperLU_DIST` with METIS ordering for fill-in reduction [22]. Thus, DAG nodes refer to dense matrix-vectors, and edges between DAG nodes represent data dependencies. DAG nodes in the same level can be solved concurrently, and DAG levels must be solved sequentially. When it comes to multi-GPU SpTRSV, we can further remove the edges between DAG nodes that assigned to the same GPU. This is because thread blocks can each be executed independently and thus may execute in parallel. That is to say, DAG nodes located in one GPU can be solved concurrently by multiple thread blocks. Ultimately, the edges in the refined DAG represent only GPU-GPU messages.

Algorithm 2 details the extended GPU SpTRSV model. The SpTRSV time is modeled based on the refined DAG. The matrix features required to build the model are number of supernodes (N_{super}), number of nonzeros of each DAG node (w_i and h_i). The computation time of each GPU (process) is the accumulation time of DAG levels. In each level, memory bandwidth scales with the number of DAG nodes until the aggregate memory bandwidth reaches the peak (line 10-14). The empirical HBM bandwidth (bw_p) is 828 GB/s [23]. According to the white paper of NVIDIA Tesla V100 accelerator (V100 [2]), the maximum number of thread blocks per V100 is $\frac{80SMs \cdot 64warps}{8warps} = 640$, where 8 warps per thread block is based on our design. Therefore, we set the lower bound of memory bandwidth per thread block (bw_m) to 1.3 GB/s. However, the number of active thread blocks can be much smaller than the maximum of 640 due to either data dependencies or hardware limit. That is to say, in some cases, the bw_m can be larger than 1.3 GB/s. Let's consider the question of how many thread blocks can leverage a full bandwidth of a SM with dependencies. Ideally, the smallest number is two. One thread block spin waits the dependency and the other one can perform other independent computation work. Thus, the bandwidth per thread block $bw_m = \frac{828GB/s}{80SMs \cdot 2warps} = 5.2$ GB/s. Correspondingly, MAX_TB=$80 \cdot 2 = 160$. Ultimately, the upper and lower bound of bw_m is 5.2 GB/s and 1.3 GB/s.

The communication time is modeled according to the number of messages and message size of each DAG node on the critical path. We count the $out_{b\_l\_p\_c}$ (the number of broadcast messages happened in column c of level l in GPU p) and $out_{r\_l\_p\_c}$ (the number of reduction messages occurring in row c on level l in GPU p) according to the process decomposition. In column broadcast, each message has a size of the width

of block column i (w_i). The latency of multiple sends can be overlapped because all the messages are coming from the same producer. Let us assume there are P processes participating in the block column broadcast. When using a binary communication tree, each process that participates in the block column broadcast sends at most two messages to its children. This reduces the send message count of the corresponding process by $log_2 P$. Ultimately, for each GPU the accumulated communication time of each DAG level is the final communication time. The communication time on each level is estimated using the number of non-overlapped messages in GPU p (line 30-33). The row reduction follows the same manner. Each message size equals the height of block rows i (h_i). Thus, each GPU has a total SpTRSV time which equals to the accumulated time of computation and communication of DAG nodes on its critical path. We then take the longest SpTRSV time among the GPUs as the final SpTRSV time, and the critical path of that GPU is the final critical path.

We introduce a refinement feature in the model (line 44). Once we model the total SpTRSV time of using P GPUs, we compare the T_{tot} with the single GPU time. If the speedup is larger than the superlinear speedup P, we believe such discrepancy is due to the optimized memory bandwidth per thread block. Recall that by default every 160 DAG nodes (thread blocks) can achieve the peak memory bandwidth. Thus, at the end of DAG levels, each thread block may achieve a higher memory bandwidth than the initialized $bw_m = \frac{bw_p}{Nsuper/P}$ if the number of unsolved DAG nodes is less than the number of supernodes per GPU. (lines 19-23). In the refinement phase, we turn off that memory bandwidth adjustment, and use the the initialized bw_m instead of that optimized one.

BW_g and L_g are parameterized by benchmarked message sizes using a round-trip ping pong benchmark. When estimating the communication time, we round up (optimistic) the message size to the next power of two to match the corresponding BW_g in the model.

6 Results

In this section, we report experiment results and analysis of our multi-GPU SpTRSV using CUDA streams, including strong scaling performance evaluation with different process decompositions, and the analysis of the observed performance. We then discuss the key matrix features to determine the potential benefit of a matrix to use multiple GPUs.

6.1 Experimental Setup: Results presented in this paper were obtained on the GPU-accelerated partition on Summit at OLCF. Each of the Summit nodes contains two IBM POWER9 processors and six NVIDIA Tesla V100 accelerators. The GPUs within a node are connected by NVIDIA's NVLink interconnect. Summit nodes are connected using EDR InfiniBand interconnect. In all experiments, the SpTRSV runs on GPUs using double-precision real matrices. We use CUDA 10 and NVSHMEM 1.1.3 with GDRcopy 2.0.

Table 2 presents the key features of the matrices used in this paper. These matrices have also been used in various computational research [24–27]. Matrix S1 comes from M3D-C1, a fusion simulation code used for magnetohydrodynamics modeling of plasma [25]. All other matrices are publicly available through the SuiteSparse Matrix Collection [28]. The selected matrices cover a wide range of matrix properties (i.e., matrix size, sparsity structure, the number of level-sets, and application domain). The matrices are first factorized via `SuperLU_DIST` with METIS ordering for fill-in reduction [22]. The resultant lower triangular matrices are used with the proposed multi-GPU implementation.

6.2 Scalability Evaluation: Figure 4 shows our speedups (using the optimal process decomposition in each concurrency) compared to the single GPU version of cusparse_csrsv2(). Our single GPU implementation outperforms cusparse_csrsv2() by up to 1.9× speedup. Our multi-GPU SpTRSV provides a performance improvement of up to 6.1× when using twelve GPUs. Therefore, our implementation enables GPU-accelerated, distributed memory computing via NVSHMEM.

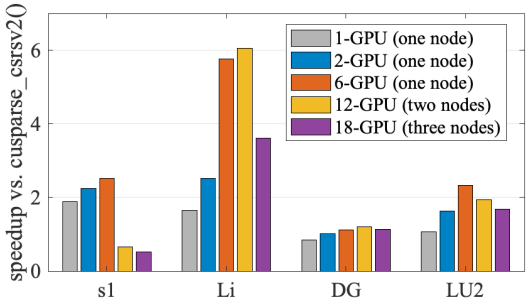
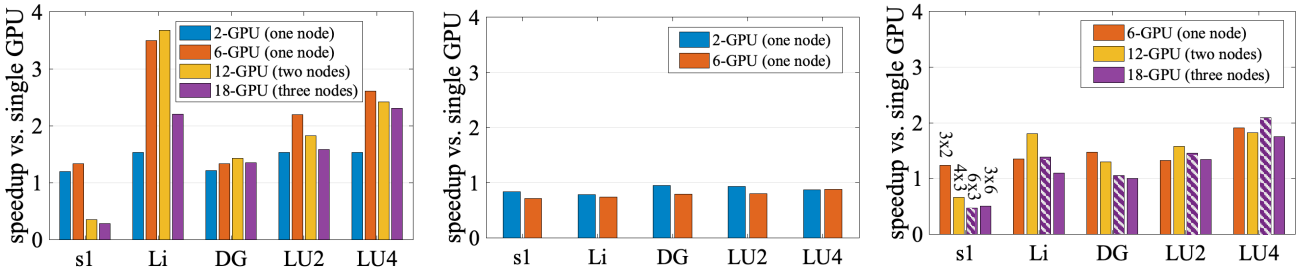


Figure 4: Multi-GPU Lower triangular solve time compared to *cusparse_csrsv2()*. Our implementation achieves up to 6.1× speedup with a scale from single GPU to eighteen GPUs (three Summit nodes).

Figure 5a shows the lower triangular solve speedup using a P×1 process decomposition (column broadcast) compared to our single GPU implementation. Our multi-GPU implementation using CUDA streams attains a speedup of up to 3.7× when using up to twelve GPUs (two nodes). Figure 5b presents the lower triangular solve speedup using a 1×P process decomposition (row reduction) on one Summit node. In this

Table 2: Test matrices.

Matrix	#supernodes	nnz in L	Levels	Maximum Speedup			
				2 GPUs	6 GPUs	12 GPUs	18 GPUs
S1	9,827	8.80E+08	388	1.2×	1.3×	0.7×	0.8×
DG_GrapheneDisorder (DG)	2,000	9.66E+08	199	1.2×	1.3×	1.4×	1.3×
LU_C_BN_C_2by2 (LU2)	1,216	8.54E+08	264	1.5×	2.2×	1.8×	1.9×
LU_C_BN_C_4by2 (LU4)	2,383	1.87E+09	320	1.5×	2.6×	2.4×	2.2×
Li4244 (Li)	362	5.18E+08	188	1.5×	3.5×	3.7×	2.7×



(a) P×1 process decomposition (column broadcast)

(b) 1×P process decomposition (row reduction).

(c) 2D process decomposition

Figure 5: Multi-GPU lower triangular solve time compared to our single GPU implementation. Our Multi-GPU implementation achieves up to 3.5× speedup on one Summit node, and achieves up to 3.7× speedup between two to three nodes using a P×1 process decomposition.

case, the performance of the single GPU outperforms the multi-GPU implementation. Figure 5c shows the speedups of using a 2D process decomposition (both column broadcast and row reduction). Performance results demonstrate that our implementation favors row parallelism over column parallelism, because the latter involves row reductions which are more expensive than column broadcasts.

Our multi-GPU SpTRSV is often able to exploit multiple GPUs on one node. On the other hand, performance is challenged when using GPUs that span multiple nodes but rarely falls off a cliff. Thus, when limited GPU memory capacity necessitates spreading a matrix over multiple nodes, our implementation can deliver acceptable performance whereas a single GPU implementation would be incapable of holding the matrix.

It is worth mentioning that although SpTRSV is challenged beyond twelve GPUs, it can scale to 4,096 processes on CPUs [12]. This is an artifact of faster GPU computational performance being offset by a much lower inter-node messaging performance. Specifically, a V100 provides 7 TFLOP/s of performance while a KNL core only provides about 0.4 TFLOP/s. At the same time, the one-sided NVSHMEM messaging performance is about 7× slower than foMPI which is used in work [12]. According to the microbenchmarks of our largest typical message size 1024 bytes, work [12] uses a network with 545 MB/s bandwidth, while the current NVSHMEM bandwidth is only 75 MB/s.

6.3 Strong Scaling Performance Analysis: We first discuss two key observations in Section 6.2: (1) speedups on multiple nodes are diminished, and (2) the selected matrices demonstrate very different scaling behaviours. We then demonstrate (3) the predictive ability of our model, which can help users to determine the number of GPUs that produces the fastest run time.

Inter-GPU Networking Performance. The smaller speedups on multiple nodes are due to the low performance of the inter-node GPU-GPU network. Figure 6 highlights the NSVHMEM SEND bandwidth between two GPUs (processes) of intra-socket, intra-node and inter-node using three different callsite scopes: **Thread block:** Use all threads in thread blocks to put data to the target GPU (process) by `nvshmem_double_put_nbi_block`, and then perform a `nvshmem_fence`. Finally, use thread 0 to send notification via `nvshmemx_int_signal`. **Warp:** Use one warp in each thread block to put data to the target GPU with `nvshmemx_double_put_warp`, and the rest remain the same with *thread block*. **Thread:** Use one thread in each thread block to put data to the target GPU with `nvshmem_double_put`, and the rest remain the same with *thread block*.

The GPU-GPU bandwidth using *thread blocks* outperforms the performance using warps and threads by 2× and 9× on average. Using thread blocks or warps deliver the same performance as using threads for inter-node communication. This is because only one single thread in a thread block/warp can issue an RMA write operation to the destination GPU over InfiniBand.

Ultimately, one should remember that the performance of one-sided messaging libraries can vary sub-

stantially. For example, on the Cray Aries network, Cray's one-sided implementation is 2.7× slower than Cray's two-sided [12] yet ETH's foMPI is 3× faster. Here, one-sided NVSHMEM is 2.3× slower than IBM Spectrum MPI over InfiniBand network on Summit [29].

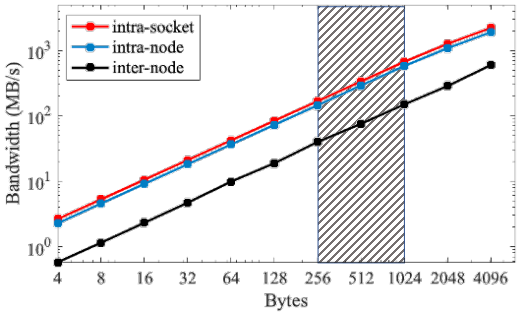


Figure 6: NVSHMEM SEND (thread block) bandwidth using two GPUs on Summit. The shadowed stripe highlights the typical message size in SpTRSV of 256 bytes to 1,024 bytes. Intra-socket NVSHMEM SEND outperforms intra-node NVSHMEM SEND (avg. 1.2×) and inter-node NVSHMEM SEND (avg. 3.9×).

Process Decomposition and Performance. Recall that the 1×P implementation leverages only a single thread to perform NVSHMEM SENDs (Algorithm 1 line 21 and 49), the overall SpTRSV performance using a 1×P implementation (Figure 5b) is limited by the resultant low NVSHMEM SEND throughput. In a 2D process decomposition, as we increase the number of GPUs participating in the row reduction, the number of messages (single thread NVSHMEM SEND) in the row reduction increases, and thus the performance decreases. Essentially, the smaller speedup for the 2D process decomposition compared to the P×1 implementation is again due to the low NVSHMEM SEND throughput (single thread) in row reductions.

Matrix Properties and Performance. Matrix DG and Li have a similar number of DAG levels: 199 and 188, respectively. Since DG has more nonzeros (966 million nonzeros) than Li (518 million nonzeros), one might assume that DG scales better than Li. However, the reality is that Li achieves 2.7× speedup on average (up to 3.7× on twelve GPUs) while DG has 1.3× speedup on average (up to 1.4× on six GPUs). Such a discrepancy is due to ignoring communication and memory bandwidth.

According to the model in Algorithm 2, matrix Li has 162 message on the critical path using two GPUs, 270 messages using six GPUs, and 292 message using twelve GPUs, while DG has 1,000, 898 and 571 messages, respectively. Hence, one can immediately understand that the large number of messages of DG makes its scaling performance worse than matrix Li.

Another aspect is the achieved memory bandwidth per thread block. Matrix Li has only 362 supernodes in total. Therefore, it can achieve 4.6 GB/s ($\frac{828}{362/2}$) when using two GPUs and 5.2 GB/s (upper bound of memory bandwidth per thread block, Algorithm 2 line 7) when using more than two GPUs. While DG achieves only 1.3 GB/s (lower bound of memory bandwidth per thread block, Algorithm 2 line 6) with 1,000 supernodes per GPU when using two GPUs, and 4.0 GB/s when using twelve GPUs (highest speedup, 1.4×). A smaller number of supernodes (thread blocks) per GPU produces less memory contention. Thus, each thread block can achieve a higher memory bandwidth.

Ultimately, matrix Li achieves the best scaling performance among the selected matrices. This is due the weak data dependency (only 188 DAG levels and a small number of messages on the critical path) and high memory bandwidth per thread block (5.2 GB/s). Even though matrix DG has a similar number of levels as matrix Li, it does not scale as well as Li due to the limited memory bandwidth per thread block and relatively larger number of messages on the critical path.

Model prediction. Figure 7 visualized the modeled time and measured times of the S1 matrix (achieves the smallest speedup at scale, up to 1.3×, among the matrices in Table 2) and the Li matrix (highest speedup, up to 3.7×). The total modeled SpTRSV time equals the accumulation of the communication time on the critical path (yellow bar) and the computation time on the critical path (blue bar). In addition to understanding and explaining observed performance, the SpTRSV model can also help identify the number of GPUs that produces the fastest run time.

The overall sweet spot of S1 is six GPUs within a node. From the model, we see that the numbers of messages on the critical paths are very similar: 7,922 messages (six GPUs) and 7,408 message (twelve GPUs), but the network bandwidth decreases as more nodes are used. Ultimately, the resultant low inter-node NVSHMEM SEND throughput makes the run time of S1 increase when using more than one node.

Unlike the S1 matrix, the communication time of Li only takes 33% when using up to eighteen GPUs. It's the computation time dominants the total run time. From one to six GPUs, we see a nearly linear reduction in computation time of matrix Li because the achieved memory bandwidth increases: 1.3 GB/s per thread block using one and two GPUs (362 supernodes using one GPU, and 181 supernodes per GPU using two GPUs) and 5.2 GB/s per thread block using six GPUs (55 supernodes per GPU).

Rather than using average parallelism of a matrix as metrics, our model incorporates the number of levels

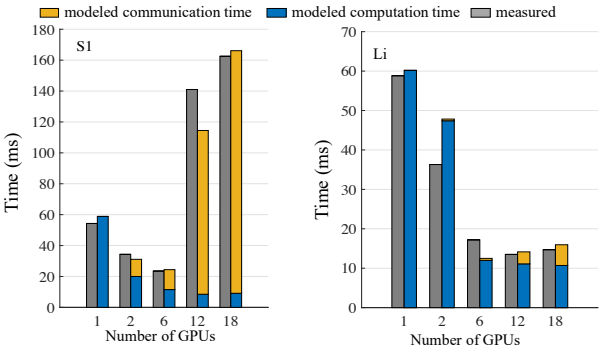


Figure 7: The performance model can help identify the number of GPUs that produces the fastest run time. The total modeled SpTRSV time equals the accumulation of the blue bar and the yellow bar.

(dependency), number of non-zeros on the critical path, and architecture features (memory bandwidth, and network bandwidth). The model highlights that the improvement in memory bandwidth per thread block (not just aggregate bandwidth) can help reduce computation time for matrices like S1, while improving NVSHMEM SEND throughput can help attain better scalability for matrices like Li.

7 Related Work

Exploring high performance SpTRSV is becoming ever more crucial on GPU-accelerated architectures. Most existing parallel GPU triangular solvers focus on optimizing single GPU performance [6, 30–33]. Due to the complex data dependencies in SpTRSV, algorithm optimization has been mainly based on the level-set methods and color-set methods for various parallel architectures. Additionally, there is research focused on optimizing the block structure [34], or analyzing nonzero layout and selecting the best sparse kernels by using machine learning and deep learning methods [35–37]. Our work explore the benefits of using multiple GPUs, and we believe that our proposed method of coupled producer-consumer parallelism using CUDA streams can bring insightful experience for DAG-based computations on emerging accelerated architectures.

The existing work of distributed-memory SpTRSV have mainly conducted on CPU platforms. Using 1D or 2D process layouts to improve load balance is discussed in work [38–40]. An asynchronous binary tree is proposed to reduce the communication latency [17]. Venkat et al. [41, 42] developed several techniques that generate wavefront parallelization with faster level-set scheduling. One-sided communication is used in work [12] to implement a synchronization-free task queue to manage messages between producer-consumer pairs. Xie et al. propose a multi-GPU SpTRSV using a 1×P process decomposition [43]. The inter-GPU communications are performed via NVSHMEM warp-level get, and it outperforms *cusparse_csrsc2()* by up to 3.2× on average using 16 GPUs on one DGX-2 node with 20% parallel efficiency. In comparison, our new algorithm achieves up to 2.9× speedup on average using six GPUs on one Summit node with 58% parallel efficiency. Hamidouche et al. implement a multi-GPU SpTRSV on AMD GPUs using ROC_SHMEM [21]. They achieved up to a 3.7× speedup compared to a baseline that used intra-kernel communication (rely on CPU threads to perform network operations on behalf of the GPU) rather than the optimal single GPU solution. It is worth mentioning that our work starts from a faster baseline. In addition, comparisons of AMD ROC_SHMEM on AMD GPUs using a column-based approach against NVIDIA NVSHMEM on NVIDIA GPUs using a supernodal approach imperil meaningful insights as too many variables were changed.

8 Conclusion

We use CUDA streams to perform coupled producer-consumer parallelism in multi-GPU SpTRSV. Over the range of two to eighteen GPUs, our multi-GPU SpTRSV implementation improve solve time by up to 3.7× compared to our single GPU implementation, and up to 6.1× compared to *cusparse_csrsv2()*. In order to assess our observed performance relative to machine capabilities and matrix features, we constructed a critical path performance model. Our SpTRSV model endows users with far greater insights as to how different aspects of multi-GPU architectures and matrix features constrain performance, e.g., the limited achieved memory bandwidth per thread block constrains the scaling performance of matrix S1, while low NVSHMEM SEND throughput constrains the performance when using 1×P and 2D decompositions.

Fusion simulations like M3DC1 [25] and NIMROD [44], solve highly ill-conditioned linear systems. One successful method is to use GMRES with a block-Jacobi preconditioner, where each diagonal block is solved by SuperLU_DIST. There is no inter-block communication within the preconditioner. Our results highlight the computational importance of keeping block size small enough so that it fits on a single node.

In the future, we will continue to refine the model to highlight the performance nuances, use the model to identify potentially superior process mappings, and port our supernodal-based triangular solver to other emerging accelerators. More broadly, we will explore the value of our multi-CUDA stream approach in other domains.

References

[1] Top500 Highlights - November 2020. URL https://www.top500.org/lists/top500/2020/11/highs/.

[2] Nvidia Tesla. V100 gpu architecture. *Online verfügbar unter http://images. nvidia. com/content/volta-architecture/pdf/volta-architecture-whitepaper. pdf, zuletzt geprüft am*, 21, 2018.

[3] István Z Reguly, Gihan R Mudalige, Carlo Bertolli, Michael B Giles, Adam Betts, Paul HJ Kelly, and David Radford. Acceleration of a full-scale industrial CFD application with OP2. *IEEE Transactions on Parallel and Distributed Systems*, 27(5): 1265–1278, 2016.

[4] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on uncertainty in artificial intelligence (UAI)*, pages 340–349, 2010.

[5] Maxim Naumov. Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the gpu. *NVIDIA Corp., Westford, MA, USA, Tech. Rep. NVR-2011*, 1, 2011.

[6] Weifeng Liu, Ang Li, Jonathan Hogg, Iain S Duff, and Brian Vinter. A synchronization-free algorithm for parallel sparse triangular solves. In *European Conference on Parallel Processing*, pages 617–630. Springer, 2016.

[7] NVIDIA NVSHMEM Documentation. URL https://docs.nvidia.com/hpc-sdk/nvshmem/index.html.

[8] Jeff R Hammond, Sayan Ghosh, and Barbara M Chapman. Implementing OpenSHMEM using MPI-3 one-sided communication. In *Workshop on OpenSHMEM and Related Technologies*, pages 44–58. Springer, 2014.

[9] M. Sourouri, T. Gillberg, S. B. Baden, and X. Cai. Effective multi-gpu communication using multiple cuda streams and threads. In *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 981–986, 2014. doi: 10.1109/PADSW.2014.7097919.

[10] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2009.

[11] Nan Ding and Samuel Williams. An instruction roofline model for gpus. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 7–18. IEEE, 2019.

[12] Nan Ding, Samuel Williams, Yang Liu, and Xiaoye S Li. Leveraging one-sided communication for sparse triangular solvers. In *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing*, pages 93–105. SIAM, 2020.

[13] Edward Anderson and Youcef Saad. Solving sparse triangular linear systems on parallel computers. *International Journal of High Speed Computing*, 1 (01):73–95, 1989.

[14] Joel H Saltz. Aggregation methods for solving sparse triangular systems on multiprocessors. *SIAM journal on scientific and statistical computing*, 11(1):123–144, 1990.

[15] Scott Beamer, Krste Asanović, and David Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 21(3-4):137–148, 2013.

[16] James W Demmel, Stanley C Eisenstat, John R Gilbert, Xiaoye S Li, and Joseph WH Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

[17] Yang Liu, Mathias Jacquelin, Pieter Ghysels, and Xiaoye S Li. Highly scalable distributed-memory sparse triangular solution algorithms. In *Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing*, pages 87–96. SIAM, 2018.

[18] Alexandros V Gerbessiotis and Leslie G Valiant. Direct bulk-synchronous parallel algorithms. *Journal of parallel and distributed computing*, 22(2): 251–267, 1994.

[19] An Introduction to CUDA-Aware MPI. URL https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/.

[20] ROC_SHMEM. https://github.com/ROCm-Developer-Tools/ROC_SHMEM, 2020.

[21] Khaled Hamidouche and Michael LeBeane. Gpu initiated openshmem: correct and efficient intra-kernel networking for dgpus. In *Proceedings of*

the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pages 336–347, 2020.

[22] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998. doi: 10.1137/S1064827595287997. URL https://doi.org/10.1137/S1064827595287997.

[23] Charlene Yang, Thorsten Kurth, and Samuel Williams. Hierarchical roofline analysis for gpus: Accelerating performance optimization for the nersc-9 perlmutter system. *Concurrency and Computation: Practice and Experience*, 32(20):e5547, 2020.

[24] Ulrike Baur, Peter Benner, Andreas Greiner, Jan G Korvink, Jan Lienemann, and Christian Moosmann. Parameter preserving model order reduction for MEMS applications. *Mathematical and Computer Modelling of Dynamical Systems*, 17(4):297–317, 2011.

[25] S C Jardin, N Ferraro, X Luo, J Chen, J Breslau, K E Jansen, and M S Shephard. The M3D-C1 approach to simulating 3D 2-fluid magnetohydrodynamics in magnetic fusion experiments. *J. Phys. Conf. Ser.*, 125(1):012044, 2008. URL http://stacks.iop.org/1742-6596/125/i=1/a=012044.

[26] Alexander Ludwig. The Gauss–Seidel–quasi-Newton method: A hybrid algorithm for solving dynamic economic models. *Journal of Economic Dynamics and Control*, 31(5):1610–1632, 2007.

[27] Gary Kumfert and Alex Pothen. Two improved algorithms for envelope and wavefront reduction. *BIT Numerical Mathematics*, 37(3):559–590, 1997.

[28] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011. ISSN 0098-3500. doi: 10.1145/2049662.2049663. URL http://doi.acm.org/10.1145/2049662.2049663.

[29] Summit User Guide. URL https://docs.olcf.ornl.gov/systems/summit_user_guide.html.

[30] Jiya Su, Feng Zhang, Weifeng Liu, Bingsheng He, Ruofan Wu, Xiaoyong Du, and Rujia Wang. Capellinisptrsv: A thread-level synchronization-free sparse triangular solve on gpus. In *49th International Conference on Parallel Processing-ICPP*, pages 1–11, 2020.

[31] Ruipeng Li and Chaoyu Zhang. Efficient parallel implementations of sparse triangular solves for gpu architectures. In *Proceedings of the 2020 SIAM Conference on Parallel Processing for Scientific Computing*, pages 106–117. SIAM, 2020.

[32] Zhengyang Lu, Yuyao Niu, and Weifeng Liu. Efficient block algorithms for parallel sparse triangular solve. In *49th International Conference on Parallel Processing-ICPP*, pages 1–11, 2020.

[33] Ernesto Dufrechou and Pablo Ezzatti. A new gpu algorithm to compute a level set-based analysis for the parallel solution of sparse triangular systems. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 920–929. IEEE, 2018.

[34] Ehsan Totoni, Michael T Heath, and Laxmikant V Kale. Structure-adaptive parallel solution of sparse triangular linear systems. *Parallel Computing*, 40 (9):454–470, 2014.

[35] Guangming Tan, Junhong Liu, and Jiajia Li. Design and implementation of adaptive spmv library for multicore and many-core architecture. *ACM Transactions on Mathematical Software (TOMS)*, 44(4):46, 2018.

[36] Yue Zhao, Jiajia Li, Chunhua Liao, and Xipeng Shen. Bridging the gap between deep learning and sparse matrix format selection. In *ACM SIGPLAN Notices*, volume 53, pages 94–108. ACM, 2018.

[37] Jiajia Li, Guangming Tan, Mingyu Chen, and Ninghui Sun. SMAT: an input adaptive auto-tuner for sparse matrix-vector multiplication. In *ACM SIGPLAN Notices*, volume 48, pages 117–126. ACM, 2013.

[38] Ehsan Totoni, Michael T. Heath, and Laxmikant V. Kale. Structure-adaptive parallel solution of sparse triangular linear systems. *Parallel Computing*, 40(9):454 – 470, 2014. ISSN 0167-8191. doi: https://doi.org/10.1016/j.parco.2014.06.006. URL http://www.sciencedirect.com/science/article/pii/S0167819114000799.

[39] Xiaoye S. Li and James W. Demmel. Making sparse Gaussian elimination scalable by static pivoting. In *Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, SC '98, pages 1–17, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-89791-984-X. URL http://dl.acm.org/citation.cfm?id=509058.509092.

[40] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw.*, 29(2):110–140, June 2003. ISSN 0098-3500. doi: 10.1145/779359.779361. URL `http://doi.acm.org/10.1145/779359.779361`.

[41] Anand Venkat, Mahdi Soltan Mohammadi, Jongsoo Park, Hongbo Rong, Rajkishore Barik, Michelle Mills Strout, and Mary Hall. Automating wavefront parallelization for sparse matrix computations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 41. IEEE Press, 2016.

[42] Anand Venkat, Mary Hall, and Michelle Strout. Loop and data transformations for sparse matrix code. In *ACM SIGPLAN Notices*, volume 50, pages 521–532. ACM, 2015.

[43] Chenhao Xie, Jieyang Chen, Jesun S Firoz, Jiajia Li, Shuaiwen Leon Song, Kevin Barker, Mark Raugas, and Ang Li. Fast and scalable sparse triangular solver for multi-gpu based hpc architectures. *arXiv preprint arXiv:2012.06959*, 2020.

[44] C.R. Sovinec, A.H. Glasser, T.A. Gianakon, D.C. Barnes, R.A. Nebel, S.E. Kruger, S.J. Plimpton, A. Tarditi, M.S. Chu, and the NIMROD Team. Nonlinear magnetohydrodynamics with high-order finite elements. *J. Comp. Phys.*, 195:355, 2004.

A Dynamic Program for Computing the Joint Cumulative Distribution Function of Order Statistics*

Rigel Galgana † Cengke Shi ‡ Amy Greenwald § Takehiro Oyakawa ¶

Abstract

We consider the problem of computing the joint cumulative distribution function of $d \in \mathbb{N}$ select order statistics of $n \geq d$ independent random variables representing $m \leq n$ distinct populations. We present an efficient dynamic programming algorithm for this problem that is polynomial in *both* d and n, though exponential in m, hence most practical when $m \ll n$. Previously, the fastest known algorithms were exponential in d, and either exponential in m or exponential in n, as well.

Like others before us, we reduce the problem to a combinatorial one of tossing balls into bins, and then we calculate the probability of satisfying various bin conditions: i.e., of sufficiently many balls landing in bins. The key observation that leads to such a significant speedup is that one need not keep track of the exact configurations of balls in bins to determine if the requisite bin conditions are satisfied. Instead, the relevant information can be compressed by merging configurations that satisfy the same bin conditions.

1 Introduction

Let $X_1, \ldots, X_n$ denote a collection of n real-valued random variables. The ith order statistic, denoted as $X_{(i)}$, is defined as the ith smallest value of $X_1, \ldots, X_n$.

Order statistics are used to tackle fundamental problems in statistics, such as outlier detection [8] and multiple comparisons testing [6], and many problems encountered in empirical process theory [18] and in goodness-of-fit testing [19]. They are also relevant in various other disciplines, such as wireless communication [9] and digital signal processing [7].

One reason for their widespread use is that estimators based on order statistics, such as L-estimators, have strong robustness guarantees. Consequently, there is interest in better understanding the behavior of order statistics, for example, by computing the joint cumulative distribution function (cdf) $F_{X_\mathcal{C}}(\mathbf{x})$ of a set $\mathcal{C} = (c_1, \ldots, c_d)$ of $d \in \mathbb{Z}_{>0}$ select order statistics of $n \geq d$ independent random variables, where $\mathbf{x} \in \mathbb{R}^d$:[1]

$$\begin{aligned} F_{X_\mathcal{C}}(\mathbf{x}) &= F_{X_{(c_1)},\ldots,X_{(c_d)}}(x_1, \ldots, x_d) \\ &= \mathbb{P}\left(X_{(c_1)} \leq x_1, \ldots, X_{(c_d)} \leq x_d\right) \\ &= \int_{t_1,\ldots,t_d \in \mathbb{R}^d \text{ s.t. } t_i \leq x_i, \forall i \in [d]} f_{X_\mathcal{C}}(\mathbf{t})\, d\mathbf{t} \ . \end{aligned}$$

Here, $f_{X_\mathcal{C}}(\mathbf{x})$ denotes the corresponding joint probability density function. Given a natural number $a \geq 1$, we use the notation $[a]$ to denote the set of integers, 1 through a, inclusive, and $[a^*]$ to denote $[a] \cup \{0\}$.

Assuming n independent and identically distributed (i.i.d.) random variables, each distributed according to F, the CDF of a single order statistic $X_{(i)}$ is (e.g., [14]):

$$F_{X_{(i)}}(x) = \sum_{j=i}^{n} \binom{n}{j} F(x)^j (1 - F(x))^{n-j} \ .$$

The joint CDF of a small number of order statistics is also relatively simple in the i.i.d. case. For example, assuming two order statistics s.t. $x_i \leq x_j$, their joint CDF is given by (e.g., [14]): $F_{X_{(i)},X_{(j)}}(x_i, x_j) = \sum_{s=j}^{n} \sum_{r=i}^{s} \frac{n!}{r!(s-r)!(n-s)!} F(x_i)^r \left(F(x_j) - F(x_i)\right)^{s-r} \left(1 - F(x_j)\right)^{n-s}$.

A generalization of this formula to the joint CDF of a set $\mathcal{C}$ of d select order statistics at $\mathbf{x} \in \mathbb{R}^d$ is a special case of the Bapat-Beg theorem [10]. Letting $x_0 = -\infty$ and $x_{d+1} = +\infty$,

$$F_{X_\mathcal{C}}(\mathbf{x}) = n! \sum_{\mathbf{i} \in \mathcal{I}} \prod_{j=1}^{d+1} \frac{\left[F(x_j) - F(x_{j-1})\right]^{i_j - i_{j-1}}}{(i_j - i_{j-1})!} \ . \tag{1.1}$$

Here, $\mathcal{I}$ is the set of all non-decreasing $(d+1)$-vectors such that the jth entry is at least c_j. More precisely, $\mathcal{I} \equiv \{\mathbf{i} \in [n^*]^d \mid c_j \leq i_j \leq c_{j+1}, \forall j \in [d]\}$, where $i_0 = 0$

*The first two authors contributed equally to this work. The last two authors are supported by NSF Award CMMI-1761546.

†This research was carried out while a student at Brown University. Email: rigel_galgana@alumni.brown.edu

‡This research was carried out while a student at Brown University. Email: cengkeshi@gmail.com

§Department of Computer Science, Brown University. Email: amy_greenwald@brown.edu

¶Department of Computer Science, Brown University. Email: takehiro_oyakawa@brown.edu

[1] WLOG, we assume $\mathcal{C}$, the index set, is ordered, and the entries in $\mathbf{x} \in \mathbb{R}^d$ are strictly increasing, as otherwise, the problem may be simplified to exclude out-of-order entries. For example, if $x_1 \geq x_2$, then $F_{X_{(1)},X_{(2)}}(x_1, x_2) = F_{X_{(1)}}(x_1)$. Similarly, if $c_1 \geq c_2$ while $x_1 < x_2$, then $F_{X_{(c_1)},X_{(c_2)}}(x_1, x_2) = F_{X_{(c_1)}}(x_1)$.

and $i_{d+1} = n$, a set whose size grows exponentially with $d = |\mathcal{C}|$. Without the assumption of independent or identically distributed random variables, this computation becomes even more intractable.

While vanilla Monte Carlo approximation may sometimes suffice to estimate the joint CDF, when the probabilities are very small, or if exact values are required, simulation-based methods tend to fail. To address these issues, several exact algorithms have been proposed. One such method, due to Bapat and Beg [10], which relies on the computation of block permanents, yields a closed-form expression for the joint CDF in the general case of $m = n$ independent but not necessarily identically distributed random variables. With the further assumption that $m \ll n$, Glueck et al. [12] improved upon the Bapat-Beg theorem by simplifying the computation of each block permanent. Efficient, polynomial-time algorithms are also known for the very special case in which $m = 2$ and $n = d$, as shown in von Schroeder and Dickhaus [17].[2]

Still, to the best of our knowledge, no procedure is known for the general case of computing the joint CDF of d select order statistics among n random variables in time polynomial in both these quantities. In this paper, we present such an algorithm. Moreover, our algorithm generalizes to a heterogeneous population in which the random variables are drawn from multiple homogeneous populations, although our algorithm is exponential in the number of populations, m.

For notational simplicity, we focus primarily on the special case of computing the joint CDF of d order statistics of n i.i.d. random variables, as the key idea that underlies our solution manifests itself in the i.i.d. case. In this special case, our dynamic programming algorithm has time complexity $O(dn^2)$ and space complexity $O(dn)$, so it is polynomial in both n and d. The previous best known algorithms for this case were given by Glueck et al. [12] and Boncelet Jr. [11], both of which are exponential in d.

We then explain how our algorithm generalizes to the case of a heterogeneous population described by m independent random variables (i.e., m homogeneous populations), with n_l individuals of each type $l \in [m]$. Our approach has time complexity $O\left(d \prod_{l=1}^{m} n_l^2\right)$ and space complexity $O\left(d \prod_{l=1}^{m} n_l\right)$, so it remains polynomial in n and d, but is exponential in m. Hence, in the non-i.i.d. case, our algorithm exhibits the greatest improvement over existing algorithms when d is large.

This paper is outlined as follows. In the next section, we translate the problem of interest into an equivalent combinatorial problem in the case of i.i.d. random variables. We then relate our problem formulation to past work (Section 3). Next, we present our key insight, which allows us to write down a more compact recurrence relation than previously known, based on which we derive an efficient dynamic program (Section 4). We then present experiments (Section 5) that compare empirical the run-times of the Bapat-Beg theorem, Glueck et al.'s, Boncelet Jr.'s, and our algorithm, before concluding with applications (Section 6).

2 The Combinatorial Problem

In this section, we describe how the computation of the joint CDF of a collection of order statistics of independent random variables can be reduced to an equivalent combinatorial problem of tossing balls into bins. Later, we develop an efficient dynamic program for this computation based on our combinatorial problem statement. Boncelet Jr. [11] also noted that this computation can be expressed combinatorially, and likewise developed a corresponding dynamic program; but his problem statement is less compact than ours, and his dynamic program is correspondingly less efficient. On the other hand, his dynamic program handles dependent as well as independent random variables, whereas ours only handles the latter.[3]

We assume a heterogeneous population characterized by m independent random variables (i.e., m homogeneous populations), with n_l individuals of each type $l \in [m]$. More specifically, for all types $l \in [m]$, we assume n_l random variables distributed according to F_i^l: i.e., $X_i^l \sim F_i^l$, for all $i \in [n_l]$. We are interested in computing the joint CDF $F_{\mathcal{C}}(\mathbf{x})$ at a point $\mathbf{x} \in \mathbb{R}^d$ of d select order statistics $\mathcal{C} = \{c_1, \ldots, c_d\}$. To restate this computation as a problem involving balls and bins, we start by defining $x_0 = -\infty$ and $x_{d+1} = +\infty$, and decomposing the real line into $d + 1$ intervals, $I_1, \ldots, I_{d+1}$, hereafter bins: $(-\infty, +\infty) = (x_0, x_{d+1}) = \cup_{j=1}^{d}(x_{j-1}, x_j] \cup (x_d, x_{d+1}) = \cup_{j=1}^{d+1} I_j = I_{1:d+1}$. Here, $I_{j:k} = \cup_{i=j}^{k} I_i$ denotes a "superbin:" i.e., the union of the jth through kth bins, inclusive. The probability that random variable X_i^l realizes a value in the superbin $I_{1:j}$ is then given by $F_i^l(x_j)$.

We use the indicator variable $\mathbf{1}_{\{X_i^l \in I_j\}}$ to denote the event "the ith ball of type l resides in the jth bin." Moreover, we let $p_j^l = \mathbb{P}\left(\{X_i^l \in I_j\}\right) = F_i^l(x_j) - F_i^l(x_{j-1})$ denote the probability that a ball of type l resides in the jth bin, for all $j \in [d]$, with $p_{d+1}^l = 1 - \sum_{j=1}^{d} p_j^l$. (We drop the index i when defining this probability, since the populations are homogeneous.)

[2] We do not discuss von Schroeder and Dickhaus' algorithms here, since our algorithms achieve similar complexity guarantees in the more general case, when $m > 2$ and $d < n$.

[3] Furthermore, Boncelet Jr.'s algorithm is more efficient when d is small and m is large, because it is constant in m.

We can now reinterpret the joint CDF computation of interest as a combinatorial problem—specifically, one of n tossing balls into d bins with the aforementioned probabilities. To do so, we adopt the following notation:

- For $l \in [m]$, for $i \in [n_l^*]$, we define the sequence of random variables $C_j^l(i) \doteq \sum_{h=1}^{i} \mathbf{1}_{\{X_h^l \in I_j\}} \in [i^*]$ as "the number of balls of type l among i such balls that lands in the jth bin." For $i \in [n^*]$, we then define $C_j(i) \doteq \sum_{l=1}^{m} C_j^l(i) \in [i^*]$ as "the total number of balls (of any type) among i balls that lands in the jth bin." We also use the random vector $\mathbf{C}(i) = (C_1(i), \ldots, C_d(i)) \in [n^*]^d$ to denote "the total number of balls (of any type) among i balls that lands in each of the first d bins." **N.B.** These definitions are more pertinent in Boncelet Jr.'s algorithm, which iterates over balls, whereas our approach is better understood as computing probabilities after all n balls have been tossed.

- Next, we define $C_j^l \doteq C_j^l(n_l) \in [n_l^*]$ as "the number of balls of type l that reside in the jth bin." Overloading notation, we also use the random variable $C_{j:j'}^l \doteq \sum_{i=j}^{j'} C_i^l \in [n_l^*]$ as "the total number of balls of type l that resides in superbin $I_{j:j'}$." Analogously, we define $C_j \doteq \sum_{l=1}^{m} C_j^l \in [n^*]$ as "the number of balls (of any type) that reside in the jth bin" and $C_{j:j'} \doteq \sum_{i=j}^{j'} C_i \in [n^*]$ as "the number of balls (of any type) that reside in superbin $I_{j:j'}$."

- Moreover, we use the random vectors $\mathbf{C}^l \doteq (C_1^l, \ldots, C_d^l) \in [n_l^*]^d$ to denote "the number of balls of type l that reside in each of the first d bins" and $\mathbf{C}_j \doteq (C_j^1, \ldots, C_j^m) \in \bigotimes_{l=1}^{m} [n_l^*]$ to denote "the number of balls of each type that reside in bin j." We also define $\mathbf{C}_{j:j'} \doteq \sum_{i=j}^{j'} \mathbf{C}_i \in \bigotimes_{l=1}^{m} [n_l^*]$ as "the total number of balls of each type that resides in superbin $I_{j:j'}$."

- We define the event $D_j \doteq \{C_{1:j} \geq c_j\}$ as "at least c_j balls reside in superbin $I_{1:j}$" for $j \in [d]$.

- Overloading notation again, we also define the event $D_{j:j'} \doteq \cap_{i=j}^{j'} D_i$ as "at least c_i balls reside in superbin $I_{1:i}$, for all $i \in \{j, \ldots, j'\}$."

To explain the connection between the original problem of computing $F_{X_c}(\mathbf{x})$ and this combinatorial setup, note that the event $X_{(c_j)} \leq x_j$ means "the c_jth smallest value of the n random variables is less than or equal to x_j" The equivalent event in the combinatorial setup is "there are at least c_j balls in superbin $I_{1:j}$: i.e., the event D_j holds. We call D_j the jth bin condition.

Having spelled out this relationship, the problem of interest can now be expressed as follows: $F_{X_c}(\mathbf{x}) =$

$$\mathbb{P}\left(\bigcap_{j=1}^{d} \left\{X_{(c_j)} \leq x_j\right\}\right) = \mathbb{P}\left(\bigcap_{j=1}^{d} \left\{C_{1:j} \geq c_j\right\}\right)$$

$= \mathbb{P}(D_{1:d})$. In other words, we are interested in computing the probability of satisfying all d bin conditions.

Definition 1. *Given an index set $\mathcal{C}$ of size d; n independent random variables representing m homogeneous populations of sizes $\mathbf{n} \doteq (n_1, \ldots, n_m)$ with $\sum_{l=1}^{m} n_l = n$; and m $(d+1)$-vectors of probabilities $\mathbf{p}^l = (p_1^l, \ldots, p_{d+1}^l)$, one per $l \in [m]$; we define the combinatorial problem* MJCDF$(\mathcal{C}, \mathbf{n}, (\mathbf{p}^l)_{l \in [m]})$ *as computing the probability $\mathbb{P}(D_{1:d})$.*

This general problem definition captures two special cases of interest: the case of one population (i.i.d. random variables), which we dub IIDJCDF$(\mathcal{C}, n, \mathbf{p})$, and the case where $m = n$ populations, which we dub IJCDF$(\mathcal{C}, n, P)$—I for independent—where $P \in \mathbb{R}^{n \times d+1}$. In the former case, it suffices to define a single vector of probabilities $p_j = \mathbb{P}(\{X_i \in I_j\})$, representing the probability that a ball resides in the jth bin, rather than m such vectors. In the latter case, we write $P_{ij} = \mathbb{P}(\{X_i \in I_j\})$ rather than p_j^l, since each population type l is associated with a unique individual i.

3 Related Work

We now understand the problems at hand as ones of computing the probability of satisfying d bin conditions: i.e., $\mathbb{P}(D_{1:d})$. One straightforward avenue to solving these problems is to compute the probability of realizing all possible combinations of balls in bins that satisfy the requisite bin conditions. Unsurprisingly, this approach is computationally infeasible, as the number of combinations grows exponentially with both n and d. This combinatorial explosion is apparent in Bapat-Beg's theorem [10], as we will see shortly, and again in Section 5 (the experiments section).

Theorem 3.1. (Bapat and Beg [10]) *Given an instance of* IJCDF$(\mathcal{C}, n, P)$*, the joint distribution $F_{X_c}(\mathbf{x})$ of the $\mathcal{C}$ order statistics evaluated at the point $\mathbf{x} \in \mathbb{R}^d$ is:*

$$F_{X_c}(\mathbf{x}) = \sum_{\mathbf{i} \in \mathcal{I}} \frac{Per_{i_1,\ldots,i_d}(X_1, \ldots, X_n)}{\prod_{j=1}^{d+1}(i_j - i_{j-1})!} \quad . \tag{3.2}$$

Here, as in Equation (1.1)*, $\mathcal{I}$ is the set of all non-decreasing $(d+1)$-vectors such that the jth entry is at least c_j. Further, $Per_{i_1,\ldots,i_d}(X_1, \ldots, X_n)$ is the permanent of the block matrix $\mathcal{P}$, where $\mathcal{P}_{ij} = P_{it}$, for all $i, j \in [n]$ and $t = \min_s\{i \mid i_s \geq j\}$.*

Note that each instance of $\mathbf{i} = (i_1, \ldots, i_d) \in \mathcal{I}$ corresponds to some $(C_{1:1}, \ldots, C_{1:d}) \in [n^*]^d$ such that $\{C_{1:j} \geq c_j\}$, for all $j \in [d]$. In other words, $\mathcal{I}$ is the set of all possible aggregated ball-count configurations that satisfy the bin conditions! This observation suggests that the summand corresponding to $\mathbf{i}$ is the probability of that particular ball-count configuration. A proof of this observation is provided by Hande [13]. Using Equation 3.2, we thus obtain the following relationship: $F_{X_\mathcal{C}}(\mathbf{x}) =$

$$= \sum_{\mathbf{i} \in \mathcal{I}} \mathbb{P}\left(\bigcap_{j=1}^{d} \{C_{1:j} = i_j\}\right) = \mathbb{P}\left(\bigcap_{j=1}^{d} \{C_{1:j} \geq c_j\}\right) \tag{3.3}$$

$\mathbb{P}(D_{1:d})$.

While the Bapat-Beg theorem gives us a closed-form solution to IJCDF, the computation of matrix permanents is #P-hard [4]; the fastest known procedure for a square matrix of size n has time complexity $O(n2^{n-1})$ [3]. As Equation (3.2) sums over the set $\mathcal{I}$, which is of size $O(n^d)$, with each summand requiring the computation of a block permanent of a matrix of size n, this approach yields a total time complexity of $O(n^{d+1}2^{n-1})$, and is thus exponential in both n and d.

In the i.i.d. setting, however, the block permanent has a simpler form. That is, Equation (3.2) reduces to Equation (1.1), which has time complexity $O(dn^d)$, so is exponential only in d, not in n. This observation suggests that the inclusion of more than one of the same type of random variable allows for an improvement in the time complexity of evaluating these block permanents. Glueck, et al. [12] observed that in the multiple populations setting, the computation of the block permanents simplifies as follows:

THEOREM 3.2. (GLUECK, ET AL. [12]) *Given an instance of* MJCDF$(\mathcal{C}, \mathbf{n}, (\mathbf{p}^l)_{l\in[m]})$*, the joint distribution* $F_{X_\mathcal{C}}(\mathbf{x})$ *of the* $\mathcal{C}$ *order statistics evaluated at a point* $\mathbf{x} \in \mathbb{R}^d$ *is given by:*

$$F_{X_\mathcal{C}}(\mathbf{x}) = \sum_{\mathbf{i}\in\mathcal{I}} \sum_{[\lambda_{jl}]\in\Lambda(\mathbf{i})} \prod_{j=1}^{d+1} \prod_{l=1}^{m} \frac{n_s!}{\lambda_{jl}!} \left(p_j^l\right)^{\lambda_{jl}} \quad . \tag{3.4}$$

Here, $\Lambda(\mathbf{i}) \in \mathbb{N}^{d+1\times m}$ *is the set of* $d+1 \times m$ *matrices of natural numbers such that:*

1. $\lambda_{jl} \geq 0$*, for all* j, l*,*
2. $\sum_{j=1}^{d+1} \lambda_{jl} = n_j$*, for all* l*,*
3. $\sum_{l=1}^{m} \lambda_{jl} = i_j - i_{j-1}$*, for all* j*.*

The $\Lambda(\mathbf{i})$ matrix, while initially cryptic, has the interpretation that $\lambda_{j,l}$ is the number of random variables drawn from F_l (i.e., balls of type l) that land in the jth bin. Thus, Glueck et al. track the number of balls *of each type* in each bin. The following relationship between Glueck et al.'s method and Bapat-Beg's, and thus between IJCDF and IIDJCDF, holds by Equation (3.3):

$$\begin{aligned}
F_{X_\mathcal{C}}(\mathbf{x}) &= \sum_{\mathbf{i}\in\mathcal{I}} \sum_{[\lambda_{jl}]\in\Lambda(\mathbf{i})} \prod_{j=1}^{d+1} \prod_{l=1}^{m} \frac{n_l!}{\lambda_{jl}!} \left(p_j^l\right)^{\lambda_{jl}} \\
&= \sum_{\mathbf{i}\in\mathcal{I}} \sum_{[\lambda_{jl}]\in\Lambda(\mathbf{i})} \mathbb{P}\left(\bigcap_{j=1}^{d+1} \bigcap_{l=1}^{m} \{C_j^l = \lambda_{jl}\}\right) \\
&= \sum_{\mathbf{i}\in\mathcal{I}} \mathbb{P}\left(\bigcap_{j=1}^{d+1} \{C_j = i_j - i_{j-1}\}\right) \\
&= \sum_{\mathbf{i}\in\mathcal{I}} \mathbb{P}\left(\bigcap_{j=1}^{d+1} \{C_{1:j} = i_j\}\right) \\
&= \sum_{\mathbf{i}\in\mathcal{I}} \mathbb{P}\left(\bigcap_{j=1}^{d} \{C_{1:j} = i_j\}\right) \quad .
\end{aligned}$$

The third equality follows as $\sum_{l=1}^{m} \lambda_{jl} = i_j - i_{j-1}$. Additionally, the last line in this derivation follows as the event $\{C_{1:d+1} = i_{d+1} = n\}$ is always true.

The Glueck et al. compression scheme reduces the time complexity of applying the Bapat-Beg theorem from $O(n^d)$, the size of $\mathcal{I}$ in the i.i.d. case, to $O\left(\prod_{l=1}^{m} n_l^d\right)$, the size of $\mathcal{I}$ assuming multiple populations. While polynomial in n, this quantity is exponential in d and m. Setting $m = 1$ recovers Equation (1.1) and thus solves IIDJCDF, while $m = n$ recovers the Bapat-Beg theorem and thus solves IJCDF.

Boncelet Jr. [11] proposed a dynamic programming solution to IJCDF$(\mathcal{C}, n, \mathbf{p})$, which improves upon the complexity of Bapat-Beg's analytic form by incrementally tallying the probabilities of all ball-count configurations as additional balls are thrown. Recall that $\mathbf{C}(i) \in [n^*]^d$ is a random vector that denotes the total number of balls (of any type) among i total balls that land all $d+1$ bins. In particular, $\mathbf{C}(i-1)$ denotes this random vector before throwing the ith ball, and $\mathbf{C}(i)$ denotes this random vector after.

THEOREM 3.3. (BONCELET JR. [11]) *Given an instance of* IJCDF$(\mathcal{C}, n, P)$ *and constant* $\mathbf{k} \in [n^*]^d$*, we have the following recurrence relation:* $\mathbb{P}(\mathbf{C}(0) = \mathbf{0}) = 1$ *and for all* $i \in [n]$*,*

$$\begin{aligned}
\mathbb{P}(\mathbf{C}(i) = \mathbf{k}) &= (1 - P_{i,d+1})\, \mathbb{P}(\mathbf{C}(i-1) = \mathbf{k}) \\
&+ \sum_{\{j|C_j>0\}} (P_{i,j})\, \mathbb{P}(\mathbf{C}(i-1) = \mathbf{k} - \mathbf{1}_j) \quad .
\end{aligned} \tag{3.5}$$

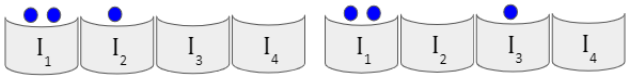


Figure 1: Consider $n = 4$ and assume $C_1 = 2$, so that D_3 is the next unsatisfied bin condition. A single ball residing in superbin $I_{2:3}$ satisfies D_3, whether it resides in I_2 or I_3.

The solution to IJCDF$(\mathcal{C}, n, P)$ *is then* $\sum_{\mathbf{k}\in\mathcal{I}} \mathbb{P}(\mathbf{C}(n) = \mathbf{k})$*, where* $\mathcal{I}$ *is again the set of* valid *ball-count configurations: i.e., those satisfying* $D_{1:d}$*.*

This recurrence relation corresponds to a dynamic program that runs n iterations, one per ball. During each iteration, $O(n^d)$ table entries are updated by summing over $O(d)$ elements. Boncelet Jr.'s algorithm thus has time and space complexity of $O(dn^{d+1})$ and $O(n^d)$, respectively.

Boncelet Jr.'s dynamic program, while less efficient than ours in the case of independent random variables, handles dependent random variables, simply by replacing the marginal probability distributions in Equation (3.5) with conditional probability distributions. In particular, we can "remember" the locations of previous balls, and then condition the ball-count configuration vector on the current random variable's *boundary* set: i.e., the minimal set S_i of balls such that X_i is conditionally independent of $\{X_1, \ldots, X_{i-1}\}\backslash S_i$ given S_i. (For example, in a Markov chain, the boundary set S_i of X_{i+1} is X_i, since X_{i+1} is conditionally independent of $\{X_1, \ldots, X_{i-1}\}$ given X_i.) We direct the reader to Boncelet Jr.'s original paper [11] for further details.

4 Our Solution

All three of the aforementioned approaches, the Bapat-Beg theorem, Glueck et al.'s method, and Boncelet Jr.'s algorithm, either implicitly or explicitly enumerate all possible ball-count configurations, and then sum the probabilities of those that satisfy $D_{1:d}$. By expressing this problem in terms of bin conditions rather than ball-count configurations, we arrive at a lower complexity solution, which does not require this expensive summation. The key insight that enables this complexity savings is the following: *The precise configuration of the balls that reside in superbin* $I_{1:j}$ *is irrelevant from the point of view of satisfying bin conditions* $j' \geq j$. For example, in Figure 1, from the point of view of satisfying bin condition 4, it does not matter whether the balls are configured as shown on the left or on the right.

Consider an instance of IIDJCDF$(\mathcal{C}, n, \mathbf{p})$. We define $\phi : [n^*] \to [d^*]$ s.t. $\phi(k)$ is the largest index j for which $c_j \leq k$: i.e., $\phi(k) = \max_j\{j \mid c_j \leq k\}$, with $\phi(0) \doteq 0$. One way to interpret $\phi(k)$ is as follows: if $C_{1:j'} = k$ for some $j' \leq j$, then $\phi(k)$ is the largest index j for which the jth bin condition is satisfied. For example, $\phi(k)$ is simply k when $d = n$, as $c_j = j$.

Our algorithm proceeds by computing the probabilities of satisfying the bin conditions, one by one. We begin by considering all the ways in which the first bin condition can be satisfied (i.e., with any of 1 through n balls residing in the first bin), together with the corresponding probabilities. As above, assume $k \in [n^*]$ balls reside in superbin $I_{1:j}$ so that $\{C_{1:j} = k\}$. The next unsatisfied bin condition is then $D_{\phi(k)+1}$. For this bin condition to hold, it is necessary that $\{C_{j+1:\phi(k)+1} \geq c_{\phi(k)+1} - k\}$. The probability of this event depends on $\sum_{i=j+1}^{\phi(k)+1} p_i$. Moreover, according to our key insight, from the point of view of satisfying future bin conditions, the $(\phi(k)+1)$st and beyond, it does not matter exactly how the balls in bins $j+1$ through $\phi(k)+1$ are configured. Consequently, we can treat all these bins as a single superbin, to arrive at a smaller version of IIDJCDF with $n-k$ balls and $d-\phi(k)$ bins, in which we eliminate bins 1 through j, collapse bins $j+1$ through $\phi(k)+1$ into a new first bin, and re-index the remaining bins $\phi(k)+2, \ldots, d$ as 2 through $d-\phi(k)$. The bin probabilities in this smaller problem require conditioning on $\{C_{1:j} = k\}$:

$$\begin{aligned}\tilde{\mathbf{p}} &= \left(\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_{d-\phi(k)}, \tilde{p}_{d-\phi(k)+1}\right) \\ &= \frac{1}{1-\sum_{i=1}^{j} p_i}\left(\sum_{i=j+1}^{\phi(k)+1} p_i, p_{\phi(k)+2}, \ldots, p_d, p_{d+1}\right) .\end{aligned}$$

Therefore, embedded within the IIDJCDF problem with n balls and d bins is a smaller subproblem with the exact same structure, but only $n-k$ balls and $d-\phi(k)$ bins: i.e., IIDJCDF$(\{c_{\phi(k)+1} - k, \ldots, c_d - k\}, n-k, \tilde{\mathbf{p}})$.

The main contribution of this paper is a recurrence relation for IIDJCDF, and more generally MJCDF, which reveals the aforedescribed substructure, and yields a corresponding dynamic programming solution that is polynomial in both n and d. Our algorithm iterates over bins, recursively computing the probability of $D_{\phi(k)+1:d}$, conditioned on k balls residing in superbin $I_{1:j}$. We prove that balls that reside in superbin $I_{j+1:\phi(k)+1}$ can be applied towards satisfying the $(\phi(k)+1)$st bin condition regardless of their precise configuration, which relieves us of the computational burden of tracking exact ball-count configurations.

4.1 Key Insight In this section, we prove our key insight and derive its main implication, based on which we arrive at a novel recurrence relation that solves IIDJCDF$(\mathcal{C}, n, \mathbf{p})$. We assume i.i.d. random variables throughout, for notational simplicity only. And we note, at the cost of maintaining vectors of counts of balls of each type, all of our techniques carry over to the

multiple populations case (see Section 4.5). We now state our key insight:

LEMMA 4.1. *For all $j' > j \in [d]$, and given $k_i \in [n^*]$ for all $i \in [j]$, it holds that:*

$$\mathbb{P}\left(D_{j':d} \mid \cap_{i=1}^{j}\{C_i = k_i\}\right) = \mathbb{P}(D_{j':d}|\{C_{1:j} = \sum_{i=1}^{j} k_i\}) \ .$$

This key insight enables our next theorem: given $\{C_{1:j} = k\}$, the probability of $D_{\phi(k)+1:d}$ does not depend on exactly how $i-k$ additional balls might be distributed in the $\phi(k)+1$st superbin, which is how the events $\{C_{j+1:\phi(k)+1} = i-k\} \cap \{C_{1:j} = k\}$ and $\{C_{1:\phi(k)+1} = i\}$ differ.

THEOREM 4.1. *For all $k < i \in [n]$ and $j \in [d]$ s.t. $c_j \leq k$ (i.e., the jth bin condition is satisfied), the following holds, assuming i.i.d. random variables:*

$$\begin{aligned}&\mathbb{P}\left(D_{\phi(k)+1:d} \mid \{C_{j+1:\phi(k)+1} = i-k\} \cap \{C_{1:j} = k\}\right)\\ &= \mathbb{P}\left(D_{\phi(i)+1:d} \mid \{C_{1:\phi(k)+1} = i\}\right) \ .\end{aligned}$$

This result, which provides the basis for our recurrence relation (Equation 4.6), follows as a direct result of Lemma 4.1, as the probability of $D_{\phi(k)+1:d}$ conditioned on the number of the balls in the first j bins and the number in bins $j+1$ through $\phi(k)+1$ is unchanged when conditioned on the total number in the first $\phi(k)+1$ bins, assuming the total number of balls in both conditions is the same.

4.2 Recurrence Relation We are now equipped to express $\mathbb{P}(D_{1:d})$ as a recurrence relation. In the base case, $k \geq c_d$ balls reside in the jth superbin, so all d requisite bin conditions are satisfied: i.e., for all $j \in \{1,\ldots,d\}$ and $k \in \{c_d,\ldots,n\}$ s.t. $c_j \leq k$, $\mathbb{P}(D_{1:d} \mid \{C_{1:j} = k\}) = 1$. Theorem 4.2 encapsulates the inductive step:

THEOREM 4.2. *For all $j \in [d]$ and $k \in [n]$ s.t. $c_j \leq k$ or $(j,k) = (0,0)$:*

$$\begin{aligned}(4.6)\quad &\mathbb{P}\left(D_{\phi(k)+1:d} \mid \{C_{1:j} = k\}\right)\\ &= \sum_{i=c_{\phi(k)+1}}^{n} \mathbb{P}\left(D_{\phi(i)+1:d} \mid \{C_{1:\phi(k)+1} = i\}\right)\\ &\qquad \mathbb{P}\left(\{C_{j+1:\phi(k)+1} = i-k\} \mid \{C_{1:j} = k\}\right) \ .\end{aligned}$$

A naive approach to solving this recurrence relation would take exponential time, because recursively expanding the summation in Equation 4.2 would yield exponentially many terms. Fortunately, there are only $O(dn)$ different subproblems as j and k only range from 1 to d and 1 to n, respectively. Using this observation, we now present a dynamic program that solves IIDJCDF in $O(dn)$ space and $O(dn^2)$ time.

4.3 Dynamic Program We now present a procedure (Algorithm 4.1) by which to construct a two-dimensional dynamic programming table $T(j,k)$, for $j \in [d]$ and $k \in \{c_j,\ldots,n\}$, and we prove that $\mathbb{P}\left(D_{\phi(k)+1:d} \mid \{C_{1:j} = k\}\right) = T(j,k)$, and moreover, that $\mathbb{P}(D_{1:d}) = T(0,0)$.

Note that $T(j,k)$ for all $c_j > k$ represent probability 0 events, as $\{C_{1:j} = k\}$ never holds when $k < c_j$, because $k < c_j$ balls cannot satisfy the jth bin condition. These table entries are hence irrelevant to the dynamic programming computation. On the other hand, the event $\{C_{j:0} = k\}$ is always true for all $k \in [n^*]$ and $j \in [d]$, as the superbins $C_{j:0}$ do not exist. Of particular interest, $\mathbb{P}(\{C_{1:0} = 0\}) = 1$.

ALGORITHM 4.1.
Joint CDF of order statistics for i.i.d. random variables
Require: $n, \mathcal{C}$ of size $d, \mathbf{p} = (p_1,\ldots,p_d)$
Ensure: $T(0,0) = \mathbb{P}(D_{1:d})$
for $k \leftarrow c_d$ to n **do**
  for $j \leftarrow 1$ to $d+1$ **do**
    $T(j,k) \leftarrow 1$
  end for
end for
for $k \leftarrow c_d - 1$ to c_1 **do**
  for $j \leftarrow \phi(k)$ to 1 **do**
    $\tilde{p}_{jk} = \sum_{h=j+1}^{\phi(k)+1} p_h / \left(1 - \sum_{h=1}^{j} p_h\right)$
    $T(j,k) = \sum_{i=c_{\phi(k)+1}}^{n} T(\phi(k)+1, i)$
      $\text{Binomial}_{n-k,\tilde{p}_{jk}}(i-k)$
  end for
end for
return $T(0,0) = \sum_{i=c_1}^{n} T(1,i)\,\text{Binomial}_{n,p_1}(i)$

4.4 Correctness and Efficiency We now establish the correctness and efficiency of Algorithm 4.1.

LEMMA 4.2. *If $\tilde{p}_{jk} = \sum_{h=j+1}^{\phi(k)+1} p_h / \left(1 - \sum_{h=1}^{j} p_h\right)$, then for all $k < i \in [n]$ and $j \in [d]$ s.t. $c_j \leq k$ and $(j,k) = (0,0)$, $\mathbb{P}\left(\{C_{j+1:\phi(k)+1} = i-k\} \mid \{C_{1:j} = k\}\right) = \text{Binomial}_{n-k,\tilde{p}_{jk}}(i-k)$.*

THEOREM 4.3. *$\mathbb{P}\left(D_{\phi(k)+1:d} \mid \{C_{1:j} = k\}\right) = T(j,k)$, for all $k \in [n]$ and $j \in [d]$ s.t. $c_j \leq k$ and $(j,k) = (0,0)$.*

COROLLARY 4.1. *$\mathbb{P}(D_{1:d}) = T(0,0)$.*

Proof. By Theorem 4.3, $\mathbb{P}\left(D_{\phi(0)+1:d} \mid \{C_{1:0} = 0\}\right) = T(0,0)$. The result follows immediately, since $\phi(0) = 0$ and $\mathbb{P}\left(\{C_{1:0} = 0\}\right) = 1$. □

We have thus established Algorithm 4.1's correctness.

THEOREM 4.4. *Algorithm 4.1 has space complexity $O(dn)$ and time complexity $O(dn^2)$.*

Proof. There are $O(dn)$ table entries, each of which requires computing a summation over $O(n)$ entries. Hence, the time complexity of Algorithm (4.1) is $O(dn^2)$ and the space complexity is $O(dn)$. □

It is important to note that there is some subtlety in this complexity analysis, as the choice of order statistics (i.e., the values in the order statistic set $\mathcal{C}$) also impact the time and space complexity. Recall that the computation of $T(j,k)$ requires a summation over table entries $T(\phi(k)+1, i)$, for $i \geq c_{\phi(k)+1}$. In addition, table entries $T(j,k)$ where $k < c_1$ are irrelevant. Consequently, when c_1 is large, we can ignore many table entries, thereby improving efficiency. Similarly, when c_d is small, the base case covers many of the table entries. While we could perhaps factor this complexity dependence of $\mathcal{C}$'s values into our theoretical analysis, we instead discuss this nuance in our experiments, which demonstrate how run time varies with respect to $\mathcal{C}$.

4.5 Multiple Populations Throughout our exposition, we have assumed i.i.d. random variables. Following [10] and [12], we can relax this assumption, and instead assume that there are m homogeneous populations of sizes $n_1, \ldots, n_m$, as in $\text{MJCDF}(\mathcal{C}, \mathbf{n}, (\mathbf{p}^l)_{l\in[m]})$. Algorithm 4.1 generalizes almost immediately to this problem. Rather than simply tracking how many balls fall into bins, say k balls in the first j bins, we keep track of how many balls *of each type* fall into the first j bins. That is, instead of conditioning on $\{C_{1:j} = k\}$, we condition on $\{\mathbf{C}_{1:j} = \mathbf{k}\}$: i.e., there are k_l balls of type l in the first j bins, for all $l \in [m]$. The recurrence relation now requires summing over all such $\mathbf{k}$ such that the next unfulfilled bin condition is satisfied. More precisely, we generalize all Lemma 4.1 (our key insight), Theorem 4.1, Theorem 4.2 (our recurrence relation), and Algorithm 4.1 below; we omit the proofs, however, as they add nothing new.

The more general algorithm requires that we likewise generalize the ϕ operation, so we define $\phi(\mathbf{k})$ to be the largest index $j' \in [d]$ s.t. $\sum_{l=1}^{m} k_l \geq c_{j'}$: i.e., the largest index j' s.t. the j'th bin condition is satisfied, with $\phi(\mathbf{0}) = 0$. We also define $\|\mathbf{k}\|$ to be the sum of all the entries in $\mathbf{k}$, and overload $<$ (and $\leq$) between vectors to mean entry-wise less than (or equal to).

LEMMA 4.3. *For all $j' > j \in [d]$, and given $k_i \in [n^*]$ for all $i \in [j]$, it holds that:*

$$\mathbb{P}\left(D_{j':d} \mid \bigcap_{i\in[j], l\in[m]} \{C_{i,l} = k_{i,l}\}\right) = \mathbb{P}\left(D_{j':d} \mid \bigcap_{l\in[m]} \left\{C^l_{1:j} = \sum_{i=1}^{j} k_{i,l}\right\}\right) .$$

This generalization of our key insight implies a more general version of the ensuing theorem:

THEOREM 4.5. *For all $\mathbf{k} < \mathbf{i} \leq \mathbf{n}$ and $j \in [d]$ s.t. $c_j \leq \|\mathbf{k}\|$ (i.e., the jth bin condition is satisfied), the following holds, assuming independent random variables:*

$$\mathbb{P}\left(D_{\phi(\mathbf{k})+1:d} \mid \{\mathbf{C}_{j+1:\phi(\mathbf{k})+1} = \mathbf{i} - \mathbf{k}\} \cap \{\mathbf{C}_{1:j} = \mathbf{k}\}\right) = \mathbb{P}\left(D_{\phi(\mathbf{i})+1:d} \mid \{\mathbf{C}_{1:\phi(\mathbf{k})+1} = \mathbf{i}\}\right) .$$

In turn, these results imply a more general recurrence relation. In the base case, where $\phi(\mathbf{k}) = c_d$, we have $\mathbb{P}\left(D_{1:d} \mid \{\mathbf{C}_{1:j} = \mathbf{k}\}\right) = 1$. The inductive step is then given by the following theorem:

THEOREM 4.6. *For all $j \in [d]$ and $\mathbf{k} \in [n]$ s.t. $c_j \leq \|\mathbf{k}\|$ and $(j, \mathbf{k}) = (0, \mathbf{0})$,*

$$\mathbb{P}(D_{\phi(\mathbf{k})+1:d} \mid \{\mathbf{C}_{1:j} = \mathbf{k}\}) = \sum_{\{\mathbf{i}:\mathbf{k}\leq\mathbf{i}\leq\mathbf{n}, \|\mathbf{i}\|\geq c_{\phi(\mathbf{k})+1}\}} \Big(\mathbb{P}(D_{\phi(\mathbf{k})+1:d} \mid \{\mathbf{C}_{1:\phi(\mathbf{k})+1} = \mathbf{i}\}) \prod_{l=1}^{m} \text{Binomial}_{n_l - k_l, \tilde{p}_{j,\mathbf{k},l}}(i_l - k_l)\Big) .$$

The time and space complexities of our dynamic programming solution to $\text{MJCDF}(\mathcal{C}, \mathbf{n}, (\mathbf{p}^l)_{l\in[m]})$ are $O\left(d\prod_{l=1}^{m} n_l\right)$ and $O\left(\prod_{l=1}^{m} n_l\right)$, respectively. As compared to Glueck et al.'s algorithm, the time dependence of our's on d is polynomial, rather than an exponential.

5 Performance Comparison

In this section, we describe experiments designed to compare our procedure (Algorithm 4.1) to that of Bapat-Beg, Glueck et al., and Boncelet Jr. We conducted three series of tests. The first compares algorithms for a single population, where we vary $\mathbf{n}$ and $\mathcal{C}$. These tests show that our algorithm is indeed faster than previous algorithms when m is small, as our theoretical analysis suggests. The second and third tests are designed to verify our algorithm's time complexity in terms of n, d, and to explore how it depends on the choice of order statistics $\mathcal{C}$—something that is not captured by our theoretical analysis.

All experiments were run on a machine with an Intel Xeon E3-1240 v5 CPU and 15.6 GB memory.

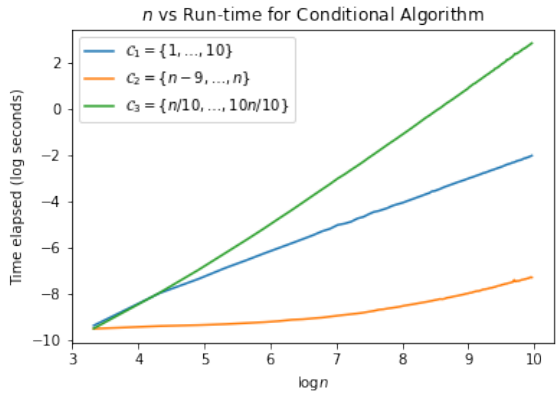


Figure 2: Log-log plot for fixed d, varying n. The plot suggests polynomial behavior in n. The effect of $\mathcal{C}$ can be seen in the different slopes, with $\mathcal{C}_1$ (the blue curve) and $\mathcal{C}_2$ (orange) running in time approximately linear and constant in n, respectively, versus $\mathcal{C}_3$ (green), which takes roughly quadratic time.

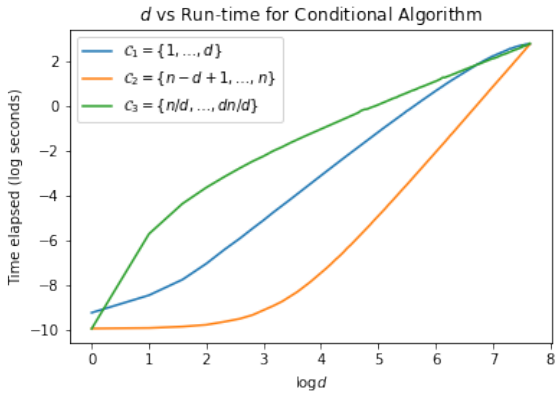


Figure 3: Log-log plot for fixed n, varying d. The plot suggests polynomial behavior in d, though the effects of $\mathcal{C}$ are more prevalent. In particular, the run time for $\mathcal{C}_3$ increases rapidly for small d and then tapers off (the green curve), whereas the opposite behavior is seen for $\mathcal{C}_2$ (orange) d order statistics.

5.1 Test Cases All random variables in all our experiments are Unif$(0,1)$: i.e., for $i \in [n]$, we assume $X_i \overset{\text{iid}}{\sim} \text{Unif}(0,1)$. Further, we choose as our bounds $x_i = i/(d+1)$: i.e., the expected value of the ith order statistic of $X_{1:n}$, though we note that the choice of bounds does not affect run time, so long as they are non-decreasing.

In our first set of tests, we choose $n \in \{6, 12, 18, 24, 30\}$ and $\mathcal{C} = [d]$ for $d \in \{1, \ldots, 5\}$. We compare our algorithm to those of Bapat-Beg, Glueck et al., and Boncelet Jr. We then repeat this experiment with only our algorithm and Glueck et al.'s method, with the same values of n evenly split across $m = 2$ populations. We exclude Boncelet Jr.'s and Bapat-Beg's algorithms from these latter experiments as their time and space complexities are independent of m, so the results in the two setups would not differ.

For our second set of tests, we hold d fixed and vary n, and for our third set, we do the reverse. In the second set, we choose $n \in \{10, 20, \ldots, 1000\}$ and fix $d = 10$. In the third set, we fix the value of n at 200 and choose $d \in \{1, \ldots, 200\}$. Additionally, we consider three choices of $\mathcal{C}$, involving the first, the last, and d evenly spread out order statistics.

5.2 Test Results Tables 1 through 4 depict the elapsed run times in seconds for the first battery of tests. Our algorithm is by far the fastest of the four, and Bapat-Beg's, the slowest. It also seems that when $m = 1$, Glueck et al.'s algorithm is marginally faster than that of Boncelet Jr.'s.

Tables 5 and 6 depict elapsed run times in seconds for the second battery of tests. In this two population experiment, our algorithm is again faster than that of Glueck et al., particularly as the problem size grows.

Figures 2 and 3 depict elapsed run times for the second and third tests, where we hold one of n and d fixed while varying the other. In these log-log plots, we can see that our algorithm exhibits worst-case polynomial behavior in both n and d, as expected, regardless of the choice of $\mathcal{C}$. We can also see that the choice of $\mathcal{C}$ can impact the run time, with consecutive order statistics (at either end of the spectrum) being faster to compute for than order statistics that are evenly spread throughout.

Additionally, we ran an ordinary least squares regression, regressing the log times on $\log n$ for the second test, and on $\log d$ for the third. Regressing on $\log n$, we find slopes of roughly 1.05, 0.44, and 1.95, respectively, when using $\mathcal{C}_1 = \{1, \ldots, 10\}$, $\mathcal{C}_2 = \{n-9, \ldots, n\}$, and $\mathcal{C}_3 = \{\frac{n}{10}, \ldots, \frac{10n}{10}\}$, respectively. Regressing on $\log d$ yields slopes of 1.71, 2.41, and 1.17, respectively. This means that the dependence on d is polynomial, with the slope as the corresponding exponent, which seems to suggest super-linear—roughly quadratic—behavior in d. This quadratic dependence on d can be explained by the choice of $\mathcal{C}$, as the size of the dynamic programming table using $\mathcal{C}_3$ is $O(dn)$, while it is only $O(d^2)$ using $\mathcal{C}_1$ and $\mathcal{C}_2$. Combining the dependence on n and d, the time complexity using $\mathcal{C}_3$ is approximately $O(dn^2)$, while the others are approximately $O(d^2 n)$. Since $n \geq d$, the time complexity using $\mathcal{C}_3$ is still $O(dn^2)$.

Experiment 1

n	6	12	18	24	30
$d=1$	8.3E-5	1.3E-4	1.7E-4	2.4E-4	2.9E-4
$d=2$	1.2E-4	2.0E-4	2.7E-4	3.5E-4	4.2E-4
$d=3$	1.9E-4	3.2E-4	4.6E-4	5.9E-4	7.2E-4
$d=4$	2.6E-4	4.8E-4	7.1E-4	9.4E-4	1.2E-3
$d=5$	3.5E-4	6.9E-4	1.0E-3	1.4E-3	1.7E-3

Table 1: Our Algorithm's Elapsed Time. Our algorithm is significantly faster than existing methods, whose run times appear in the neighboring tables.

n	6	12	18	24	30
$d=1$	7.5E-5	2.2E-4	2.1E-4	2.8E-4	3.6E-4
$d=2$	6.7E-4	9.4E-4	2.1E-3	3.8E-3	5.7E-3
$d=3$	6.6E-4	4.7E-3	1.5E-2	3.4E-2	6.4E-2
$d=4$	1.3E-3	1.8E-2	8.3E-2	2.5E-1	6.0E-1
$d=5$	2.2E-3	6.1E-2	4.0E-1	1.5E0	4.4E0

Table 2: Glueck et al.'s Algorithm Elapsed Time. It is slightly faster than Boncelet Jr.'s algorithm (Table 3).

n	6	12	18	24	30
$d=1$	1.2E-4	3.8E-4	5.7E-4	1.0E-3	1.6E-3
$d=2$	4.3E-4	2.7E-3	7.9E-3	1.8E-2	3.3E-2
$d=3$	1.5E-3	1.5E-2	6.7E-2	1.9E-1	4.4E-1
$d=4$	4.3E-3	7.1E-2	4.3E-1	1.6E0	4.5E0
$d=5$	1.1E-2	2.8E-1	2.3E0	1.1E1	3.7E1

Table 3: Boncelet Jr.'s Algorithm Elapsed Time. While slower than our algorithm (Table 1) and Glueck et al.'s (Table 2) when $m=1$, Boncelet Jr.'s algorithm's run times do not increase with m.

n	6	12	18	24	30
$d=1$	1.3E-3	3.5E-1			
$d=2$	3.8E-3	2.2E0			
$d=3$	9.3E-3	1.0E1			
$d=4$	1.7E-2	3.7E1			
$d=5$	2.5E-2	1.1E2			

Table 4: Bapat-Beg Theorem Elapsed Time. Values for the larger population sizes ($n \in \{18, 24, 30\}$) are omitted because the run times blew up.

Experiment 2

$\mathbf{n}$	$[3,3]$	$[6,6]$	$[9,9]$	$[12,12]$	$[15,15]$
$d=1$	1.8E-4	3.9E-4	7.0E-4	1.1E-3	1.8E-3
$d=2$	3.0E-4	8.2E-4	1.7E-3	2.8E-3	4.2E-3
$d=3$	6.0E-4	1.9E-3	4.2E-3	7.3E-3	1.1E-2
$d=4$	1.0E-3	3.8E-3	8.4E-3	1.5E-2	2.3E-2
$d=5$	1.3E-3	6.2E-3	1.5E-2	2.7E-2	4.3E-2

Table 5: Our Algorithm's Elapsed Time. As compared to Glueck et al.'s, the improvement in run time at larger values of d is significant. We remark there is a noticeable increase in run time in the $m=2$ case as compared to the $m=1$ case (Table 1).

$\mathbf{n}$	$[3,3]$	$[6,6]$	$[9,9]$	$[12,12]$	$[15,15]$
$d=1$	1.6E-4	4.7E-4	9.0E-4	1.7E-3	
$d=2$	1.0E-3	8.6E-3	3.2E-2	8.8E-2	
$d=3$	4.0E-3	8.6E-2	6.1E-1	2.7E0	
$d=4$	1.1E-2	6.0E-1	7.5E0	5.0E1	
$d=5$	2.1E-2	3.0E0	6.5E1	6.5E2	

Table 6: Glueck et al.'s Algorithm Elapsed Time. Values for the larger population sizes ($\mathbf{n}=[15,15]$) are omitted because the run times blew up. While faster than Boncelet Jr.'s algorithm (Table 3) when $m=1$, when $m=2$ it is noticeably slower.

6 Applications

As alluded to in the introduction, there are many applications of the joint distributions of order statistics.

One straightforward application of the algorithm developed in this paper is to auction design. It is well known in auction theory [15] that *reserve prices*—prices above which auction participants must bid—can boost revenue. In repeated, online auctions (for example, Google Adwords), where the auctioneer can model the participants' bid distributions, the auctioneer might be interested in the probability of making a sale at a given reserve price. This probability depends on the distribution of the highest-order statistic. More generally, in position (or slot) auctions [16], such as those used by search engines to sell ads alongside organic search results, the auctioneer can boost its revenue by setting multiple reserve prices, one per slot. The probability of selling d advertising slots given multiple reserve prices depends on the joint distribution of the d highest-order statistics. More specifically, assuming n bidders with n corresponding independent bid distributions, $F_1, \ldots, F_n$, and d ordered items, with reserve prices $x_1, \ldots, x_d$, the probability of interest is exactly $\textsc{ijcdf}(\{n+1-d, \ldots, n\}, n, P)$, where $P_{i,j} = F_i(x_j) - F_i(x_{j-1})$.

As an example of an application with multiple

populations of random variables, where our algorithm is particularly effective, consider, for example, the case of an airline selling three types of goods: economy, business, and first-class tickets. Assume there are $a > b > c$ economy, business, and first-class tickets, respectively, and the airline aims to sell them at prices $x_1 \leq x_2 \leq x_3$, respectively. Based on a data analysis involving features such as income, age, gender, etc., all of which can affect a customer's willingness to pay for a flight, the airline has determined that there are n potential customers who can be broken down into m groups such that all n_k customers in group k value the flight at $X_k \overset{\text{iid}}{\sim} F_k$. In service of optimizing its revenue, the airline is interested in the probability of selling all available tickets in all three classes. The airline is thus interested in a joint order statistic distribution, namely the probability that there are at least $c_1 \doteq a + b + c$ customers with values greater than x_1, of which $c_2 \doteq b + c$ have values greater than x_2, of which $c_3 \doteq c$ have values greater than x_3. More specifically, the probability of interest is $\textsc{ijcdf}(\{n+1-c_1, n+1-c_2, n+1-c_3\}, n, P)$, where $P_{i,j} = F_i(x_j) - F_i(x_{j-1})$.

Finally, a third application of joint order statistic distributions can be seen in job shop scheduling. In one of the more basic versions of the problem, there are n jobs of size $X_1, \ldots, X_n \sim F$ and d machines with capacities $x_1, \ldots, x_d$. Machine j can accept at most one job i, and only if its capacity x_j is at least the job's size X_i. One quantity of interest is the probability that all machines are in use: i.e., that no machines are idle. This problem can be rephrased as computing the probability that the largest job $X_{(n)}$ is of size at most x_d, the second largest job $X_{(n-1)}$ is of size at most x_{d-1}, and so forth. With the relation to order statistics made clear, the quantity of interest is then given by $\textsc{iidjcdf}(\{n+1-d, \ldots, n\}, n, (F(x_1), \ldots, F(x_d), 1 - F(x_d)))$.

7 Summary and Future Directions

In this paper, we studied the problem of computing the joint distribution of d order statistics. Following Boncelet Jr., we pose this problem combinatorially, as one of tossing balls into bins. We argued that while Boncelet Jr.'s approach of tallying precise ball-count configurations—which requires both time and space that is exponential in d—is sufficient, it is not necessary. We instead restate the problem as one of satisfying bin conditions. This insight enables us to solve the problem by conditioning on the number of balls in a range of bins whose respective bin conditions have been satisfied, a novel formulation that enables a solution that is polynomial in d, whereas all previously studied algorithms are exponential in this quantity.

Supporting our theoretical analysis, our experiments indicate that our algorithm is particularly effective in cases where d is large and the random variables are either i.i.d. or are independently distributed from only a small number $m \ll n$ of populations. In the cases where m is large our algorithm remains exponential. Similarly, when d is small, existing algorithms are competitive with our own.

There are several natural extensions that could follow from our work. A few of them are listed below:

1. Can we draw connections between Boncelet Jr.'s algorithm (or ours) and computing permanents?
2. Can we lower bound the run time of a solution to problem in the multiple populations case, or might there be an algorithm that is also polynomial in the number of populations?
3. How does the numerical error suffered by our algorithms compare to that of algorithms, such as those studied in von Schroeder and Dickhaus [17]?
4. Can we devise an algorithm to compute the joint probability $\mathbb{P}\left(\bigcap_{c_j \in \mathcal{C}} \{X_{(c_j)} \in B_j\}\right)$ for arbitrary intervals $B_1, \ldots, B_d$ over $\mathbb{R}$? This probability often a quantity of interest when analyzing empirical processes, evaluating the robustness of rank-based metrics such as income percentiles in censuses, or computing value-at-risk in financial risk management.
5. Lastly, can we apply our key insight, which allowed us to disregard specific ball-count configurations, to improve the performance of existing methods such as Boncelet Jr.'s algorithm, which is efficient in m and can handle dependent random variables?

8 Acknowledgements

We would like to thank the anonymous reviewers for sharing their insights, which undoubtedly improved the presentation of these ideas.

References

[1] J. W. H. Liu, *A compact row storage scheme for cholesky factors using elimination trees*, ACM TOMS, 12 (1986), pp. 127–148.

[2] N. Balakrishnan and C. R. Rao, *Order Statistics: Theory & Methods*, Handbook of Statistics, 16 (1998)

[3] J. Ryser, *Combinatorial Mathematics*, The Carus Mathematical Monographs, 14 (1963)

[4] L. G. Valiant, *The Complexity of Computing the Permanent*, Theoretical Computer Science, 1979, pp. 189–201.

[5] H. A. David and H. N. Nagaraja, *Order Statistics, Third Edition*, 3 (2003)

[6] Y. Benjamini and Y. Hochberg, *Controlling the false discovery rate: a practical and powerful approach to multiple testing*, Journal of the Royal statistical society: series B (Methodological), 57 (1995), pp. 289–300.

[7] I. Pitas and A. Venetsanopoulos, *Order statistics in digital image processing*, Proceedings of the IEEE, 80 (1992).

[8] N. Balakrishnan, *Permanents, order statistics, outliers, and robustness*, Revista matemática complutense, 20 (2007), pp. 7–107.

[9] H. Yang and M. Alouini, *Order statistics in wireless communications: diversity, adaptation, and scheduling in MIMO and OFDM systems*, 2007

[10] R. B. Bapat and M. I. Beg, *Order statistics for nonidentically distributed variables and permanents*, The Indian Journal of Statistics, Series A, 1989, pp. 79–93.

[11] C. G. Boncelet Jr., *Algorithms to computer order statistic distributions*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. 868–876.

[12] D. H. Glueck, A. Karimpour-Fard, J. Mandel, L. Hunter, and K. E. Muller, *Fast computation by block permanents of cumulative distribution functions of order statistics from several populations*, Communications in Statistics: theory and methods, 37 (2008), pp. 2815–2824.

[13] S. Hande, *A Note on Order Statistics for Nondentically Distributed Variables*, The Indian Journal of Statistics, Series A (1961–2002), 56 (1994), pp. 365–368.

[14] G. Cassela and R. Berger, *Statistical Inference (2nd ed.)*, 2002.

[15] R. B. Myerson, *Optimal Auction Design* Mathematics of Operations Research, 6 (1981).

[16] H. L. Varian, *Position Auctions*, International Journal of Industrial Organization, 25 (2007), pp. 1163–1178.

[17] J. von Schroeder and T. Dickhaus, *Efficient calculation of the joint distribution of order statistics*, Computational Statistics and Data Analysis, 144 (2020).

[18] G. Shorack and J. Wellner, *Empirical Processes with Applications to Statistics*, Society for Industrial and Applied Mathematics (1996)

[19] EMA/CPMP, *Points to Consider on Multiplicity Issues in Clinical Trials.*

Efficient signed backward substitution for piecewise affine functions via path problems in a directed acyclic graph*

Torsten Bosse† Ralf Seidler† H. Martin Bücker†‡

Abstract

We introduce an efficient signed backward substitution for a highly-structured system. More precisely, the problem is to find a vector u of dimension s that solves the system of piecewise affine equations $u = c + L|u|$, where L is a strictly lower left triangular $s \times s$ matrix, c denotes a given vector of dimension s, and the notation $|\cdot|$ indicates the component-wise absolute value of a vector. The novel approach is based on a Neumann series reformulation and attempts to exploit a high degree of parallelism. We provide an analysis of its parallel run-time and show that it is suited for large, sparse systems whose triangular matrix has a small switching depth. The general idea behind this approach which is also used in the convergence proof is based on modelling the switching depth by a graph theoretic model. The key observation is that the computation of the switching depth corresponds to a single-pair shortest path problem in a directed acyclic graph. The proposed method is implemented and numerically evaluated using several examples whose problem structures are representative of various applications in scientific computing.

1 Introduction

Combinatorial scientific computing [8, 11] is concerned with the solution of problems arising from scientific computing by means of discrete mathematics. The specific problem from scientific computing addressed in this article is the solution of a system of piecewise affine equations. This article shows its intimate connection to the shortest path problem in a directed acyclic graph.

To introduce the origin of this problem, let $L \in \mathbb{R}^{s\times s}$ be a strictly lower left triangular (s.l.t.) matrix and $c \in \mathbb{R}^s$ a vector of dimension $s \in \mathbb{N}$. Furthermore, the component-wise absolute value of a vector $u \in \mathbb{R}^s$ is denoted by $|u|$ where $|\cdot| : \mathbb{R}^s \to \mathbb{R}^s$. Given L and c, we are interested in the solution $u^* = u^*(c, L) \in \mathbb{R}^s$ of the **S**trictly **L**ower **T**riangular **S**ystem

$$u = c + L|u| \tag{SLTS}$$

by a signed backward substitution. In the sequel, this problem is abbreviated by `SLTS(c,L)`. Although it is closely related to the backward substitution for triangular systems of linear equations [4], it is currently not among the standard numerical linear algebra kernels. For example, it is not available in any of the common linear algebra packages like BLAS [12], LAPACK [1], MATLAB [10], BLIS [18] or Armadillo [15]. One of the reasons why linear algebra packages tend to omit this problem might be the lack of applications and, thus, the need for highly efficient implementations.

Our motivation to study this problem stems from modelling piecewise-smooth functions

$$y = F(x), \quad F : \mathbb{R}^n \to \mathbb{R}^m,$$

by piecewise affine approximations

$$\Delta y = \Delta F(x, \Delta x), \quad \Delta F : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m.$$

In contrast to smooth functions, where the linear approximation $F(x + \Delta x) \approx F(x) + F'(x)\Delta x$ at a point $x \in \mathbb{R}^n$ in direction $\Delta x \in \mathbb{R}^n$ can be represented by a simple matrix-vector product, the piecewise affine approximations $F(x+\Delta x) \approx F(x)+\Delta F(x, \Delta x)$ require a more sophisticated algebraic representation; see [9, 16].

One representation is the abs-normal form (ANF) given by the system of piecewise affine equations

$$\begin{bmatrix}\Delta z\\ \Delta y\end{bmatrix} = \begin{bmatrix}a\\ b\end{bmatrix} + \begin{bmatrix}Z & L\\ J & Y\end{bmatrix}\begin{bmatrix}\Delta x\\ |\Delta z|\end{bmatrix}. \tag{1.1}$$

Here, $a = a(x) \in \mathbb{R}^s$, $b = b(x) \in \mathbb{R}^m$, $Z = Z(x) \in \mathbb{R}^{s\times n}$, $L = L(x) \in \mathbb{R}^{s\times s}$, $J = J(x) \in \mathbb{R}^{m\times n}$, $Y = Y(x) \in \mathbb{R}^{m\times s}$ are certain vectors and matrices that contain derivative information of F. As discussed in [5], this information can be evaluated by techniques of algorithmic differentiation [14, 7]. Without loss of generality, the matrix L is strictly lower left triangular [6, 17]. For any direction Δx, this triangular structure allows to evaluate not only the artificial vector of *switching*

*Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – INST 275/334–1 FUGG; INST 275/363–1 FUGG

†Institute for Computer Science, Friedrich Schiller University Jena, 07737 Jena, Germany.

‡Michael Stifel Center Jena for Data-driven and Simulation Science, 07737 Jena, Germany.

variables $\Delta z \in \mathbb{R}^s$, but also the piecewise affine approximation $\Delta y = \Delta F(x, \Delta x)$. Setting

$$u = \Delta z \quad \text{and} \quad c = a + Z\Delta x$$

in (1.1), shows that a basic building block of the ANF consists of the evaluation of the switching vector Δz which corresponds to the solution of a strictly lower triangular system of the form (SLTS).

From a conceptual point of view, solving (SLTS) for u is not difficult and its unique solution u^* can be easily determined by sequentially computing the entries u_i^* in the order $i = 1, \ldots, s$. However, the sequential nature of this problem is computationally challenging since it is not favourable for modern computer architectures that support parallel computations, including GPUs. The novel contribution of this article is to develop, analyse, and implement efficient parallel methods that are based on a reformulation of (SLTS) using Neumann series. The new methods do not only deliver performance superior to the naive approach in certain cases, but also allow the implementation of matrix-free methods, where the matrix L is not explicitly known and only matrix-vector products are available.

The efficiency of these new methods crucially depends on a certain parameter to be defined as follows. A matrix L is called nil-potent if there is a natural number k such that $L^k = 0$ and the smallest number with this property is called the *degree of nil-potency*, denoted by the symbol $\nu(L)$. If L is sparse it is common to abstract from the nonzero values by taking into account only the nonzero pattern. The nonzero pattern of a matrix L is denoted by the symbol $\mathcal{L}$. In general, the nonzero pattern of powers for L and $\mathcal{L}$ differ, namely, the pattern of L^k does not necessarily coincide with $\mathcal{L}^k$ for $k > 1$. The reason is that numerical computations on nonzero values incidentally can lead to zero values in L^k. The idea behind considering the pattern rather than the values is to neglect these numerical cancellations. The smallest number $k \in \mathbb{N}$ such that $\mathcal{L}^k = 0$ is called the *switching depth of a matrix* L denoted by $\mu(L)$. Notice that, in general, $\mu(L) \neq \nu(L)$; see § 3 for an example.

The switching depth $\mu(L)$ reflects the depth of nested absolute value functions in (SLTS) induced by an s.l.t. matrix L. Thus, it can serve as an indicator of the computational complexity of the piecewise affine approximation represented by an ANF (1.1). As observed in [3], it is reasonable to consider sparse matrices L involving certain (block) structures. The sparsity can lead to values of the switching depth $\mu(L)$ that are comparatively small, even if the dimension s of the matrix L is large. In these situations, the efficiency of algorithms for evaluating an ANF crucially depends on recognizing and exploiting small values of $\mu(L)$.

The structure of this article is as follows: In § 2, we present three basic approaches to solve (SLTS) using `dot`-products and `axpy` operations. This section also contains a novel approach based on an alternative reformulation of (SLTS) using Neumann series and a modified version best suited for sparse problems with a small switching depth. The run-time analysis and convergence proof for the modified Neumann approach is based on a mathematical rigorous definition and a graph theoretic model for the switching depth, which is presented in § 3. The theoretical results are validated by a set of numerical experiments, which are provided in § 4 before the conclusions.

2 Solving Signed Lower Triangular Systems

As mentioned in the introduction, (SLTS) is closely related to backward substitution for triangular systems. In detail, if the (element-wise) signatures

$$\sigma = \sigma(u) = \operatorname{sign}(u) \in \{-1, 1\} \in \mathbb{R}^s$$

of the solution vector u^* for `SLTS(c,L)` are known, then (SLTS) reduces to the solution of the triangular system

$$(I - L\Sigma_{u^*})u = c. \tag{2.2}$$

Here, $I \in \mathbb{R}^{s\times s}$ denotes the identity matrix and

$$\Sigma_u = \Sigma(u) = \operatorname{diag}(\sigma(u)) \in \mathbb{R}^{s\times s}$$

is the diagonal matrix that contains the signatures σ of a vector $u \in \mathbb{R}^s$ on its main diagonal such that $\Sigma_u u = |u|$. In most cases the correct signatures σ^* of the solution u^* are not known a priori and, thus, standard methods are not applicable. For a dense matrix L, it is possible to formulate the (simple) Algorithm 1 that exploits the triangular structure of L and finds the entries u_i of the solution of `SLTS(c,L)` sequentially without knowing their correct signatures σ_i^* a priori.

Algorithm 1 $[u] = \texttt{SLTS}[c, L]$

1: Set $u = (c_1, 0, \ldots, 0)^\top$
2: **for** $i = 2, \ldots, s$ **do**
3: Update $u_i = c_i + \sum_{j=1}^{i-1} L_{i,j}|u_j|$
4: **end for**
5: **return** u

Obviously, this algorithm is optimal with respect to the number of required numerical operations. Two alternative implementations of this algorithm are given by Algorithms 2 and 3, which are based on either `dot`-products or `axpy`s, respectively. Although, both alternatives require some more operations than the naive implementation of Algorithm 1, their implementation is

likely to be superior in terms of *parallel run-time* since they allow to reuse existing standard routines that are highly optimised for parallel hardware architectures.

Algorithm 2 $[u] = \texttt{SLTS\_DOT}[c, L]$

1: Set $u = (c_1, 0, \ldots, 0)^\top$
2: **for** $i = 2, \ldots, s$ **do**
3: Update $u_i = c_i + \langle L_{i,:}, |u| \rangle$
4: **end for**
5: **return** u

Algorithm 3 $[u] = \texttt{SLTS\_AXPY}[c, L]$

1: Set $u = (c_1, c_2, \ldots, c_s)^\top$
2: **for** $i = 2, \ldots, s$ **do**
3: Update $u = u + |u_{i-1}| L_{:,i-1}$
4: **end for**
5: **return** u

Here, the term *parallel run-time* refers to the number of steps to perform arithmetic operations on an ideal parallel architecture. That is, we do not take into account any costs for communication and/or storage access. Of course, this definition of parallel run-time is unrealistic. However, its simplicity offers the advantage to abstract from practical issues like data distribution, communication including collective operations, synchronization, contention, and concurrent access to shared resources. In fact, for the purpose of this article, it is sufficient to assume that the parallel run-time to compute a matrix-vector product Ax with $A \in \mathbb{R}^{m \times n}$ and $x \in \mathbb{R}^n$ is given by

$$T_{\mathrm{MVP}}(n) = O(\log n) \tag{2.3}$$

using $mn/2$ processors. Each of the m scalar results of a matrix-vector product are computed independently by an inner product that is carried out in $O(\log n)$ using $n/2$ processors in a tree-like fashion. If the matrix is sparse with at most r nonzeros per row we replace (2.3) by $O(\log r)$ using $mr/2$ processors.

The evaluation of the ANF (1.1) consists of several parts: (i) matrix-vector products involving the matrices Z and J both with n columns and the matrix Y with s columns, (ii) the solution of (SLTS) involving an $s \times s$ matrix L, and (iii) vector-vector operations on vectors of length n and s. Under the assumption (2.3), the parallel run-time for evaluating parts (i) and (iii) of the ANF is bounded from above by $O(\log s)$ and $O(\log n)$. However, the parallel run-time for part (ii) which consists of the solution of (SLTS) to find Δz is bounded from below by $\Omega(s)$. The reason is that any of the three methods presented in Algorithms 1–3 involves data dependencies that result from the serial nature of the s instances of the for-loop. The iteration i needs the values computed in the previous iteration $i-1$. To summarise, the problem (SLTS) can constitute the computational bottleneck in the evaluation of (1.1); see [3] for further information.

As a remedy to go below the lower bound $\Omega(s)$ of Algorithms 1–3, we propose an alternative approach to solve (SLTS) that is based on Neumann series for matrices [4]:

COROLLARY 2.1. (NEUMANN) *For any $A \in \mathbb{R}^{s \times s}$ with $\|A\| < 1$, the matrix $I - A$ is nonsingular and*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

Therefore, assume for a moment that the correct signatures σ^* of the solution u^* of `SLTS(c,L)` are known and are used to define the signature matrix Σ_{u^*}. Then, equation (2.2) and Corollary 2.1 allow to reformulate `SLTS(c,L)` as the finite Neumann series

$$u = (I - L\Sigma_{u^*})^{-1} c = \sum_{k=0}^{s} (L\Sigma_{u^*})^k c \tag{2.4}$$

due to the strictly lower triangular structure of $L\Sigma_{u^*}$. This observation and the triangular structure give rise to the iterative approach given in Algorithm 4 that solves (SLTS) by successively determining the correct signatures and, thus, the solution u^*.

Algorithm 4 $[u] = \texttt{SLTS\_Neumann\_Naive}[c, L]$

1: Set $u = c$
2: **for** $j = 1, \ldots, s$ **do**
3: Set $\Sigma_u = \mathrm{diag}(\mathrm{sign}(u))$
4: Set $u^{\mathrm{new}} = c$
5: **for** $l = 1, \ldots, s$ **do**
6: Evaluate $u^{\mathrm{new}} = L\Sigma_u u^{\mathrm{new}} + c$
7: **end for**
8: Update $u = u^{\mathrm{new}}$
9: **end for**
10: **return** u

The strategy of Algorithm 4 is as follows: Pick any vector $u \in \mathbb{R}^s$ as an initial guess for the solution u^*. For example, $u = c$ seems to be a reasonable choice since it immediately yields the correct solution if $c = 0$ or $L = 0$. The algorithm then proceeds in an outer iteration until it finds the solution u^*. In each outer iteration, the algorithm determines an improved approximation u^{new} of u^* by evaluating (2.4) in an inner iteration using the diagonal matrix Σ_u, which is defined by the signatures of the current approximation u.

PROPOSITION 2.1. *Under the assumption* (2.3) *Algorithm 4 solves* `SLTS(c,L)` *within a parallel run-time of* $O(s^2 \log s)$ *using* $s^2/2$ *processors.*

Proof. The proof follows by an inductive argument and the triangular structure of L. In detail, assume that u is the current approximation of u^*, such that the first $1 \leq k \in \mathbb{N}$ signatures coincide, namely, $\operatorname{sign}(u_i) = \operatorname{sign}(u_i^*)$ for all $i = 1, \ldots, k$. Then, the triangular structure of L guarantees that the first $k+1$ signatures of u^{new} and u^* coincide after evaluating $u^{\text{new}} = \sum_{k=0}^{s} (L\Sigma_u)^k c$ in the inner loop with the signature matrix Σ_u defined by the signatures of u, i.e., $\operatorname{sign}(u_i^{\text{new}}) = \operatorname{sign}(u_i^*)$ for all $i = 1, \ldots, k+1$. Thus, the algorithm determines at least one additional correct signature per outer iteration such that u^* is found after at most s outer iterations. The run-time $O(s^2 \log s)$ is due to the required operations for evaluating $u^{\text{new}} = \sum_{k=0}^{s} (L\Sigma_u)^k c$ in each of the s sequential outer iterations. ∎

For dense problems, the parallel run-time for Algorithm 4 of order $O(s^2 \log s)$ exceeds the lower bound $\Omega(s)$ for the first three presented methods. Therefore, we propose to reduce its parallel run-time by introducing the modified Algorithm 5. This modification avoids the inner loop at all and simply uses the signatures of the latest computed approximation u of u^*.

Algorithm 5 $[u] =$ `SLTS_Neumann`$[c, L]$

1: Set $u = [0, 0, \ldots, 0]^\top$
2: Set $u^{\text{new}} = c$
3: **while** $(u^{\text{new}} \neq u)$ **do**
4: Update $u = u^{\text{new}}$
5: Set $\Sigma_u = \operatorname{diag}(\operatorname{sign}(u))$
6: Evaluate $u^{\text{new}} = L\Sigma_u u + c$
7: **end while**
8: **return** u

As proven in the next section, the modified approach converges to the solution of `SLTS(c,L)` and is usually more efficient than all of the previously suggested algorithms.

PROPOSITION 2.2. *Algorithm 5 finds a solution* u^* *of* `SLTS(c,L)` *and requires at most* $\mu(L)$ *iterations. Under the assumption* (2.3) *its parallel run-time is* $O(\mu(L) \log r)$ *using* $sr/2$ *processors, where* $r \in \mathbb{N}$ *denotes the maximal number of nonzero entries for each row of* L.

Obviously, Algorithm 5 is superior for problems that are sparse and have a small switching depth $\mu(L)$. Besides taking advantage of the implicit dependency structure hidden in the matrix L, the algorithm offers a number of further advantages. For example, it does not require any explicit representation of L and allows matrix-free techniques to perform a matrix-vector product $L\tilde{v}$ for given vector $\tilde{v}$. If the matrix L is the derivative of a function, these matrix-vector products are efficiently evaluated by the forward mode of algorithmic differentiation. Also, Algorithm 5 enables "warm-start" strategies by omitting the assignment $u^{\text{new}} = c$ in the initialisation and allowing the user to provide an adequate initial guess to u. Both features are beneficial in the context of ANFs, where L corresponds to a certain derivative of a piecewise smooth function F as discussed in [2, 3]. Here usually a sequence of ANFs needs to be evaluated for slightly different entries giving rise to evaluate the slightly perturbed problem `SLTS(`$\tilde{c}$`,`$\tilde{L}$`)`. Depending on the function F and the perturbation, it is expected that the solution u^* of `SLTS(`c`,`L`)` is also an approximation for the solution $\tilde{u}^*$ of `SLTS(`$\tilde{c}$`,`$\tilde{L}$`)`. Thus, using an initialisation that replaces c by u^*, Algorithm 5 tends to find the solution $\tilde{u}^*$ even before reaching the upper bound on the iterations given by $\mu(L)$.

The proof of Algorithm 5 is based on a graph theoretic model of the switching depth introduced in the next section.

3 Graph Theoretic Model

In this section, a mathematical rigorous model of the switching depth is introduced that is based on graphs. This model does not only allow to efficiently compute the a priori unknown switching depth, but also helps to prove the correctness and the parallel run-time for Algorithm 5.

The following definition represents the dependency structure in (SLTS) induced by the nonzero structure of the matrix L.

DEFINITION 3.1. *Let* $L \in \mathbb{R}^{s \times s}$ *be an s.l.t. matrix with entries* $L_{i,j}$. *The directed acyclic graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *representing the rows and columns by the vertices* $\mathcal{V} = \{v_1, \ldots, v_s\}$ *and the nonzeros by the directed edges*

$$\mathcal{E} = \{(v_i, v_j) \in \mathcal{V} \times \mathcal{V} \mid L_{i,j} \neq 0\}$$

is called the induced graph of L *and denoted by* $\mathcal{G}(L)$.

The graph theoretic representation of L as a directed acyclic graph (DAG) allows a mathematical rigorous definition of the switching depth.

DEFINITION 3.2. *Let* $\mathcal{P}$ *be the set of all paths in the directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *induced by* L, *namely,*

$$\mathcal{P} = \big\{(v_{i_1}, v_{i_2}, \ldots, v_{i_{\ell+1}}) \mid v_{i_j} \in \mathcal{V} \text{ and } (v_{i_j}, v_{i_{j+1}}) \in \mathcal{E} \text{ for } 1 \leq j \leq \ell\big\}, \tag{3.5}$$

and denote by $\langle p \rangle = \ell \in \mathbb{N}$ *the length of a path* $p = (v_{i_1}, v_{i_2}, \ldots, v_{i_{\ell+1}}) \in \mathcal{P}$ *with* ℓ *edges. Then, the switching depth* $\mu(L)$ *of* L *is defined as one more than the maximal length of all paths*

$$\mu(L) = 1 + \max_{p \in \mathcal{P}} \langle p \rangle.$$

In other words, the switching depth of L exceeds by one the longest path in the induced graph $\mathcal{G}(L)$. Recall from § 1 that the switching depth is also defined as the smallest number k such that $\mathcal{L}^k = 0$. This alternative definition coincides with Definition 3.2 because it is well-known [13] that the entry in row i and column j of the matrix $\mathcal{L}^k$ is equal to m if and only if there are m paths from vertex v_i to vertex v_j via exactly k edges.

The difference between the dimension s, the degree of nil-potency $\nu(L)$, and the nil-potency neglecting numerical cancellations, i.e., the switching depth $\mu(L)$, is illustrated in the next example.

Example. Consider the s.l.t. matrix $L \in \mathbb{R}^{s \times s}$ with $s = 5$ given by

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ +1 & +1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \\ +1 & 0 & +1 & +1 & 0 \end{bmatrix}. \tag{3.6}$$

Its induced graph $\mathcal{G}(L)$ visualised in Figure 1 consists of five vertices and seven edges. Obviously, the degree of nil-potency is given by $\nu(L) = 2$ since $L^2 = 0$. However, its switching depth is $\mu(L) = 3$ because the paths (v_5, v_3, v_1), (v_5, v_4, v_1) and (v_5, v_4, v_2) have length 2 and there is no longer path. Observe that $\mu(L) < s$. ∎

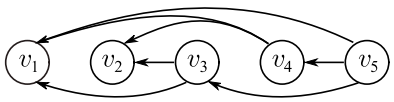


Figure 1: The graph $\mathcal{G}(L)$ induced by L in (3.6) whose longest path is of length $\mu(L) - 1 = 2$.

The graph model can be used to efficiently determine the switching depth, which can be found by a weighted shortest path reformulation on a DAG.

PROPOSITION 3.1. *For any s.l.t. matrix* $L \in \mathbb{R}^{s \times s}$ *with* $R \in \mathbb{N}$ *nonzero entries, computing the switching depth* $\mu(L)$ *is equivalent to finding a single-pair shortest path problem on a DAG with* $s + 2$ *vertices and* $R + 2s$ *weighted edges.*

Proof. The directed graph $\mathcal{G}(L) = (\mathcal{V}, \mathcal{E})$ induced by L consists of $|\mathcal{V}| = s$ vertices and $|\mathcal{E}| = R$ edges. Now, consider the weighted extended graph $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ that is defined by adding to $\mathcal{G}(L)$ two vertices

$$\widetilde{\mathcal{V}} = \mathcal{V} \cup \{v_0, v_{s+1}\}$$

and $2s$ directed edges

$$\widetilde{\mathcal{E}} = \mathcal{E} \cup \{(v_i, v_0), 1 \le i \le s\} \cup \{(v_{s+1}, v_i), 1 \le i \le s\}.$$

All edges in $\widetilde{\mathcal{E}}$ are weighted by the value -1. Since $\mathcal{G}$ is a DAG, by construction $\widetilde{\mathcal{G}}$ is also a DAG. Then, finding the length of a longest path in the (unweighted) graph $\mathcal{G}$ is equivalent to determining the weight of a shortest path from v_{s+1} to v_0 in the (weighted) graph $\widetilde{\mathcal{G}}$, where the weight of a path $\widetilde{p}$ is the sum of the weights of its constituent edges. Let p denote a longest path in $\mathcal{G}$ and let $\langle \widetilde{p} \rangle_{\mathrm{w}}$ be the weight of a shortest path $\widetilde{p}$ in $\widetilde{\mathcal{G}}$. By construction, we find the connection

$$\langle p \rangle = -\langle \widetilde{p} \rangle_{\mathrm{w}} - 2 \tag{3.7}$$

between the length and weight of a path in an unweighted and weighted DAG, respectively. ∎

The idea to compute the switching depth via shortest paths can be used to determine the *depths* of all vertices in the extended graph $\widetilde{\mathcal{G}}$.

DEFINITION 3.3. *Let* $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ *be the extended DAG associated with the DAG induced by* $L \in \mathbb{R}^{s \times s}$*. The depth* $d_v \in \mathbb{N}$ *of a vertex* $v \in \widetilde{\mathcal{V}}$ *is defined as the weight of a shortest path from* v *to* v_0*. That is, let*

$$\mathcal{P}_v = \big\{(v_{i_1}, v_{i_2}, \ldots, v_{i_{\ell+1}}) \mid v_{i_1} = v, v_{i_{\ell+1}} = v_0 \in \widetilde{\mathcal{V}}, \\ v_{i_j} \in \widetilde{\mathcal{V}} \text{ and } (v_{i_j}, v_{i_{j+1}}) \in \widetilde{\mathcal{E}} \text{ for } 1 \le j \le \ell\big\}$$

denote the set of all paths of any length ℓ *that begin at a vertex* v *and end at vertex* v_0*. Then, the depth of* v *is defined as*

$$d_v = -\min_{p \in \mathcal{P}_v} \langle p \rangle_{\mathrm{w}}.$$

Neglecting the edge weights in $\widetilde{\mathcal{G}}$, this definition sets the depth d_v to the number of edges of the longest path from a vertex v to the vertex v_0. These depths in the extended graph $\widetilde{\mathcal{G}}$ are used in the following proposition.

PROPOSITION 3.2. *Let* $\widetilde{\mathcal{G}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ *denote the edge-weighted extended DAG from the proof of Proposition 3.1. For all vertices* $v \in \widetilde{\mathcal{V}}$*, introduce the depth* d_v *of a vertex* v *as a (vertex) weight. Furthermore, let the symbol* u_i^k *denote the entry at position* $1 \le i \le s$ *of the vector* u^{new} *in the iteration* k *of the while loop in Algorithm 5. Then,* u_i^k *is equal to the correct value of the entry at position* i *of the solution,* u_i^**, and will not change in all following iterations of the while loop if* $d_{v_i} \le k$*.*

Proof. The proposition is proofed by induction on the iteration k of the while loop. For the base $k = 1$, we observe that any vertex v_i with depth $d_{v_i} \leq 1$ has the only successor v_0. So, there is no vertex v_j with $1 \leq j \leq s$ such that $(v_i, v_j) \in \widetilde{\mathcal{E}}$. This statement is equivalent to the row i of L containing solely zero values. Due to $u = L\Sigma_u u + c$, this zero row i in L leads to $u_i^1 = c_i = u_i^*$. That is, u_i^1 is equal to the value of the correct solution u_i^*. Notice that the data dependencies encoded in $\widetilde{\mathcal{G}}$ show that the values of u_i^1 with $d_{v_i} \leq 1$ will not change in any of the following iterations.

For the induction step, we assume that

$$u_i^k = u_i^* \text{ if } d_{v_i} \leq k. \tag{3.8}$$

We then need to show that $u_i^{k+1} = u_i^*$ if $d_{v_i} \leq k+1$. We proof this induction step indirectly and assume that there is a position i with $d_{v_i} \leq k+1$ but $u_i^{k+1} \neq u_i^*$. Since the row i of L denoted by $L_{i,:}$ and c_i are fixed and Algorithm 5 performs the update

$$u_i^{k+1} = L_{i,:}\Sigma_{u^k} u^k + c_i,$$

there must be an incorrect value in any of the entries of the vector from the previous iteration k, denoted by u^k. Assume, without loss of generality, that there is an incorrect value in u_j^k. This incorrect value at position j will only influence the value of u_i^{k+1} if $L_{i,j} \neq 0$. So, there is an edge $(v_i, v_j) \in \widetilde{\mathcal{E}}$ which means that the depth of the vertex v_j is given by $d_{v_j} = k$. However, this contradicts the hypothesis (3.8) which shows the value u_j^k to be correct. ∎

The construction introduced in the previous proposition constitutes the essence of the proof of Proposition 2.2.

Proof. [Proposition 2.2] According to Proposition 3.2 the values of u_i^k for $1 \leq i \leq s$ are correct at the latest when the iteration k arrives at $k = \max_{1 \leq i \leq s} d_{v_i}$, where the depths are taken with respect to the extended graph $\widetilde{\mathcal{G}}$. That is, the maximal number of iterations of Algorithm 5 is given by the maximal depth of any vertex v_i with $1 \leq i \leq s$. Since vertex v_0 is included and v_{s+1} is excluded, this depth equals the number of edges of a shortest path $\widetilde{p}$ in $\widetilde{\mathcal{G}}$ excluding v_{s+1} and all its incident edges. Since all edges in $\widetilde{\mathcal{G}}$ are weighted by the value -1, we have $k = -\langle \widetilde{p} \rangle_{\mathrm{w}}$. Since v_{s+1} is removed from $\widetilde{\mathcal{G}}$, the connection between the shortest path in $\widetilde{\mathcal{G}}$ and the longest path p in $\mathcal{G}$ is similar to (3.7) and is given by $\langle p \rangle = -\langle \widetilde{p} \rangle_{\mathrm{w}} - 1$. So, the maximal number of iterations is $k = \langle p \rangle + 1$ which, by definition, is the switching depth $\mu(L)$.

Using assumption (2.3) the parallel run-time is $O(\mu(L) \log r)$ using $sr/2$ processors. ∎

Recall that Proposition 2.2 specifies the parallel run-time of Algorithm 5 to solve `SLTS(c,L)` as $O(\mu(L) \log r)$ using $sr/2$ processors. On the other hand, problem `SLTS(c,L)` can always be solved using Algorithm 3 within parallel run-time of $\Theta(s)$ using s processors. Disregarding the number of processors, the choice, which of the two algorithms is more efficient and shall be used depends on $\mu(L)$. Namely, if $\mu(L) \log r \geq s$ then one should use Algorithm 3, whereas $\mu(L) \log r \leq s$ suggests to use Algorithm 5. Hence, Problem `SLTS(c,L)` can be solved in a maximal parallel run-time of order $O\big(\min(s, \mu(L) \log r)\big)$ but requires a pre-test to verify or falsify $\mu(L) \log r \leq s$.

Certainly, the parallel run-time of this pre-test should not exceed the parallel run-time of the problem itself. Thus, computing the switching depth by solving a shortest path problem as stated in Proposition 3.1 is computationally not feasible in practice. The reason is that any serial or parallel shortest path algorithm for this problem depends on the number of vertices and/or edges given by the dimension s and/or the total number of nonzero entries in L denoted by $R \in \{0, \ldots, s(s-1)/2\}$, respectively. Typically, this parallel run-time exceeds the parallel run-time of $O\big(\min(s, \mu(L) \log r)\big)$.

However, it is possible to formulate Algorithm 6 that does not exceed this run-time order and computes a lower bound for the switching depth to decide whether Algorithm 3 or Algorithm 5 should be used.

PROPOSITION 3.3. *Given the sparsity pattern $\mathcal{L}$ of an s.l.t. matrix L, Algorithm 6 determines a lower bound μ_{low} for the switching depth $\mu(L)$ that can be used to verify if $\mu(L) \log r \geq s$. Under the assumption* (2.3) *its parallel run-time is given by $O\big(\min(s, \mu(L) \log r)\big)$ using $s^2/2$ processors.*

Proof. Observe that the graph definition of the switching depth, Definition 3.2, only depends on the sparsity pattern $\mathcal{L} = (\mathcal{L}_{i,j})$ of L, where

$$\mathcal{L}_{i,j} = \begin{cases} 1 & \text{if } L_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}.$$

Since $\mathcal{L}$ only has non-negative entries, no incidental cancellations can occur in $\mathcal{L}^k$ and, thus, the switching depth $\mu(L) = \mu(\mathcal{L})$ coincides with the nil-potency degree $\nu(\mathcal{L})$. Therefore, any lower bound μ_{low} on the nil-potency degree $\nu(\mathcal{L})$ is also a lower bound for $\mu(\mathcal{L})$. To find such a lower bound, Algorithm 6 sequentially tests if the patterns of $\mathcal{L}^2, \mathcal{L}^3, \ldots$ become zero and tries to find a lower bound μ_{low} for the smallest exponent k for which $\mathcal{L}^k = 0$. The iteration is stopped prematurely if the lower bound μ_{low} exceeds the threshold $s/\log r$. Since $\mu_{\mathrm{low}} \leq \mu(L)$ and since computing the pattern of

the product $\mathcal{L}u$ requires a parallel run-time of $O(\log r)$ using $sr/2$ processors, the criteria $\mu_{\text{low}} \log r \leq s$ implies that the overall complexity of the loop and, thus, the parallel run-time of the algorithm on $s^2/2$ processors is bounded by $O\big(\min(s, \mu(L) \log r)\big)$. ∎

Algorithm 6 $[\mu_{\text{low}}] = \texttt{SLTS\_PreTest}[\mathcal{L}]$

```
Set r to the maximal number of nonzeros per row in L
Set u = [0, 0, ..., 0]^T and u^new = [1, 1, ..., 1]^T
Initialise μ_low = 0
if (r = 0) then
    return μ_low
end if
while (u^new ≠ [0, ..., 0]^T) and (μ_low log r ≤ s) do
    Set μ_low = μ_low + 1
    Update u = min(1, u^new) (component-wise)
    Evaluate u^new = Lu
end while
return μ_low
```

Combining the pre-test described in Algorithm 6 with Algorithms 3 and 5 then yields an efficient method for solving (SLTS) that is suited for parallel hardware architectures and automatically takes benefit from small switching depths as is stated in the following theorem.

THEOREM 3.1. *Under the assumption* (2.3) *problem* (SLTS) *can be solved within a parallel run-time of* $O\big(\min(s, \mu(L) \log r)\big)$ *using* $s^2/2$ *processors.*

Proof. Follows from Propositions 2.2 and 3.3. ∎

An experimental validation of these theoretical results is presented in the next section.

4 Experimental Evaluation

We report the results of numerical experiments which are carried out to validate the proposed Neumann method and compare its efficiency with other approaches. Therefore, the proposed signed lower triangular solver from § 2 is tested on several examples with common structures.

EXPERIMENT 1. *All matrices used in the experiments have different properties and characteristics. In detail, each matrix L has a certain sparsity pattern $\mathcal{L}$ with switching depth $\mu(\mathcal{L})$, which can be (partly) controlled by the parameter $q \in (0, 1] \subset \mathbb{R}$. The experiments comprise five sparse s.l.t. matrices $L \in \mathbb{R}^{s \times s}$ with corresponding vectors $c \in \mathbb{R}^s$ as well as a rescaled test problem with a dense s.l.t. matrix $L \in \mathbb{R}^{\lceil qs \rceil \times \lceil qs \rceil}$ and $c \in \mathbb{R}^{\lceil qs \rceil}$:*

- $\mathcal{L}_{\text{AntiDiag}}$ - *Anti-diagonal matrices that contain nonzero entries only on the $\lceil 2qs \rceil$ main dense anti-diagonals.*
- $\mathcal{L}_{\text{Arrow}}$ - *Arrow-shaped matrices that contain nonzero entries only on the first $\lceil qs \rceil$ dense columns and last $\lceil qs \rceil$ dense rows.*
- $\mathcal{L}_{\text{Block}}$ - *Block-diagonal matrices, where the size of the dense triangular blocks equals to $\lceil qs \rceil$.*
- $\mathcal{L}_{\text{Dense}}$ - *Dense s.l.t. matrices with rescaled dimension $\tilde{s} = \lceil qs \rceil$, namely, $\mathcal{L}_{\text{Dense}} \in \mathbb{R}^{\lceil qs \rceil \times \lceil qs \rceil}$.*
- $\mathcal{L}_{\text{Sprand}}$ - *Sparse s.l.t. matrices with randomly distributed entries, where q represents the relative amount of nonzeros entries.*
- $\mathcal{L}_{\text{SubDiag}}$ - *Sub-diagonal matrices that contain entries only on the first $\lceil qs \rceil$ dense lower diagonals.*

The sparsity patterns for these matrices are visualised in Figure 2. For all examples, the entries of the dense vector $c \in \mathbb{R}^s$ and the matrices $L \in \mathbb{R}^{s \times s}$ are chosen uniformly from the interval $[-1, 1] \subset \mathbb{R}$. Moreover, all (nonzero) rows in the matrices $L \in \mathbb{R}^{s \times s}$ are normalised to avoid amplification effects. The switching depths, the number of nonzero entries $R \in \mathbb{N}$ of $\mathcal{L}$, and the maximal number $r \in \mathbb{N}$ of nonzeros per row are given in Table 1.

Table 1: Switching depth μ, number of nonzero entries R, and maximal number r of nonzeros per row.

Pattern $\mathcal{L}$	$\mu(\mathcal{L})$	R	r
$\mathcal{L}_{\text{AntiDiag}}$	$\lceil qs \rceil$	$O(qs^2)$	$O(qs)$
$\mathcal{L}_{\text{Arrow}}$	$\lceil qs \rceil$	$O(qs^2)$	$O(s)$
$\mathcal{L}_{\text{Block}}$	$\lceil qs \rceil$	$O(qs^2)$	$O(qs)$
$\mathcal{L}_{\text{Dense}}$	$\lceil qs \rceil$	$O(q^2 s^2)$	$O(qs)$
$\mathcal{L}_{\text{Sprand}}$	$\{1, \ldots, s\}$	$O(qs^2)$	$O(s)$
$\mathcal{L}_{\text{SubDiag}}$	s	$O(qs^2)$	$O(qs)$

The examples from the experiment are used for runtime measurements to compare a MATLAB (R2020b) implementation using dense GPUArrays for the proposed Neumann and AXPY approaches, namely, to contrast Algorithms 3 and 5, respectively. For all examples, the original dimension is set to $s = 2^{12}$ and varying parameters $q \in \{2^0, 2^{-1}, \ldots, 2^{-11}\}$.

The experiments were performed on a standard ThinkPad T480s running Windows 10 that uses an Intel(R) Core(TM) i7-8550U CPU with 1.80 GHz, 16 GB RAM, and an Nvidia MX150 GPU with 2 GB RAM.

The measured double-precision run-times (in seconds) for the AXPY and Neumann methods visualised in Figure 3 represent an average of at least 50 repetitions. The corresponding standard variance is indicated by the coloured area. The figure also shows the run-times of MATLAB's built-in solver for triangular linear systems. Moreover, it contains the run-times for the

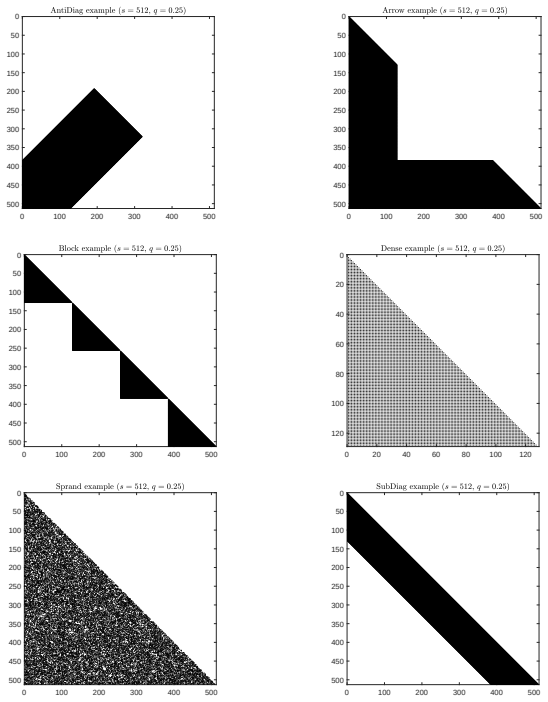


Figure 2: Sparsity pattern of $\mathcal{L}_{\text{AntiDiag}}$, $\mathcal{L}_{\text{Arrow}}$, $\mathcal{L}_{\text{Block}}$, $\mathcal{L}_{\text{Dense}}$, $\mathcal{L}_{\text{Sprand}}$, and $\mathcal{L}_{\text{SubDiag}}$ for $s = 512$ with $q = 0.25$.

combined method, namely, the `SLTS_PreTest` described in Algorithm 6 followed either by the Neumann or the AXPY method as indicated by the grey background.

In general, Figure 3 indicates that the Neumann method is often several magnitudes faster than the AXPY method for sparsity patterns $\mathcal{L}$ with large dimension s and comparatively small switching depth $\mu(\mathcal{L})$.

In detail, it can be observed that the run-time of the AXPY method depends only on the dimension s of the matrix and neither on the parameter q nor on the sparsity pattern $\mathcal{L}$ of the specific example. The dependence on the dimension can be observed for the case $\mathcal{L}_{\text{Dense}}$, where the dimension is set to $\lceil qs \rceil$.

In contrast to the AXPY method, the run-time of the Neumann approach strongly correlates to s, q, and the switching depth. For the considered examples, the measured run-times of the Neumann approach can be explained via the switching depth $\mu(\mathcal{L})$, which is proportional to q and correlates to the number of iterations required by Neumann algorithm to converge. For $\mathcal{L}_{\text{AntiDiag}}$, $\mathcal{L}_{\text{Arrow}}$, $\mathcal{L}_{\text{Block}}$, and $\mathcal{L}_{\text{Sprand}}$, the number of iterations are in accordance with the switching depth presented in Table 1 and the resulting run-times exhibit the expected behaviour as can be deduced from Figure 4. The same holds true for the case $\mathcal{L}_{\text{Dense}}$, where the run-time is (almost) quadratic.

The only example, which is not in accordance with theory and whose results are omitted, is the sub-diagonal case $\mathcal{L}_{\text{SubDiag}}$. Here, one would expect that the run-time of the Neumann approach grows proportional to the factor q since the factor corresponds to the number of sub-diagonals and, thus, to the computational effort for every iteration in the Neumann approach. In particular, the number of iterations should be constant for all q and coincide with the switching depth, namely, $\mu(\mathcal{L}_{\text{SubDiag}}) = s$. However, it was observed that the number of iterations is decreasing, which can (partly) be explained by an unfortunate choice for the entries, which allows the Neumann method to find the solution prematurely due to structural numerical cancellations. For the case L_{SubDiag}, this effect is avoided by the augmented version $L_{\text{SubDiag(v.2.0)}} = L_{\text{SubDiag}} + D$ using the sub-diagonal matrix $D \in \mathbb{R}^{s \times s}$ with

$$D_{i,j} = \begin{cases} j & \text{if } i = j + 1, \\ 0 & \text{otherwise} \end{cases}.$$

The results for the augmented version visualised in Figure 3 are in accordance with theory. In particular, it can be observed that the number of Neumann iterations coincides with the switching depth $\mu(\mathcal{L}_{\text{SubDiag(v.2.0)}}) = s$ and that the run-time is proportional to the number $r = \lceil qs \rceil$ of nonzero entries per row.

According to Figure 3, the run-time of the `combined` approach correlates to the minimum of the run-times for the AXPY and Neumann methods with some additional overhead for executing the `SLTS_PreTest`. The overhead is negligible in real-world use-cases since it only needs to be evaluated once and can be performed offline. In most cases, the test gives a good estimate of when to use the AXPY or Neumann approach, as indicated by the two different background colours. However, the test is rather conservative and, in certain cases, suggests the use of AXPY in favour of Neumann whose run-time is smaller. For example, it suggests to use the Neumann approach only up to a factor $q \leq 2^{-4}$ for L_{Block}, whereas the measured run-times in Figure 3 clearly show that it is better to use Neumann up to a factor $q \leq 2^{-1}$.

A precise `SLTS_PreTest` would identify values of q that are near to where the curves of the run-time of Neumann and AXPY intersect. This intersection does not only depend on the quantities $\mu(L)$, r, and s but also on the hardware effecting the actual run-times. Therefore, the precision of our conservative `SLTS_PreTest` also depends on these three quantities and the hardware. The influence of changing one of these factors is depicted in Figure 5, where the dimension s is now increased to 2^{13}. Compared to Figure 3 the `SLTS_PreTest` tends to estimate the intersection point of the Neumann and AXPY run-time curves more precisely.

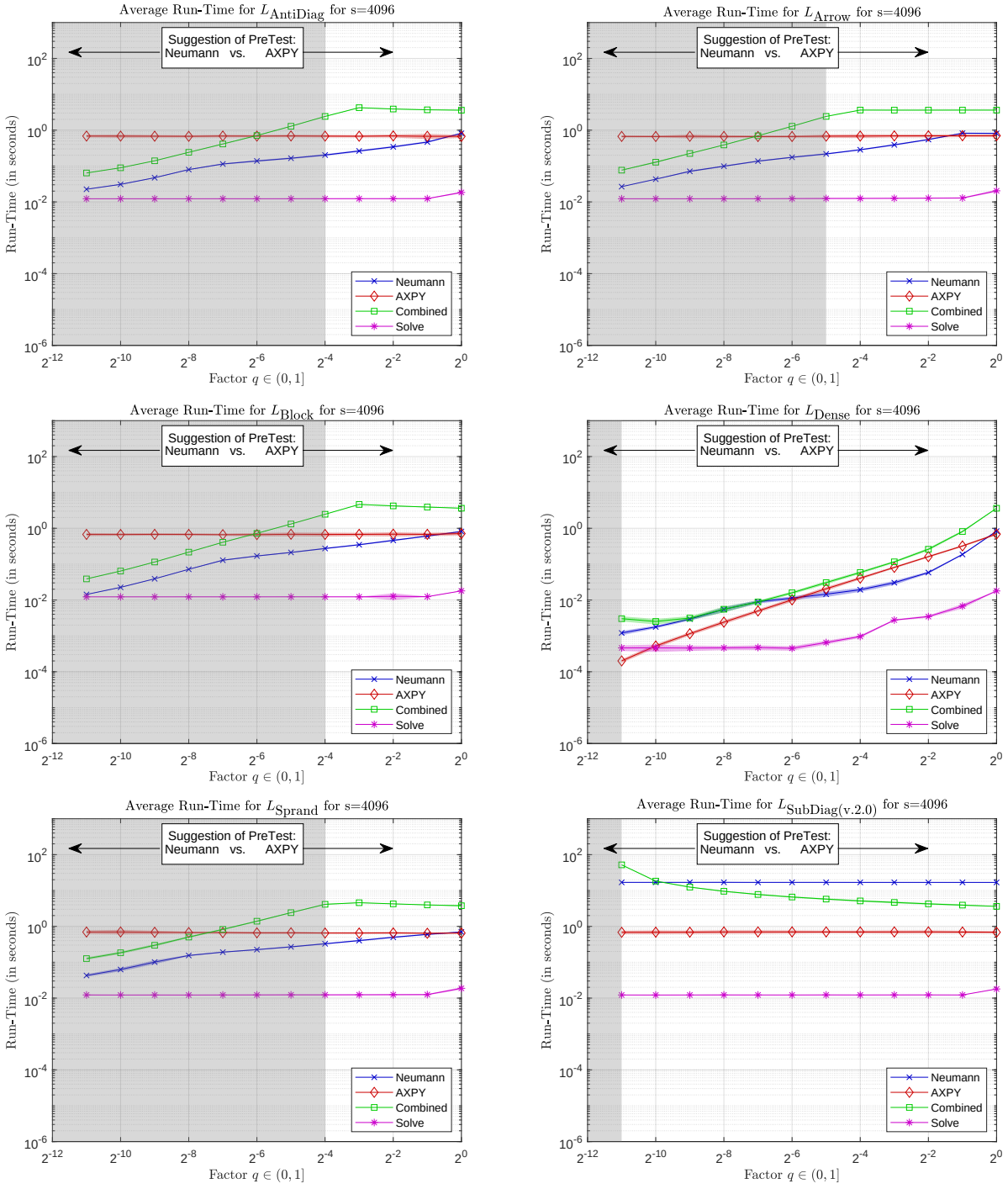


Figure 3: Run-time measurements (in seconds) to solve (SLTS) with Algorithms 3 and 5 for L_{AntiDiag}, L_{Arrow}, L_{Block}, L_{Dense}, L_{Sprand}, and $L_{\text{SubDiag(v.2.0)}}$ compared to MATLAB's triangular solver and the combined method based on Algorithm 6 automatically determining whether to use the Neumann (grey background) or AXPY method.

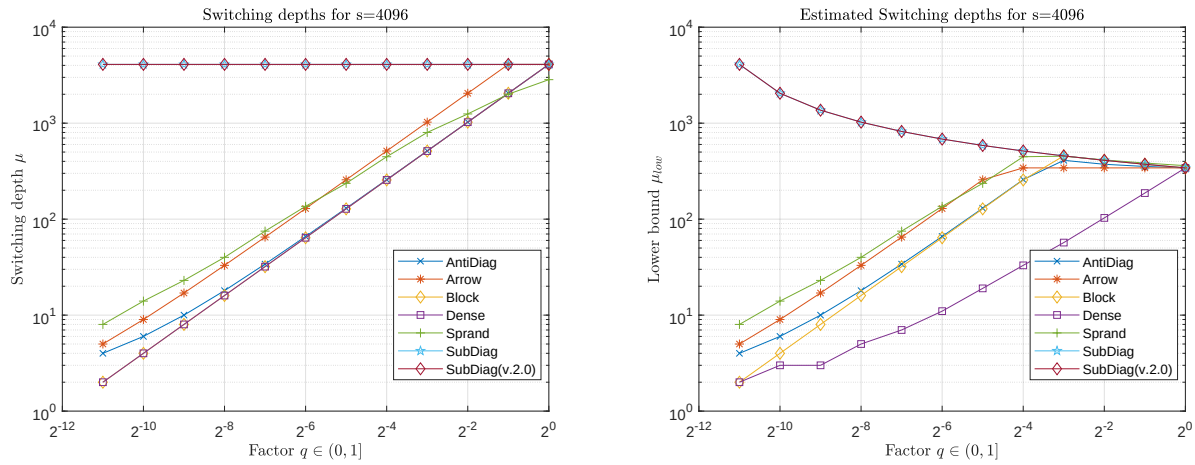


Figure 4: Switching depth μ for L_{AntiDiag}, L_{Arrow}, L_{Block}, L_{Dense}, L_{Sprand}, L_{SubDiag}, and $L_{\text{SubDiag(v.2.0)}}$ w.r.t. the factor q (left) and its estimated lower bound μ_{low} computed by `SLTS_PreTest` described in Algorithm 6 (right).

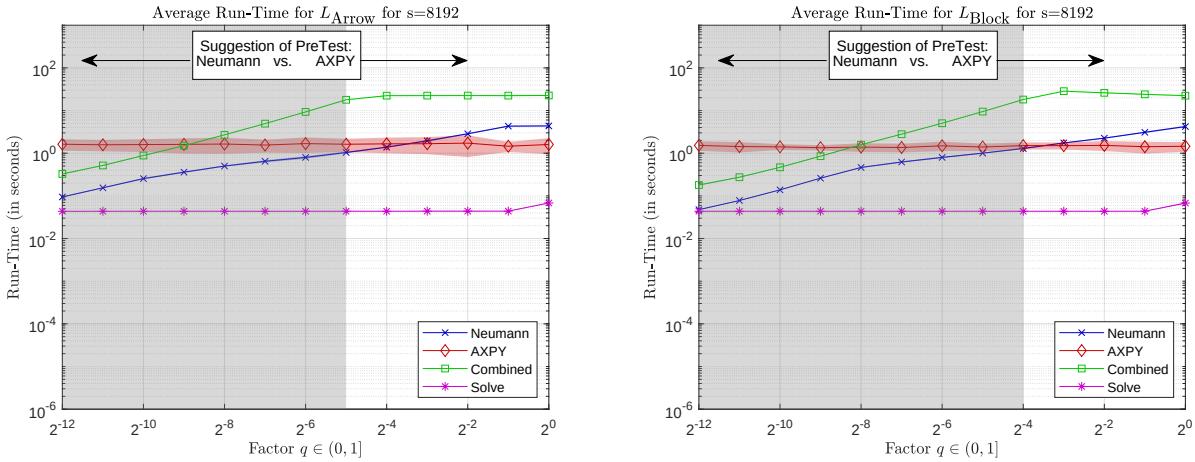


Figure 5: Run-time measurements (in seconds) to solve (SLTS) with Algorithms 3 and 5 for L_{Arrow} and L_{Block} for larger matrices with $s = 2^{13}$ compared to MATLAB's built-in triangular solver and the combined method.

5 Conclusions

We considered the efficient numerical solution of signed lower triangular systems. These systems are motivated as an important part in the evaluation of the algebraic ANF representation for piecewise affine functions, which arise as approximations of piecewise smooth functions. It is observed that the sequential dependency structure of these problems is challenging for modern parallel computer architectures. Therefore, Algorithm 5 is developed, which uses a Neumann series reformulation of the system. This algorithm benefits from implicit hidden structures and problem dependent characteristics like a small switching depth. The convergence and run-time bounds for the algorithm are proven by a graph theoretic model for the switching depth. The algorithm is numerically tested on several numerical examples representing typical use-cases and compared to a naive solver for the system based on a sequential approach using axpy operations. The results of the run-time experiments indicate that the proposed algorithm is capable of easily outperforming naive implementations by several orders of magnitude on state-of-the-art hardware for problems with large dimensions but small switching depth.

Acknowledgement

The experiments are prepared on resources of Friedrich Schiller University Jena supported by DFG grants INST 275/334–1 FUGG and INST 275/363–1 FUGG.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 3rd ed., 1999.

[2] T. Bosse, *(Almost) Matrix-free solver for piecewise linear functions in abs-normal form*, Numerical Linear Algebra with Applications, 26 (2019), p. e2258.

[3] T. Bosse and S. Narayanan, *Study of the numerical efficiency of structured abs-normal forms*, Optimization Methods & Software, (2019), pp. 1–25.

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, US, 3rd ed., 1996.

[5] A. Griewank, *On stable piecewise linearization and generalized algorithmic differentiation*, Optimization Methods and Software, 28 (2013), pp. 1139–1178.

[6] A. Griewank, J.-U. Bernt, M. Radons, and T. Streubel, *Solving piecewise linear systems in abs-normal form*, Linear Algebra and its Applications, 471 (2015), pp. 500–530.

[7] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Society for Industrial and Applied Mathematics, Philadelphia, PA, US, 2nd ed., 2008.

[8] B. Hendrickson and A. Pothen, *Combinatorial scientific computing: The enabling power of discrete algorithms in computational science*, in High Performance Computing for Computational Science – VECPAR 2006, M. Daydé, J. M. L. M. Palma, Á. L. G. A. Coutinho, E. Pacitti, and J. C. Lopes, eds., vol. 4395 of Lecture Notes in Computer Science, Berlin, Heidelberg, 2007, Springer Berlin Heidelberg, pp. 260–280.

[9] D. M. W. Leenaerts and W. M. V. Bokhoven, *Piecewise Linear Modeling and Analysis*, Kluwer Academic Publishers, Norwell, MA, US, 1998.

[10] The Mathworks, Inc., *MATLAB (R2020b)*, Natick, Massachusetts, 2020.

[11] U. Naumann and O. Schenk, eds., *Combinatorial Scientific Computing*, Computational Science Series, Chapman and Hall/CRC, Boca Raton, FL, USA, 2012.

[12] Netlib, *BLAS (Basic Linear Algebra Subprograms)*. online, http://www.netlib.org/blas, 2017.

[13] C. M. Radar, *Conceted components and minimum paths*, in Graph Algorithms in the Language of Linear Algebra, J. Kepner and J. Gilbert, eds., SIAM, Philadelphia, 2011, ch. 3, pp. 19–27.

[14] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, vol. 120 of Lecture Notes in Computer Science, Springer, Berlin, 1981.

[15] C. Sanderson and R. Curtin, *Armadillo: a template-based C++ library for linear algebra*, Journal of Open Source Software, 1(2) (2016).

[16] S. Scholtes, *Introduction to Piecewise Differentiable Equations*, SpringerBriefs in Optimization, Springer, Dordrecht, 2012.

[17] T. Streubel, A. Griewank, M. Radons, and J.-U. Bernt, *Representation and analysis of piecewise linear functions in abs-normal form*, in System Modeling and Optimization, C. Pötzsche, C. Heuberger, B. Kaltenbacher, and F. Rendl, eds., Berlin, Heidelberg, 2014, Springer Berlin Heidelberg, pp. 327–336.

[18] F. G. Van Zee and R. A. van de Geijn, *BLIS: A framework for rapidly instantiating BLAS functionality*, ACM Transactions on Mathematical Software, 41 (2015), pp. 14:1–14:33.

Multidimensional Included and Excluded Sums

Helen Xu* Sean Fraser* Charles E. Leiserson*

Abstract

This paper presents algorithms for the ***included-sums*** and ***excluded-sums*** problems used by scientific computing applications such as the fast multipole method. These problems are defined in terms of a d-dimensional array of N elements and a binary associative operator $\oplus$ on the elements. The included-sum problem requires that the elements within overlapping boxes cornered at each element within the array be reduced using $\oplus$. The excluded-sum problem reduces the elements outside each box. The ***weak*** versions of these problems assume that the operator $\oplus$ has an inverse $\ominus$, whereas the ***strong*** versions do not require this assumption. In addition to studying existing algorithms to solve these problems, we introduce three new algorithms.

The ***bidirectional box-sum (BDBS)*** algorithm solves the strong included-sums problem in $\Theta(dN)$ time, asymptotically beating the classical ***summed-area table (SAT)*** algorithm, which runs in $\Theta(2^d N)$ and which only solves the weak version of the problem. Empirically, the BDBS algorithm outperforms the SAT algorithm in higher dimensions by up to 17.1×.

The ***box-complement*** algorithm solves the strong excluded-sums problem in $\Theta(dN)$ time, asymptotically beating the state-of-the-art ***corners*** algorithm by Demaine *et al.*, which runs in $\Omega(2^d N)$ time. The box-complement algorithm empirically outperforms the corners algorithm by about 1.4× given similar amounts of space in three dimensions.

If the assumptions for the weak excluded-sums problem can be satisfied, the ***bidirectional box-sum complement (BDBSC)*** algorithm, which is a trivial extension of the BDBS algorithm, can beat box-complement by up to a factor of 4.

1 Introduction

Many scientific computing applications require reducing many (potentially overlapping) regions of a tensor, or multidimensional array, to a single value for each region quickly and accurately. For example, the integral-image problem (or summed-area table) [3, 6] preprocesses an image to answer queries for the sum of elements in arbitrary rectangular subregions of a matrix in constant time. The integral image has applications in real-time image processing and filtering [11]. The fast multipole method (FMM) is a widely used numerical approximation for the calculation of long-ranged forces in various N-particle simulations [1, 9]. The essence of the FMM is a reduction of a neighboring subregion's elements, excluding elements too close, using a multipole expansion to allow for fewer pairwise calculations [5, 7]. Specifically, the multipole-to-local expansion in the FMM adds relevant expansions outside some close neighborhood but inside some larger bounding region for each element [1, 14]. High-dimensional applications include the FMM for particle simulations in 3D space [4,10] and direct summation problems in higher dimensions [13].

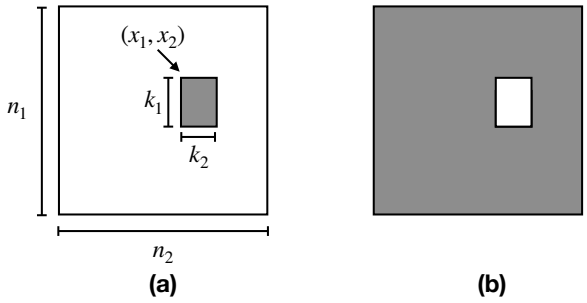


Figure 1: An illustration of included and excluded sums in 2 dimensions on an $n_1 \times n_2$ matrix using a (k_1, k_2)-box. **(a)** For a coordinate (x_1, x_2) of the matrix, the included-sums problem requires all the points in the $k_1 \times k_2$ box cornered at (x_1, x_2), shown as a grey rectangle, to be reduced using a binary associative operator $\oplus$. The included-sums problem requires that this reduction be performed at *every* coordinate of the matrix, not just at a single coordinate as is shown in the figure. **(b)** A similar illustration for excluded sums, which reduces the points outside the box.

These problems give rise to the excluded-sums problem [8], which underlies applications that require reducing regions of a tensor to a single value using a binary associative operator. For example, the excluded-sums problem corresponds to the translation of the local expansion coefficients within each box in the FMM [9]. The problems are called "sums" for ease of presentation, but the general problem statements (and therefore algorithms to solve the problems) apply to any context involving a ***monoid*** $(S, \oplus, e)$, where S is a set of values,

*Computer Science and Artificial Intelligence Laboratory. Massachusetts Institute of Technology. Email: {hjxu, sfraser, cel}@mit.edu.

$\oplus$ is a binary associative operator defined on S, and $e \in S$ is the identity for $\oplus$.

Although the excluded-sums problem is particularly challenging and meaningful for multidimensional tensors, let us start by considering the problem in only 2 dimensions. And, to understand the excluded-sums problem, it helps to understand the included-sums problem as well. Figure 1 illustrates included and excluded sums in 2 dimensions, and Figure 2 provides examples using addition as the $\oplus$ operator. We have an $n_1 \times n_2$ matrix $\mathcal{A}$ of elements over a monoid $(S, \oplus, e)$. We also are given a "box size" $\mathbf{k} = (k_1, k_2)$ such that $k_1 \leq n_1$ and $k_2 \leq n_2$. The ***included sum*** at a coordinate (x_1, x_2), as shown in Figure 1(a), involves ***reducing*** — accumulating using $\oplus$ — all the elements of $\mathcal{A}$ inside the $\mathbf{k}$-box ***cornered*** at (x_1, x_2), that is,

$$\bigoplus_{y_1=x_1}^{x_1+k_1-1} \bigoplus_{y_2=x_2}^{x_2+k_2-1} \mathcal{A}[y_1, y_2] \ ,$$

where if a coordinate goes out of range, we assume that its value is the identity e. The ***included-sums problem*** computes the included sum for all coordinates of $\mathcal{A}$, which can be straightforwardly accomplished with four nested loops in $\Theta(n_1 n_2 k_1 k_2)$ time. Similarly, the ***excluded sum*** at a coordinate, as shown in Figure 1(b), reduces all the elements of $\mathcal{A}$ outside the $\mathbf{k}$-box cornered at (x_1, x_2). The ***excluded-sums problem*** computes the excluded sum for all coordinates of $\mathcal{A}$, which can be straightforwardly accomplished in $\Theta(n_1 n_2 (n_1 - k_1)(n_2 - k_2))$ time. We shall see much better algorithms for both problems.

Excluded Sums and Operator Inverse One way to solve the excluded-sums problem is to solve the included-sums problem and then use the inverse $\ominus$ of the $\oplus$ operator to "subtract" out the results from the reduction of the entire tensor. This approach fails for operators without an inverse, however, such as the maximum operator max. As another example, the FMM involves solving the excluded-sums problem over a domain of functions which cannot be "subtracted," because the functions exhibit singularities [8]. Even for simpler domains, using the inverse (if it exists) may have unintended consequences. For example, subtracting finite-precision floating-point values can suffer from catastrophic cancellation [8, 17] and high round-off error [12]. Some contexts may permit the use of an inverse, but others may not.

Consequently, we refine the included- and excluded-sums problems into ***weak*** and ***strong*** versions. The weak version requires an operator inverse, while the strong version does not. Any algorithm for the included-sums problem trivially solves the weak excluded-sums problem, and any algorithm for the strong excluded-sums problem trivially solves the weak excluded-sums problem. This paper presents efficient algorithms for both the weak and strong excluded-sums problems.

$$\begin{pmatrix} 1 & 3 & 6 & 2 & 5 \\ 3 & 9 & 1 & 1 & 2 \\ 5 & 1 & 5 & 3 & 2 \\ 4 & 3 & 2 & 0 & 9 \\ 6 & 2 & 1 & 7 & 8 \end{pmatrix} \quad \begin{pmatrix} 16 & 19 & 10 & 10 & 7 \\ 18 & 16 & 10 & 8 & 4 \\ 13 & 11 & 10 & 14 & 11 \\ 15 & 8 & 10 & 24 & 17 \\ 8 & 3 & 8 & 15 & 8 \end{pmatrix} \quad \begin{pmatrix} 75 & 72 & 81 & 81 & 84 \\ 73 & 75 & 81 & 83 & 87 \\ 78 & 80 & 81 & 77 & 80 \\ 76 & 83 & 81 & 67 & 74 \\ 83 & 78 & 83 & 76 & 83 \end{pmatrix}$$

(a) (b) (c)

Figure 2: Examples of the included- and excluded-sums problems on an input matrix in 2 dimensions with box size $(3, 3)$ using the max operator. **(a)** The input matrix. The square shows the box cornered at $(3, 3)$. **(b)** The solution for the included-sums problem with the $+$ operator. The highlighted square contains the included sum for the box in (a). The included-sums problem requires computing the included sum for every element in the input matrix. **(c)** A similar example for excluded sums. The highlighted square contains the excluded sum for the box in (a).

Summed-area Table for Weak Excluded Sums The ***summed-area table (SAT)*** algorithm uses the classical summed-area table method [3, 6, 15] to solve the weak included-sums problem on a d-dimensional tensor $\mathcal{A}$ having N elements in $O(2^d N)$ time. The SAT algorithm cannot be used to solve the strong included-sums problem, however, because it requires an operator inverse. The summed-area table algorithm can easily be extended to an algorithm for weak excluded-sums by totaling the entire tensor and subtracting the solution to weak included sums. We will call this algorithm the ***SAT complement (SATC) algorithm***.

Corners Algorithm for Strong Excluded Sums The naive algorithm for strong excluded sums that just sums up the area of interest for each element runs in $O(N^2)$ time in the worst case, because it wastes work by recomputing reductions for overlapping regions. To avoid recomputing sums, Demaine *et al.* [8] introduced an algorithm that solve the strong excluded-sums problem in arbitrary dimensions, which we will call the ***corners algorithm***.

Given a d-dimensional tensor of N elements, the corners algorithm takes $\Omega(2^d N)$ time to compute the excluded sum in the best case because there are 2^d corners and each one requires $\Omega(N)$ time to add its contribution to each excluded box [16]. The bound is tight: given $\Theta(dN)$ space, the corners algorithm takes $\Theta(2^d N)$ time. With $\Theta(N)$ space, the corners algorithm takes $\Theta(d2^d N)$ time [16].

Algorithm	*Source*	*Time*	*Space*	*Included or Excluded?*	*Strong or Weak?*
Naive included sum	[This work]	$\Theta(KN)$	$\Theta(N)$	Included	Strong
Naive included sum complement	[This work]	$\Theta(KN)$	$\Theta(N)$	Excluded	Weak
Naive excluded sums	[This work]	$\Theta(N^2)$	$\Theta(N)$	Excluded	Strong
Summed-area table (SAT)	[6, 15]	$\Theta(2^d N)$	$\Theta(N)$	Included	Weak
Summed-area table complement (SATC)	[6, 15]	$\Theta(2^d N)$	$\Theta(N)$	Excluded	Weak
Corners(c)	[8]	$\Theta((d+1/c)2^d N)$	$\Theta(cN)$	Excluded	Strong
Corners spine	[8]	$\Theta(2^d N)$	$\Theta(dN)$	Excluded	Strong
Bidirectional box sum (BDBS)	[This work]	$\Theta(dN)$	$\Theta(N)$	Included	Strong
Bidirectional box sum complement (BDBSC)	[This work]	$\Theta(dN)$	$\Theta(N)$	Excluded	Weak
Box-complement	[This work]	$\Theta(dN)$	$\Theta(N)$	Excluded	Strong

Table 1: A summary of all algorithms for excluded sums in this paper. All algorithms take as input a d-dimensional tensor of N elements. We include the runtime, space usage, whether an algorithm solves the included- or excluded-sums problem, and whether it solves the strong or weak version of the problem. We use K to denote the volume of the box (in the runtime of the naive algorithm).

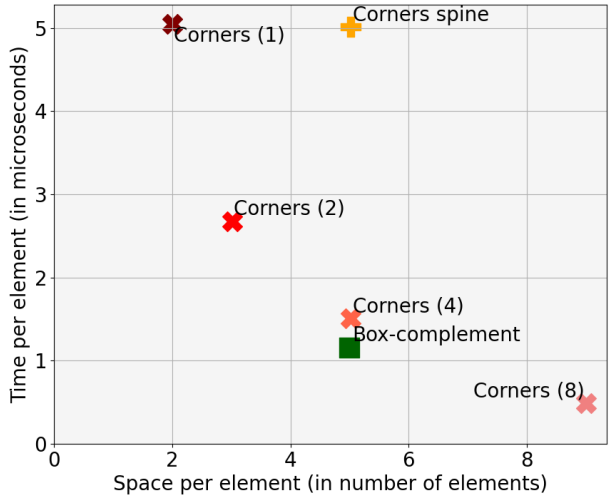


Figure 3: Space and time per element of the corners and box-complement algorithms in 3 dimensions. We use `Corners(c)` and `Corners Spine` to denote variants of the corners algorithm with extra space. We set the number of elements $N = 681472 = 88^3$ and the box lengths $k_1 = k_2 = k_3 = 4$ (for $K = 64$).

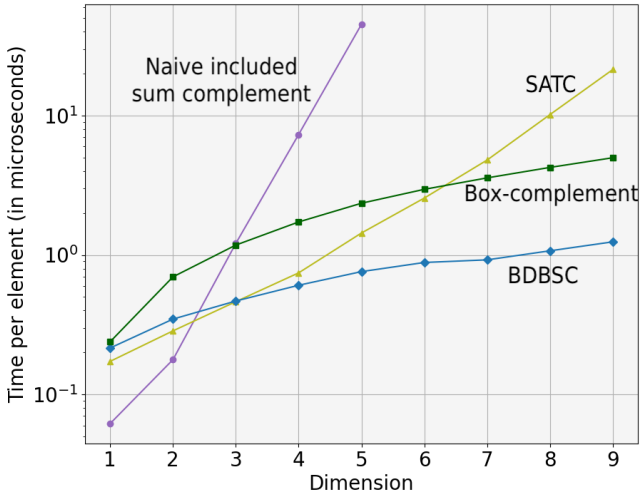


Figure 4: Time per element of algorithms for excluded sums in arbitrary dimensions. The number of elements N of the tensor in each dimension was in the range $[2097152, 134217728]$ (selected to be a exact power of the number of dimensions). For each number of dimensions d, we set the box volume $K = 8^d$.

Contributions This paper presents algorithms for included and strong excluded sums in arbitrary dimensions that improve the runtime from exponential to linear in the number of dimensions. For strong included sums, we introduce the ***bidirectional box-sum*** (BDBS) algorithm that uses prefix and suffix sums to compute the included sum efficiently. The BDBS algorithm can be easily extended into an algorithm for weak excluded sums, which we will call the ***bidirectional box-sum complement*** (BDBSC) algorithm. For strong excluded sums, the main insight in this paper is the formulation of the excluded sums in terms of the "box complement" on which the ***box-complement*** algorithm is based. Table 1 summarizes all algorithms considered in this paper.

Figure 3 illustrates the performance and space usage of the box-complement algorithm and variants of the 3D corners algorithm. Since the paper that introduced the corners algorithm stopped short of a general construction in higher dimensions, the 3D case is the highest dimensionality for which we have implementations of the box-complement and corners algorithm. The 3D case is of interest because applications such as the FMM often present in three dimensions [4, 10]. We find that the box-complement algorithm outperforms the corners algorithm by about 1.4× when given similar amounts of space, though the corners algorithm with twice the space as box-complement is 2× faster. The box-complement algorithm uses a fixed (constant) factor of extra space,

while the corners algorithm can use a variable amount of space. We found that the performance of the corners algorithm depends heavily on its space usage. We use `Corners(c)` to denote the implementation of the corners algorithm that uses a factor of c in space to store leaves in the computation tree and gather the results into the output. Furthermore, we also explored a variant of the corners algorithm [16] called `Corners spine`, which uses extra space to store the spine of the computation tree and asymptotically reduce the runtime.

Figure 4 demonstrates how algorithms for weak excluded sums scale with dimension. We omit the corners algorithm because the original paper stopped short of a construction of how to find the corners in higher dimensions. We also omit an evaluation of included-sums algorithms because the relative results of all algorithms would be the same. The naive and summed-area table perform well in lower dimensions but exhibit crossover points (at 3 and 6 dimensions, respectively) because their runtimes grow exponentially with dimension. In contrast, the BDBS and box-complement algorithms scale linearly in the number of dimensions and outperform the summed-area table method by at least 1.3× after 6 dimensions. The BDBS algorithm demonstrates the advantage of solving the weak problem, if you can, because it is always faster than the box-complement algorithm, which doesn't exploit an operator inverse. Both algorithms introduced in this paper outperform existing methods in higher dimensions, however.

To be specific, our contributions are as follows:

- the bidirectional box-sum (BDBS) algorithm for strong included sums;
- the bidirectional box-sum complement (BDBSC) algorithm for weak excluded sums;
- the box-complement algorithm for strong excluded sums;
- theorems showing that, for a d-dimensional tensor of size N, these algorithms all run in $\Theta(dN)$ time and $\Theta(N)$ space;
- implementations of these algorithms in `C++`; and
- empirical evaluations showing that the box-complement algorithm outperforms the corners algorithm in 3D given similar space and that both the BDBSC algorithm and box-complement algorithm outperform the SATC algorithm in higher dimensions.

Outline The rest of the paper is organized as follows. Section 2 provides necessary preliminaries and notation to understand the algorithms and proofs. Section 3 presents an efficient algorithm to solve the included-sums problem, which will be used as a key subroutine in the box-complement algorithm. Section 4 formulates the excluded sum as the "box-complement," and Section 5 describes and analyzes the resulting box-complement algorithm. Section 6 presents an empirical evaluation of algorithms for excluded sums. Finally, we provide concluding remarks in Section 7.

2 Preliminaries

This section reviews tensor preliminaries used to describe algorithms in later sections. It also formalizes the included- and excluded-sums problems in terms of tensor notation. Finally, it describes the prefix- and suffix-sums primitive underlying the main algorithms in this paper.

Tensor Preliminaries We first introduce the coordinate and tensor notation we use to explain our algorithms and why they work. At a high level, tensors are d-dimensional arrays of elements over some monoid $(S, \oplus, e)$. In this paper, tensors are represented by capital script letters (e.g., $\mathcal{A}$) and vectors are represented by lowercase boldface letters (e.g., $\mathbf{a}$).

We shall use the following terminology. A d-dimensional ***coordinate domain*** U is the cross product $U = U_1 \times U_2 \times \ldots \times U_d$, where $U_i = \{1, 2, \ldots, n_i\}$ for $n_i \geq 1$. The ***size*** of U is $n_1 n_2 \cdots n_d$. Given a coordinate domain U and a monoid $(S, \oplus, e)$ as defined in Section 1, a ***tensor*** $\mathcal{A}$ can be viewed for our purposes as a mapping $\mathcal{A} : U \to S$. That is, a tensor maps a coordinate $\mathbf{x} \in U$ to an ***element*** $\mathcal{A}[\mathbf{x}] \in S$. The ***size*** of a tensor is the size of its coordinate domain. We omit the coordinate domain U and monoid $(S, \oplus, e)$ when they are clear from context.

We use Python-like ***colon notation*** $x : x'$, where $x \leq x'$, to denote the half-open interval $[x, x')$ of coordinates along a particular dimension. If $x : x'$ would extend outside of $[1, n]$, where n is the maximum coordinate, it denotes only the coordinates actually in the interval, that is, the interval $\max\{1, x\} : \min\{n + 1, x'\}$. If the lower bound is missing, as in $: x'$, we interpret the interval as $1 : x'$, and similarly, if the upper bound is missing, as in $x :$, it denotes the interval $[x, n]$. If both bounds are missing, as in $:$, we interpret the interval as the whole coordinate range $[1, n]$.

We can use colon notation when indexing a tensor $\mathcal{A}$ to define ***subtensors***, or ***boxes***. For example, $\mathcal{A}[3 : 5, 4 : 6]$ denotes the elements of $\mathcal{A}$ at coordinates $(3, 4), (3, 5), (4, 4), (4, 5)$. For full generality, a box B ***cornered*** at coordinates $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_d)$, where $x_i < x'_i$ for all $i = 1, 2, \ldots, d$, is the box $(x_1 : x'_1, x_2 : x'_2, \ldots, x_d : x'_d)$. Given a ***box size*** $\mathbf{k} = (k_1, \ldots, k_d)$, a ***$\mathbf{k}$-box cornered*** at coordinate $\mathbf{x}$ is the box cornered at $\mathbf{x}$ and $\mathbf{x}' = (x_1 + k_1, x_2 + k_2, \ldots, x_d + k_d)$. A ***(tensor) row*** is a box with a single value in

each coordinate position in the colon notation, except for one position, which includes that entire dimension. For example, if $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is a coordinate of a tensor $\mathcal{A}$, then $\mathcal{A}[x_1, x_2, \ldots, x_{i-1}, :, x_{i+1}, x_{i+2}, \ldots, x_d]$ denotes a row along dimension i.

The colon notation can be combined with the reduction operator $\oplus$ to indicate the reduction of all elements in a subtensor:

$$\bigoplus \mathcal{A}[x_1 : x'_1, x_2 : x'_2, \ldots, x_d : x'_d] = \bigoplus_{y_1 \in [x_1, x'_1)} \bigoplus_{y_2 \in [x_2, x'_2)} \cdots \bigoplus_{y_d \in [x_d, x'_d)} \mathcal{A}[y_1, y_2, \ldots, y_d] \ .$$

Problem Definitions We can now formalize the included- and excluded-sums problems from Section 1.

Definition 1. (Included and Excluded Sums) *An algorithm for the **included-sums problem** takes as input a d-dimensional tensor $\mathcal{A} : U \to S$ with size N and a box size $\mathbf{k} = (k_1, k_2, \ldots, k_d)$. It produces a new tensor $\mathcal{A}' : U \to S$ such that every output element $\mathcal{A}'[\mathbf{x}]$ holds the reduction under $\oplus$ of elements within the $\mathbf{k}$-box of $\mathcal{A}$ cornered at $\mathbf{x}$. An algorithm for the **excluded-sums problem** is defined similarly, except that the reduction is of elements outside the $\mathbf{k}$-box cornered at $\mathbf{x}$.*

In other words, an included-sums algorithm computes, for all $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in U$, the value $\mathcal{A}'[\mathbf{x}] = \bigoplus \mathcal{A}[x_1 : x_1 + k_1, x_2 : x_2 + k_2, \ldots, x_d : x_d + k_d]$. It's messier to write the output of an excluded-sums problem using colon notation, but fortunately, our proofs do not rely on it.

As we have noted in Section 1, there are weak and strong versions of both problems which allow and do not allow an operator inverse, respectively.

Prefix and Suffix Sums The ***prefix-sums operation*** [2] takes an array $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ of n elements and returns the "running sum" $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, where

$$b_k = \begin{cases} a_1 & \text{if } k = 1, \\ a_k \oplus b_{k-1} & \text{if } k > 1 \ . \end{cases} \tag{2.1}$$

Let Prefix denote the algorithm that directly implements the recursion in Equation 2.1. Given an array $\mathbf{a}$ and indices $start \leq end$, the function Prefix($\mathbf{a}$, $start$, end) computes the prefix sum in the range $[start, end]$ of $\mathbf{a}$ in $O(end - start)$ time. Similarly, the ***suffix-sums operation*** is the reverse of the prefix sum and computes the sum right-to-left rather than left-to-right. Let Suffix($\mathbf{a}$, $start$, end) be the corresponding algorithm for suffix sums.

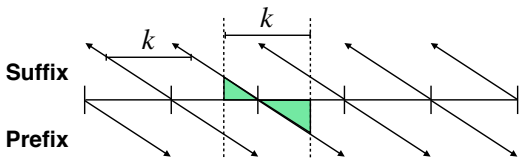


Figure 5: An illustration of the computation in the bidirectional box-sum algorithm. The arrows represent prefix and suffix sums in runs of size k, and the shaded region represents the prefix and suffix components of the region of size k outlined by the dotted lines.

3 Included Sums

This section presents the ***bidirectional box-sum algorithm (BDBS) algorithm*** to compute the included sum along an arbitrary dimension, which is used as a main subroutine in the box-complement algorithm for excluded sums. As a warm-up, we will first describe how to solve the included-sums problem in one dimension and then extend the technique to higher dimensions. We include the one-dimensional case for clarity, but the main focus of this paper is the multidimensional case. The full version of the paper includes all the pseudocode and omitted proofs for BDBS in 1D [16].

Included Sums in 1D Before investigating the included sums in higher dimensions, let us first turn our attention to the 1D case for ease of understanding. We will sketch an algorithm BDBS-1D which takes as input a list A of length N and a (scalar) box size[1] k and outputs a list A' of corresponding included sums. At a high level, the BDBS-1D algorithm generates two intermediate lists A_p and A_s, each of length N, and performs N/k prefix and suffix sums of length k on each intermediate list. By construction, for $x = 1, 2, \ldots, N$, we have $A_p[x] = A[k\lfloor x/k \rfloor : x + 1]$, and $A_s[x] = A[x : k\lceil (x+1)/k \rceil]$.

Finally, BDBS-1D uses A_p and A_s to compute the included sum of size k for each coordinate in one pass. Figure 5 illustrates the ranged prefix and suffix sums in BDBS-1D, and Figure 6 presents a concrete example of the computation.

BDBS-1D solves the included-sums problem on an array of size N in $\Theta(N)$ time and $\Theta(N)$ space. First, it uses two temporary arrays to compute the prefix and suffix as illustrated in Figure 5 in $\Theta(N)$ time. It then makes one more pass through the data to compute the included sum, requiring $\Theta(N)$ time.

[1] For simplicity in the algorithm descriptions and pseudocode, we assume that $n_i \bmod k_i = 0$ for all dimensions $i = 1, 2, \ldots, d$. In implementations, the input can either be padded with the identity to make this assumption hold, or it can add in extra code to deal with unaligned boxes.

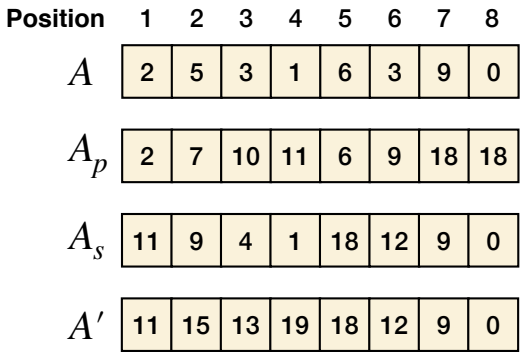


Figure 6: An example of computing the 1D included sum using the bidirectional box-sum algorithm, where $N = 8$ and $k = 4$. The input array is A, the k-wise prefix and suffix sums are stored in A_p and A_s, respectively, and the output is in A'.

Generalizing to Arbitrary Dimensions The main focus of this work is multidimensional included and excluded sums. Computing the included sum along an arbitrary dimension is almost exactly the same as computing it along 1 dimension in terms of the underlying ranged prefix and suffix sums. We sketch an algorithm BDBS that generalizes BDBS-1D to arbitrary dimensions.

Let $\mathcal{A}$ be a d-dimensional tensor with N elements and let $\mathbf{k}$ be a box size. The BDBS algorithm computes the included sum along dimensions $i = 1, 2, \ldots, d$ in turn. After performing the included-sum computation along dimensions $1, 2, \ldots, i$, every coordinate in the output $\mathcal{A}_i$ contains the included sum in each dimension up to i:

$$\mathcal{A}_i[x_1, x_2, \ldots, x_d] = \bigoplus \mathcal{A}[\underbrace{x_1 : x_1 + k_2, \ldots, x_i : x_i + k_i}_{i}, \underbrace{x_{i+1}, \ldots, x_d}_{d-i}].$$

Overall, BDBS computes the full included sum of a tensor with N elements in $\Theta(dN)$ time and $\Theta(N)$ space by performing the included sum along each dimension in turn.

Although we cannot directly use BDBS to solve the strong excluded-sums problem, the next sections demonstrate how to use the BDBS technique as a key subroutine in the box-complement algorithm for strong excluded sums.

4 Excluded Sums and the Box Complement

The main insight in this section is the formulation of the excluded sum as the recursive "box complement". We show how to partition the excluded region into $2d$ non-overlapping parts in d dimensions. This decomposition of the excluded region underlies the box-complement for strong excluded sums in the next section.

First, let's see how the formulation of the "box complement" relates to the excluded sum. At a high level, given a box B, a coordinate $\mathbf{x}$ is in the "i-complement" of B if and only if $\mathbf{x}$ is "out of range" in some dimension $j \leq i$, and "in the range" for all dimensions greater than i.

Definition 2. (Box Complement) *Given a d-dimensional coordinate domain U and a dimension $i \in \{1, 2, \ldots, d\}$, the **i-complement** of a box B cornered at coordinates $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{x}' = (x'_1, \ldots, x'_d)$ is the set*

$$C_i(B) = \{(y_1, \ldots, y_d) \in U : \text{ there exists } j \in [1, i] \text{ such that } y_j \notin [x_j, x'_j), \text{ and for all } m \in [i+1, d], y_m \in [x_m, x'_m)\}.$$

Given a box B, the reduction of all elements at coordinates in $C_d(B)$ is exactly the excluded sum with respect to B. The box complement recursively partitions an excluded region into disjoint sets of coordinates.

Theorem 4.1. (Recursive Box-complement) *Let B be a box cornered at coordinates $\mathbf{x} = (x_1, \ldots, x_d)$ and $\mathbf{x}' = (x'_1, \ldots, x'_d)$ in some coordinate domain U. The i-complement of B can be expressed recursively in terms of the $(i-1)$-complement of B as follows:*

$$C_i(B) = (\underbrace{:, \ldots, :}_{i-1}, : x_i, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}) \cup (\underbrace{:, \ldots, :}_{i-1}, x'_i :, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}) \cup C_{i-1}(B),$$

where $C_0(B) = \emptyset$.

Proof. For simplicity of notation, let $\text{RHS}_i(B)$ be the right-hand side of the equation in the statement of Theorem 4.1. Let $\mathbf{y} = (y_1, \ldots, y_d)$ be a coordinate. In order to show the equality, we will show that $\mathbf{y} \in C_i(B)$ if and only if $\mathbf{y} \in \text{RHS}_i(B)$.

Forward Direction: $\mathbf{y} \in C_i(B) \rightarrow \mathbf{y} \in \text{RHS}_i(B)$.
We proceed by case analysis when $\mathbf{y} \in C_i(B)$. Let $j \leq i$ be the highest dimension at which $\mathbf{y}$ is "out of range," or where $y_j < x_j$ or $y_j \geq x'_j$.

Case 1: $j = i$.
Definition 2 and $j = i$ imply that either $y_i < x_i$ or $y_i \geq x'_i$, and $x_m \leq y_m \leq x'_m$ for all $m > i$. By definition, $y_i < x_i$ implies $\mathbf{y} \in (:, \ldots, :, : x_i, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d)$. Similarly, $y_i \geq x'_i$ implies $\mathbf{y} \in (:, \ldots, :, x'_i :, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d)$. These are exactly the first two terms in $\text{RHS}_i(B)$.

Case 2: $j < i$.
Definition 2 and $j < i$ imply that $\mathbf{y} \in C_{i-1}(B)$.

Backwards Direction: $\mathbf{y} \in \mathrm{RHS}_i(B) \to \mathbf{y} \in C_i(B)$. We again proceed by case analysis.

Case 1: $\mathbf{y} \in \big(:, \ldots, :, : x_i, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d\big)$ or $\mathbf{y} \in \big(:, \ldots, :, x'_i :, x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d\big)$.
Definition 2 implies $\mathbf{y} \in C_i(B)$ because there exists some $j \le i$ (in this case, $j = i$) such that $y_j < x_j$ and $x_m \le y_m < x'_m$ for all $m > i$.

Case 2: $\mathbf{y} \in C_{i-1}(B)$.
Definition 2 implies that there exists j in the range $1 \le j \le i-1$ such that $y_j < x_j$ or $y_j \ge x'_j$ and that for all $m \ge i$, we have $x_m \le y_m < x'_m$. Therefore, $\mathbf{y} \in C_{i-1}(B)$ implies $\mathbf{y} \in C_i(B)$ since there exists some $j \le i$ (in this case, $j < i$) where $y_j < x_j$ or $y_j \ge x'_j$ and $x_m \le y_m < x'_m$ for all $m > 1$.

Therefore, $C_i(B)$ can be recursively expressed as $\mathrm{RHS}_i(B)$. ∎

In general, unrolling the recursion in Theorem 4.1 yields $2d$ disjoint partitions that exactly comprise the excluded sum with respect to a box.

COROLLARY 4.1. (EXCLUDED-SUM COMPONENTS) *The excluded sum can be represented as the union of $2d$ disjoint sets of coordinates as follows:*

$$C_d(B) = \bigcup_{i=1}^{d} \Bigg((\underbrace{:, \ldots, :}_{i-1}, : x_i, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}) \cup (\underbrace{:, \ldots, :}_{i-1}, x_i + k_i :, \underbrace{x_{i+1} : x'_{i+1}, \ldots, x_d : x'_d}_{d-i}) \Bigg) .$$

We use the box-complement formulation in the next section to efficiently compute the excluded sums on a tensor by reducing in disjoint regions of the tensor.

5 Box-Complement Algorithm

This section describes and analyzes the box-complement algorithm for strong excluded sums, which efficiently implements the dimension reduction in Section 4. The box-complement algorithm relies heavily on prefix, suffix, and included sums as described in Sections 2 and 3.

Given a d-dimensional tensor $\mathcal{A}$ of size N and a box size $\mathbf{k}$, the box-complement algorithm solves the excluded-sums problem with respect to $\mathbf{k}$ for coordinates in $\mathcal{A}$ in $\Theta(dN)$ time and $\Theta(N)$ space. The full version of the paper contains all omitted pseudocode and proofs for the box-complement algorithm [16].

Algorithm Sketch At a high level, the box-complement algorithm proceeds by dimension reduction. That is, the algorithm takes d dimension-reduction steps, where each step adds two of the components from Corollary 4.1 to each element in the output tensor. In the ith dimension-reduction step, the box-complement algorithm computes the i-complement of B (Definition 2) for all coordinates in the tensor by performing a prefix and suffix sum along the ith dimension and then using the BDBS technique along the remaining $d-i$ dimensions. After the ith dimension-reduction step, the box-complement algorithm operates on a tensor of $d-i$ dimensions because i dimensions have been reduced so far via prefix sums. Figure 7 presents an example of the dimension reduction in 2 dimensions, and Figure 8 illustrates the recursive box-complement in 3 dimensions.

Prefix and Suffix Sums In the ith dimension reduction step, the box-complement algorithm uses prefix and suffix sums along the ith dimension to reduce the elements "out of range" along the ith dimension in the i-complement. That is, given a tensor $\mathcal{A}$ of size $N = n_1 \cdot n_2 \cdots n_d$ and a number $i < d$ of dimensions reduced so far, we define a subroutine PREFIX-ALONG-DIM$(\mathcal{A}, i)$ that fixes the first i dimensions at $n_1, \ldots, n_i$ (respectively), and then computes the prefix sum along dimension $i+1$ for all remaining rows in dimensions $i+2, \ldots, d$ in $O\Big(\prod_{j=i+1}^{d} n_j\Big)$ time.

The subroutine PREFIX-ALONG-DIM computes the reduction of elements "out of range" along dimension i. That is, after PREFIX-ALONG-DIM$(\mathcal{A}, i)$, for each coordinate $x_{i+1} = 1, 2, \ldots, n_{i+1}$ along dimension $i+1$, every coordinate in the (dimension-reduced) output $\mathcal{A}'$ contains the prefix up to that coordinate in dimension $i+1$:

$$\mathcal{A}'[\underbrace{n_1, \ldots, n_i}_{i}, x_{i+1}, \underbrace{x_{i+2}, \ldots, x_d}_{d-i-1}] = \bigoplus \mathcal{A}[\underbrace{n_1, \ldots, n_i}_{i}, : x_{i+1} + 1, \underbrace{x_{i+2}, \ldots, x_d}_{d-i-1}].$$

Since the similar subroutine SUFFIX-ALONG-DIM has almost exactly the same analysis and structure, we omit its discussion.

Included Sums In the ith dimension reduction step, the box-complement algorithm uses the BDBS technique along the ith dimension to reduce the elements "in range" along the ith dimension in the i-complement. That is, given a tensor $\mathcal{A}$ of size $N = n_1 \cdot n_2 \cdots n_d$ and a number $i < d$ of dimensions reduced so far, we define a subroutine BDBS-ALONG-DIM.

BDBS-ALONG-DIM computes the included sum for each row along a specified dimension after dimension reduction. Let $\mathcal{A}$ be a d-dimensional tensor, $\mathbf{k}$ be a box size, i be the number of reduced dimensions so far,

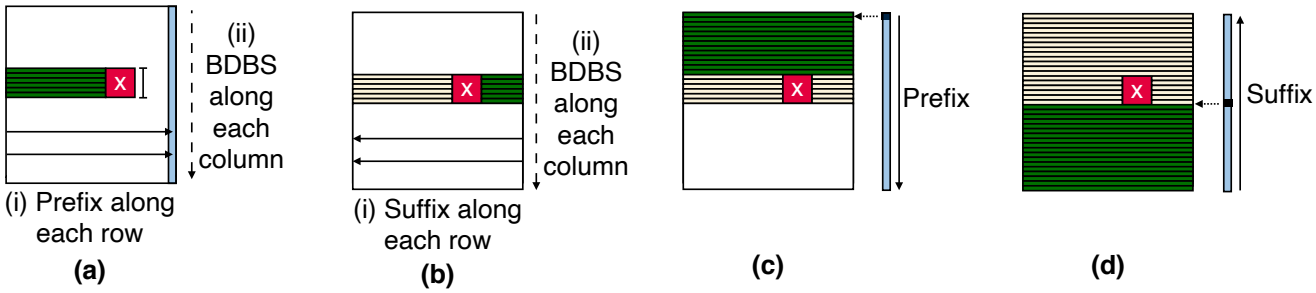


Figure 7: Steps for computing the excluded sum in 2 dimensions with included sums on prefix and suffix sums. The steps are labeled in the order they are computed. The 1-complement **(a)** prefix and **(b)** suffix steps perform a prefix and suffix along dimension 1 and an included sum along dimension 2. The numbers in **(a)**,**(b)** represent the order of subroutines in those steps. The 2-complement **(c)** prefix and **(d)** suffix steps perform a prefix and suffix sum on the reduced array, denoted by the blue rectangle, from step **(a)**. The red box denotes the excluded region, and solid lines with arrows denote prefix or suffix sums along a row or column. The long dashed line represents the included sum along each column.

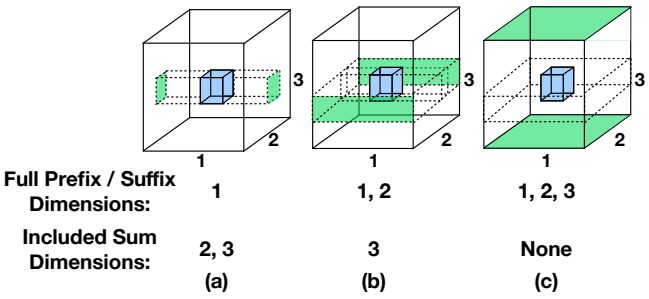


Figure 8: An example of the recursive box-complement in 3 dimensions with dimensions labeled $1, 2, 3$. The subfigures **(a)**, **(b)**, and **(c)** illustrate the 1-, 2-, and 3-complement, respectively. The blue region represents the coordinates inside the box, and the regions outlined by dotted lines represent the partitions defined by Corollary 4.1. For each partition, the face against the edge of the tensor is highlighted in green.

and j be the dimension to compute the included sum along such that $j > i$. BDBS-Along-Dim$(\mathcal{A}, \mathbf{k}, i, j)$ computes the included sum along the jth dimension for all rows $(n_1, \ldots, n_i, :, \ldots, :)$. That is, for each coordinate $\mathbf{x} = (n_1, \ldots, n_i, x_{i+1}, \ldots, x_d)$, the output tensor $\mathcal{A}'$ contains the included sum along dimension j:

$$\mathcal{A}'[\mathbf{x}] = \bigoplus \mathcal{A}[\underbrace{n_1, \ldots, n_i}_{i}, \underbrace{x_{i+1}, \ldots, x_j}_{j-i}, x_{j+1} : x_{j+1} + k_{j+1}, \underbrace{x_{j+2}, \ldots, x_d}_{d-j-1}].$$

BDBS-Along-Dim$(\mathcal{A}, \mathbf{k}, i, j)$ takes $\Theta\left(\prod_{\ell=i+1}^{d} n_\ell\right)$ time because it iterates over $\left(\prod_{\ell=i+1}^{d} n_\ell\right)/n_{j+1}$ rows and runs in $\Theta(n_{j+1})$ time per row. It takes $\Theta(N)$ space using the same technique as BDBS-1D.

Adding in the Contribution Each dimension-reduction step must add its respective contribution to each element in the output. Given an input tensor $\mathcal{A}$ and output tensor $\mathcal{A}'$, both of size N, the function Add-Contribution takes $\Theta(N)$ time to add in the contribution with a pass through the tensors.

Putting It All Together Finally, we will see how to use the previously defined subroutines to describe and analyze the box-complement algorithm for excluded sums. Figure 9 presents pseudocode for the box-complement algorithm. Each dimension-reduction step has a corresponding prefix and suffix step to add in the two components in the recursive box-complement. Given an input tensor $\mathcal{A}$ of size N, the box-complement algorithm takes $\Theta(N)$ space because all of its subroutines use at most a constant number of temporaries of size N, as seen in Figure 9.

Given a tensor $\mathcal{A}$ as input, the box-complement algorithm solves the excluded-sums problem by computing the recursive box-complement components from Corollary 4.1. By construction, for dimension $i \in [1, d]$, the prefix-sum part of the ith dimension-reduction step outputs a tensor $\mathcal{A}_p$ such that for all coordinates $\mathbf{x} = (x_1, \ldots, x_d)$, we have

$$\mathcal{A}_p[x_1, \ldots, x_d] = \bigoplus \mathcal{A}[\underbrace{:, \ldots, :}_{i}, : x_{i+1}, \underbrace{x_{i+2} : x_{i+2} + k_{i+2}, \ldots, x_d : x_d + k_d}_{d-i-1}].$$

Similarly, the suffix-sum step constructs a tensor $\mathcal{A}_s$ such that for all $\mathbf{x}$,

$$\mathcal{A}_s[x_1, \ldots, x_d] = \bigoplus \mathcal{A}[\underbrace{:, \ldots, :}_{i}, x_{i+1} + k_{i+1} :, \underbrace{x_{i+2} : x_{i+2} + k_{i+2}, \ldots, x_d : x_d + k_d}_{d-i-1}].$$

We can now analyze the performance of the box-complement algorithm.

THEOREM 5.1. (TIME OF BOX-COMPLEMENT) *Given a d-dimensional tensor $\mathcal{A}$ of size $N = n_1 \cdot n_2 \cdot \ldots \cdot n_d$,* BOX-COMPLEMENT *solves the excluded-sums problem in $\Theta(dN)$ time.*

Proof. We analyze the prefix step (since the suffix step is symmetric, it has the same running time). Let $i \in \{1, \ldots, d\}$ denote a dimension.

The ith dimension reduction step in BOX-COMPLEMENT involves 1 prefix step and $(d-i)$ included sum calls, which each take $O\left(\prod_{j=i}^{d} n_j\right)$ time. Furthermore, adding in the contribution at each dimension-reduction step takes $\Theta(N)$ time. The total time over d steps is therefore $\Theta\left(\sum_{i=1}^{d}\left((d-i+1)\prod_{j=i}^{d} n_j + N\right)\right)$. Adding in the contribution is clearly $\Theta(dN)$ in total.

Next, we bound the runtime of the prefix and included sums. In each dimension-reduction step, reducing the number of dimensions of interest exponentially decreases the size of the considered tensor. That is, dimension reduction exponentially reduces the size of the input: $\prod_{j=i}^{d} n_j \leq N/2^{i-1}$. The total time required to compute the box-complement components is therefore

$$\sum_{i=1}^{d}(d-i+1)\prod_{j=i}^{d} n_j \leq \sum_{i=1}^{d}(d-i+1)\frac{N}{2^{i-1}} \leq 2(d+2^{-d}-1)N = \Theta(dN).$$

Therefore, the total time of BOX-COMPLEMENT is $\Theta(dN)$. ∎

6 Experimental Evaluation

This section presents an empirical evaluation of strong and weak excluded-sums algorithms. In 3 dimensions, we compare strong excluded-sums algorithms: specifically, we evaluate the box-complement algorithm and variants of the corners algorithm and find that the box-complement outperforms the corners algorithm given similar space. Furthermore, we compare weak excluded-sums algorithms in higher dimensions. Lastly, to simulate a more expensive operator than numeric addition when reducing, we introduce an artificial slowdown in the binary associative operator in 3D. The full version of the paper includes details of the experimental setup and additional results [16].

Strong Excluded Sums in 3D Figure 3 summarizes the results of our evaluation of the box-complement and corners algorithm in 3 dimensions with a box length of $k_1 = k_2 = k_3 = 4$ (for a total box volume of $K = 64$) and number of elements $N = 681472 = 88^3$. We tested with

BOX-COMPLEMENT($\mathcal{A}$, $\mathbf{k}$)

```
1  // Input: Tensor A with d-dimensions, box size k
   // Output: Tensor A' with size and dimensions
   // matching A containing the excluded sum.
2  init A' with the same size as A
3  A_p ← A; A_s ← A
4  // Current dimension-reduction step
5  for i ← 1 to d
6  // Saved from previous dimension-reduction step.
7      A_p ← A reduced up to dimension i − 1
8      A_s ← A_p    // Save input to suffix step
9      // PREFIX STEP
       // Reduced up to i dimensions.
10     PREFIX-ALONG-DIM along
11       dimension i on A_p.
12     A ← A_p    // Save for next round
13     // Do included sum on dimensions [i + 1, d].
14     for j ← i + 1 to d
15         // A_p reduced up to i dimensions
16         BDBS-ALONG-DIM on
17           dimension j in A_p
18     // Add into result
19     ADD-CONTRIBUTION from A_p into A'
20
21     // SUFFIX STEP
       // Do suffix sum along dimension i
22     SUFFIX-ALONG-DIM along
23       dimension i in A_s
24     // Do included sum on dimensions [i + 1, d]
25     for j ← i + 1 to d
26         // A_s reduced up to i dimensions
27         BDBS-ALONG-DIM on
28           dimension j in A_s
29     // Add into result
30     ADD-CONTRIBUTION from A_s into A'
31 return A'
```

Figure 9: Pseudocode for the box-complement algorithm. For ease of presentation, we omit the exact parameters to the subroutines and describe their function in the algorithm.

varying N but found that the time and space per element were flat. We found that the box-complement algorithm outperforms the corners algorithm by about 1.4× when given similar amounts of space, though the corners algorithm with 2× the space as the box-complement algorithm was 2× faster.

We explored two different methods of using extra space in the corners algorithm based on the computation tree of prefixes and suffixes: (1) storing the spine of the computation tree to asymptotically reduce the running time, and (2) storing the leaves of the computation tree to reduce passes through the output. Although storing the

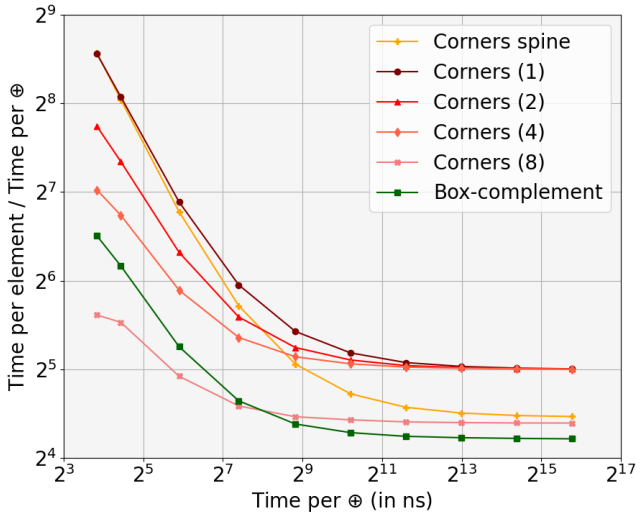


Figure 10: The scalability of excluded-sum algorithms as a function of the cost of operator $\oplus$ on a 3D domain of $N = 4096$ elements. The horizontal axis is the time in nanoseconds to execute $\oplus$. The vertical axis represents the time per element of the given algorithm divided by the time for $\oplus$. We inflated the time of $\oplus$ using increasingly large arguments to the standard recursive implementation of a Fibonacci computation.

leaves does not asymptotically affect the behavior of the corners algorithm, we found that reducing the number of passes through the output has significant effects on empirical performance. Storing the spine did not improve performance, because the runtime is dominated by the number of passes through the output.

Excluded Sums With Different Operators Most of our experiments used numeric addition for the $\oplus$ operator. Because some applications, such as FMM, involve much more costly $\oplus$ operators, we studied how the excluded-sum algorithms scale with the cost of $\oplus$. To do so, we added a tunable slowdown to the invocation of $\oplus$ in the algorithms. Specifically, they call an unoptimized implementation of the standard recursive Fibonacci computation. By varying the argument to the Fibonacci function, we can simulate $\oplus$ operators that take different amounts of time.

Figure 10 summarizes our findings. For inexpensive $\oplus$ operators, the box-complement algorithm is the second fastest, but as the cost of $\oplus$ increases, the box-complement algorithm dominates. The reason for this outcome is that box-complement performs approximately 12 $\oplus$ operations per element in 3D, whereas the most efficient corners algorithm performs about 22 $\oplus$ operations. As $\oplus$ becomes more costly, the time spent executing $\oplus$ dominates the other bookkeeping overhead.

Weak Excluded Sums in Higher Dimensions Figure 4 presents the results of our evaluation of weak excluded-sum algorithms in higher dimensions. For all dimensions $i = 1, 2, \ldots, d$, we set the box length $k_i = 8$ and chose a number of elements N to be a perfect power of the dimension i. Table 1 presents the asymptotic runtime of the different excluded-sum algorithms.

The weak naive algorithm for excluded sums with nested loops outperforms all of the other algorithms up to 2 dimensions because its runtime is dependent on the box volume, which is low in smaller dimensions. Since its runtime grows exponentially with the box length, however, we limited it to 5 dimensions.

The summed-area table algorithm outperforms the BDBS and box-complement algorithms up to 6 dimensions, but its runtime scales exponentially in d.

Finally, the BDBS and box-complement algorithms scale linearly in the number of dimensions and outperform both naive and summed-area table methods in higher dimensions. Specifically, the box-complement algorithm outperforms the summed-area table algorithm by between 1.3× and 4× after 6 dimensions. The BDBS algorithm demonstrates an advantage to having an inverse: it outperforms the box-complement algorithm by 1.1× to 4×. Therefore, the BDBS algorithm dominates the box-complement algorithm for weak excluded sums.

7 Conclusion

In this paper, we introduced the box-complement algorithm for the excluded-sums problem, which improves the running time of the state-of-the-art corners algorithm from $\Omega(2^d N)$ to $\Theta(dN)$ time. The space usage of the box-complement algorithm is independent of the number of dimensions, while the corners algorithm may use space dependent on the number of dimensions to achieve its running-time lower bound.

Acknowledgments

The idea behind the box-complement algorithm was originally conceived jointly with Erik Demaine many years ago, and we thank him for helpful discussions. Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] Rick Beatson and Leslie Greengard. A short course on fast multipole methods. *Wavelets, Multilevel Methods and Elliptic PDEs*, pages 1–37, 1997.

[2] Guy E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November 1990.

[3] Derek Bradley and Gerhard Roth. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, 12(2):13–21, 2007.

[4] Hongwei Cheng, William Y. Crutchfield, Zydrunas Gimbutas, Leslie F. Greengard, J. Frank Ethridge, Jingfang Huang, Vladimir Rokhlin, Norman Yarvin, and Junsheng Zhao. A wideband fast multipole method for the Helmholtz equation in three dimensions. *Journal of Computational Physics*, 216(1):300–325, 2006.

[5] Ronald Coifman, Vladimir Rokhlin, and Stephen Wandzura. The fast multipole method for the wave equation: A pedestrian prescription. *IEEE Antennas and Propagation Magazine*, 35(3):7–12, 1993.

[6] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, page 207–212, New York, NY, USA, 1984. Association for Computing Machinery.

[7] Eric Darve. The fast multipole method: numerical implementation. *Journal of Computational Physics*, 160(1):195–240, 2000.

[8] E. D. Demaine, M. L. Demaine, A. Edelman, C. E. Leiserson, and P. Persson. Building blocks and excluded sums. *SIAM News*, 38(4):1–5, 2005.

[9] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 135(2):280–292, August 1997.

[10] Nail A. Gumerov and Ramani Duraiswami. *Fast multipole methods for the Helmholtz equation in three dimensions*. Elsevier, 2005.

[11] Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo Lastra. Fast summed-area table generation and its applications. In *Computer Graphics Forum*, volume 24, pages 547–555. Wiley Online Library, 2005.

[12] Nicholas J. Higham. The accuracy of floating point summation. *SIAM J. Scientific Computing*, 14:783–799, 1993.

[13] William B March and George Biros. Far-field compression for fast kernel summation methods in high dimensions. *Applied and Computational Harmonic Analysis*, 43(1):39–75, 2017.

[14] Xiaobai Sun and Nikos P Pitsianis. A matrix version of the fast multipole method. *Siam Review*, 43(2):289–300, 2001.

[15] Ernesto Tapia. A note on the computation of high-dimensional integral images. *Pattern Recognition Letters*, 32(2):197–201, 2011.

[16] Helen Xu, Sean Fraser, and Charles E. Leiserson. Multidimensional included and excluded sums, 2021. https://arxiv.org/abs/2106.00124.

[17] Gernot Ziegler. Summed area computation using ripmap of partial sums, 2012. GPU Technology Conference (talk).

Efficient Parallel Sparse Symmetric Tucker Decomposition for High-Order Tensors

Shruti Shivakumar*, Jiajia Li†, Ramakrishnan Kannan‡, Srinivas Aluru§

Abstract

Tensor based methods are receiving renewed attention in recent years due to their prevalence in diverse real-world applications. There is considerable literature on tensor representations and algorithms for tensor decompositions, both for dense and sparse tensors. Many applications in hypergraph analytics, machine learning, psychometry, and signal processing result in tensors that are both sparse and symmetric, making it an important class for further study. Similar to the critical Tensor Times Matrix chain operation (TTMc) in general sparse tensors, the Sparse Symmetric Tensor Times Same Matrix chain (S^3TTMc) operation is compute and memory intensive due to high tensor order and the associated factorial explosion in the number of non-zeros. In this work, we present a novel compressed storage format CSS for sparse symmetric tensors, along with an efficient parallel algorithm for the S^3TTMc operation. We theoretically establish that S^3TTMc on CSS achieves a better memory versus run-time trade-off compared to state-of-the-art implementations. We demonstrate experimental findings that confirm these results and achieve up to 2.9× speedup on synthetic and real datasets.

1 Introduction

Tensors are higher dimension generalizations of matrices, and are used to represent multi-dimensional data. Symmetric tensors are an important class of tensors, arising in diverse fields such as psychometry, signal processing, machine learning, and hypergraph analytics [42, 31, 2, 14, 12, 18]. Estimating means of Gaussian graphical models and independent component analysis often utilize symmetric tensors for decompositions [4, 15, 2]. Hypergraphs, generalizations of graphs which allow edges to span multiple vertices, have become ubiquitous in understanding real world networks and multi-entity interactions [10, 11]. Affinity relations in a hypergraph can be represented as a high-order adjacency tensor which is sparse and symmetric [13, 29, 2, 44]. While mathematical research on symmetric tensors is longstanding [30, 20, 7, 9], emerging massive data in these applications has sparked the demand for scalable, efficient algorithms that utilize advances in numerical linear algebra, numerical optimization, as well as high performance computing. State-of-the-art tensor libraries [35, 21, 40] incorporate high performance tensor methods for general sparse tensors; however, to the best of our knowledge, they lack specialized algorithms for sparse tensors that are symmetric.

A representative tensor approach in dimensionality reduction and hypergraph clustering is tensor decomposition, and the popular Tucker decomposition [19] is studied in this work. A key kernel in symmetric Tucker decomposition is the *Sparse Symmetric Tensor Times Same Matrix chain* (S^3TTMc) operation[1], a specialization of the fundamental tensor times matrix chain operation (TTMc) for symmetric tensors. In symmetric Tucker decomposition, instead of different matrix factors in the chain, the same matrix is repeatedly used.

In high-performance algorithms for sparse tensor decomposition, a balance between two desirable but competing goals is pursued - (i) compressed storage format to represent the sparse tensor, and (ii) efficient computation based on it to perform the decomposition. For general sparse tensors, the two goals could align well [35, 22, 24, 27] since when a sparse tensor is stored compactly, the memory footprint of the tensor - which could be a big portion of the memory traffic during computation - is reduced. Moreover, superior sparse tensor compression can co-exist with better data locality. For example, a compact Compressed Sparse Row (CSR) representation of a sparse matrix demonstrates better data locality from the reuse of row indices. A similar phenomenon occurs in Compressed Sparse Fibers (CSF)

*Georgia Institute of Technology, Atlanta, GA, USA

†Pacific Northwest National Laboratory, Richland, WA, USA; William & Mary, Williamsburg, VA, USA. This research was also partially funded by the US Department of Energy under Award No. 66150 and the Laboratory Directed Research and Development program at PNNL under contract No. ND8577.

‡Oak Ridge National Laboratory, Oak Ridge, TN, USA. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (`http://energy.gov/downloads/doe-public-access-plan`).

§Georgia Institute of Technology, Atlanta, GA, USA

[1]For notational convenience, we call S^3TTMc in lieu of SSTTSMc.

[35] and Hierarchical COOrdinate (HiCOO) [22] formats from different perspectives.

For sparse symmetric tensors, efficient computation and compressed storage format are *contending characteristics*, especially for high dimensional data. Fig. 1 illustrates the trade-off between efficiently computing S^{3}TTMc and the size of compressed storage data structures for symmetric tensors. The *Unique COOrdinate* (*UCOO*) representation achieves compact storage by saving the coordinates (indices) along with the value of each non-zero exactly once. However, the compression comes at the price of slow S^{3}TTMc computation, as retrieving all index permutations of every non-zero increases redundant computation and limits parallelism. On the other hand, by saving all index permutations of each non-zero, we can overcome the computation challenges in *UCOO*. Thus, S^{3}TTMc becomes a general TTMc where state-of-the-art high performance libraries, such as SPLATT [35] and ParTI! [21], can be leveraged. Unfortunately, this approach leads to $\mathcal{O}(N!)$ increase in storage, where N is the tensor order. With the pervasiveness of high order adjacency tensors representing massive hypergraphs, the memory overhead is formidable. For example, an order-14 sparse symmetric tensor generated from Amazon reviews [26] (Details in Sec. 6.2) causes $14! = 8.7 \times 10^{10}$ factor increase in storage to save all index permutations. Thus, one is forced to choose between efficient algorithms that use storage formats that cannot be sustained for high-order tensors (*SPLATT*), and efficient storage formats (*UCOO*) that do not permit efficient algorithms.

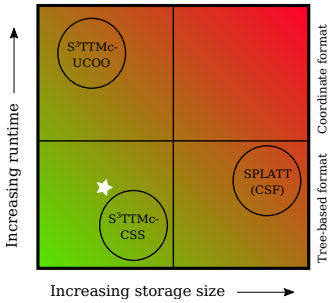


Figure 1: Comparison of trade-off between run-time and storage size for proposed S^{3}TTMc-CSS computation to different algorithms and storage formats.

In this paper, we propose a *computation-aware compact storage format*, termed the Compressed Sparse Symmetric (CSS) format, which has storage requirement of the same order as *UCOO* while being able to perform S^{3}TTMc more efficiently than state-of-the-art TTMc implementations, such as SPLATT, as shown in Fig. 1. CSS is especially suited for high-order tensors since its size is bounded by 2^N, asymptotically smaller than $N!$. We present a novel shared-memory parallel algorithm S^{3}TTMc-CSS to efficiently perform S^{3}TTMc using CSS. We demonstrate through experiments that the performance of our parallel algorithm concurs with Fig. 1.

Our main contributions are summarized as follows:

- We propose the novel computation-aware CSS structure for sparse symmetric tensors. (Sec. 4)
- We design and implement an efficient multi-core parallel S^{3}TTMc algorithm, called S^{3}TTMc-CSS based on the CSS format. (Sec. 5)
- We perform a thorough cost analysis of S^{3}TTMc-CSS and outline a detailed comparison to baseline implementations and their formats. (Secs. 4.2 and 5.5)
- Our S^{3}TTMc-CSS bridges the gap between efficient S^{3}TTMc computation and compact storage - S^{3}TTMc-CSS is (i) up to 2.9× faster than SPLATT while requiring up to five orders of magnitude lesser memory, and (ii) at least two orders of magnitude faster than *S^{3}TTMc-UCOO* while only requiring up to 6× memory than *UCOO*. (Sec. 6)
- We report, for the first time, the Tucker decomposition of an order-14 tensor from a real world hypergraph. (Sec. 6.6)

2 Background

2.1 Sparse Symmetric Tensors An order-N symmetric tensor $\mathcal{X} \in \mathbb{R}^{I \times I \times \ldots I}$ can be generated from an N-uniform hypergraph on I vertices, where every hyperedge spanning multiple vertices has the same cardinality N. The tensor order N is the number of modes or dimensions. The non-zero values of the symmetric tensor $\mathcal{X}$ remain unchanged under any permutation of its indices. That is, for every $e = \{v_{i_1}, v_{i_2}, \ldots, v_{i_N}\} \in E$, $\mathcal{X}_{\sigma(\mathbf{i})} = w(e)$, where $\sigma(\mathbf{i})$ denotes any of the $N!$ permutations of the index tuple $\mathbf{i} = (i_1, \ldots, i_N)$. We refer to a non-zero $\mathcal{X}_{\mathbf{i}} \in unz(\mathcal{X})$ as an *index-ordered unique (IOU) non-zero* if the index set $\mathbf{i}$ is ordered $(i_1 < i_2 < \ldots < i_N)$. Thus, $nz(\mathcal{X})$ is divided into a set of equivalence classes, $unz(\mathcal{X})$ of size $unnz \ll nnz$, under the permutation relation σ. The IOU non-zero $\mathcal{X}_{\mathbf{i}}$ represents an equivalence class containing all permutations $\sigma(\mathbf{i})$ of the index set $\mathbf{i}$. A subtensor of a tensor $\mathcal{X}$ is obtained by fixing indices along some dimensions while varying along the others, denoted by ':'. The matricization of $\mathcal{X}$ along any mode n $(1 \leq n \leq N)$ flattens/unfolds the tensor into a matrix $\mathbf{X}_{(n)} \in \mathbb{R}^{I \times I^{N-1}}$. Following Kolda and Bader [19], we denote vectors using bold lowercase letter (e.g., $\mathbf{i}$, $\mathbf{j}$), matrices using bold uppercase letters (e.g., $\mathbf{U}$), and tensors using bold calligraphic letters (e.g., $\mathcal{X}$).

2.2 Sparse Tensor Formats Multiple compressed data structures, with varying levels of compactness, have been proposed for general sparse tensors. Popular among these include COO [19], CSF [35], MM-CSF [27],

F-COO [24], and HiCOO [23] formats. The simplest storage format for sparse tensors is the COOrdinate (COO) format. COO stores each non-zero as a tuple $(i_1, i_2, \ldots i_N; v)$, where $i_n, n = 1, \ldots, N$ is an index coordinate and v is the non-zero value. The Compressed Sparse Fiber (CSF) format is a higher order analogue to the Compressed Sparse Row (CSR) sparse matrix format. It is a compressed storage format structured as a tree, where every level corresponds to a tensor mode. Every non-zero in a sparse tensor is represented by a root to leaf path.

Algorithm 1: Symmetric HOOI

Result: Core tensor $\mathcal{C}$, and orthonormal matrix $\mathbf{U}$

1 **while** $\mathcal{C}$ *not converged* **do**
2 $\quad \mathcal{Y} = \mathcal{X} \times_{-1} [\mathbf{U}]$ // S^{3}TTMc
3 $\quad \mathbf{U} \leftarrow R$ left singular vectors of matricized $\mathcal{Y}$
4 $\quad \mathcal{C} = \mathcal{Y} \times_1 \mathbf{U}$

2.3 Sparse Symmetric Tucker Decomposition

Symmetric Tucker decomposition of an order-N sparse symmetric tensor $\mathcal{X} \in \mathbb{R}^{I \times I \ldots I}$ finds a dense orthonormal matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ and an order-N dense symmetric core tensor $\mathcal{C} \in \mathbb{R}^{R \times R \ldots R}$ such that $\arg\min_{\mathcal{C},\mathbf{U}} ||\mathcal{X} - \mathcal{C} \times_1 \mathbf{U} \times_2 \mathbf{U} \ldots \times_N \mathbf{U}||$. In hypergraph clustering and dimensionality reduction, the matrix rank R corresponds to number of clusters and size of reduced space respectively, and is usually a small value. We show the symmetric higher order iterations (HOOI) algorithm (Algorithm 1) by introducing the symmetry property into the popular HOOI approach to Tucker decomposition [19]. The operation in Line 2, *Sparse Symmetric Tensor Times Same Matrix chain* (S^{3}TTMc), is observed to be computationally expensive in large data, and will be our focus.

$$\mathcal{Y} = \mathcal{X} \times_{-1} [\mathbf{U}] = \mathcal{X} \times_2 \mathbf{U} \times_3 \mathbf{U} \ldots \times_N \mathbf{U} \tag{2.1}$$

$$\implies \mathbf{Y}(i_1, :) = \sum_{\mathcal{X}_\mathbf{i} \in nz(\mathcal{X})} \mathcal{X}_\mathbf{i} \left\{ \bigotimes_{j=2}^{N} \mathbf{U}(i_j, :) \right\} \tag{2.2}$$

$$= \sum_{\mathcal{X}_{i_1,:,\ldots}} \mathbf{U}(i_2, :) \otimes \ldots \left(\sum_{\mathcal{X}_{\ldots i_{N-1},:}} \mathcal{X}_{\ldots i_{N-1},:} \mathbf{U}(i_N, :) \right) \tag{2.3}$$

S^{3}TTMc, given by Eq. (2.1), is represented by a sequence of tensor times matrix products (TTM) [34] on all but one mode of the symmetric tensor $\mathcal{X}$. TTM of an order-N symmetric tensor $\mathcal{X} \in \mathbb{R}^{I \times I \ldots I}$ with a dense matrix $\mathbf{U} \in \mathbb{R}^{I \times R}$ along mode n, denoted by $\mathcal{Y} = \mathcal{X} \times_n \mathbf{U}$, is defined for $r \in \{1, \ldots, R\}$ as $\mathcal{Y}_{i_1 \ldots i_{n-1} r i_{n+1} \ldots i_N} = \sum_{i_n=1}^{I} \mathcal{X}_{i_1 \ldots i_{n-1} i_n i_{n+1} \ldots i_N} \mathbf{U}_{i_n r}$.

The Kronecker product of vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$, denoted by $\mathbf{u} \otimes \mathbf{v} \in \mathbb{R}^{mn}$, is the vectorized outer product [19]. The sparsity of $\mathcal{X}$ favors rewriting S^{3}TTMc using Kronecker products in Eq. (2.2), similar to the approach used in the work [16, 32], by only doing meaningful computation corresponding to its non-zeros (with index permutations), thus obtaining the matricized output, $\mathbf{Y}$. Eq. (2.3) also utilizes the distributivity of Kronecker products to reuse intermediate results across multiple non-zeros, named *non-zero memoization*.

2.4 Other Related Work

Sparse Tensor Decompositions State-of-the-art sparse tensor CANDECOMP/PARAFAC (CP) decomposition [36, 22, 27] and Tucker decomposition [25, 34, 32] research targets general sparse tensors. Kaya *et al.* proposed a dimension tree data structure that partitions mode indices in a hierarchical manner [17] to efficiently utilize intermediate results among TTM operations, so-called *operation memoization*, thus reducing the cost of CP decomposition. The *operation memoization* [17], orthogonal with our *nonzero memoization*, could be utilized to further enhance S^{3}TTMc performance in the future.

Dense Symmetric Tensor Decompositions There are optimized approaches designed for dense symmetric tensor methods [8]. Ballard *et al.* [5] designed a symmetric storage format for dense tensor eigenvalue algorithms on GPUs. Schatz *et al.* [28] proposed a blocked compact symmetric storage to reduce computation redundancy by utilizing temporary tensors. Solomonik *et al.* proposed symmetric tensor contractions with fewer multiplications [37] and proved lower bounds for communication [38]. Cai *et al.* [6] explored symmetry to reduce redundancy in computation and storage in INDSCAL (individual differences in scaling) decomposition. To the best of our knowledge, the work presented in this paper is the first to effectively utilize symmetry in sparse tensors for space- and performance-efficiency.

3 Baseline Implementations

We use three existing implementations based on two general sparse tensor formats as our baselines – *UCOO*, the 'unique COO' format that stores only IOU non-zeros, and *FCSF*, the 'full CSF' format which stores all index permutations. Two S^{3}TTMc implementations using *UCOO* are *S^{3}TTMc-UCOO* and Cyclops Tensor Framework (CTF) [40, 39]; we use *SPLATT* [33] to evaluate S^{3}TTMc implementation using *FCSF*.

$$\mathbf{Y}(ind, :) = \sum_{\substack{\mathcal{X}_\mathbf{i} \in unz(\mathcal{X}) \\ ind \in \mathbf{i}}} \mathcal{X}_\mathbf{i} \sum_{\mathbf{j} = \sigma(\mathbf{i} \setminus ind)} \left\{ \bigotimes_{k=2}^{N} \mathbf{U}(j_k, :) \right\} \tag{3.4}$$

3.1 *S^{3}TTMc-UCOO and CTF* Among the state-of-the-art formats, COO directly supports storing only IOU non-zeros of a sparse symmetric tensor because of its invariance with mode order [23]. Note that

in Eq. (2.2), an IOU non-zero $\mathcal{X}_{i_1,\dots i_N}$ contributes $(N-1)!$ terms to the summation for each of the rows $\mathbf{Y}(i_1,:), \mathbf{Y}(i_2,:), \dots, \mathbf{Y}(i_N,:)$, i.e., every permutation $(j_1,\dots j_N) = \sigma(i_1, i_2, \dots i_N)$ contributes a term $(\mathbf{U}(j_2,:) \otimes \dots \otimes \mathbf{U}(j_N,:))$ to the summation in Eq. (2.2) for $\mathbf{Y}(j_1,:)$. We implement a parallel *S³TTMc-UCOO* baseline by grouping terms in Eq. (2.2) and accessing only the IOU non-zeros $unz(\mathcal{X})$ for the summation, as illustrated in Eq. (3.4). However, computing S³TTMc on *UCOO* is inefficient since updating the rows of $\mathbf{Y}$ in parallel raises memory contention, thus requiring atomic operations as all IOU non-zeros whose indices contain i_n contribute to $\mathbf{Y}(i_n,:)$. We evaluate the performance of our implementation of Eq. (3.4) and the existing Cyclops Tensor Framework (CTF) [40], both using the *UCOO* format, in computing S³TTMc in *S³TTMc-UCOO* and *CTF* baselines respectively. A disadvantage of both baselines is that they cannot perform *non-zero memoization*, i.e. memoizing intermediate permutation results among IOU non-zeros, without maintaining additional information on index distribution.

3.2 *SPLATT using FCSF format* The CSF format is more storage-efficient than COO but suffers from enforcing mode ordering. If we only store IOU non-zeros in a CSF representation, in order to update $\mathbf{Y}(i_n,:)$ for a given IOU non-zero $\mathcal{X}_{\mathbf{i}}$, traversing the path corresponding to $\mathcal{X}_{\mathbf{i}}$ $\mathcal{O}(N!)$ times is unavoidable. This is because CSF only considers non-zero memoization among IOU non-zeros, but not within the permutations of an IOU non-zero. Thus, by assuming a machine with sufficiently large memory, we store all index permutations of IOU non-zeros in CSF format and use the state-of-the-art TTMc algorithm for general sparse tensors - SPLATT [35] - as our third baseline to compute S³TTMc.

4 CSS Structure

Our objective is to design a data structure for sparse symmetric tensors which reduces memory consumption and avoids redundant saving of symmetric information, overcoming one of the key challenges affecting the *FCSF* format. Moreover, the format must enable two types of non-zero memoization for intermediate Kronecker product results in the S³TTMc operation - (i) between IOU non-zeros, stored in a tree structure similar to the CSF format, and (ii) within permutations of an IOU non-zero, uniquely designed for symmetry in this work. We trade off memory requirement for efficient S³TTMc computation, thereby outperforming the baselines using *UCOO* representation.

Inspired by the CSF format, we design a tree structure that stores only IOU non-zeros. We name our tree data structure as the *Compressed Sparse Symmetric* (CSS) format. Consider a symmetric tensor $\mathcal{X} \in \mathbb{R}^{I\times I\times\dots I}$ of order N. CSS is structured as a forest with $N-1$ levels constructed from IOU non-zeros, $unz(\mathcal{X})$. We denote the path down the CSS tree from level 1 to node i_l at level l by $\mathcal{P}_{\mathbf{i}_l} = (i_1, i_2, \dots i_l)$. Then, $\mathcal{P}_{\mathbf{i}_l}$ corresponds to an ordered subsequence of size l contained in at least one ordered index tuple $\mathbf{j}$, i.e. $\mathcal{P}_{\mathbf{i}_l} = (j_{k_1}, j_{k_2}, \dots j_{k_l}) \subseteq \mathbf{j}$ while maintaining the order $k_1 < \dots < k_l$. Storing all ordered subsequences of IOU non-zeros highlights the computation-aware property of the CSS format - both forms of non-zero memoization are achieved by our S³TTMc operation with minimal additional index information. In the example shown in Fig. 2, the CSS format constructed from the 6 non-zeros of the order-4 tensor in Table 1 has 3 levels. Nodes $(2,3,5)$ and $(2,8)$ form two paths of the non-zero $(2,3,5,8)$. Due to the lexicographical order, the path $(2,8)$ has to stop at the maximum index 8, thus its length is only 2. Each leaf node at level $(N-1)$ stores an array containing the left-out index $i_N = \mathbf{i} \backslash \mathcal{P}_{\mathbf{i}_{N-1}}$ and the non-zero value (shown in blue in Fig. 2 and Table 1). In the example, the array $[8, 1.0]$ is affiliated to node-5 in path $\mathcal{P}_{\mathbf{5}} = (2,3,5)$. The leaf nodes in levels higher than $N-1$ are essential to efficiently computing S³TTMc (Details in Sec. 5).

Note that the paths $\mathcal{P}_{\mathbf{i}_l}$ correspond to l-length subsequences of index tuples, while $\mathbf{i}$ refer to the index tuples themselves. Thus, each IOU non-zero corresponds to multiple paths $\mathcal{P}_{\mathbf{i}_l}$ to levels $l = 1, 2, \dots N-1$. In Fig. 2, we mark the nodes on paths constructed for the first non-zero $(2,3,5,8)$ in red. There are six paths of length 1 from level 1 to level 2, and four paths of length 2 from level 1 to level 3; thus fourteen paths in total of lengths 0, 1 and 2. Generally, each IOU non-zero contributes $\binom{N}{l}$ nodes to level l and totally (2^N-1) nodes to the CSS representation. Across multiple non-zeros, their paths/nodes could be reused, e.g., path $(2,3)$ belongs to both $(2,3,5,8)$ and $(1,2,3,9)$ non-zeros. Thus, the total number of nodes in CSS tree depends on the IOU non-zero distribution. We empirically observe that for a uniform distribution of IOU non-zeros in the tensor, the number of nodes in CSS is much smaller than $(2^N-1)unnz$ due to the overlap of nodes between non-zeros. Such a construction not only provides a compact representation of a symmetric tensor, but also ensures uniqueness in representation. We present two key properties of the CSS data structure that are useful in the construction of the S³TTMc-CSS algorithm below.

PROPERTY 4.1. *Any path in the CSS forest exists only if it is a subsequence of the index tuple of at least one IOU non-zero. Moreover, the ordering of the node values in every path further ensures unique representation of permutations of subsets of all IOU non-zeros.*

Due to Property 4.1, there are exactly $N = \binom{N}{N-1}$

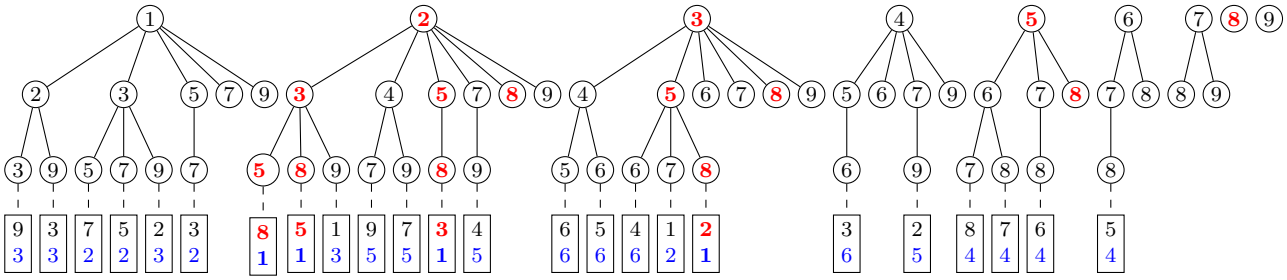


Figure 2: CSS format for the sparse symmetric tensor in Table 1

i_1	**2**	1	1	5	2	3
i_2	**3**	3	2	6	4	4
i_3	**5**	5	3	7	7	5
i_4	**8**	7	9	8	9	6
vals	**1**	2	3	4	5	6

Table 1: A sparse symmetric tensor $\mathcal{X} \in \mathbb{R}^{10\times10\times10\times10}$ with 6 IOU nonzeros.

distinct paths to the leaf nodes at level $(N-1)$ for any IOU non-zero. In the example in Fig. 2, four paths repeatedly save the index tuple $(2,3,5,8)$ at level-3 with values.

PROPERTY 4.2. *There exists an injective mapping between a set of IOU non-zeros of an order-N sparse symmetric tensor and the CSS forest constructed from it.*

While the CSS format still has some redundancy in terms of storing subsequences of index tuples of IOU non-zeros, it is a necessity for efficient S^{3}TTMc computation (discussed in Sec. 5). Moreover, CSS is practical, as it is less expensive than the state-of-the-art general sparse formats (compared below).

4.1 Space Complexity For a single IOU non-zero, the number of nodes at level l is $\binom{N}{l}$ for CSS. We need to save both pointers and indices for every level, which doubles the storage of CSS. The affiliated leaf level indices and non-zero values are stored in each of the N nodes at level $(N-1)$ generated by this non-zero. Thus, the space requirements for storing one IOU non-zero unz, and for storing $unnz$ non-zeros are given by Eq. (4.5) and Eq. (4.6) respectively.

$$S^{CSS}_{unz} = 2\beta_{int}\sum_{l=1}^{N-1}\binom{N}{l} + N(\beta_{int}+\beta_{dbl}) \tag{4.5}$$

$$S^{CSS}_{total} = 2\beta_{int}\sum_{l=1}^{N-1} nnz_l + unnz\cdot N(\beta_{int}+\beta_{dbl}) \tag{4.6}$$

where nnz_l is the number of nodes at level l in the CSS structure. The overlap in the nodes across IOU non-zeros is inherent in the nnz_l parameter, thus $nnz_l < \binom{N}{l}\cdot unnz$. However, as the CSS structure has a combinatorial construction procedure, the sizes of the levels follow the trend of the binomial coefficient, $nnz_1 < nnz_2 \ldots < nnz_{N/2} > \ldots > nnz_{N-1}$. We use a binary search tree for inserting nodes in CSS in order to maintain the ordering of nodes while traversing depth-wise and level-wise through the CSS tree. Thus, the time complexity of CSS construction is $\mathcal{O}\big(unnz \log I\big(2^N - N\big)\big)$.

4.2 Comparison to baselines

UCOO: Storing IOU non-zeros in *UCOO* is more compact than CSS. However, as we will demonstrate in Sec. 6, S^{3}TTMc using the CSS format is more work efficient than using *UCOO*.

$$S^{UCOO}_{total} = (N\beta_{int} + \beta_{dbl})unnz \tag{4.7}$$

FCSF: Though CSS and *FCSF* both use the tree structure to explore overlap between non-zeros, unlike CSS, *FCSF* does not recognize the symmetry feature of a tensor, and has $N!\cdot unnz$ root to leaf paths. We give the space analysis for *FCSF* by referring to the work [34]. For a single IOU non-zero unz, each *FCSF* level has $\binom{N}{l}l!$ nodes for all permutations, consuming a factorial order more space than CSS (Eq. (4.8)). The storage overhead of $nnz = unnz\cdot N!$ non-zeros in $\mathcal{X}$ is given by Eq. (4.9).

$$S^{FCSF}_{unz} = 2\beta_{int}\sum_{l=1}^{N-1}\binom{N}{l}l! + N!\,(\beta_{int}+\beta_{dbl}) \tag{4.8}$$

$$S^{FCSF}_{total} = 2\beta_{int}\sum_{l=1}^{N-1} nnz'_l + N!\,unnz(\beta_{int}+\beta_{dbl}) \tag{4.9}$$

where nnz'_l is the number of nodes at level l in the CSF format with $nnz'_1 < nnz'_2 < \ldots < nnz'_{N-1}$. Though the last term in Eq. (4.6) is much smaller than Eq. (4.9), it is still difficult to directly compare them due to the significant overlapping among nodes reflected in nnz_l and nnz'_l respectively. We will experimentally validate our analysis so far by demonstrating the space advantages of CSS on synthetic and real data in Sec. 6.

5 S^{3}TTMc Computation

In this section, we propose an algorithm to compute S^{3}TTMc for sparse symmetric tensors stored in CSS format. We seek to remove the arithmetic redundancies in the implementation of Eq. (3.4) for *S^{3}TTMc-UCOO* baseline, and demonstrate how CSS inherently supports a memoization strategy that achieves this goal.

5.1 Formulation We recursively define vector function $\mathcal{K}$ which allows for further factorization of the Kronecker products.

$$\mathcal{K}(\mathbf{i}) = \begin{cases} \mathbf{U}(i,:) & |\mathbf{i}| = 1 \\ \sum_{j \in i_1 \dots i_l} \mathbf{U}(j,:) \otimes \mathcal{K}(\mathbf{i} \setminus j) & |\mathbf{i}| = l > 1 \end{cases} \tag{5.10}$$

If we unroll the recursion in Eq. (5.10), note that $\mathcal{K}(\mathbf{i})$ is the sum of Kronecker products of rows of $\mathbf{U}$ for every permutation $\mathbf{j} = \sigma(\mathbf{i})$ (Eq. (5.11)). Then, Eq. (3.4) can be succinctly factorized using Eq. (5.10), as seen in Eq. (5.12).

$$\mathcal{K}(\mathbf{i}) = \sum_{\mathbf{j}=\sigma(\mathbf{i})} \otimes_k \mathbf{U}(j_k,:) \tag{5.11}$$

$$\mathbf{Y}(ind,:) = \sum_{\substack{\mathcal{X}_\mathbf{i} \in unz(\mathcal{X}) \\ ind \in \mathbf{i}}} \mathcal{X}_\mathbf{i} \, \mathcal{K}(\mathbf{i} \setminus ind) \tag{5.12}$$

The computation-aware property of the CSS format enables us to then use our new formulation to achieve non-zero memoization, hence sharing intermediate $\mathcal{K}$ vectors across multiple IOU non-zeros and within permutations of a given IOU non-zero. We will explore parallel approaches to computing Eq. (5.12) in the remainder of this section.

5.2 Naive Approach Consider node i_l in the CSS data structure at level l with path $\mathcal{P}_{\mathbf{i}_l} = (i_1, i_2, \dots, i_l)$. Let node $\mathcal{P}_{\mathbf{i}_l}(l)$ store $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$. From Eq. (5.10), we note that $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ depends only on $\mathcal{K}(\mathcal{P}'_k), \mathcal{P}'_k = \mathcal{P}_{\mathbf{i}_l} \setminus i_k, 1 \leq k \leq l$. Thus, in order to compute $\mathcal{K}$ at node i_l, we need results from nodes at level $l-1$ that correspond to the last node in paths $\mathcal{P}'$.

A naive implementation of S^{3}TTMc performs a level-wise traversal of CSS to compute the $\mathcal{K}$ for nodes at level l. It memoizes $\mathcal{K}(\mathcal{P})$ at all nodes in level $l-1$, and thus simply retrieves $\mathcal{K}(\mathcal{P}'_k), 1 \leq k \leq l$ to compute $\mathcal{K}(\mathcal{P}_l)$. The memory required for memoization has an upper bound determined by the level l ($1 \leq l < N-2$) with the largest $nnz_l R^l + nnz_{l+1} R^{l+1}$. However, this approach runs into memory bottlenecks because (i) the memory overhead of memoizing becomes large with increasing tensor order, and (ii) while computing $\mathcal{K}(\mathcal{P})$ for nodes at level l, the nodes at level $l-1$ that contribute to the Kronecker product of nodes at level l are not consecutive in memory, and this step becomes quite expensive due to the random accesses in a large $\mathcal{K}(\mathcal{P})$ array. Moreover, the naive approach disregards the ordering of nodes in CSS, due to which it overestimates the memory requirement for memoization. We implement an efficient memoization strategy by only storing the minimum number of intermediate $\mathcal{K}(\mathcal{P})$ vectors needed to perform S^{3}TTMc on the CSS tree while still maintaining computational efficiency, as can be seen from lines 1-4 in the S^{3}TTMc-CSS algorithm (Algorithm 2).

5.3 Optimizations

Memoization overhead We seek to reduce the memory blowup due to non-zero memoization in the naive algorithm. Consider the set of paths $SP(\mathcal{P}_{\mathbf{i}_l}) = \{\mathcal{P}'_k : \mathcal{P}'_k = \mathcal{P}_{\mathbf{i}_l} \setminus i_k, 1 \leq k \leq l\}$ for some path $\mathcal{P}_{\mathbf{i}_l}$ in CSS. From the construction of the CSS format, every path $\mathcal{P}'$ in $SP(\mathcal{P}_{\mathbf{i}_l})$ is either (i) a subpath of $\mathcal{P}_{\mathbf{i}_l}$ or, (ii) node $\mathcal{P}'(l')$ at any level $l' < l$ is to the right of $\mathcal{P}_{\mathbf{i}_l}(l')$, i.e. path $\mathcal{P}'$ is to the right of $\mathcal{P}_{\mathbf{i}_l}$. This property narrows the positions of nodes storing $\mathcal{K}(\mathcal{P}'), \mathcal{P}' \in SP(\mathcal{P}_{\mathbf{i}_l})$ by imposing an ordering to the paths in $SP(\mathcal{P}_{\mathbf{i}_l})$. Thus, we switch to a depth-wise traversal of CSS where, for computing $\mathcal{K}(\mathcal{P}_{\mathbf{i}_{N-1}})$, we store $\mathcal{K}$ vector of only contributing paths in $SP(\mathcal{P}_{\mathbf{i}_l})$ for every level $2 \leq l \leq N-1$ and discard $\mathcal{K}$ vectors of paths that are to the left of $\mathcal{K}(\mathcal{P}_{\mathbf{i}_{N-1}})$. Given a set of IOU non-zeros, the size of memory that needs to be allocated for this memoization strategy is determined by the node p_m at level $N-2$ with the largest number of children. That is, the maximum number of $\mathcal{K}$ vectors that need to be stored before at least one of them is discarded is the sum of the number of $\mathcal{K}$ vectors required to compute $\mathcal{K}(\mathbf{j})$ for every child j of p_m.

Symbolic parent graph Note that since $SP(\mathcal{P}_{\mathbf{i}_l})$ are not disjoint sets, we need to ensure that $\mathcal{K}$ of overlapping elements are not computed multiple times. We resolve this problem by constructing an auxiliary data structure called the *symbolic parent graph* $SG(V,E)$ that retrieves all elements in $SP(\mathcal{P}_{\mathbf{i}_l})$ for every path $\mathbf{i}_l$ in CSS data structure, where

$$V = \bigcup_{l=1}^{N-1} V_l \quad \Big| \quad \begin{aligned} &E = \{(u,v) : u \in V_{j+1}, v \in V_j \cap \\ &\{w : w = \mathcal{P}'(j), \mathcal{P}' \in SP(\mathcal{P}_u)\}, 1 \leq j < N-1\} \end{aligned}$$

Here, V_l is the set of all nodes of the CSS tree at level l, and for every node u in level l, $deg(u) = l$. Moreover, $SG(V,E)$ is a union of nnz_{N-1} directed acyclic graphs, i.e., $SG = \bigcup_{i=1}^{nnz_{N-1}} \mathcal{T}_i^{SG}$, where $\mathcal{T}_i^{SG}$ is the DAG with node i as the first node in its topological sorting. From the construction, we infer that $i \in nz_{N-1}$. The construction of the symbolic parent graphs makes it easy to retrieve the nodes in the CSS tree that contribute to the partial Kronecker product at any given node. The space complexity of this auxiliary graph is $S_{SG} = \beta_{int} \sum_{l=2}^{N-1} l \cdot nnz_l$. The time complexity of $SG(V,E)$ construction is $\mathcal{O}(|V|+|E|) = \mathcal{O}(\sum_{l=2}^{N-1} l \cdot nnz_l)$, which is smaller than the S^{3}TTMc cost by $\mathcal{O}(R^l)$ per CSS level (Eq. (5.14)).

5.4 Parallelism The symbolic parent graph SG is used to perform a parallel depth-wise traversal of CSS to compute the intermediate $\mathcal{K}(\mathcal{P})$ results. The leaf nodes

Algorithm 2: Rank-R S^{3}TTMc-CSS

Result: Matricized S^{3}TTMc, $\mathbf{Y} \in \mathbb{R}^{I \times R^{N-1}}$

Construct symbolic parent graph SG

$p_m = \arg\max_{j \in nz_{N-2}} |child(j)|$

Allocate memoization workspace $WS = \sum_{l=1}^{N-2} R^l \big|\{n : n \in \mathcal{T}_j^{SG}, n \in nz_l \forall j \in child(p_m)\}\big|$

parfor $i_{N-1} = 1, \dots nnz_{N-1}$

  while $u_l \in nodes(\mathcal{T}_{i_{N-1}}^{SG})$ *at level* l **do**

    if $l < 2$ **then**

      $\mathcal{K}(u_l) = \mathbf{U}(u_l)$

    else

      for $\mathcal{P}' \in SP(u_l)$ **do**

        $n = \mathcal{P}_{u_l} \setminus \mathcal{P}'$

        $\mathcal{K}(\mathcal{P}_{u_l}) \leftarrow \mathcal{K}(\mathcal{P}_{u_l}) + \mathbf{U}(n,:) \otimes \mathcal{K}(\mathcal{P}')$

parfor $i_{N-1} = 1, \dots nnz_{N-1}$

  while $ind \in$ *nonzero array of node* i_{N-1} **do**

    $\mathcal{P} = (i_1, \dots i_{N-1})$

    AtomicAdd$\big(\mathbf{Y}(ind,:), \mathcal{K}(\mathcal{P}) \cdot \mathcal{X}_{(\mathcal{P},ind)}\big)$

of CSS are distributed over p threads, each of which is responsible for computing $\mathcal{K}(\mathcal{P})$ for all of its leaf nodes, as shown in Algorithm 2. We refer to the set of nodes in level $l-1$ that contribute to $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ of node i_l as the symbolic parents of i_l, i.e., $SP(\mathcal{P}_{\mathbf{i}_l})$. Thread local workspace is allocated based on the strategy described in Sec. 5.3 to reduce memory allocated for memoization.

Thus, we can propagate intermediate Kronecker products to nodes down the levels of the CSS tree using the edges of SG. At the leaf nodes at level $N-1$, the final $\mathcal{K}(\mathcal{P}_{\mathbf{i}_{N-1}})$ vectors are multiplied with the non-zero values at every leaf. Note that while there are no synchronization constraints when computing $\mathcal{K}$ for the nodes of the CSS tree, atomics are required while accumulating results into the matricized output $\mathbf{Y}$.

S^{3}TTMc-CSS can be directly used in Algorithm 1 to gain better performance for Tucker decomposition.

5.5 Cost comparison with baselines We compare the number of floating point operations in S^{3}TTMc-CSS (C^{CSS}) to *SPLATT* (C^{SPLATT}) and *S^{3}TTMc-UCOO* (C^{UCOO}) [2]. We provide a cost model for each of the algorithms, and analyze their performance for rank R-S^{3}TTMc.

Flop count of S^{3}TTMc-CSS We assume $nnz_l \approx \binom{N}{l} unnz$ for $N-1 \geq l \geq \lceil \frac{N}{2} \rceil$. The assumption stems from two observations on the CSS structure - (i) the number of nodes at level l is proportional to the number of distinct subsequences of length l from the set of IOU non-zeros (Sec. 4.1), and (ii) for high-order tensors, frequency of overlap of subsequences between IOU non-zeros reduces deeper down the tree i.e. branching in the tree decreases deeper into the CSS structure. As level $\lceil N/2 \rceil$ has the largest number of nodes, we restrict our assumption to $N-1 \geq l \geq \lceil \frac{N}{2} \rceil$. The cost of computing $\mathcal{K}(\mathcal{P}_{\mathbf{i}_l})$ for any path $\mathcal{P}_{\mathbf{i}_l}$ is the sum of Kronecker products $\mathcal{K}(\mathcal{P}'_k) \otimes \mathbf{U}(i_k,:)$ for each path $\mathcal{P}'_k \in SP(\mathcal{P}_{\mathbf{i}_l}), 1 \leq k \leq l$, as shown in eq. (5.13). We then write the total flop count of S^{3}TTMc computation as the sum of $c(l)$ over all levels of the CSS tree, along with the cost of scaling the $\mathcal{K}$ vectors at level $N-1$ by the non-zero value, as shown in Eq. (5.14).

$$c(l; N, R) = (2l-1) R^l \binom{N}{l} unnz \tag{5.13}$$

$$C^{CSS} = \sum_{l=2}^{N-1} c(l; N, R) + 2 \cdot unnz \cdot N R^{N-1} \tag{5.14}$$

Note that for $l < \lceil \frac{N}{2} \rceil$, we have $nnz_l < nnz_{l+1}$ (Sec. 4), and thus $c(l) < c(l+1)$. For $l \geq \lceil \frac{N}{2} \rceil$, we analyze the extrema of $c(l; N, R)$ to obtain the level l_0 for which C^{CSS} is dominated by $c(l_0; N, R)$.

As $c(l; N, R)$ is log-concave, on inspecting the first derivative of $\log c(l; N, R)$, we have $\log c'(l) > 0$ at $l = \lceil \frac{N}{2} \rceil$ and

$$\frac{d}{dl} \log c(l) \Big|_{l=N-1} = \log R - \left(H_{N-1} - \frac{2N-1}{2N-3} \right) \tag{5.15}$$

where H_n is the n^{th}-harmonic number [43]. We know that H_N closely approximates the natural logarithm function, i.e., $\log N \leq H_N \leq 1 + \log N$. Eq. (5.15) is negative for high-order tensors for suitable values of rank R, as $H_{N-1} - \frac{2N-1}{2N-3}$ is unbounded for large N. This implies that the sign of $(\log c(l))'$ determines if C^{CSS} is dominated by the cost contribution of the last level of CSS, or if there exists a level $N-1 > l_0 \geq \lceil \frac{N}{2} \rceil$ that obtains the maximum $c(l; N, R)$.

Comparison to *S^{3}TTMc-UCOO* *S^{3}TTMc-UCOO* implements Eq. (3.4) without any memoization of intermediate Kronecker products shared between IOU non-zeros. Its cost model is given by $C^{UCOO} = 2R^{N-1} N! \, unnz$.

Irrespective of the level contributing the largest cost to C^{CSS}, it is clear that $\log\left(\frac{C^{CSS}}{C^{UCOO}}\right) < 0$. S^{3}TTMc-CSS performs less work than *S^{3}TTMc-UCOO* for high-order sparse symmetric tensors.

Comparison to *SPLATT* The cost model of *SPLATT* in terms of the number of floating point operations is given by Eq. (5.16).

$$C^{SPLATT} = \sum_{l=2}^{N-1} 2R^{N-l+1} nnz'_l + 2 \cdot unnz \cdot N! \cdot R \tag{5.16}$$

The dominant term in Eq. (5.16) is the computation at level N [34] due to the growth of the factorial term with increasing tensor order.

[2] Since CTF targets sparse symmetric tensor contractions, a more general case compared to TTM, and *S^{3}TTMc-UCOO* outperforms CTF in Sec. 6, we ignore its analysis.

THEOREM 5.1. *S^3TTMc-CSS outperforms SPLATT in terms of number of floating point operations for rank R-S^3TTMc operation on order-N sparse symmetric tensors, if Eq.* (5.15) *is negative.*

Proof. Based on the sign of Eq. (5.15), we compare the flop counts of S^{3}TTMc-CSS and *SPLATT* baseline.

- Eq. (5.15) is negative, implying that there exists a level $N-1 > l_0 \geq \lceil \frac{N}{2} \rceil$ such that $c(l_0; N, R)$ is the dominant term, i.e., $(\log c(l))' = 0$ at $l = l_0$. Then, applying AM-HM inequality, we have

$$\log R \geq \frac{2(2l_0 - N)}{N+1} - \frac{2}{2l_0 - 1} \tag{5.17}$$

  Requiring that $R > 1$ further restricts the feasible values of l_0 to $\frac{N}{2} + 1 \leq l_0 < N - 1$. Using the inequality derived in Eq. (5.17) along with the main condition for this case, i.e., $0 < \log R < H_{N-1} - \frac{2N-1}{2N-3}$, we can write

$$\begin{aligned} \log\left(\frac{C^{CSS}}{C^{SPLATT}}\right) &= \log \frac{(2l_0 - 1)R^{l_0}\binom{N}{l_0} unnz}{RN! \cdot unnz} \\ &< \log \frac{2l_0(N-1)^{l_0 - 1}}{l_0^{l_0}(N - l_0)^{N - l_0}} \end{aligned} \tag{5.18}$$

  Note that Eq. (5.18) is an increasing function in l_0 for $\frac{N}{2} + 1 \leq l_0 < N - 1$ for a given tensor order, and as $N \to \infty$, we see that Eq. (5.18) is negative, i.e., S^{3}TTMc-CSS performs less work than *SPLATT*.

- Eq. (5.15) is non-negative, implying that the dominant term in C^{CSS} is $c(N-1; N, R)$. Then,

$$\begin{aligned} \log \frac{C^{CSS}}{C^{SPLATT}} &= \log \frac{(2N-1)NR^{N-1} \cdot unnz}{RN! \cdot unnz} \\ &> \log\left(\frac{2N-1}{N-1}\right) - \frac{4N-5}{(N-1)(2N-3)} > 0 \end{aligned}$$

  Thus, for $\log R \geq H_{N-1} - \frac{2N-1}{2N-3}$, Algorithm 2 could perform more work than SPLATT.

For S^{3}TTMc, if rank and tensor order satisfy $\log R < H_{N-1} - \frac{2N-1}{2N-3}$, our algorithm is more cost-efficient than *SPLATT*. ∎

Table 2 lists a range of (N, R) pairs using N_{min} and R_{max} values such that for a S^{3}TTMc with tensor order $N \geq N_{min}$ and matrix rank $R \leq R_{max}$, Eq. (5.15) is negative, thus the performance of S^{3}TTMc-CSS is theoretically superior to SPLATT.

R_{max}	2	4	8	16
N_{min}	5	8	14	26

Table 2: S^{3}TTMc-CSS outperforms *SPLATT* when its tensor order $N \geq N_{min}$ and matrix rank $R \leq R_{max}$.

6 Experiments

6.1 Platform and Experimental Configurations

Our experiments are conducted on a shared memory multiprocessor with two 24-core Intel Xeon Gold 6238M processors and 1.5 TBytes memory. The code is written in C++ and parallelized using OpenMP, configured to double-precision floating point arithmetic and 64-bit unsigned integers. It is compiled with GCC 8.3.0 and Netlib LAPACK 3.8.0 for linear algebra routines.

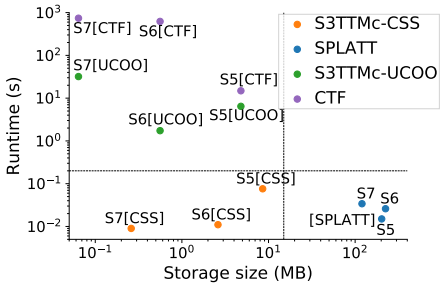
6.2 Datasets We tested eleven synthetic and two real-world symmetric tensors of varying orders and number of IOU non-zeros, as shown in Table 3. Since real hypergraphs often exhibit strong interactions among groups of nodes, we use this property to generate synthetic hypergraphs with four clusters, with each cluster having a uniform distribution of IOU non-zeros. The *Walmart* and *Amazon* datasets are constructed from real-world hypergraphs, and have been used to evaluate the performance of hypergraph clustering algorithms. *Walmart*, an order-8 symmetric tensor [1], is generated from a hypergraph representing Walmart's user relations where hyperedges are co-purchased products. *Amazon*, an order-14 symmetric tensor [26], is extracted from a user-review hypergraph with product reviews as hyperedges. Only one CSF tree and the fastest leaf-to-root TTMc algorithm outlined in the work [34] are used from SPLATT owing to the symmetry feature. CTF [39, 40] is run with LAPACK 3.8.0 [3] and HPTT 1.0.5 [41] functionalities. The thirteen datasets have been divided into three categories - *small*, *large*, and *huge* - depending on the ability of each implementation to compute S^{3}TTMc successfully within a reasonable time limit. The choice of R for rank-R S^{3}TTMc on each dataset is determined by our cost analysis in Sec. 5.5. Among all datasets in *small* and *large* categories, except for dataset S5, R is set to the largest R_{max}. We chose $R = 3$ for S5 with the purpose to demonstrate a case where *SPLATT* outperforms S^{3}TTMc-CSS and verify Theorem 5.1.

6.3 Overall Performance We compare the performance of our S^{3}TTMc-CSS with *SPLATT*, *S^3TTMc-UCOO* and *CTF* for all three categories of datasets.

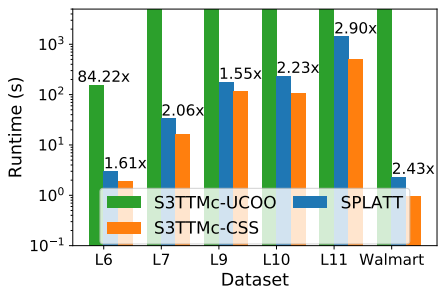
Small datasets Fig. 3a demonstrates the trade-off in the runtime of the S^{3}TTMc operations and the size of the symmetric tensor storage format observed in each of the implementations. The size of the symmetric tensors chosen is small enough to ensure that all baseline implementations run successfully without timing out. We observe three groups in Fig. 3a - (i) *S^3TTMc-UCOO* and CTF implementations, which are the most compact but exhibit high runtimes; (ii) *SPLATT* us-

Table 3: Description of symmetric sparse tensors and implementations that can successfully compute S^{3}TTMc within a reasonable time limit.

Category	Tensor	Order	Dim	*unnz*	*R*	Implementations			
						S^{3}TTMc-UCOO	CTF (*UCOO*)	*SPLATT (FCSF)*	S^{3}TTMc-CSS
Small	S5	5	100	100K	3	✓	✓	✓	✓
	S6	6	100	10K	2	✓	✓	✓	✓
	S7	7	50	1K	3	✓	✓	✓	✓
Large	L6	6	400	1M	2	✗	✗	✓	✓
	L7	7	400	1M	3	✗	✗	✓	✓
	L9	9	400	10K	5	✗	✗	✓	✓
	L10	10	400	1K	5	✗	✗	✓	✓
	L11	11	400	100	6	✗	✗	✓	✓
	Walmart	8	15624	3082	4	✗	✗	✓	✓
Huge	H8	8	400	1M	4	✗	✗	✗	✓
	H12	12	400	10K	3	✗	✗	✗	✓
	H13	13	400	10K	3	✗	✗	✗	✓
	Amazon	14	12981	2131	2	✗	✗	✗	✓



(a)



(b)

Dataset	Size (MB)	Time (s)
H8	1660.00	308.82
H12	578.42	179.07
H13	1150.00	1000.15
Amazon	479.28	15.76

(c)

Figure 3: Comparison of overall performance of S^{3}TTMc-CSS with *SPLATT*, *S^{3}TTMc-UCOO* and CTF for datasets in Table 3. (a) Storage-runtime trade off of all algorithms for *small* tensors. The grouping of runtimes and storage size is similar to Fig. 1. (b) S^{3}TTMc runtime for *large* tensors. *S^{3}TTMc-UCOO* completes only for dataset L6. The speed-up of CSS over each of the baselines is indicated over the corresponding bar. (c) Storage-runtime trade off of S^{3}TTMc-CSS for *huge* tensors. *SPLATT* runs out of memory for this category.

ing *FCSF*, which has a massive memory overhead but is significantly faster than *S^{3}TTMc-UCOO* and CTF using *UCOO*; and (iii) S^{3}TTMc-CSS, which is comparable to *UCOO* in terms of storage requirement while still being faster than *SPLATT*, *S^{3}TTMc-UCOO* and CTF. Fig. 3a aligns perfectly with Fig. 1 and shows that S^{3}TTMc-CSS does an effective trade-off between storage and efficient computation.

S^{3}TTMc-CSS is faster than *SPLATT* for S6 and S7 datasets, as the rank R chosen satisfies Theorem 5.1. However, as we have chosen $R > \exp(H_4 - 1.285)$ (Eq. (5.15)), S^{3}TTMc-CSS performs more work than *SPLATT* and is hence slower.

Large datasets The lack of scalability of both *S^{3}TTMc-UCOO* and CTF with increasing tensor order in Fig. 3a makes them an infeasible baseline for comparison on *large* datasets. While CTF runs out of memory for all datasets, *S^{3}TTMc-UCOO* does not complete in reasonable time for order-7 tensors and higher, as can be seen in Fig. 3b. For the best thread configuration, we achieve up to 2.9× speedup over *SPLATT*, the fastest state-of-the-art baseline (details in Sec. 3). Moreover, our algorithm is always faster than *SPLATT* for *large* symmetric tensors.

Huge datasets Fig. 3c presents the performance of S^{3}TTMc-CSS on *huge* tensors for which *SPLATT* runs out of memory. H8 symmetric tensor has the largest memory requirement at 1.66GB, while S^{3}TTMc on H13 has the longest execution time at 1000 seconds.

6.4 Thread Scalability We analyze thread scalability of S^{3}TTMc-CSS and *SPLATT* for a representative order-10 tensor, L10. Note that we have excluded *S^{3}TTMc-UCOO* and CTF as they do not run to completion for any of the *large* datasets, owing to their lack of scalability with increasing tensor order. *SPLATT* and S^{3}TTMc-CSS show similar scalability due to the similarities in tree layout of the data structure, and its depth wise traversal to compute S^{3}TTMc, as can be seen in Fig. 4. However, S^{3}TTMc-CSS is faster than *SPLATT*, achieving up to 2.46× speedup over *SPLATT* for 8 threads. The scaling becomes poor for higher or-

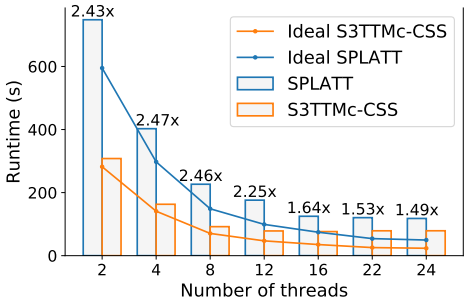


Figure 4: Multithreading scalability of S^{3}TTMc-CSS compared to *SPLATT* for order-10 tensor L10. The speed-up of CSS over *SPLATT* is specified above the blue bars.

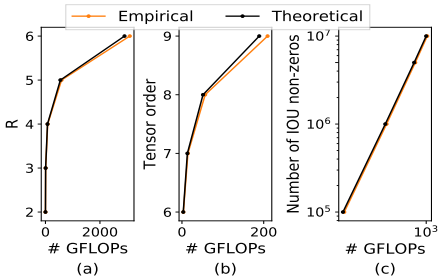


Figure 5: Effect of (a) matrix rank R on S^{3}TTMc-CSS for order-11 tensor L11, (b) tensor order N for rank 2-S^{3}TTMc on tensors having 1M IOU non-zeros, and (c) number of IOU non-zeros in order-7 tensor on S^{3}TTMc-CSS with $R = 3$.

der tensors using more threads, as can be seen in Fig. 4, because of the size of memory allocated per thread increases, resulting in degrading data locality and cache behavior. Another effect of increasing thread memory is the worsening data locality in the Kronecker product computations. Though the memoization strategy described in Algorithm 2 minimizes the number of stored partial Kronecker products, the overhead due to random accesses in the intermediate workspace still remains, and manifests as suboptimal scaling.

6.5 Parameter sweep We analyze the behavior of S^{3}TTMc-CSS as a function of three key parameters affecting the S^{3}TTMc operation - (i) rank R chosen for S^{3}TTMc, (ii) tensor order, and (iii) number of IOU non-zeros in the tensor. We compare the number of floating point operations performed by S^{3}TTMc-CSS with the theoretical number of FLOPs defined in Eq. (5.14). Note that empirical observations closely follow our cost model in Sec. 5.5. Fig. 5a shows the behavior of the S^{3}TTMc operation with varying matrix rank for order-11 tensor, L11. From Eq. (5.14), we see that C^{CSS} grows exponentially with R for a given symmetric tensor, and we correspondingly observe nonlinear scaling in Fig. 5a. This is different from the close to linear scalability of general sparse TTMc [25]. Similarly, a non-linear growth in the number of FLOPs is observed with increasing tensor order (Fig. 5b). We have chosen $R = 2$ - which is R_{max} for order-6 tensors (Table 2) - as it satisfies Theorem 5.1 for all tensor orders considered in the figure. It is difficult to estimate the ideal scaling trend with both R and N from Eq. (5.14) due to the dependence on the level of the CSS tree that dominates the cost of C^{CSS}. For the number of IOU non-zeros *unnz*, Eq. (5.14) indicates linear scaling, the same as observed in Fig. 5c for rank-3 S^{3}TTMc on order-7 tensor. With increasing *unnz*, we also observe non-linear growth in storage size, similar to *FCSF*; though size of *UCOO* increases linearly. There is, however, less significant change in the size of CSS format with increasing mode size I due to the inherent sparsity of the tensor.

6.6 Symmetric Tucker Decomposition We compute symmetric Tucker decomposition on the largest real-world dataset in Table 3, the Amazon dataset. As all the three baselines run out of memory, we use S^{3}TTMc-CSS to compute S^{3}TTMc for the decomposition (Line 2 in Algorithm 1). We are able to run rank-2 symmetric Tucker decomposition for the order-14 Amazon dataset in $\sim$ 1.79hrs, which shows the applicability of this work in the analysis of real world hypergraphs. S^{3}TTMc-CSS provides a practical framework to use symmetric Tucker decomposition in a hierarchical clustering approach to hypergraph analytics.

7 Conclusions

This work focuses on the S^{3}TTMc operation - the specialization of the frequently used tensor times matrix chain operation to symmetric tensors - used in symmetric tensor decomposition. We propose a computation-aware storage format CSS designed for symmetric tensors, with which we can efficiently perform S^{3}TTMc in parallel. We underscore the utility of our algorithm by demonstrating a better trade-off between memory and run-time over SPLATT, S^{3}TTMc-UCOO, and CTF for multiple synthetic and real-world datasets. In the future, we intend to explore the applicability of CSS to other tensor operations like the Matricized Tensor Times Khatri-Rao Product (MTTKRP) and decompositions like tensor train. We plan to improve the access patterns of intermediate results while executing S^{3}TTMc-CSS in parallel, adopt operation memoization, and apply our efficient S^{3}TTMc operation to hypergraph analytics.

Acknowledgments

This research is supported in part by a grant from NVIDIA Corporation.

References

[1] Ilya Amburg, Nate Veldt, and Austin Benson. Clustering in graphs and hypergraphs with categorical edge labels. In *Proceedings of The Web Conference 2020*, pages 706–717, 4 2020.

[2] Animashree Anandkumar, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. Tensor Decompositions for Learning Latent Variable Models. Technical report, 2014.

[3] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK users' guide*. SIAM, 1999.

[4] Joseph Anderson, Mikhail Belkin, Navin Goyal, Luis Rademacher, and James Voss. The More, the Merrier: the Blessing of Dimensionality for Learning Large Gaussian Mixtures. Technical report, 5 2014.

[5] Grey Ballard, Tamara Kolda, and Todd Plantenga. Efficiently computing tensor eigenvalues on a GPU. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1340–1348. IEEE, 5 2011.

[6] Muthu Baskaran, Benoˆit Meister, Richard Lethin, and Jonathan Cai. Optimization of symmetric tensor computations. In *IEEE Conference on High Performance Extreme Computing (HPEC), Waltham, MA, USA*. IEEE, September, Sep 2015.

[7] Jerome Brachat, Pierre Comon, Bernard Mourrain, and Elias Tsigaridas. Symmetric tensor decomposition. *European Signal Processing Conference*, pages 525–529, 1 2009.

[8] Johan Cambré, Lieven De Lathauwer, and Bart De Moor. Best rank-(R, R, R) super-symmetric tensor approximation - A continuous-time approach. In *Proceedings of the IEEE Signal Processing Workshop on Higher-Order Statistics, SPW-HOS 1999*, pages 242–246. Institute of Electrical and Electronics Engineers Inc., 1999.

[9] Pierre Comon, Gene Golub, Lek-Heng Lim, and Bernard Mourrain. Symmetric tensors and symmetric tensor rank. 2 2008.

[10] Ernesto Estrada and Juan A. Rodriguez-Velazquez. Complex Networks as Hypergraphs. *Physica A: Statistical Mechanics and its Applications*, 364:581–594, 5 2005.

[11] Suzanne Renick Gallagher, Micah Dombrower, and Debra S. Goldberg. Using 2-node hypergraph clustering coefficients to analyze disease-gene networks. In *ACM BCB 2014 - 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 647–648, New York, New York, USA, 9 2014. Association for Computing Machinery, Inc.

[12] Debarghya Ghoshdastidar and Ambedkar Dukkipati. Consistency of Spectral Partitioning of Uniform Hypergraphs under Planted Partition Model. Technical report.

[13] Debarghya Ghoshdastidar and Ambedkar Dukkipati. Uniform Hypergraph Partitioning: Provable Tensor Methods and Sampling Techniques. *Journal of Machine Learning Research*, 18(50):1–41, 2017.

[14] Debarghya Ghoshdastidar and Ad@csa Iisc Ernet In. A Provable Generalized Tensor Spectral Method for Uniform Hypergraph Partitioning Ambedkar Dukkipati. Technical report.

[15] Navin Goyal, Santosh Vempala, and Ying Xiao. Fourier PCA and Robust Tensor Decomposition. *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 584–593, 6 2013.

[16] Oguz Kaya and Bora Ucar. High Performance Parallel Algorithms for the Tucker Decomposition of Sparse Tensors. In *2016 45th International Conference on Parallel Processing (ICPP)*, pages 103–112. IEEE, 8 2016.

[17] Oguz Kaya and Bora Uçar. Parallel candecomp/parafac decomposition of sparse tensors using dimension trees. *SIAM Journal on Scientific Computing*, 40(1):C99–C130, 2018.

[18] Zheng Tracy Ke, Feng Shi, and Dong Xia. Community Detection for Hypergraph Networks via Regularized Tensor Power Iteration. Technical report, 2020.

[19] T. Kolda and B. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[20] Tamara G. Kolda. Numerical Optimization for Symmetric Tensor Decomposition. 10 2014.

[21] Jiajia Li, Yuchen Ma, and Richard Vuduc. ParTI! : A Parallel Tensor Infrastructure for multicore CPUs and GPUs, 10 2018.

[22] Jiajia Li, Jimeng Sun, and Richard Vuduc. HiCOO: Hierarchical Storage of Sparse Tensors. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 238–252. IEEE, 11 2018.

[23] Jiajia Li, Jimeng Sun, and Richard Vuduc. HiCOO: Hierarchical storage of sparse tensors. In *Proceedings of the ACM/IEEE International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Dallas, TX, USA, November 2018.

[24] B. Liu, C. Wen, A. D. Sarwate, and M. M. Dehnavi. A unified optimization approach for sparse tensor operations on GPUs. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 47–57, Sept 2017.

[25] Yuchen Ma, Jiajia Li, Xiaolong Wu, Chenggang Yan, Jimeng Sun, and Richard Vuduc. Optimizing sparse tensor times matrix on gpus. *Journal of Parallel and Distributed Computing*, 2018.

[26] Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, 2019.

[27] Israt Nisa, Jiajia Li, Aravind Sukumaran-Rajam, Prasant Singh Rawat, Sriram Krishnamoorthy, and P. Sadayappan. An efficient mixed-mode representation of

sparse tensors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, pages 49:1–49:25, New York, NY, USA, 2019. ACM.

[28] Martin D. Schatz, Tze Meng Low, Robert A. van de Geijn, and Tamara G. Kolda. Exploiting Symmetry in Tensors for High Performance: Multiplication with Symmetric Tensors. *SIAM Journal on Scientific Computing*, 36(5):C453–C479, 1 2014.

[29] Amnon Shashua, Ron Zass, and Tamir Hazan. Multi-way Clustering Using Super-symmetric Non-negative Tensor Factorization. Technical report, 2005.

[30] Decompositionsamantha Sherman and Tamara G Kolda. Estimating Higher-Order Moments Using Symmetric Tensor DecompositionSamanthaDecomposition. Technical report.

[31] Nicholas D. Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E. Papalexakis, and Christos Faloutsos. Tensor Decomposition for Signal Processing and Machine Learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 7 2017.

[32] Shaden Smith and George Karypis. Tensor-matrix products with a compressed sparse tensor. In *Proceedings of the 5th Workshop on Irregular Applications Architectures and Algorithms - IA3 '15*, pages 1–7, New York, New York, USA, 2015. ACM Press.

[33] Shaden Smith and George Karypis. SPLATT: The Surprisingly ParalleL spArse Tensor Toolkit. `http://cs.umn.edu/~splatt/`, 2016.

[34] Shaden Smith and George Karypis. Accelerating the tucker decomposition with compressed sparse tensors. In *European Conference on Parallel Processing*. Springer, 2017.

[35] Shaden Smith, Niranjay Ravindran, Nicholas Sidiropoulos, and George Karypis. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. In *Proceedings of the 29th IEEE International Parallel & Distributed Processing Symposium*, IPDPS, 2015.

[36] Shaden Smith, Niranjay Ravindran, Nicholas D. Sidiropoulos, and George Karypis. SPLATT: Efficient and Parallel Sparse Tensor-Matrix Multiplication. In *Proceedings - 2015 IEEE 29th International Parallel and Distributed Processing Symposium, IPDPS 2015*, 2015.

[37] Edgar Solomonik. *Provably Efficient Algorithms for Numerical Tensor Algebra*. PhD thesis, EECS Department, University of California, Berkeley, Sep 2014.

[38] Edgar Solomonik, James Demmel, and Torsten Hoefler. Communication lower bounds for tensor contraction algorithms. Technical report, ETH Zürich, 2015.

[39] Edgar Solomonik and Torsten Hoefler. Sparse Tensor Algebra as a Parallel Programming Model. Technical report, 2015.

[40] Edgar Solomonik, Devin Matthews, Jeff Hammond, and James Demmel. Cyclops Tensor Framework: Reducing Communication and Eliminating Load Imbalance in Massively Parallel Contractions. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pages 813–824. IEEE, 5 2013.

[41] Paul Springer, Tong Su, and Paolo Bientinesi. Hptt: A high-performance tensor transposition c++ library. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, pages 56–62, 2017.

[42] Ledyard R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 9 1966.

[43] Eric W. Weisstein. Harmonic Number.

[44] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with Hypergraphs: Clustering, Classification, and Embedding. Technical report.

The Traveling Firefighter Problem

Majid Farhadi*
Georgia Tech
farhadi@gatech.edu

Alejandro Toriello
Georgia Tech
atoriello@isye.gatech.edu

Prasad Tetali
Georgia Tech
tetali@math.gatech.edu

Abstract

We introduce the L_p Traveling Salesman Problem (L_p-TSP), given by an origin, a set of destinations, and underlying distances. The objective is to schedule a destination visit sequence for a traveler of unit speed to minimize the Minkowski p-norm of the resulting vector of visit/service times. For $p = \infty$ the problem becomes a path variant of the TSP, and for $p = 1$ it defines the Traveling Repairman Problem (TRP), both at the center of classical combinatorial optimization.

L_p-TSP can be formulated as a convex mixed-integer program and enables a smooth interpolation between path-TSP and TRP, corresponding to optimal routes from the perspective of a server versus the customers, respectively. The parameter p can affect fairness or efficiency of the solution: The case $p = 2$, which we term the Traveling Firefighter Problem (TFP), models the scenario when the cost/damage due to a delay in service is quadratic in time.

We provide a polynomial-time reduction of L_p-TSP (losing a factor of $1 + \varepsilon$ in performance) to the segmented-TSP, a routing problem that defines a constant $O(1+\varepsilon^{-2})$ number of deadlines by which given numbers of vertices should be visited. Subsequently we derive polynomial-time approximation schemes for L_p-TSP in the Euclidean metric and the tree metric (for which the problem is strongly NP-hard).

We also study the all-norm-TSP, in which the objective is to find a route that is (approximately) optimal with respect to the minimization of any norm of the visit times. We improve the approximation bound for this problem to 8, down from 16, and further prove an impossibility for an approximation factor better than 1.78, even in line metrics. Finally, we show the performance of our algorithm can be optimized for a specific norm, particularly yielding a 5.65-approximation for the TFP on general metrics.

*Supported by aco.gatech.edu, triad.gatech.edu. Part of this work was conducted when authors visited Simons Institute for the Theory of Computing.

We leave open several interesting directions to further develop this line of research.

1 Introduction

The L_p-TSP is a routing problem seeking to minimize the L_p norm of the vector of visit/service times to a set of customer locations. It generalizes and interpolates between two well-studied problems, the path variant of the TSP and the TRP, also known as the Minimum Latency Problem, which are the two extreme cases in which one minimizes either the largest or the sum of service times. By assuming the server's speed is constant, we use time and distance interchangeably in the remainder of the paper.

As one motivating example, the L_p-TSP for $p = 2$, which we call the Traveling Firefighter Problem (TFP), abstracts a macro-scale optimal strategy for dispatching a firefighter to minimize the total damage due to fires at various locations. Ride-sharing is another use-case of our problem. For example, devising the return route of a school bus, should we optimize the fuel consumption (i.e. the driver's time en route) or the average student waiting time? Is it fairer to further penalize larger waiting times, e.g. minimizing the sum of squares/cubes of the waiting times? Before formulating the problems under study we introduce some notation.

Notation. $[z]$ denotes the set $\{1, \cdots, z\}$ for any positive integer z. $\tilde{O}(\cdot)$ is equivalent to $O(\cdot)$, treating $\varepsilon > 0$ as a constant. A metric over a set of nodes V is a distance function $d : V \times V \to \mathbb{R}_{\geq 0}$ that satisfies symmetry, $d(x, y) = d(y, x) \quad \forall x, y \in V$, identity, $d(x, x) = 0$, and the triangle inequality, $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in V$.

The inputs to the problem are the set of vertices V, including both the destinations and the server's starting location, $s \in V$, and the underlying metric $d(\cdot, \cdot)$ over V, corresponding to distances (or times) between vertex pairs.

A feasible solution or route is a permutation σ over V that starts at the origin; σ_i denotes the i^{th} vertex to be visited, so we always have $\sigma_1 = s$. $\mathcal{F}$ denotes the set of all feasible solutions. The i^{th} smallest visit time,

due to a solution $\sigma \in \mathcal{F}$, is denoted T_i^σ, i.e.

$$T_i^\sigma = \begin{cases} 0 & i = 1 \\ \sum_{j=2}^{i} d(\sigma_{j-1}, \sigma_j) & i \in \{2, \cdots, n\} \end{cases}$$

The visit time for vertex v is denoted ℓ_v^σ. Similarly, $\ell_s^\sigma = 0$ and $\ell_v^\sigma = \sum_{i=2}^{\sigma_i = v} d(\sigma_{i-1}, \sigma_i) \quad \forall v \neq s$.

Definition 1. (L_p-TSP) *The input of the optimization problem L_p-TSP is a set of destinations V, a starting vertex $s \in V$, and a metric $d : V \times V \to \mathbb{R}_{\geq 0}$. The objective is to find a feasible route, $\sigma \in \mathcal{F}$, starting at s and visiting all $v \in V$, that minimizes the Minkowski p-norm of the visit times, i.e. $\min_{\sigma \in \mathcal{F}} \|\ell^\sigma\|_p$, where*

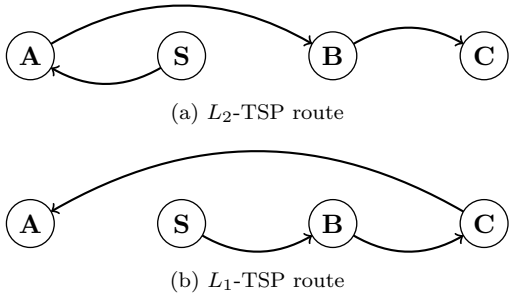
$$\|\ell^\sigma\|_p := \left(\sum_{v \in V} |\ell_v^\sigma|^p \right)^{\frac{1}{p}}.$$

When the problem/objective is clear from context, we denote an optimal route as OPT and the answer of our algorithm as ALG.

A Better Objective. One can verify that the objective (norm) affects various aspects of the routing problem, such as efficiency. L_p-TSP enables a smooth transition between two extreme objectives. For larger p's the objective is strongly affected by dominating (larger) entries of the delay vector and minimizing $\|\ell\|_p$ is more to the benefit of the server. In contrast, smaller p's provide a further aggregated measure of the amount of time that the customers have waited. This trade-off can also be interpreted from a fairness perspective, as increasing p discourages the longest waiting time from becoming too large.

Firefighter Example. We further elaborate on why $p \notin \{1, \infty\}$ can be a useful objective by considering the routing of a firefighter.[1] Consider a set of wildfires in dispersed locations, and suppose a skilled firefighter extinguishes any fire the second they arrive at a location; the firefighter must choose the order in which to visit and extinguish the fires. One possible strategy is to choose a sequence in order to finish extinguishing all fires as soon as possible, which corresponds to solving the L_∞-TSP (the path-TSP). However, this may not be the best solution for the firefighter: L_∞-TSP minimizes the latest visit time for all fires, while the cost could be affected by all visit times; thus an aggregated measure could be a better objective.

[1] Over the past decade, and for the first time on record, the annual number of acres burned in the United States exceeded 10 million; this occurred twice [Hoo18]. In July 2019, a record 2.4 million acres of the Amazon rainforest were torched [Bor19]. The optimal allocation, scheduling and routing of firefighting resources may help address this global challenge.



(a) L_2-TSP route

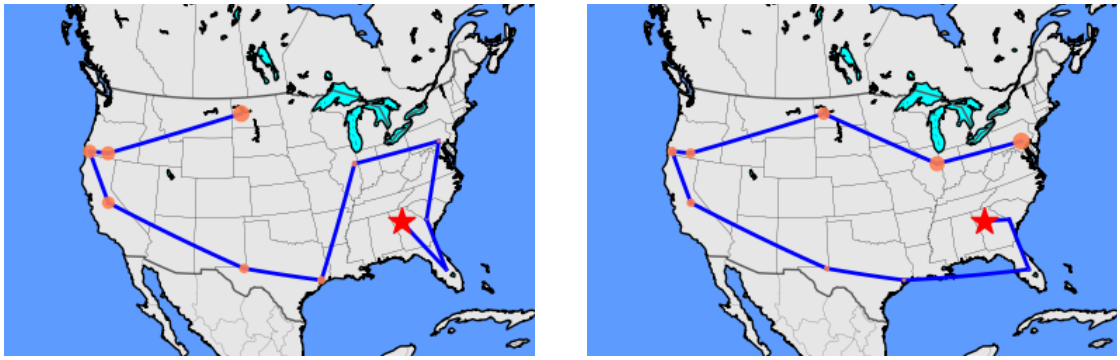
(b) L_1-TSP route

Figure 1: Different norms cause different routes.

For example, consider the following simple dynamics for the spread of wildfires over uniform territory, ignoring possible differences such as vegetation, weather and wind. After every second, a flammable point of territory is ignited if it is within a unit distance from the burning flames. Under these dynamics, the area of land scorched by the fire is a quadratic function of the elapsed time. Therefore, the damage due to the delay on the i^{th} visit can be better represented by $\ell_{\sigma_i}^2$ and minimizing $\sum_v \ell_v^2 = \|\ell\|_2^2$, or equivalently $\|\ell\|_2$, is a better objective for this scenario compared to other norms, particularly $\|\ell\|_1$ or $\|\ell\|_\infty$. Motivated by this example, we term L_2-TSP as the Traveling Firefighter Problem (TFP).

In an applied setting, this approach may require some refinement but the basic idea still applies. Land and weather asymmetries can be modeled by a multiplicity of vertices: If a fire spreads twice as fast in area, we can represent it by two vertices overlapping in the metric. One could also generalize the objective to a weighted sum of the squared delays and/or discretize large fires into smaller ones. Moreover, the time required to extinguish a fire can be accounted for by adding a new edge, hanging from the original destination at a distance proportional to the time required to contain that fire, and moving the destination to the other endpoint of the new edge.

The Argument versus The Objective. The set of feasible routes for all L_p-TSP problems is the same, while the objectives are different. For any $p \neq q$, there exist instances where the two optimal routes are different. As a simple example, consider four vertices **S**, **A**, **B**, and **C** over a line metric at locations $0, -1-\varepsilon, +1$, and $+2$ respectively. Starting at **S**, the route **SABC** is optimal for L_2-TSP with corresponding objective of $\|(0, 1+\varepsilon, 3+2\varepsilon, 4+2\varepsilon)\|_2 \simeq \sqrt{26}$, in contrast to the optimal route for L_1-TSP that is **SBCA** with the objective $\|(0, 1, 2, 5+\varepsilon)\|_1 = 8+\varepsilon$. Figure 1 depicts this example.



(a) Path-TSP route

(b) TFP route

Figure 2: Optimal path-TSP versus TFP routes for 10 U.S. locations; the areas of the discs at visit locations are proportional to the expected (quadratic) damage at each location, due to the delay along the route. Here, taking the TFP route (instead of the TSP) reduces total damage by 5%.

Consider the effect of a "wrong" norm on the optimal route. Figure 2 depicts an example considering 10 locations with simultaneous wildfires in the United States.[2] Dispatching a firefighter from Atlanta, and traveling 100 times faster than the spread of fire, TFP would look quite different than L_∞-TSP. In particular, the TFP reduces the total damage by 5% compared to the path-TSP solution.

As another example, consider destinations on the x axis, with many fires located at $+1$ and a single fire at $-1+\varepsilon$. Starting at $x=0$, the optimal L_∞-TSP route moves left first and then right, resulting in a solution roughly three times as expensive as the optimal TFP route, in terms of the L_2 objective. In fact, the relative performance of an L_∞-TSP route for the L_2 objective can be *unbounded*. For instance, consider the Euclidean metric over the plane with $n-2$ fires located at complex coded locations

$$\{e^{2\pi\cdot\frac{k}{n}i} : k \in \{1,\cdots,n-2\}\},$$

and m distinct fires located at $e^{2\pi\frac{n-1-\varepsilon}{n}i}$. Starting at $(1,0) = e^{0i}$ and moving at unit speed, for $n\to\infty$ and $m/n\to\infty$ the damage (squared delay) due to L_∞-TSP route (by walking in the wrong direction around the circle) converges to $4\pi^2$, while for the optimal TFP solution it goes to zero.

All-norm-TSP. We observed that TSP may not be a good solution for other L_p objectives. One may consider whether a different L_p-TSP is hopefully good for all norms. In this line, a natural question is whether there exists a single route that is approximately optimal concerning the minimization of any norm of the visit times, and whether we can efficiently find one. This can be viewed also as an online problem where the adversary chooses the norm, e.g. L_p, and the objective is to provide a competitive solution with respect to the optimal route.

[2]Wildfires observed during the first week of Dec 2019, according to satellite data [NAS].

Definition 2. (all-norm-TSP) *Given $s, V \ni s, d : V\times V\to\mathbb{R}_{\geq 0}$ as before, the objective is to choose a route that minimizes the maximum possible ratio between a symmetric norm of the visit time vector of the output route σ and the optimal route for that norm,*

$$\min_{\sigma\in\mathcal{F}}\sup_{\|\cdot\|}\frac{\|\ell_\sigma\|}{\min_{\sigma'\in\mathcal{F}}\|\ell_{\sigma'}\|}.$$

Golovin et al. [GGKT08] introduced this problem as the *all-norm-TSP* and gave an algorithm that outputs a route that is a 16-approximation with respect to any norm of the visit time vector. The concept of all-norm minimization has been of interest in many applications, e.g., for routing, load balancing [KRT99], and machine scheduling [AERW04, BP03].

1.1 Main Results & Proof Ideas. Optimizing the non-linear objective of L_p-TSP can be computationally challenging. The problem is already NP-hard even in the linear case, $p=1$, on a tree metric [Sit02], i.e. when the metric is pairwise distances over a graph on V that forms a tree. In contrast, TSP on trees is solvable, in linear time.

Archer and Williamson [AW03] showed that a $(1+\varepsilon)$-approximate solution for TRP exists that is a concatenation of $O(\log n\cdot\varepsilon^{-1})$ TSP paths. This can be generalized as the following Lemma, which enables a quasipolynomial time approximation scheme for weighted trees.

LEMMA 1.1. ([AW03]) *L_p-TSP can be $(1+\varepsilon)$-approximated by a concatenation of $O(\log n \cdot \varepsilon^{-1})$ TSP paths.*

Proof. Without loss of generality we can assume integral and polynomially bounded input distances, $d(i,j) \in \{0\} \cup [O(n^2/\varepsilon)] = \{0\} \cup [\tilde{O}(n^2)]$, as an appropriate quantization by rounding only adds a multiplicative error of $1 + O(\varepsilon/n^2)$ to any edge and keeps any norm of a valid tour, within a factor $(1 \pm \varepsilon)$.

Now break the optimal route according to time spots $1, (1+\varepsilon), \cdots, (1+\varepsilon)^\gamma$ where $\gamma = O(\log n \cdot \varepsilon^{-1})$ because $O(n^3 \cdot \varepsilon^{-1})$ is a bound on the length of the optimal route. Replacing each sub-route between two consecutive time spots, with the shortest path TSP over the same points does not increase any visit time beyond a factor $1+\varepsilon$, and hence preserves any norm of the visit times by a factor of $1+\varepsilon$. ∎

The above approach can only lead to a pseudo-polynomial time approximation algorithm, even for trees. Reducing the problem to many shortest paths cannot lead to an efficient polynomial-time reduction, because a concatenation of $o(\log n)$ path-TSP routes cannot approximate L_1-TSP within a constant factor [Sit14]. To further reduce the number of paths, we use the notion of segmented-TSP, introduced by Sitters [Sit14], to enable some dependence between consecutive deadlines. This problem requires a (sequence of monotonically non-decreasing) number of destinations to be visited by a number of deadlines, formulated as follows.

DEFINITION 3. (SEGMENTED-TSP) *Given $V \ni s, d : V \times V \to \mathbb{R}_{\geq 0}$ as before, in addition to integer numbers $n_1 \leq n_2 \leq \cdots \leq n_k \leq |V| = n$ and fractional numbers $t_1 \leq t_2 \leq \cdots \leq t_k$ as inputs, the segmented-TSP problem is a decision problem to verify whether a route exists that visits at least n_i distinct vertices by time t_i for all $i \in [k]$, starting at s.*

Approximation of segmented-TSP, i.e. a decision problem, can be defined as follows.

DEFINITION 4. *An α-approximate solution to a segmented-TSP instance, must visit the first n_i vertices by the modified deadline $\alpha \cdot t_i$, $\forall i \in [k]$, if an answer to the original segmented-TSP exists.*

We generalize a main result of Sitters [Sit14] that showed TRP can be reduced to (a polynomially many number of approximate) segmented-TSP problems with a constant number of deadlines.

THEOREM 1. *[L_p-TSP $\Rightarrow$ segmented-TSP] Let $\varepsilon > 0$ be a constant and $\mathcal{A}$ be an α-approximation algorithm for segmented-TSP for some $k = O(1+\varepsilon^{-2})$ of our choice. There is a $(1+\varepsilon) \cdot \alpha$-approximation algorithm for L_p-TSP that calls $\mathcal{A}$ (on the same network (V, s, d)) for a strongly polynomial number of times.*

Theorem 1, proved in Section 2, along with a PTAS for segmented-TSP [Sit14] on tree metrics and Euclidean metrics imply the following results.

COROLLARY 1.1. *There exist polynomial-time approximation schemes (PTAS) for L_p-TSP on weighted trees and Euclidean plane metrics.*

Note that due to our reduction of L_p-TSP to segmented-TSP, any approximation results for segmented-TSP, on specific metrics, would follow for L_p-TSP, at the cost of an additional factor of $(1+\varepsilon)$ to the approximation bound. Our results can be generalized to the case of multiple travelers, starting from arbitrary locations, as discussed in Section 5.

For general metrics, a PTAS is unlikely, as the problem becomes Max-SNP-hard. On the other hand, the constant factor approximability of L_p-TSP for general metrics is immediate due to the 16-approximation of the all-norm-TSP by Golovin et. al. [GGKT08]. In this line, we improve the approximation bound for all-norm-TSP by a factor of 2.

THEOREM 2. *[all-norm-TSP] There is a polynomial-time algorithm to find a route that is 8-approximate with respect to the minimization of any symmetric norm of the visit times (including L_p-TSP, TFP, and TRP).*

Theorem 2 is proved in Section 3. Our algorithm builds on the partial covering idea, presented as Algorithm 1, that was pioneered by Blum et al. [BCC+94] for TRP, and was developed through subsequent studies [GK98, CGRT03, GGKT08, BKL21].

Algorithm 1 Routing via Partial Covering

```
procedure GEOMETRIC-COVERING(V, s, d)
    Algorithm Parameters: b ∈ (0, ∞), c ∈ (1, ∞)
    i ← 0
    while there remains destinations to visit do
                                   ▷ Conducting sub-tours
        C_i ← a maximal route of length ≤ b · c^i.
        Travel through C_i (and return to the origin)
        i ← i + 1
    return an ordering σ of V according to their
(first) visit time through the above loop.
```

The parameters b and c strongly affect the performance of the algorithm. For all-norm-TSP, we choose

$b = \min_{i,j \in V} d(i,j)$ and $c = 2$, as in [GGKT08]. The key difference of our algorithm is in line 6. Instead of an approximation algorithm for k-TSP, i.e. a route of minimum length that visits k-vertices, we utilize a milder relaxation, which is a tree (instead of a route/stroll) rooted at s, including k vertices, and of total length not larger than an optimal k-TSP. Such a *good k-tree* can be found in polynomial time using the primal-dual method that solves a Lagrangian relaxation of k-TSP [CGRT03, ALW08, PS14].

On the other hand, we provide a first impossibility result for all-norm-TSP, notably beyond known inapproximability bounds for specific L_p-TSP problems. The following is proved in Section 3.2.

THEOREM 3. *There is no approximation algorithm for all-norm-TSP with multiplicative factor better than* 1.78*, independent of $P = NP$ or other complexity hypotheses.*

The above result reaffirms the need for approximation algorithms specifically designed for each norm. Along this line, we present a randomized 5.65 approximation algorithm for the Traveling Firefighter Problem.

THEOREM 4. *There is a randomized, polynomial-time* 5.65*-approximation algorithm for TFP on general metrics.*

Theorem 4 is proved in Section 4, for which we adapt the ideas by Chaudhuri et. al. [CGRT03] and optimize the parameter c in line 2 of Algorithm 1. Choosing $b \in [1, c]$ at random, with a distribution of uniform density for $\log(b)$, simplifies the analysis, while one can efficiently de-randomize the algorithm by quantization of b.

1.2 Literature Review Traveling Salesman Problem is a principal problem in computer science, combinatorial optimization, and operations research, and its first formulations date back as early as 19^{th} century (c.f. [ABCC06]). Since the celebrated 3/2-approximation algorithm of Christofides-Serdyukov [Chr76, Ser78], for its tour-variant on general metrics, TSP has been extensively studied for half a century [Wol80, SW90, BP90, Goe95, CV00, GLS05, BC11, SWVZ12, HNR19]. Very recently, Karlin, Klein, and Oveis Gharan [KKG20] showed TSP can be approximated strictly better than 3/2, while the problem remains NP-hard to approximate within a factor of 123/122 [KLS15].

The common ground in numerous variants of TSP is that a set of vertices are to be visited in the *fastest possible* way, i.e., optimizing the time spent by the server/traveler. In contrast, the Traveling Repairman Problem (TRP), a.k.a. Minimum Latency Problem, the school bus driver problem [Wil94], hidden treasure [BCC$^+$94, KPY96, ALMS00], and the deliveryman problem [Min89, FLM93, MDZL08], optimizes the route purely from the perspective of clients, i.e., the total waiting time to be visited, and is another extensively studied combinatorial optimization problem [ACP$^+$86, PY93, BCC$^+$94, GK98, CGRT03, AW03] with a state-of-the-art approximation factor of $\simeq 3.59$ for general metrics [CGRT03, PS14].

Containment of fires can be abstracted from various perspectives. Hartnell [Har95] modeled a constant speed spread of fires through edges of a graph, along which the firefighters also displace. Many objectives, such as minimization of the number of burned vertices, or the required time to contain the fire(s) are studied in this model and the problem is an active area of research. See [FM09, KLL14, ABZ18, ABK20, DFH21] and references therein.

Generalizing the objective to Minkowski norm of the solution has allowed interpolating other classical problems, e.g., L_p set cover problem [GGKT08, BBFT20] that further united the greedy algorithms for set-cover and the minimum-sum-set-cover [FLT04] problems. Set cover is better approximable for $p = 1$ than $p = \infty$, while TSP ($p = \infty$) is currently better approximated than TRP ($p = 1$). Nevertheless, this order is not expected to be reversed as TRP is intrinsically a harder problem. Moreover, in contrast to the concordance among L_p set cover problems, for which the same greedy algorithm gives best (possible) bounds for any $p \in [1, \infty)$, state-of-the-art algorithms for TSP and TRP are significantly different, and potentially far from the best possible. This is yet another motivation to study L_p-TSP, ultimately towards unified best algorithms for all underlying problems.

1.3 Paper Organization. In Section 2 we provide a reduction from L_p-TSP to segmented-TSP and subsequent approximation schemes for Euclidean and weighted-tree metrics. In Section 3, we present the 8-approximation for all-norm-TSP in general metrics, along with a first inapproximability bound. This will also be a preliminary for optimized algorithm for L_2-TSP in Section 4, where we present a 5.65-approximation for the Traveling Firefighter Problem on general metrics. We discuss generalizations to multiple vehicle scenarios in Section 5 and conclude the paper with a set of interesting open problems to continue this line of research.

2 Reducing L_p-TSP to Segmented TSP

In this section, we provide our reduction from L_p-TSP to polynomially many instances of segmented-TSP. Let us restate Theorem 1.

THEOREM 1. *[L_p-TSP $\Rightarrow$ segmented-TSP] Let $\varepsilon > 0$ be a constant and $\mathcal{A}$ be an α-approximation algorithm for segmented-TSP for some $k = O(1+\varepsilon^{-2})$ of our choice. There is a $(1+\varepsilon)\cdot\alpha$-approximation algorithm for L_p-TSP that calls $\mathcal{A}$ (on the same network (V, s, d)) for a strongly polynomial number of times.*

In particular, a corollary of the above Theorem (and the following Lemma) is a $(1+\varepsilon)$ approximation algorithm for any L_p-TSP on weighted-tree metrics - where the problem becomes strongly NP-hard even for $p = 1$ - as well as the Euclidean plane.

LEMMA 2.1. ([SIT14]) *Segmented TSP, for any constant number of segments M, can be solved in polynomial time for weighted trees, and $1+\varepsilon$ approximated for unweighted Euclidean metric.*

COROLLARY 1.1. *There exist polynomial-time approximation schemes (PTAS) for L_p-TSP on weighted trees and Euclidean plane metrics.*

In the rest of this section, we prove Theorem 1, by providing a dynamic programming algorithm that approximates L_p-TSP using polynomially many calls to (approximate) segmented-TSP. More precisely, we show if there is an α-approximate solution for segmented-TSP, the dynamic program guarantees an approximation factor of at most $\alpha\cdot(1+\varepsilon)$ for arbitrary constant $\varepsilon > 0$.

The algorithm is presented in a few steps, each imposing no more than $1+O(\varepsilon)$ multiplicative error. To achieve exact $1+\varepsilon$ precision, one may run the algorithm for a constant fraction of the target ε.

For some k as large as $O(1+\varepsilon^{-2})$ we can ensure $c \stackrel{\text{def}}{=} (1+\varepsilon)^k \geq 3$. Let OPT^{λ_i} denote the maximal prefix of OPT route for L_p-TSP of length at most

$$\lambda_i \stackrel{\text{def}}{=} (1+\varepsilon)^{-j}\cdot c^i\,, \quad \forall i \geq 0\,,$$

where j is a fixed random number, uniformly distributed over $\{0,\cdots,k-1\}$.

Let OPT' be a tour made of sub-tours consisting of traversing OPT^{λ_i} and returning to the origin and waiting until time $3\lambda_i$ before starting the next sub-tour. To confirm the above is feasible, we need to show sub-tour $i+1$, being allowed to begin at $3\lambda_i$, does not leave before the return of previous sub-tour, i.e.,

$$3\lambda_{i-1} + 2\|T^{\mathrm{OPT}^{\lambda_i}}\|_\infty \leq \lambda_i + 2\lambda_i = 3\lambda_i$$

which is immediate having $\lambda_i = c\lambda_{i-1} \geq 3\lambda_{i-1}$.

LEMMA 2.2. *The modified tour OPT' is approximately optimal in expectation, i.e., for some $k \in O(1+\varepsilon^{-2})$*

$$\mathbb{E}_j\left[\|T^{OPT'}\|_p^p\right] \leq (1+\varepsilon)\|T^{OPT}\|_p^p\,.$$

Proof. All vertices are visited (for the first time) in the same order, by OPT and OPT'. Let the d^{th} service time by the optimal solution be

$$T_d^{\mathrm{OPT}} \in ((1+\varepsilon)^\delta, (1+\varepsilon)^{\delta+1}]$$

for some integer $\delta \geq 0$.

If this vertex is visited in the i^{th} sub-tour of OPT', we can write

$$T_d^{\mathrm{OPT}'} = T_d^{\mathrm{OPT}} + 3\lambda_{i-1}\,.$$

We can bound this additional delay by

$$T_d^{\mathrm{OPT}'} - T_d^{\mathrm{OPT}} \leq 3(1+\varepsilon)^{\delta-j'}$$

where j' has the same distribution as j.

We can prove the desired by bounding the per-vertex ratio by

$$\frac{\mathbb{E}_{j'}\left[(T_d^{\mathrm{OPT}'})^p\right]}{(T_d^{\mathrm{OPT}})^p} \leq (1+\varepsilon)$$

because $\frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \frac{a_i}{b_i}$ where $a_1,\cdots$ and $b_1,\cdots$ are positive real numbers.

Considering $p \geq 1$ and $T_d^{\mathrm{OPT}'} = T_d^{\mathrm{OPT}} + 3\lambda_{i-1}$, the left hand side of the target ratio is maximized for $T_d^{\mathrm{OPT}} = (1+\varepsilon)^\delta$, so it suffices to prove

$$\frac{\mathbb{E}_{j'}\left[((1+\varepsilon)^\delta + 3\lambda_{i-1})^p\right]}{((1+\varepsilon)^\delta)^p} \leq (1+\varepsilon)\,.$$

We will have

$$\begin{aligned}
&\frac{\mathbb{E}_{j'}\left[((1+\varepsilon)^\delta + 3\lambda_{i-1})^p\right]}{((1+\varepsilon)^\delta)^p}\\
&\leq \frac{\mathbb{E}_{j'}\left[((1+\varepsilon)^\delta + 3(1+\varepsilon)^{\delta-j'})^p\right]}{((1+\varepsilon)^\delta)^p}\\
&= \mathbb{E}_{j'}\left[(1+3(1+\varepsilon)^{-j'})^p\right]\\
&= 1 + \mathbb{E}_{j'}\left[(1+3(1+\varepsilon)^{-j'})^p - 1^p\right]\\
&\leq 1 + \frac{1}{k}\sum_{j'=0}^{k-1}(3p)^p(1+\varepsilon)^{-j'}\\
&\leq 1 + \frac{(3p)^p}{k}\cdot\frac{1}{1-(1+\varepsilon)^{-1}}\\
&= 1 + \frac{(3p)^p}{k}\cdot\frac{1+\varepsilon}{\varepsilon}\,.
\end{aligned}$$

To get the desired (from the last inequality) it suffices to assume

$$k \geq \frac{(3p)^p(1+\varepsilon)}{\varepsilon^2}\,.$$

□

We can now complete the proof of Theorem 1, similar to the main idea of Sitters [Sit14], presented as follows.

Lemma 2.2 implies for some $j \in [k]$, where $k = O(1+\varepsilon^{-2})$, there exists a near optimal routing, OPT', that for each $i \in [\tilde{O}(n^2)]$, visits *new vertices* only during $[3\lambda_{i-1}, \lambda_i]$, and returns to the origin and remains there until

$$3\lambda_i = \frac{3}{(1+\varepsilon)^j} \cdot (1+\varepsilon)^{ki}\,.$$

We can search for such a path by reconstructing OPT^{λ_i} for all i and upper bounding the consequent $\|T^{\mathrm{OPT}'}\|_p^p$ using dynamic programming.

Define $D[i][d]$ as (an upper bound on) the contribution of visit times of vertices that are visited by OPT^{λ_i}, to $\|T^{\mathrm{OPT}'}\|_p^p$, further assuming the number of these vertices is d.

We can compute $D[i][d]$ considering $O(n^k)$ cases of $(m_1, m_2, \cdots, m_k)$ where m_r denotes the number of vertices that are visited by OPT^{λ_i} during

$$(3\lambda_{i-1} + \lambda_i \cdot (1+\varepsilon)^{r-k-1}, 3\lambda_{i-1} + \lambda_i \cdot (1+\varepsilon)^{r-k}]\,.$$

Note that it is necessary to have $\sum_r m_r \leq d$ and let $d' = d - \sum_r m_r$ be the number of vertices visited by $\mathrm{OPT}^{\lambda_{i-1}}$. We can write

$$D[i][d] = \min_{m_1,\cdots,m_k} D[i-1][d'] + \mathrm{Seg\text{-}TSP}_i(d', m_{[r]}) \cdot \sum_r m_r \cdot (3\lambda_{i-1} + \lambda_i \cdot (1+\varepsilon)^{r-k})^p\,,$$

where $\mathrm{Seg\text{-}TSP}_i(d', m_{[r]})$ has value 1 if segmented-TSP is feasible for visiting at least

$$d', d'+m_1, \cdots, d'+m_1+\cdots+m_r$$

vertices by deadlines

$$\lambda_{i-1}, 3\lambda_{i-1} + \lambda_i \cdot (1+\varepsilon)^{-k}, \cdots, 3\lambda_{i-1} + \lambda_i$$

is feasible, and otherwise has value ∞. Note that we have an α approximate solver for Segmented-TSP, though for convenience we can alternatively assume the traveller goes at the speed of α instead of 1, to get a $1+\varepsilon$ approximate solution to $\|T^{\mathrm{OPT}}\|_p$ by $(D[\tilde{O}(n^2)][n])^{1/p}$. In the end, moving at unit speed (instead of α) at every stage can increase (any) norm of the delay vector $\|T\|$ by a factor α so we have an $\alpha \cdot (1+\varepsilon)$ approximation, that was promised by Theorem 1.

Finally it is worth to mention that the route (instead of the value) can be reconstructed using update (parent) information of $D[\cdot][\cdot]$ and a constructive approximate solver for Seg-TSP[$\cdot$] and we can short-cut potential re-visits of vertices to have a valid Hamiltonian route.

3 All-norm TSP

Since the introduction of a first constant approximation for TRP by Blum et al. [BCC+94], partial covering through applying a geometric series of limits on the length of the sub-tours has been a core in the design of routing algorithms. In this section, we improve the 16-approximate/competitive solution for all-norm TSP by a factor of 2. We further provide a first lower bound for this problem.

3.1 Approximation for General Metrics. The idea is to iteratively cover more and more vertices by sub-tours of exponentially (geometrically) increasing length while trying to maximize the total number of vertices that are visited (not necessarily for the first time) in each iteration. Our algorithm uses the following milder relaxation of k-TSP, called a *good k-tree*, in place of line 6 in Algorithm 1.

Definition 5. *A good k-tree is a tree of size k, including s, and with a total edge-weight of no more than that of the optimal k-TSP (starting from s).*

Lemma 3.1. ([CGRT03]) *A good k-tree can be found in polynomial time.*

Chaudhuri et. al. [CGRT03] proved the above using a primal-dual approach [Gar96, AK00] that allows finding a feasible solution to the primal (integer) linear program of the k-tree problem paired with a feasible dual solution to k-TSP, that by weak duality has no less of a cost.

We are now ready to prove Theorem 2.

theorem 2. *[all-norm-TSP] There is a polynomial-time algorithm to find a route that is 8-approximate with respect to the minimization of any symmetric norm of the visit times (including L_p-TSP, TFP, and TRP).*

Proof. WLOG assume the nearest neighbor to s is at distance 1. For $k = 1, 2, \ldots, n$ find a good-k-tree. Among these, name the largest tree (with respect to number of vertices) of total length at most 2^i as G_i for $i = 0, 1, 2, \ldots$.

Let C_i be a (randomized) depth-first traversal of G_i and let C be the concatenation of C_i's for $i = 0, \ldots$. The final tour ALG will visit vertices in the order that they appear in C, which does not increase the first-visit time for any vertex, due to triangle inequality of the metric, while short-cutting vertices that are being re-visited.

Let

$$T_k^{\mathrm{OPT}} \in [2^i, 2^{i+1}) .$$

This shows the shortest (length) k-path in G is no longer than 2^{i+1} . So our good-k-tree is no longer than 2^{i+1} , hence C_{i+1} has at least k distinct vertices, allowing us to upper bound our k^{th} visit time by

$$T_k^{\mathrm{ALG}} \leq \sum_{j=0}^{i+1} |C_j| \leq \sum_{j=0}^{i+1} 2 \times 2^j < 2^{i+3} .$$

Together with $T_k^{\mathrm{OPT}} \geq 2^i$ and the above inequality we have

$$T_k^{\mathrm{ALG}} \leq 8 \times T_k^{\mathrm{OPT}} .$$

We showed $T_k^{\mathrm{ALG}} \leq 8 \cdot T_k^{\mathrm{OPT}} \quad \forall i \in [n]$, i.e., T^{OPT} is 8-submajorized by T^{ALG} in terminology of [GGKT08, HLP88]. The result is that

$$\|T^{\mathrm{ALG}}\| \leq 8 \cdot \|T^{\mathrm{OPT}}\|$$

w.r.t. any norm $\|\cdot\|$. ∎

One can verify the above algorithm performs asymptotically 3 times worse than the optimal TRP for the example with service points at $\{2^i : i \in \mathbb{N}\}$ and starting at $x = 0$.

3.2 Inapproximability. We conclude this section by providing a lower bound for all-norm TSP. We show even for line metrics, an α-approximate all-norm TSP cannot be guaranteed in general, for $\alpha < 1.78$.

THEOREM 3. *There is no approximation algorithm for all-norm-TSP with multiplicative factor better than* 1.78*, independent of $P = NP$ or other complexity hypotheses.*

Proof. Let the metric correspond to absolute differences over the real line, and start the walk from the origin $x = 0$. Our example has a similar structure as follows. There is a single destination at $x = -1$, in addition to n destinations at $x \in \left\{b^i - 1 : i \in [n]\right\}$. Choosing $b = 1+\varepsilon$ such that $b^n \gg 1$, the optimal all-norm TSP visits the longer branch first, and is optimal with respect to TRP, while for TSP (which takes the shorter branch first) it achieves an an approximation ratio of

$$\frac{2b^n + 1}{b^n + 2} .$$

For $b = 1.001$ and $n = 2256$ the above setup provides a bound of 1.71 .

We further developed a similar, yet numerically verified example depicted as Figure 3, for which no 1.78-approximate all-norm-TSP route exits. For reproducibility we include this instance in the Appendix. ∎

The above example, is yet another motivation to study and optimize routing algorithms specific to the appropriate objective/norm, as one solution *cannot* be good for all.

4 Traveling firefighter in general metrics

In this section we build upon our geometric partial-covering algorithm with good k-trees to improve approximation bound for a specific norm, i.e., the Traveling Firefighter Problem.

We achieve the improved approximation bound by randomization (of parameter b) and optimization of our analysis w.r.t. approximation guarantee for a specific norm. Our approach can provide approximation guarantees (better than 8) for other L_p-TSP problems. We present main ideas in the rest of the section by proving the following result.

THEOREM 4.1. *Traveling Firefighter Problem for general metrics can be* 5.641*-approximated in polynomial time.*

Proof. We present a randomized approximation algorithm for general metrics, that can be efficiently derandomized.

Among the set of good k-trees, pre-computed for all k , let G_i be the largest one of total length at most $b \cdot c^i$, and let C_i be a depth first traversal of that. Parameter $c > 1$ is a constant, to be optimized for performance guarantees, and

$$[1, c] \ni b = c^U$$

where U is a random variable distributed uniformly over the interval $[0, 1]$. Finally, we reverse each C_i with probability half, and concatenate C_i's (and shortcut repeated visits) to achieve the output ordering ALG .

Let the latency of the k^{th} vertex visited by the optimal route be

$$T_k^{\mathrm{OPT}} = ac^i ,$$

for some $a \in [1, c]$ and integer $i \geq 0$.

It is easy to see that our $(i + \mathbb{1}[a \geq b])^{\mathrm{th}}$ sub-tour contains at least k vertices, hence, we can bound

$$T_k^{\mathrm{ALG}} \leq X_k + \sum_{j=0}^{i-\mathbb{1}[a<b]} 2bc^j ,$$

Figure 3: An example with no better than 1.78 all-norm TSP

Where X_k is zero if our tour visits its k^{th} vertex before the $(i - \mathbb{1}[a < b])^{\text{th}}$ sub-tour, and

$$X_k \in [0, 2bc^{i+\mathbb{1}[a \geq b]}],$$

depending on its location in the sub-tour.

Due to convexity of the quadratic function we can bound the expected damage, with

$$\begin{aligned}
&\mathbb{E}\left[(T_k^{\text{ALG}})^2\right] \\
&\leq \frac{1}{2}\left(\sum_{j=0}^{i-\mathbb{1}[a<b]} 2bc^j\right)^2 \\
&+ \frac{1}{2}\left(2bc^{i+1-\mathbb{1}[a<b]} + \sum_{j=0}^{i-\mathbb{1}[a<b]} 2bc^j\right)^2 \\
&= \frac{1}{2}\left(2b\frac{c^{i+\mathbb{1}[a\geq b]}-1}{c-1}\right)^2 \\
&+ \frac{1}{2}\left(2b\frac{c^{i+1+\mathbb{1}[a\geq b]}-1}{c-1}\right)^2 \\
&\leq \frac{1}{2}\left(\frac{2bc}{c-1}\right)^2 c^{2(i+\mathbb{1}[a\geq b])} \\
&= c^{2i}\cdot\frac{2c^2}{(c-1)^2}\left(b^2c^{2\cdot\mathbb{1}[a\geq b]}\right).
\end{aligned}$$

Bounding the expected damage at the i^{th} service, considering random variable b we have

$$\begin{aligned}
&\mathbb{E}\left[(T_k^{\text{ALG}})^2\right] \\
&\leq \mathbb{E}\left[c^{2i}\cdot\frac{2c^2}{(c-1)^2}\left(b^2c^{2\mathbb{1}[a\geq b]}\right)\right] \\
&= c^{2i}\cdot\frac{2c^2}{(c-1)^2}\left(c^2\int_0^{\log_c d} c^{2U}dU + \int_{\log_c d}^1 c^{2U}dU\right) \\
&= c^{2i}\frac{2c^2}{(c-1)^2}\left(\frac{c^2(a^2-1)}{\ln c} + \frac{c^2-a^2}{\ln c}\right) \\
&= (ac^i)^2\left(\frac{2c^2\cdot(c^2-1)}{(c-1)^2\ln c}\right) \\
&= (T_k^O)^2\left(\frac{2c^2(c+1)}{(c-1)\ln c}\right).
\end{aligned}$$

We can now choose c in order to minimize the multiplicative bound

$$\frac{c+1}{c-1}\cdot 2c^2/\ln c \leq 31.82$$

that can be achieved for $c \simeq 2.54$. With this we have

$$\mathbb{E}\left[\|T^{\text{ALG}}\|_2^2\right] \leq 31.82\|T^{\text{OPT}}\|_2^2.$$

We provided a randomized algorithm, that along law of large numbers, can be applied in practice, by repeating the algorithm and reporting the best performing route. On the other hand it can be simply de-randomized by exploring all values for a dense enough quantization of b. In the first case we will have a $\sqrt{31.82} \simeq 5.641$ approximate L_2-TSP with high probability, and in latter scenario we will have deterministic result with negligible additional approximation error. □

5 Generalizations to Multiple Vehicles

In many applications we have multiple vehicles, potentially dispatching from different hubs [KP20].

Our reduction of L_p-TSP to Segmented-TSP can be generalized to multiple vehicles, with arbitrary start locations. Similar to the approach in Section 2 we can convert any optimal multi-vehicle solution to repetition of $O(\varepsilon^{-2})$ of prefix routes for each vehicle, all synchronized with a single $j \in \{0, \cdots, k-1\}$, picked uniformly at random. Lemma 2.2 can be subsequently adapted to allow limiting the search space to solutions that have all vehicles at starting locations simultaneously at all $3\lambda_i$, with negligible degrade of the optima.

Finally, adapting the dynamic programming, we can guarantee a multiplicative $O(\varepsilon)$ loss given an (approximate) solver for multi-vehicle segmented TSP with constant $O(\varepsilon^{-2})$ deadlines, and the corresponding total number of destinations to be visited (by at least one vehicle) until up to each. This is indeed fruitful as results on segmented-TSP also generalize to multi-vehicle variant, e.g., in tree-distance or Euclidean metric.

Our algorithms can be similarly adapted in the case where release dates are added for the destinations.

Generalizing results in Sections 3-4 to multi-vehicle variants of the problems is also possible. For this purpose, the k-stroll subproblem, for which we used the mild relaxation of good k-tree, can be generalized to bottleneck-stroll of Post and Swamy [PS14]. This will be at the cost of further degrades to the approximation constants and proposes interesting open problems for study. For general metrics, current best approximation

guarantees for multi vehicle $\{\text{single}, \text{multi}\}$ depot L_1-TSP is $\{7.183, 8.497\}$ [PS14].

Discussion and Open Problems

We studied combinatorial optimization problems whose objectives can be more appropriate, efficient, fair, and adjustable, depending on enormous applications of optimal routing/scheduling.

For TSP and TRP, the analyses of approximation algorithms as well as complexity results heavily rely on the linearity of the objective function. Hence, TFP and more generally L_p-TSP pose further challenging problems and require new techniques to be developed.

We developed two approaches, towards high precision and scalable approximation of the answer.

First, we provided a high precision polynomial time reduction of L_p-TSP to segmented-TSP with only a constant number of deadlines for visiting the required number of destinations. Our reduction enables approximation schemes for L_p-TSP on Euclidean as well as weighted tree metrics; this is yet another motivation to further study the segmented-TSP problem.

The other approach relied mainly on the fact that the objective is a norm of the delay vector. In this line, we developed an algorithm for all-norm-TSP, on general metrics, of approximation factor 8. We also provided a first inapproximability result for all-norm-TSP.

Last but not least, we showed how the performance of the latter algorithm can be optimized for a specific norm, particularly approximating Traveling Firefighter Problem within a factor 5.65.

In the end, in addition to further improving the approximation bounds for the problems under study, we mention but a few of many interesting open problems and potential research directions regarding L_p-TSP and all-norm-TSP.

- L_1-TSP, i.e., TRP is harder than L_∞-TSP, at least on trees. For what p is the L_p-TSP problem the hardest?
- While TRP is strongly NP-hard on weighted trees, its complexity is unknown for caterpillars [Sit02]. Similarly TFP and L_p-TSP seem challenging even on such fundamental examples, that is yet to be resolved.
- While we theoretically claimed $p = 2$ to be ideal for the Traveling Firefighter Problem, this assumption should be given further justification / investigation in practice.
- Further applications of L_p-TSP can be inspected, e.g. in optimal containment of spread of pandemics [Har04, TCM20].
- We observe $c = 2$ to be optimal for the analysis of all-norm-TSP algorithm. For TRP, the current best result is due to a base $c \approx 3.59$ for the geometric series, while $c \approx 2.54$ is better for TFP, as we discussed. In this vein, one can inspect the best base for the geometric series used by the partial covering algorithm, depending on the objective, which naturally seems to be non-increasing on p.
- Stronger impossibility results for all-norm-TSP and even larger hardness of approximation bounds for this problem seem plausible.

Acknowledgements

We thank anonymous reviewers for valuable comments.

References

[ABCC06] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook, *The traveling salesman problem: a computational study*, Princeton university press, 2006.

[ABK20] Gideon Amir, Rangel Baldasso, and Gady Kozma, *The firefighter problem on polynomial and intermediate growth groups*, Discrete Mathematics **343** (2020), no. 11, 112077.

[ABZ18] David Adjiashvili, Andrea Baggio, and Rico Zenklusen, *Firefighting on trees beyond integrality gaps*, ACM Transactions on Algorithms (TALG) **15** (2018), no. 2, 1–33.

[ACP+86] Foto Afrati, Stavros Cosmadakis, Christos H Papadimitriou, George Papageorgiou, and Nadia Papakostantinou, *The complexity of the travelling repairman problem*, RAIRO-Theoretical Informatics and Applications **20** (1986), no. 1, 79–87.

[AERW04] Yossi Azar, Leah Epstein, Yossi Richter, and Gerhard J Woeginger, *All-norm approximation algorithms*, Journal of Algorithms **52** (2004), no. 2, 120–133.

[AK00] Sanjeev Arora and George Karakostas, *A 2+ approximation algorithm for the k-MST problem*, Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 2000, pp. 754–759.

[ALMS00] Giorgio Ausiello, Stefano Leonardi, and Alberto Marchetti-Spaccamela, *On salesmen, repairmen, spiders, and other traveling agents*, Italian Conference on Algorithms and Complexity, Springer, 2000, pp. 1–16.

[ALW08] Aaron Archer, Asaf Levin, and David P Williamson, *A faster, better approximation algorithm for the minimum latency problem*, SIAM Journal on Computing **37** (2008), no. 5, 1472–1498.

[AW03] Aaron Archer and David P Williamson, *Faster approximation algorithms for the minimum latency problem*, Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2003, pp. 88–96.

[BBFT20] Nikhil Bansal, Jatin Batra, Majid Farhadi, and Prasad Tetali, *Improved approximations for min sum vertex cover and generalized min sum set cover*, arXiv preprint arXiv:2007.09172 (2020).

[BC11] Sylvia Boyd and Robert Carr, *Finding low cost TSP and 2-matching solutions using certain half-integer subtour vertices*, Discrete Optimization **8** (2011), no. 4, 525–539.

[BCC+94] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan, *On the minimum latency problem*, Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, pages 163–171. ACM (1994).

[BKL21] Marcin Bienkowski, Artur Kraska, and Hsiang-Hsuan Liu, *Traveling repairperson, unrelated machines, and other stories about average completion times*, arXiv preprint arXiv:2102.06904 (2021).

[Bor19] Alejandra Borunda, *See how much of the Amazon forest is burning, how it compares to other years*, National Geographic (2019).

[BP90] Sylvia C Boyd and William R Pulleyblank, *Optimizing over the subtour polytope of the travelling salesman problem*, Mathematical programming **49** (1990), no. 1-3, 163–187.

[BP03] Nikhil Bansal and Kirk Pruhs, *Server scheduling in the lp norm: a rising tide lifts all boat*, Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, 2003, pp. 242–250.

[CGRT03] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar, *Paths, trees, and minimum latency tours*, 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings., IEEE, 2003, pp. 36–45.

[Chr76] Nicos Christofides, *Worst-case analysis of a new heuristic for the Travelling Salesman Problem*, Tech. Report RR-388, February 1976.

[CV00] Robert Carr and Santosh Vempala, *Towards a 4/3 approximation for the asymmetric traveling salesman problem*, 116–125.

[DFH21] Arye Deutch, Ohad Noy Feldheim, and Rani Hod, *Multi-layered planar firefighting*, arXiv preprint arXiv:2105.03759 (2021).

[FLM93] Matteo Fischetti, Gilbert Laporte, and Silvano Martello, *The delivery man problem and cumulative matroids*, Operations Research **41** (1993), no. 6, 1055–1064.

[FLT04] Uriel Feige, László Lovász, and Prasad Tetali, *Approximating min sum set cover*, Algorithmica **40** (2004), no. 4, 219–234.

[FM09] Stephen Finbow and Gary MacGillivray, *The firefighter problem: a survey of results, directions and questions.*, Australas. J Comb. **43** (2009), 57–78.

[Gar96] Naveen Garg, *A 3-approximation for the minimum tree spanning k vertices*, Proceedings of 37th Conference on Foundations of Computer Science, IEEE, 1996, pp. 302–309.

[GGKT08] Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan, *All-norms and all-l_p-norms approximation algorithms*, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008.

[GK98] Michel Goemans and Jon Kleinberg, *An improved approximation ratio for the minimum latency problem*, Mathematical Programming **82** (1998), no. 1-2, 111–124.

[GLS05] David Gamarnik, Moshe Lewenstein, and Maxim Sviridenko, *An improved upper bound for the TSP in cubic 3-edge-connected graphs*, Operations Research Letters **33** (2005), no. 5, 467–474.

[Goe95] Michel X Goemans, *Worst-case comparison of valid inequalities for the TSP*, Mathematical Programming **69** (1995), no. 1-3, 335–349.

[Har95] Bert Hartnell, *Firefighter! an application of domination*, the 24th Manitoba Conference on Combinatorial Mathematics and Computing, University of Minitoba, Winnipeg, Cadada, 1995, 1995.

[Har04] Stephen G Hartke, *Attempting to narrow the integrality gap for the firefighter problem on trees.*, Discrete Methods in Epidemiology, 2004, pp. 225–231.

[HLP88] GH Hardy, JE Littlewood, and G Polya, *Inequalities cambridge univ*, Press, Cambridge (1988).

[HNR19] Arash Haddadan, Alantha Newman, and R Ravi, *Shorter tours and longer detours: uniform covers and a bit beyond*, Mathematical Programming (2019), 1–29.

[Hoo18] K Hoover, *Wildfire statistics congressional research service*, 2018.

[KKG20] Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan, *A (slightly) improved approximation algorithm for metric TSP*, arXiv preprint arXiv:2007.01409 (2020).

[KLL14] Rolf Klein, Christos Levcopoulos, and Andrzej Lingas, *Approximation algorithms for the geometric firefighter and budget fence problems*, Latin American Symposium on Theoretical Informatics, Springer, 2014, pp. 261–272.

[KLS15] Marek Karpinski, Michael Lampis, and Richard Schmied, *New inapproximability bounds for tsp*, Journal of Computer and System Sciences **81** (2015), no. 8, 1665–1677.

[KP20] Miroslav Kulich and Libor Preucil, *Multi-robot search for a stationary object placed in a known environment.*

[KPY96] Elias Koutsoupias, Christos Papadimitriou, and Mihalis Yannakakis, *Searching a fixed graph*, International Colloquium on Automata, Languages, and Programming, Springer, 1996, pp. 280–289.

[KRT99] Jon Kleinberg, Yuval Rabani, and Éva Tardos, *Fairness in routing and load balancing*, 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039), IEEE, 1999, pp. 568–578.

[MDZL08] Isabel Méndez-Díaz, Paula Zabala, and Abilio Lucena, *A new formulation for the traveling deliveryman problem*, Discrete applied mathematics **156** (2008), no. 17, 3223–3237.

[Min89] Edward Minieka, *The delivery man problem on a tree network*, Annals of Operations Research **18** (1989), no. 1, 261–266.

[NAS] *Fire information for resource management system, NASA*, https://firms.modaps.eosdis.nasa.gov.

[PS14] Ian Post and Chaitanya Swamy, *Linear programming-based approximation algorithms for multi-vehicle minimum latency problems*, Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2014, pp. 512–531.

[PY93] Christos H Papadimitriou and Mihalis Yannakakis, *The traveling salesman problem with distances one and two*, Mathematics of Operations Research **18** (1993), no. 1, 1–11.

[Ser78] AI Serdyukov, *O nekotorykh ekstremal'nykh obkhodakh v grafakh*, Upravlyayemyye sistemy **17** (1978), 76–79.

[Sit02] René Sitters, *The minimum latency problem is NP-hard for weighted trees*, International conference on integer programming and combinatorial optimization, Springer, 2002, pp. 230–239.

[Sit14] ———, *Polynomial time approximation schemes for the traveling repairman and other minimum latency problems.*, Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms, SIAM, 2014, pp. 604–616.

[SW90] David B Shmoys and David P Williamson, *Analyzing the held-karp TSP bound: A monotonicity property with application*, Information Processing Letters **35** (1990), no. 6, 281–285.

[SWVZ12] Frans Schalekamp, David P Williamson, and Anke Van Zuylen, *A proof of the Boyd-Carr conjecture*, Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms, SIAM, 2012, pp. 1477–1486.

[TCM20] Guy Tennenholtz, Constantine Caramanis, and Shie Mannor, *Sequential vaccination for containing epidemics*, medRxiv (2020).

[Wil94] Todd Gerald Will, *Extremal results and algorithms for degree sequences of graphs.*

[Wol80] Laurence A Wolsey, *Heuristic analysis, linear programming and branch and bound*, Combinatorial Optimization II, Springer, 1980, pp. 121–134.

A From Section 3

Our example for 1.78 inapproximability of all-norm-TSP has similar structure as the exponential sequence presented in Section 3 achieving the lower bound of 1.71, yet is computationally tuned and verified. This suggests the best (worst) example can have a different structure. Follows the locations of the destinations over the line to be visited by a traveler, starting at $x = 200$. All points are over the x-axis.

$$\begin{aligned} V = \{&0, 200, 202, 204, 206, 208, 210,\\ &212, 214, 216, 217, 218, 219, 220, 221,\\ &222, 223, 224, 225, 226, 228, 230, 232,\\ &234, 236, 238, 240, 242, 244, 246, 250,\\ &254, 258, 262, 266, 270, 274, 278, 282,\\ &286, 289, 292, 295, 298, 301, 304, 307,\\ &310, 313, 316, 316, 316, 316, 316, 316,\\ &316, 316, 316, 316, 316, 322, 328, 334,\\ &340, 346, 352, 358, 364, 370, 376, 382,\\ &388, 394, 400, 406, 412, 418, 424, 430,\\ &436, 446, 456, 466, 476, 486, 496, 506,\\ &516, 526, 536, 540, 544, 548, 552, 556,\\ &560, 564, 568, 572, 576, 595, 614, 633,\\ &652, 671, 690, 709, 728, 747, 766, 775,\\ &784, 793, 802, 811, 820, 829, 838, 847,\\ &856, 888, 920, 952, 984, 1016, 1048,\\ &1080, 1112, 1144, 1176, 1199, 1222,\\ &1245, 1268, 1291, 1314, 1337, 1360,\\ &1383, 1406, 1519, 1632, 1745, 1858,\\ &1971, 2084, 2197, 2310, 2423, 2536\} \end{aligned}$$

Search and evacuation with a near majority of faulty agents*

Jurek Czyzowicz† Ryan Killick‡ Evangelos Kranakis§ Grzegorz Stachowiak¶

Abstract

There are $n \geq 3$ unit speed mobile agents placed at the origin of the infinite line. In as little time as possible, the agents must find and evacuate from an exit placed at an initially unknown location on the line. The agents can communicate in the wireless mode in order to facilitate the evacuation (i.e. by announcing the target's location when it is found). However, among the agents are a subset of at most f crash faulty agents who may fail to announce the target when they visit its location.

In this paper we study this aforementioned problem for the specific case that $n = 2f + 1$. We introduce a novel type of search algorithm and analyze its competitive ratio – the supremum, over all possible target locations, of the ratio of the time the agents take to evacuate divided by the initial distance between the agents and the target. We demonstrate that the competitive ratio of evacuation is at most 7.437011 for $(n, f) = (3, 1)$; at most 7.253767 for $(n, f) = (5, 2)$ and $(7, 3)$; and at most 7.147026 for $(n, f) = (9, 4)$. For larger values of $n = 2f + 1$ we prove an asymptotic upper bound of $4 + 2\sqrt{2}$. We also adapt our evacuation algorithm for $(n, f) = (3, 1)$ to the problem of search by three agents with one byzantine fault, i.e. the faulty agent may also lie about finding the target. In doing so we improve the best known upper bound on this search problem from 8.653055 to 7.437011.

1 Introduction

Problems of search and exploration are central to many areas of computer science and mathematics and, accordingly, have received much attention in the literature. Perhaps the simplest search type problem considers the optimal trajectory of a single mobile agent tasked with finding a target placed at an unknown location on the infinite line. The goal of the agent is to minimize the competitive ratio. Independently studied by Bellman [9] and Beck [6] in the 1960's, it is now well known that the optimal trajectory for this single agent search uses a doubling strategy whereby the agent, starting at the origin, moves between points on the line at alternating positions $1, -2, 4, -8, \ldots$. It is a simple task to show that this trajectory ensures a competitive ratio of 9.

Search by multiple agents on the line is a natural extension of the single agent search. Of course, with more than one agent also arise questions about how search is affected by the presence of agents with differing capabilities/attributes. For example, one can consider agents with different speeds, and or communication abilities. A particularly interesting and important topic in group search is the development of fault tolerant search algorithms.

In this paper we study a version of fault tolerant group search on the line. Specifically, we consider the problem of *evacuation* by $n = 2f + 1$ mobile agents when at most f of these agents are faulty. The agents all begin the search at the same time from a common location and the goal is for the agents to find and exit from a target placed at an unknown location on the line. To achieve this goal the agents can co-operate by exchanging messages with one another in wireless mode (i.e. instantaneously and across any distance). This goal is impeded by the presence of f crash faulty agents who may fail to announce a detected target.

We use our results from the crash evacuation problem to improve upper bounds on the problem of search by three agents at most one of which is *byzantine* faulty. A byzantine faulty agent is similar to a crash faulty agent except that it can also lie about finding the target. When $n = 2f + 1$ it can happen that all agents are required to reach the target in order for the search to complete and so crash evacuation can be viewed as a sub-problem of the more difficult byzantine search problem.

1.1 Model We have $n = 2f + 1$ mobile agents with at most f of them faulty. Agents all begin at a common location referred to as the origin and the agents can move up to a maximum unit speed in either the positive direction (referred to as moving to the right) or the negative direction (referred to as moving to the left). An agent may change its travel direction arbitrarily often and with no associated time cost.

*Supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

†Départemant d'informatique, Université du Québec en Outaouais, Gatineau, Canada.

‡School of Comp. Sci., Carleton University, Ottawa, Canada.

§School of Comp. Sci., Carleton University, Ottawa, Canada.

¶Institute of Comp. Sci., University of Wroclaw, Wroclaw, Poland.

The agents are labelled with unique identifiers taken from the set $\{0, \ldots, n-1\}$ and can communicate with each other in the wireless mode. A parallel search algorithm specifies a unique trajectory for each agent and all agents are assumed to have full knowledge of these trajectories. It follows that the only kind of message broadcast by an agent will be a notification that it has detected the target at its current location. If an agent does not broadcast a message while visiting a location then it is assumed that the agent did not detect the target at that location.

Each agent is aware of the number f of faults, however, the identity of the faulty agents is unknown. The fault model considered for the agents is that of crash or silent faults. In this model an agent may fail to announce the target when it is detected, however it cannot send a message falsely claiming that it has found the target when it has not (this is known as the byzantine model). The presence of faulty agents thus implies that an agent cannot necessarily trust that the target is not at a location previously visited by another agent. In order to be sure that a target is not at a particular location x, it will be required that the location x has been visited by at least one provably reliable agent. With at most f faults, at least $f+1$ agents must visit x in order to have this guarantee.

It is possible that the faulty agents do not follow the trajectories assigned to them. However, all agents are aware of the trajectories of the other agents, and any agent that is found to be not following its assigned trajectory can be reliably identified as faulty. We will assume that the identity and behavior of the faulty agents is controlled by an adversary who will always act in a way to maximize the competitive ratio. We may therefore safely assume that any such premature identification will not occur. Each agent will therefore follow the trajectory initially assigned to it until they either find the target or they receive an announcement that the target has been found. Since announcements can always be trusted, agents will immediately move to the announced location in order to complete the evacuation.

1.2 Preliminaries and notation We begin with the following definitions which apply to a parallel search algorithm for $n = 2f+1$ agents, f of which are faulty.

DEFINITION 1.1. *The evacuation time E_f^x is the worst case time required for all reliable agents to reach a target at location x.*

DEFINITION 1.2. *The search time S_f^x is the worst case time until the first reliable agent reaches x.*

DEFINITION 1.3. *The competitive ratio $R_f = \sup_x \frac{E_f^x}{|x|}$ represents the worst case ratio of the evacuation time to the lower bound $|x|$ on the time required to find the target.*

Since crash faulty agents fail silently (i.e. they cannot lie about finding the target), it follows that any announcement made by an agent must be truthful. As a result the only sensible thing for the (reliable) agents to do once an announcement has been made is to immediately move to the announced target's location. This observation has two important implications. First, it implies that we can define a parallel evacuation algorithm entirely by the trajectories of the agents. Second, it implies that we can express the evacuation time as the sum of S_f^x and the distance between the target and the agent most distant from the target at the time S_f^x. This last point leads us to make the following definition.

DEFINITION 1.4. *Define i_f^x and Δ_f^x as the identity of, and distance between, the agent most distant from location x at the time S_f^x.*

With this definition we can express the evacuation time as follows

$$E_f^x = S_f^x + \Delta_f^x. \tag{1.1}$$

We consider agent trajectories defined by sequences of turning points – points on the line at which agents change their movement direction, and between which the agents move at constant unit speed. We use the notation $d_{i,j}$ to refer to the turning point j of agent i. We will assume that the turning points alternate on either side of the origin with increasing absolute values, i.e. if $d_{i,j} > 0$ then $d_{i,j+1} < 0$ and $|d_{i,j+1}| > |d_{i,j}|$.

For these types of trajectories one must make additional assumptions in order to achieve a constant competitive ratio. To see why this is, imagine we have a set of trajectories with first turning points $d_{i,0}$ and assume that for the majority of the agents we have $d_{i,0} > \delta$ for some $\delta > 0$. Then the target can be placed at location $-\epsilon$ with $\epsilon > 0$ arbitrarily small and all agents that initially moved to the left are made to be faulty. The first time a reliable agent can reach the target is then 2δ and the competitive ratio is at least $2\delta/\epsilon$.

To overcome this problem one usually makes the assumption that the agents are aware of a lower bound on the distance to the target. Then, by making the first turning points much smaller than this lower bound, a finite competitive ratio is possible. Alternatively, one can assume that the agents do not have first turning points. In other words, one assumes that the turning point sequence $d_{i,j}$ extends to $j = -\infty$ and

the agents have always been moving. Although less realistic, we find the latter assumption to be more elegant mathematically and we will take this approach here.

We end this section with a lemma which specifies how the target will be placed in the worst case.

LEMMA 1.1. *The supremum of $E_f^x/|x|$ always occurs when x is a turning point.*

1.3 Related work Search problems are optimization problems generally concerned with minimizing the time required for a set of mobile agents to find a hidden target in a given environment. One usually assumes that the environment is known in advance and the focus is on studying the effects on the search time under different assumptions on the agent capabilities. Searching in an unknown environment implies exploration where quite often there are additional/alternative goals the agents are required to achieve, e.g. mapping and/or positioning the searchers within the environment [1, 2, 16, 20].

When the environment is known, search is a pure optimization problem. The study of search by a single agent on the infinite line was initiated independently by Bellman [9] and Beck [6, 7, 8] where, among other things, the authors demonstrate the now well known result that a single searcher cannot find a hidden target at initial distance d from the searcher in time less than $9d$. The work by Bellman and Beck gave rise to a number of variants of search on the line. Notable is the work of Heath and Fristedt [18], Fristedt [17], and Gal [19]. Also notable is the works of Baeza-Yates et. al. [3, 4] where, among other things, the authors study problems of search by agents in environments different from the line, e.g. in the plane or at the origin of w concurrent rays (known as the "Lost Cow" problem). Group search was initiated in [10] where the problem of evacuation by multiple agents that can communicate face-to-face was studied. More recently, search on the line was considered when: the agents have distinct speeds [5]; turning costs are included [15]; the concern is to minimize the energy consumed during the search [11, 12].

Search on the line with possibly faulty searchers was initiated in [14] wherein the authors introduce optimal trajectories – the *proportional schedules* – for search by n agents at most f of which are crash faulty. This work is particularly relevant to the problem we study here. It should also be noted that the optimality of the proportional schedules for search was only established at a later time in [21]. Search with byzantine faults was first studied in [13] wherein the authors prove a number of lower bounds and upper bounds on the problem. Many of these upper bounds were later improved in [22], where, in particular, the authors demonstrate that the proportional schedules of [14] can be used to achieve an upper bound of 8.653055 on the problem of search by three agents, one of which is byzantine faulty.

1.4 Results and outline Our main result is the development and analysis of a novel search type algorithm for the evacuation problem with crash faulty agents. A summary of the resulting upper bounds are listed in Table 1 along with the best known lower bounds. We also prove an asymptotic upper bound on the evacuation by $n = 2f+1$ agents of $4+2\sqrt{2}$ (Theorem 2.2) and improve the upper bound on search by three agents, at most one of which is byzantine faulty, from 8.653055 to 7.437011 (Theorem 4.1). The best known lower bound on this search problem is 5.233069.

Table 1: Best known upper and lower bounds on the competitive ratio of evacuation by $n = 2f+1$ agents. Upper bounds are due to this work. Lower bounds are due to [14] and [21].

n	f	UB	LB	Theorem (this work)
3	1	7.437011	5.233069	3.2
5	2	7.253767	4.434326	3.1
7	3	7.253767	4.076343	3.1
9	4	7.147026	3.870110	3.1

In Section 2.1 we analyze the competitive ratio of evacuation for the proportional schedules – a family of trajectories first developed in [14] for the purpose of search by crash faulty agents. We use this section to prove our asymptotic upper bound (Theorem 2.2) and also to build intuition on how we can improve upon these trajectories. In Section 3 we introduce a generalization of the proportional schedules and analyze separately the cases that $f > 1$ (Subsection 3.1) and $f = 1$ (Subsection 3.2). In Section 4 we show that our evacuation algorithm for three agents also leads to an improvement on the competitive ratio of search by three agents, one of which is byzantine faulty. Finally, in Section 5 we conclude with a brief discussion of open problems. Due to space considerations, the proofs for many of the more straightforward lemmas/theorems are not included.

2 Crash faulty evacuation

With the evacuation time expressed as in (1.1) it is clear that in order to optimize the evacuation time one needs to consider a trade-off between the search time S_f^x and the distance Δ_f^x. This is in contrast to the normal search problem which aims only to optimize S_f^x. Nevertheless, one can imagine that an algorithm that optimizes S_f^x would still provide a good starting point for studying

the evacuation problem. Since it just so happens that an optimal algorithm for crash faulty search is known, we will use this approach to study the evacuation problem.

2.1 Proportional schedules An optimal algorithm for the crash faulty search problem was introduced in [14]. This algorithm is referred to as a *proportional schedule* and is defined by the collection of n trajectories represented by the sequences of turning points $d_{i,j} = r^{2i/n}(-r)^j$ where $r > 1$ is a real number parameter. Figure 1 depicts an example proportional schedule for the case $(n, f) = (5, 2)$ using a space-time diagram which plots an agent's position on the x axis with time on the y-axis. Our strategy for analyzing this algorithm derives from equation (1.1), i.e. we will compute $S_f^x = S_f^x(r)$ and $\Delta_f^x = \Delta_f^x(r)$. Our goal will be to prove the following Theorem 2.1.

THEOREM 2.1. *The proportional schedules have competitive ratio* $1 + \frac{2r}{r+r^{2/n}} + \frac{4r^{2+1/n}}{(1+\epsilon)(r+r^{2/n})(r-1)} \le R_f < 1 + \frac{2r}{r+1} + \frac{4r^{2+1/n}}{(r+1)(r-1)}$ *for all* $\epsilon > 0$.

We make note of the following properties of the turning-points $d_{i,j}$ which hold for any agent i, turning point j, and integer k: $d_{i+k,j} = r^{2k/n} d_{i,j}$, and $d_{i,j+k} = (-r)^k d_{i,j}$. Particularly useful is the fact that

$$d_{i+kn,j} = r^{2k} d_{i,j} = d_{i,j+2k} \tag{2.2}$$

which allows us to refer to a turning-point j of an agent with "label" $i \ge n$ or $i < 0$ with the understanding that we are actually referring to a later/earlier turning-point of agent $i \mod n$.

LEMMA 2.1. *Define* $t_{i,j}$ *as the first time at which agent* i *reaches location* $d_{i,j}$. *Then* $t_{i,j} = \frac{r+1}{r-1}|d_{i,j}|$.

We observed in the caption of Figure 1 that the turning points of the agents all lie along a common cone. This is evidenced by the ratio $t_{i,j}/|d_{i,j}| = \frac{r+1}{r-1}$ being independent of i and j. We will make use of this property shortly.

We define the semi-open intervals $I_{i,j}$ as follows.

DEFINITION 2.1. $I_{i,j} := (d_{i,j}, d_{i+1,j}] = (1, r^{2/n}] \cdot d_{i,j}$.

The sequence of intervals $[I_{i,j}]_{i=-\infty}^{\infty}$ with j even (resp. j odd) covers the entire positive (resp. negative) half-line without overlap. Thus, for any fixed j and position x there exists a unique integer i for which the position $x \in I_{i,j}$. Using the property (2.2), we can equivalently say that there exists a unique j and $i \in \{0, \dots, n-1\}$ for which $x \in I_{i,j}$. Note also that the symmetry inherent to the trajectories implies that for a fixed $z \in (1, r^{2/n}]$ the competitive ratio when the target is placed at $x = z d_{i,j} \in I_{i,j}$ will not depend on i or j. The next lemma will allow us to compute the search time of the algorithm.

LEMMA 2.2. *Agent* $i + k$, $k = 1, \dots, n$, *first visits* $z d_{i,j} \in I_{i,j}$ *at time* $T_{i,j,k}(z) = \left(z + \frac{2r^{2k/n}}{r-1}\right)|d_{i,j}|$.

We now turn our attention to the distance Δ_f^x. We will only bound this distance and to do this we inspect the space-time points at which the trajectories of the agents intersect.

LEMMA 2.3. *Let* $(\rho_{i,j,k}, \tau_{i,j,k})$, $k = 0, 1 \dots, f$, *be the space-time point at which the trajectory of agent* i *intersects the trajectory of agent* $i + k$ *while agent* i *is traveling away from its turning point* $d_{i,j}$ *and agent* $i + k$ *is traveling towards its turning point* $d_{i+k,j}$. *Then* $\rho_{i,j,k} = \frac{r - r^{2k/n}}{r-1} d_{i,j}$, *and* $\tau_{i,j,k} = \frac{r + r^{2k/n}}{r-1}|d_{i,j}|$.

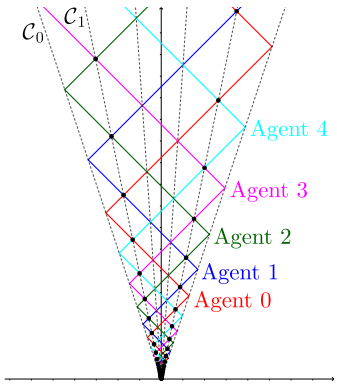


Figure 1: Example space-time trajectories when $r = 2$ and $(n, f) = (5, 2)$. Time is on the y-axis, position on the x-axis. Note that each of the turning-points lie along a cone $\mathcal{C}_0$ with slope $\pm\frac{r+1}{r-1}$. Also indicated are the points $(\rho_{i,j,1}, \tau_{i,j,1})$ which lie along the cone $\mathcal{C}_1$. For each fixed k the points $(\rho_{i,j,k}, \tau_{i,j,k})$ lie along a cone $\mathcal{C}_k$.

The space-time points $(\rho_{i,j,k}, \tau_{i,j,k})$ for $k = 1$ are depicted in Figure 1. One can observe that for a fixed k the points $(\rho_{i,j,k}, \tau_{i,j,k})$ all lie along a common cone. Let $\mathcal{C}_k$ represent the cone corresponding to points $(\rho_{i,j,k}, \tau_{i,j,k})$. Then the slope of $\mathcal{C}_k$ is $\beta_k := \tau_{i,j,k}/|\rho_{i,j,k}| = \frac{r + r^{2k/n}}{r - r^{2k/n}}$. By referring to Figure 1 one can make the important observation that there always exists an agent located between the cones $\mathcal{C}_0$ and $\mathcal{C}_1$. This observation easily leads to a bound on Δ_f^x.

OBSERVATION 2.1. *At all times* $t > 0$ *there exists an agent located within each of the intervals* $\pm\left[\frac{t}{\beta_1}, \frac{t}{\beta_0}\right]$.

LEMMA 2.4. $|x| + \frac{S_f^x}{\beta_1} \le \Delta_f^x \le |x| + \frac{S_f^x}{\beta_0}$.

We are now in a position to prove Theorem 2.1.

Proof. (Theorem 2.1) In the worst case the target is just beyond a turning point and so we assume that the target is at location $x = (1+\epsilon)d_{i,j}$, with $\epsilon > 0$ arbitrarily small. Also suppose that it is agent $k \in \{1, \ldots, f+1\}$ that is the first reliable agent to reach the target. Then by Lemma 2.2, Lemma 2.4, and equation (1.1) we have $|x| + T_{i,j,k}(1+\epsilon)\left(1+\frac{1}{\beta_1}\right) \leq E_f^x \leq |x| + T_{i,j,k}(1+\epsilon)\left(1+\frac{1}{\beta_0}\right)$. Dividing by $|x| = (1+\epsilon)|d_{i,j}|$, substituting in the expressions for $T_{i,j,k}(1)$, β_0, and β_1 then yields $1 + \frac{2r}{r+r^{2/n}} + \frac{4r^{1+2k/n}}{(1+\epsilon)(r+r^{2/n})(r-1)} \leq R_f \leq 1 + \frac{2r}{r+1} + \frac{4r^{1+2k/n}}{(1+\epsilon)(r+1)(r-1)}$. The right hand side of both inequalities increases with k and so taking $k = f+1$ yields $1 + \frac{2r}{r+r^{2/n}} + \frac{4r^{2+1/n}}{(1+\epsilon)(r+r^{2/n})(r-1)} R_f \leq 1 + \frac{2r}{r+1} + \frac{4r^{2+1/n}}{(1+\epsilon)(r+1)(r-1)}$. The theorem then follows by taking $\epsilon \to 0$ on the right hand side. ∎

Theorem 2.1 used the fact that the agent most distant from the target at time S_f^x will be somewhere between the cones $\mathcal{C}_0$ and $\mathcal{C}_1$. Since $\lim_{n\to\infty} r^{2/n} = 1$ it is clear that $\lim_{n\to\infty} \beta_1 = \beta_0$, i.e. the cones $\mathcal{C}_0$ and $\mathcal{C}_1$ approach each other as n gets large. This implies that the bounds of Theorem 2.1 will also approach each other for large n and we thus find that:

THEOREM 2.2. *The asymptotic competitive ratio of the proportional schedule algorithm is* $\lim_{f\to\infty} R_f = 7 - \frac{2(r-3)}{r^2-1}$. *If we take* $r = 3+2\sqrt{2}$ *then* $\hat{R} = 4+2\sqrt{2}$.

To compute an exact expression for the competitive ratio as a function of r we would need to determine the identity i_f^x of the agent that is most distant from x at the time S_x^f. Although this is not very difficult to do, it does not add to the results of the paper since we will be improving upon this algorithm in the next section. It is useful, however, to know the optimal competitive ratio for this algorithm for the sake of comparison and discussion. Figure 2 in Section 3 shows a plot of the optimized competitive ratios as a function of f along with the bounds from Theorem 2.1 evaluated with the optimized parameter r. Also shown is the competitive ratio when r is chosen to optimize the search time only. One can observe that in all cases except $f = 1$, the actual competitive ratio is essentially identical to the lower bound implying that the optimal choice of r places the agent i_f^x on or near the interior cone $\mathcal{C}_1$ at the time S_f^x. This observation leads one to question whether or not the single degree of freedom provided by the parameter r is sufficient to facilitate an efficient trade-off between the search time and the distance Δ_f^x. In the next section we validate this concern and show that a generalized form of the proportional schedule leads to an improvement in the competitive ratio.

3 Generalized (proportional) schedules

In this section we consider a generalized form of the proportional schedule. Put simply, we will add an extra two turning points between each pair of turning points $d_{i,j}$ and $d_{i,j+1}$ of the normal proportional schedule. We will refer to these "sub-turning points" using the notation $d_{i,j}^{(\ell)}$, $\ell = 0, 1, 2$. An intuitive parameterization of the turning points uses parameters $s \in [0, r+1]$ and $a \in [s-1, r]$ as follows

$$d_{i,j}^{(\ell)} = d_{i,j} \cdot \begin{cases} 1, & \ell = 0 \\ -a, & \ell = 1 \\ s-a, & \ell = 2 \end{cases}, \tag{3.3}$$

The parameter s controls the distance between $d_{i,j}^{(1)}$ and $d_{i,j}^{(2)}$, and a controls the location of $d_{i,j}^{(1)}$ (relative to $d_{i,j}$). With the bounds given on s and a we will have $d_{i,j}^{(1)} \in [d_{i,j}, d_{i,j+1}]$ and $d_{i,j}^{(2)} \in [d_{i,j}, d_{i,j+1}]$. One can also observe that when $s = 0$ these trajectories are identical to those of the proportional schedule and so this can be rightly called a generalization.

Although the parameterization using (r, s, a) is intuitive, it will be much more convenient to replace s with the parameter $q := \frac{r+s}{r-1}$. We will use both of these parameterizations, however, we will favor the one with q. For the parameterization with q we have

$$d_{i,j}^{(\ell)} = d_{i,j} \cdot \begin{cases} 1, & \ell = 0 \\ -a, & \ell = 1 \\ q(r-1) - r - a, & \ell = 2 \end{cases}, \tag{3.4}$$

We make note of the following identities concerning the parameters q and s: $q = \frac{r+s}{r-1}$, $q - 1 = \frac{1+s}{r-1}$, and $q + s = r(q-1)$. We will use these identities repeatedly and without reference

One approach to analyzing these trajectories would be to compute and optimize the competitive ratio as a function of the parameters (r, q, a). This would involve a nightmarish case analysis that would scare away even the most interested readers. Alas, this is not the approach we take. Instead we will describe sets of objectively good choices for the parameters q and a and express the competitive ratio of the resulting trajectories as a function of the parameter r. We will use the results from the previous section to guide us as much as possible. The analysis of the cases $f = 1$ and $f > 1$ is sufficiently different to warrant considering each separately. We will begin with the case that $f > 1$ since this case closely mirrors that of the vanilla proportional schedules.

3.1 Many faults

Recall that the worst case scenario for the proportional schedules occurs when the target is

placed just beyond a turning point $d_{i,j}$ and the first f agents that visit the target are faulty. We will refer to this scenario as Scenario A in order to refer to it quickly. Also recall that, in all cases that $f > 1$, it was optimal to choose r so that the agent i_f^x was located on or near the inner bounding cone $\mathcal{C}_1$ at the time S_f^x in order to minimize the distance Δ_f^x in the event that Scenario A occurs. Scenario A will still be a potential worst case for the generalized schedule, however, we can now use our extra degrees of freedom to ensure that agent i_f^x is located on the cone $\mathcal{C}_1$ at the time S_f^x while leaving the parameter r to facilitate a more efficient tradeoff between Δ_f^x and S_f^x. Our goal is to prove the following theorem.

THEOREM 3.1. *Fix the number of faults $f > 1$ with $n = 2f + 1$. Define the functions*

$$\hat{q}(r,u) := \frac{r^{2u/n-1}+1}{(1+r^{2/n})r^{2(u-1)/n-1} - 2r^{1/n}} \tag{3.5}$$

$$\hat{a}(r,q) := \begin{cases} q(r^{1-2/n}-1), & q \le \frac{r}{r-r^{2/n}} \\ q(r^{1-4/n}-1), & \textit{otherwise} \end{cases} \tag{3.6}$$

and the set

$$\mathcal{P} := \Bigg\{ (r,u) \mid r > 1,\ u \in \{f+3,\dots,n\}, \\ \frac{r}{r-1} \le \hat{q}(r,u) \le \min\left\{ \frac{r}{r-r^{1-2/n}}, \frac{r}{r-r^{4/n}} \right\} \Bigg\} \tag{3.7}$$

Then, for pairs $(r,u) \in \mathcal{P}$, the competitive ratio of the generalized proportional schedule with parameters $q = \hat{q}(r,u)$, and $a = \hat{a}(r,\hat{q}(r,u))$ is

$$R_f = \begin{cases} R_f^A, & q \le \frac{r}{r-r^{2/n}} \\ \max\{R_f^A,\ R_f^B\}, & \textit{otherwise.} \end{cases} \tag{3.8}$$

where $R_f^A = 1 + \frac{2q(1+2qr^{1/n})}{q+(q-1)r^{2/n}}$, and $R_f^B \le 3 + \frac{2(q-1)}{q(r^{1-4/n}-1)}\left[2r - \frac{r^{3/n}(1+2qr^{1/n})}{q+(q-1)r^{2/n}}\right]$.

The rough idea behind this theorem is as follows. Taking $q = \hat{q}(r,u)$ ensures that the agent $i+u$ will be located on the cone $\mathcal{C}_1$ (properly modified for the generalized schedules) in the event that Scenario A occurs (see Lemma 3.6). Choosing $a = \hat{a}(r,q)$ ensures that agent $i+u$ will be the most distant agent in the event of Scenario A (see Lemma 3.5). The quantity R_f^A gives the competitive ratio of Scenario A and R_f^B gives the competitive ratio of an additional potential worst case.

To proceed we need to define analogues of the quantities $t_{i,j}$, $T_{i,j,k}(z)$, $\tau_{i,j,k}$, and $\rho_{i,j,k}$ in the context of the generalized schedules.

LEMMA 3.1. *The time $t_{i,j}^{(\ell)}$ at which agent i reaches its sub-turning point $d_{i,j}^{(\ell)}$ is*

$$t_{i,j}^{(\ell)} = |d_{i,j}| \cdot \begin{cases} 2q-1, & \ell = 0 \\ 2q+a, & \ell = 1 \\ q(r+1)-r+a, & \ell = 2 \end{cases}. \tag{3.9}$$

LEMMA 3.2. *The time $T_{i,j,k}(z)$ at which agent $i+k$ first reaches location $z = zd_{i,j} \in I_{i,j}$ is*

$$T_{i,j,k}(z) = \begin{cases} T_{i,j,k}^{\circ}(z), & a \ge \frac{zr}{r^{2k/n}} \\ T_{i,j,k}^{+}(z), & a < \frac{zr}{r^{2k/n}} \end{cases}. \tag{3.10}$$

where $T_{i,j,k}^{\circ}(z) := \left[z + 2qr^{2k/n-1}\right]|d_{i,j}|$, and $T_{i,j,k}^{+}(z) := \left[z + 2(q-1)r^{2k/n}\right]|d_{i,j}|$.

The proportional schedules enjoyed the property that the $(f+1)^{st}$ agent to reach any location $x = zd_{i,j} \in I_{i,j}$ was the agent $i+f+1$. The next lemma outlines the conditions for this to also be the case with the generalized schedules.

LEMMA 3.3. *Suppose that $a \ge r^{1/n}$. Then agent $i+k$, $k = f+1,\dots,n$, will reach any position $x = zd_{i,j} \in I_{i,j}$ at the time $T_{i,j,k}(z) = T_{i,j,k}^{\circ}(z)$. Furthermore, if $q \le \frac{r}{r-r^{2/n}}$ then agents $i+1,\dots,i+f$ will reach x before agent $i+f+1$, otherwise, if $\frac{r}{r-r^{2/n}} < q \le \frac{r}{r-r^{4/n}}$ then agents $i+1,\dots,i+f-1$ will reach x before agent $i+f+1$, and agent $i+f$ will reach x before agent $i+f+1$ provided that $a \ge zr^{1/n}$.*

LEMMA 3.4. *Suppose that $a \ge r^{1/n}$. Then the point $(\rho_{i,j,k}, \tau_{i,j,k})$, $k = 0,\dots,f$, at which the trajectory of agent i intersects that of agent $i+k$ while agent i is moving away from $d_{i,j}$ and agent $i+k$ is moving towards $d_{i+k,j}$ is*

$$(\rho_{i,j,k}, \tau_{i,j,k}) = \begin{cases} (\rho_{i,j,k}^{\circ}, \tau_{i,j,k}^{\circ}), & a \ge q(r^{1-2k/n}-1) \\ (\rho_{i,j,k}^{+}, \tau_{i,j,k}^{+}), & \textit{otherwise} \end{cases} \tag{3.11}$$

where $\rho_{i,j,k}^{\circ} := q(1-r^{2k/n-1})d_{i,j}$, $\tau_{i,j,k}^{\circ} := q(1+r^{2k/n-1})|d_{i,j}|$, $\rho_{i,j,k}^{+} := [q-(q-1)r^{2k/n}]d_{i,j}$, and $\tau_{i,j,k}^{+} := [q+(q-1)r^{2k/n}]|d_{i,j}|$.

Let $\dagger = \circ, +$. Since the ratios $\tau_{i,j,k}^{\dagger}/|\rho_{i,j,k}^{\dagger}|$ are independent of i and j, for fixed k and $\dagger$ the points $(\rho_{i,j,k}^{\dagger}, \tau_{i,j,k}^{\dagger})$ all lie along a common cone. Let $\mathcal{C}_k^{\dagger}$ refer to the cone with slope $\beta_k^{\dagger} := \tau_{i,j,k}^{\dagger}/|\rho_{i,j,k}^{\dagger}|$ corresponding to the points $(\rho_{i,j,k}^{\dagger}, \tau_{i,j,k}^{\dagger})$. The proportional schedules had the property that at all times $t > 0$ there was an agent between the cones $\mathcal{C}_0$ and $\mathcal{C}_1$ and this agent was the most distant from the origin on its respective side. For the

generalized schedules we will want the same property to hold for the cones $\mathcal{C}_0^+$ and $\mathcal{C}_1^+$. We observe that

$$\beta_k^\circ = \frac{r + r^{2k/n}}{r - r^{2k/n}}, \quad \beta_k^+ = \frac{q + (q-1)r^{2k/n}}{|q - (q-1)r^{2k/n}|}. \tag{3.12}$$

We include the absolute value in the denominator of β_k^+ since it is possible that $q - (q-1)r^{2k/n} < 0$. We will later show that when $k = 1$ we will indeed have $q - (q-1)r^{2k/n} \geq 0$ as a result of the condition $q \leq \frac{r}{r - r^{1-2/n}}$ in the definition of $\mathcal{P}$.

LEMMA 3.5. *During the time interval* $[\tau_{i-1,j,1}, \tau_{i,j,1}]$ *agent* i *has the most negative/positive position when* j *is odd/even provided that* $a = q(r^{1-4/n} - 1)$ *or* $q(r^{1-4/n} - 1) < a \leq q(r^{1-2/n} - 1)$ *and* $q \leq \frac{r}{r - r^{2/n}}$.

In the next lemma we describe how we should choose q if we want the agent furthest from $d_{i,j}$ to be located on the cone $\mathcal{C}_1^+$ at the time the $(f+1)^{st}$ agent reaches $d_{i,j}$.

LEMMA 3.6. *Take* $q = \hat{q}(r,k)$. *Then agents* $i + k - 1$ *and* $i + k$, $k = f + 3, \ldots, n$, *will both be located on the cone* $\mathcal{C}_1^+$ *on the opposite side of the origin from* $d_{i,j}$ *at the time* $T^\circ_{i,j,f+1}(1)$.

We need one last lemma before proving Theorem 3.1.

LEMMA 3.7. $\hat{a}(r,q) > r^{1/n}$ *when* $r > 1$ *and* $q \geq \frac{r}{r-1}$.

We now have all of the lemmas required to prove Theorem 3.1.

Proof. (Theorem 3.1) Suppose without loss of generality that the target is at location $x = z d_{i,j} \in I_{i,j}$. In the worst case the first f agents that reach the target are faulty and x is just beyond a turning point.

Consider pairs $(r,u) \in \mathcal{P}$ and suppose that $q = \hat{q}(r,u)$ and $a = \hat{a}(r,q)$. By definition of $\mathcal{P}$ we have $r > 1$ and $\frac{r}{r-1} \leq q \leq \min\{\frac{r}{r - r^{1-2/n}}, \frac{r}{r - r^{4/n}}\}$. Then, by Lemma 3.7, we also have $a > r^{1/n}$. By Lemma 3.3 we know that agent $i + f + 1$ will thus reach location x at the time $T_{i,j,f+1}(z) = T^\circ_{i,j,f+1}(z) = (z + qr^{2(f+1)/n-1})|d_{i,j}| = (z + qr^{1/n})|d_{i,j}|$.

Agents $i+1, \ldots, i+f-1$ will all reach x before agent $i+f+1$, and agents $i+f+2, \ldots, i+n$ will reach x after agent $i+f+1$. Agent $i+f$ will reach x before agent $i+f+1$ provided that $q \leq \frac{r}{r-r^{2/n}}$ or $q > \frac{r}{r-r^{2/n}}$ and $a \geq zr^{1/n}$. We first consider the case that agent $i+f$ reaches x before agent $i+f+1$.

With $q \leq \frac{r}{r-r^{2/n}}$ or $q > \frac{r}{r-r^{2/n}}$ and $a \geq zr^{1/n}$ we know that agent $i+f+1$ will be the $(f+1)^{st}$ agent to reach the target. In the worst case the target is at position $x = (1+\epsilon)d_{i,j}$, i.e. $z = 1 + \epsilon$. The search time is therefore $S_f^x = T^\circ_{i,j,f+1}(1+\epsilon) = (1 + \epsilon + 2qr^{1/n})|d_{i,j}|$. Since $q = \hat{q}(r,u)$, we know by Lemma 3.6 that agent $i+u$ will be (one of) the agents most distant from x at time S_f^x. Moreover, this agent will be located on the cone $\mathcal{C}_1^+$ at time $T^\circ_{i,j,f+1}(1)$. We can thus conclude that $\Delta_x^f = |x| + \frac{T^\circ_{i,j,f+1}(1)}{\beta_1^+} + \epsilon|d_{i,j}|$. Using (1.1) then yields (after some manipulation) $E_f^x = (1+3\epsilon)|d_{i,j}| + \left(1 + \frac{1}{\beta_1^+}\right)(1 + 2qr^{1/n})|d_{i,j}|$. Dividing by $|x| = (1+\epsilon)|d_{i,j}|$ and taking the limit $\epsilon \to 0$ gives the competitive ratio to be $R_f^A = 1 + \left(1 + \frac{1}{\beta_1^+}\right)(1 + 2qr^{1/n})$ where we have included a superscript A to indicate that this is the competitive ratio of Scenario A.

We have from equation (3.12) that $\beta_1^+ = \frac{q + (q-1)r^{2/n}}{|q - (q-1)r^{2/n}|}$. We claim that $q - (q-1)r^{2/n} \geq 0$ as a result of the requirement that $q \leq \frac{r}{r - r^{1-2/n}}$. Indeed, we have $q - (q-1)r^{2k/n} = q(1 - r^{2/n}) + r^{2/n} \geq 0$ or $q \leq \frac{r^{2/n}}{r^{2/n} - 1} = \frac{r}{r - r^{1-2/n}}$. Now observe that $1 + \frac{1}{\beta_1^+} = 1 + \frac{q - (q-1)r^{2/n}}{q + (q-1)r^{2/n}} = \frac{2q}{q + (q-1)r^{2/n}}$ and we can conclude that the competitive ratio of Scenario A is $R_f^A = 1 + \frac{2q(1 + 2qr^{1/n})}{q + (q-1)r^{2/n}}$ as required.

Now consider the case that $q > \frac{r}{r - r^{2/n}}$ and $a < zr^{1/n}$. Recall that the condition $a < zr^{1/n}$ derived from the requirement that agent $i+f$ reaches its turning point $d^{(1)}_{i+f,j-1}$ before reaching x. The time at which agent $i+f$ reaches x is $T^+_{i,j,f}(z)$ and at this time agents $i+1, \ldots, i+f-1$ and $i+f+1$ have already visited x. The worst case for this scenario places the target just beyond the turning point $d^{(1)}_{i+f,j-1}$ and so we take $x = (1+\epsilon)d^{(1)}_{i+f,j-1} = (1+\epsilon)\frac{a}{r^{1/n}}|d_{i,j}|$. Thus, with $z = (1+\epsilon)\frac{a}{r^{1/n}}$, the search time for this case is at most $T^+_{i,j,f}(z)$, i.e. $S_f^x \leq T^+_{i,j,f}(z) = (z + 2(q-1)r^{2f/n})|d_{i,j}| = (z + 2(q-1)r^{1-1/n})|d_{i,j}|$.

At the time $T^\circ_{i,j,f+1}(1)$ the agent most distant from x was at distance $|x| + T^\circ_{i,j,f+1}(1)/\beta_1^+$. Thus, at time $T^+_{i,j,f}(z)$ this agent ill be further away from x by at most the distance $T^+_{i,j,f}(z) - T^\circ_{i,j,f+1}(1)$. We therefore have that $\Delta_f^x \leq |x| + \frac{T^\circ_{i,j,f+1}(1)}{\beta_1^+} + T^+_{i,j,f}(z) - T^\circ_{i,j,f+1}(1)$. For the evacuation time we then find (after some manipulation) $E_f^x = \left[3z + 4(q-1)r^{1-1/n} - \left(1 - \frac{1}{\beta_1^+}\right)(1 + 2qr^{1/n})\right]|d_{i,j}|$. We have $1 - \frac{1}{\beta_1^+} = 1 - \frac{q - (q-1)r^{2/n}}{q + (q-1)r^{2/n}} = \frac{2(q-1)r^{2/n}}{q + (q-1)r^{2/n}}$ and using this result in the expression for E_f^x gives $E_f^x = \left[3z + \frac{2(q-1)}{r^{1/n}}\left(2r - \frac{r^{3/n}(1 + 2qr^{1/n})}{q + (q-1)r^{2/n}}\right)\right]|d_{i,j}|$. Dividing through by $z|d_{i,j}|$ with $z = (1+\epsilon)\frac{a}{r^{1/n}}$

and taking the limit $\epsilon \to 0$ yields the competitive ratio $R_f^B \leq 3 + \frac{2(q-1)}{a}\left(2r - \frac{r^{3/n}(1+2qr^{1/n})}{q+(q-1)r^{2/n}}\right)$. Since we are assuming that $q > \frac{r}{r-r^{2/n}}$ we have $a = \hat{a}(r,q) = q(r^{1-4/n}-1)$ and we can finally conclude that $R_f^B \leq 3 + \frac{2(q-1)}{q(r^{1-4/n}-1)}\left(2r - \frac{r^{3/n}(1+2qr^{1/n})}{q+(q-1)r^{2/n}}\right)$ as required. □

Table 2: Optimal parameter choices for the generalized schedules and the corresponding competitive ratios. The competitive ratio of the optimized proportional schedules is included in the last row for comparison.

(n,f)	(5,2)	(7,3)	(9,4)	(11,5)
r	3.58545	5.97532	4.21585	3.22306
u	5	6	8	10
q	1.45340	1.45340	1.38190	1.44983
s	0.17225	1.25582	0.22813	0
a	1.67348	1.67348	2.84964	2.32740
R_f	7.25377	7.25377	7.14703	7.10648
R_f, P.S.	7.37001	7.40756	7.23077	7.10648

Figure 3 in the appendix illustrates the optimized trajectories of the agents for the cases $f = 2, 3$. Table 2 lists the (numerically) optimized competitive ratios of the generalized schedule along with the corresponding optimal pair (r, u) and the resulting parameters a and q. Figure 2 plots these competitive ratios as a function of f along with those of the optimized proportional schedule. One can observe that, in many cases, the two algorithms have identical competitive ratios. In all cases for $f > 1$ the optimal competitive ratio is defined by Scenario A. It is interesting to note that in every case except $f = 3$ we have $\hat{a}(r,q) = q(r^{1-2/n} - 1)$. In fact, the case $(n,f) = (7,3)$ seems to be unique in a number of ways. It has an identical competitive ratio as the $f = 2$ case. Moreover, the optimal parameter values for q and a are identical for $f = 3$ and $f = 2$, and, although the parameter r is not the same, one can confirm that the quantity $r^{1/n}$ is identical in both cases.

3.2 Three agents, one fault Our goal is to prove the following theorem.

THEOREM 3.2. *Suppose that* $r > 2\sqrt{2}$, *define* $\eta = \sqrt{(r + r^{2/3} + 1)^2 + 4r^{2/3}(r+1)(r - 2r^{1/3})}$, *and take*

$$a = \frac{r + 1 + r^{2/3} - \eta}{2(r - 2r^{1/3})}, \quad q = \frac{r + 1 - a}{r + 1 - 2r^{1/3}}. \tag{3.13}$$

Then the competitive ratio of the generalized schedule for $(n,f) = (3,1)$ *is*

$$R_1 = 2 + \frac{1}{r + 1 - 2r^{1/3}}[r^{1/3}(2 - r^{1/3}) + \eta] \tag{3.14}$$

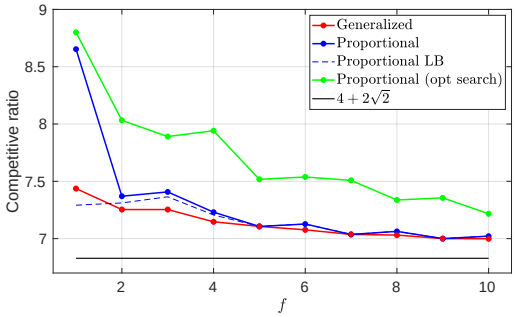


Figure 2: The competitive ratio as a function of the number of faults f for the proportional schedule (solid blue) and generalized proportional schedule (red) algorithms. The dashed blue line indicates the lower-bound of Theorem 2.1; the green line indicates the competitive ratio of the proportional schedule when we choose r to optimize only the search time; the lower black line indicates the asymptotic limit of $4 + 2\sqrt{2}$.

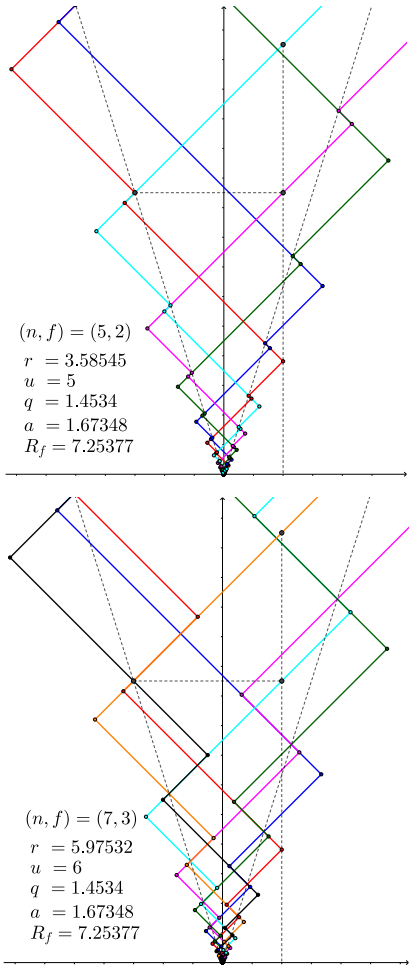


Figure 3: The optimized trajectories of the agents when $f = 2, 3$.

In particular, if $r = 6.833921$, $a = 1.699557$, *and* $q = 1.518949$ *then* $R_1 = 7.437011$.

Recall the potential worst case Scenario A for the proportional schedule – the target is placed just beyond a turning point $d_{i,j}$ and agent $i+1$ (who would be the first to reach the target) is faulty. Thus, agent $i+2$ will be the first reliable agent to reach the target and at this time agent $i = i+3 \mod 3$ will still need to evacuate. The problem with the normal proportional schedule is that agent i might be relatively far from location $d_{i,j}$ when it hears the announcement. We will thus use the extra turning points of the generalized algorithm in order to position agent i relatively close to $d_{i,j}$ at the time agent $i+2$ reaches $d_{i,j}$. In particular, if we choose q according to (3.13), then agent i will be located at its sub-turning point $d^{(2)}_{i,j}$ at exactly the same time agent $i+2$ reaches location $d_{i,j}$ (see Lemma 3.8). However, in fixing the value of q in this way we will introduce a new potentially worst-case location of the target and, choosing a according to (3.13), will ensure this new potential worst case is no worse than Scenario A (see Lemma 3.11).

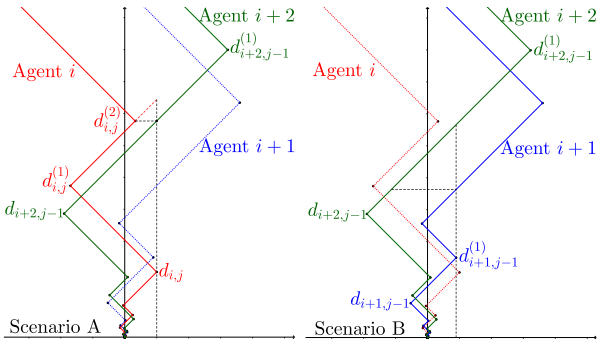


Figure 4: The two worst case scenarios for the generalized proportional schedule for $n = 3$ and $f = 1$ when the parameters a and q are chosen according to (3.13). Scenario A (resp. Scenario B) occurs when the target is just beyond the turning point $d_{i,j}$ (resp. $d^{(1)}_{i+1,j-1}$) and agent $i+1$ (resp. i) is faulty.

Figure 4 illustrates the trajectories of the three agents when a and q are chosen according to (3.13). The left figure displays Scenario A. One can observe that at the instant agent $i+2$ reaches location $d_{i,j}$ agent i will be at its turning point $d^{(2)}_{i,j}$. The right of Figure 4, which will be referred to as Scenario B, represents the new potentially worst case situation. In this scenario the target is placed at location $d^{(1)}_{i+1,j-1}$ and it is agent i that is faulty. Agent $i+1$ will be the first reliable agent to reach the target and agent $i+2$ will be the final agent to evacuate. Observe that for this scenario the announcement by agent $i+1$ will not affect the trajectory of agent $i+2$ since this agent was already on its way to $d^{(1)}_{i+1,j-1}$ at the time of the announcement. We will choose a in order to ensure that the competitive ratios resulting from Scenarios A and B will be equal.

Scenario A: This is the case depicted on the left of Figure 4. The target is just beyond the turning point $d_{i,j}$ and agent $i+2$ is the first reliable agent to reach the target. The next lemma states that if we choose s according to (3.13) then agent i will be at its turning point $d^{(2)}_{i,j}$ at the moment agent $i+2$ reaches $d_{i,j}$.

LEMMA 3.8. *If* q *is given by* (3.13) *and* $a \geq \frac{1}{r^{1/3}}$ *then agent* i *will reach its turning point* $d_{i,j+2}$ *at the same time agent* $i+2$ *reaches location* $d_{i,j}$.

We now let R^A_1 represent the competitive ratio for Scenario A. The next lemma provides an expression for this competitive ratio in terms of q and a.

LEMMA 3.9. *Take* $a \geq \frac{1}{r^{1/3}}$. *Then Scenario A has competitive ratio* $R^A_1 = 1 + 2(q+a)$.

Proof. With q given by (3.13) we know that agent $i+2$ will reach $d_{i,j}$ at the time $T_{i,j,2}(1) = t^{(2)}_{i,j}$ and at this time agent i will be located at $d^{(2)}_{i,j}$. In the worst case the target is just beyond $d_{i,j}$ and the competitive ratio is therefore $R^A_1 = 1 + \frac{t^{(2)}_{i,j} - |d^{(2)}_{i,j}|}{|d_{i,j}|} = 1 + \frac{t^{(1)}_{i,j} + |d^{(1)}_{i,j}|}{|d_{i,j}|}$. With $t^{(1)}_{i,j} = (2q+a)|d_{i,j}|$ and $|d^{(1)}_{i,j}| = a|d_{i,j}|$ we get $R^A_1 = 1 + 2(q+a)$. ∎

Scenario B: This is the case depicted on the right of Figure 4. The target is placed just beyond the turning point $d^{(1)}_{i+1,j-1}$, agent $i+1$ is the first reliable agent to reach the target, and agent $i+2$ will evacuate last. We note that for this case to occur it must be that $|d^{(1)}_{i+1,j-1}| \leq |d_{i,j}|$ which implies that $a \leq r^{1/3}$.

LEMMA 3.10. *Take* $a \leq r^{1/3}$. *Then Scenario B has competitive ratio* $R^B_1 = 1 + \frac{2qr^{2/3}}{a}$.

Proof. In the worst case the target is at location x just beyond $d^{(1)}_{i+1,j-1}$. Referring to Figure 4 one can observe that the announcement in this case will not change the trajectory of agent $i+2$ and the evacuation will complete at the time agent $i+2$ would normally reach x. The last turning point agent $i+2$ visits before visiting x is the turning point $d_{i+2,j-1}$ and thus the competitive ratio is $R^B_1 = 1 + \frac{t_{i+2,j-1} + |d_{i+2,j-1}|}{d^{(1)}_{i+1,j-1}}$. With $t_{i+2,j-1} = (2q-1)r^{1/3}|d_{i,j}|$, $|d_{i+2,j-1}| = r^{1/3}|d_{i,j}|$, and $|d^{(1)}_{i+1,j-1}| = ar^{-1/3}|d_{i,j}|$ we get $R^B_1 = 1 + \frac{(2q-1)r^{1/3} + r^{1/3}}{ar^{-1/3}} = 1 + \frac{2qr^{2/3}}{a}$. ∎

The next lemma states that choosing q and a according to (3.13) ensures that $R_1^A = R_1^B$.

LEMMA 3.11. *Take* $\frac{1}{r^{1/3}} \leq a \leq r^{1/3}$ *and* $r > 2\sqrt{2}$. *If* q *and* a *are given by* (3.13) *then* $R_1^A = R_1^B$.

To derive the expression for R_1 in Theorem 3.2 we substitute (3.13) into the expression for R_1^A in Lemma 3.9. Numerically optimizing (3.14) with respect to $r > 2\sqrt{2}$ yields the result of Theorem 3.2.

4 Byzantine search with one fault

In this section we use our evacuation algorithm for $(n, f) = (3, 1)$ to improve upon the best known upper bound for *search* by three agents, with at most one byzantine fault. For this problem we only require the agents to reliably identify the target and, in particular, they do not need to evacuate. A reliable identification of the target occurs at the first time a provably reliable agent reaches the target. We will prove the following result.

THEOREM 4.1. *Byzantine search for three agents, at most one of which is faulty, can be completed with a competitive ratio no more than* 7.43701137.

We first give a quick discussion of the model. A byzantine faulty agent is similar to a crash faulty agent except that byzantine agents can also lie about finding the target, in addition to failing to announce the target. Therefore, in this case, an announcement that the target has been found cannot necessarily be trusted. In order to confirm whether or not the target is at an announced location x at least one provably reliable agent must confirm that the target is at x. With $n = 2f + 1$, it can be required that all agents reach x before this confirmation occurs (which is, of course, just evacuation in disguise). However, in this case it is possible that the agents do not find the target at x and must continue their search. Nevertheless, in this process the reliable agents will be able to identify at least one of the faulty agents (the one who lied) and thus, if the search continues, it will be from a better standpoint in terms of the ratio of faulty agents to reliable agents. In the specific case of $(n, f) = (3, 1)$, the moment the single faulty agent is identified, both of the remaining agents will know each other is reliable.

To prove Theorem 4.1 we demonstrate that the agents can simply use the generalized schedule of Theorem 3.2 to achieve the desired competitive ratio. The proof simply checks that the faulty agent cannot achieve a worse competitive ratio by lying about the target's location.

5 Conclusions

We have studied the problem of evacuation by $n = 2f+1$ agents when at most f of the agents are crash faulty. We introduced a novel type of search algorithm which gives an improvement for the evacuation as compared to the trajectories used for optimal crash search. These trajectories also gave an improvement on the best known upper bound for the problem of search by three agents at most one of which is byzantine faulty.

Since we did not carry out a pure optimization of the parameters for our new algorithm it is possible that there are better choices of these parameters than the ones described here. Furthermore, it is possible to further generalize the trajectories of the agents with the addition of more sub-turning points, however, this seems rather unlikely to improve the results. It is also an open problem to improve lower bounds for the evacuation problem since the state of the art derives from the optimality of the search problem, i.e. evacuation has at least the same competitive ratio as search since the agents must first find the target in order to evacuate. As a result, there is a rather large gap between the upper bounds presented here and the best known lower bounds. The complexity of the trajectories described here hints that achieving a tight lower bound for this problem may be a difficult task indeed.

We have focused on the particular case that $n = 2f + 1$. When $n > 2f + 1$ the problem is trivial since we can split the agents into two groups of at least $f + 1$ agents each and send them in opposite directions. This strategy easily leads to a competitive ratio of 3, which is tight. The problem is interesting when $n < 2f + 1$ and the analysis of these cases offers an extension of this research. There are some simple observations that one can make for these cases. An upper bound of 9 is clear since we can have all agents stick together and use the doubling strategy for single agent search. However, in most cases it is expected that one can improve upon this upper bound. Indeed, for pairs $(n, f) = (3k, 2k - 1)$, $k \geq 1$, we can form 3 groups of k agents and, with at most $2k - 1$ faults, two out of three of these groups are guaranteed to contain a reliable agent. We can therefore have each group act as a single agent and perform the generalized schedule algorithm of Theorem 3.2 to achieve an upper bound of ~ 7.437. A similar trick can be used to achieve the same competitive ratio as achieved for the case $(2f+1, f)$ for all pairs $((2f + 1)k, (f + 1)k - 1)$, $k \geq 1$ and $f \geq 1$.

References

[1] S. Albers and M. R. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[2] S. Albers, K. Kursawe, and S. Schuierer. Exploring unknown environments with obstacles. *Algorithmica*, 32(1):123–143, 2002.

[3] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[4] R. Baeza-Yates and R. Schott. Parallel searching in the plane. *Computational Geometry*, 5(3):143–154, 1995.

[5] E. Bampas, J. Czyzowicz, L. Gąsieniec, D. Ilcinkas, R. Klasing, T. Kociumaka, and D. Pająk. Linear search by a pair of distinct-speed robots. *Algorithmica*, 81(1):317–342, 2019.

[6] A. Beck. On the linear search problem. *Israel J. of Mathematics*, 2(4):221–228, 1964.

[7] A. Beck. More on the linear search problem. *Israel J. of Mathematics*, 3(2):61–70, 1965.

[8] A. Beck and D. Newman. Yet more on the linear search problem. *Israel J. of Mathematics*, 8(4):419–429, 1970.

[9] R. Bellman. An optimal search. *SIAM Review*, 5(3):274–274, 1963.

[10] M. Chrobak, L. Gąsieniec, Gorry T., and R. Martin. Group search on the line. In *SOFSEM 2015*, pages 164–176, Sněžkou, Czech Republic, 2015. Springer.

[11] J. Czyzowicz, K. Georgiou, R. Killick, E. Kranakis, D. Krizanc, M. Lafond, L. Narayanan, J. Opatrny, and S. Shende. Energy consumption of group search on a line. In *ICALP 2019*, pages 137:1–137:15, Patras, Greece, 2019. LIPIcs.

[12] J. Czyzowicz, K. Georgiou, R. Killick, E. Kranakis, D. Krizanc, M. Lafond, L. Narayanan, J. Opatrny, and S. Shende. Time-energy tradeoffs for evacuation by two robots in the wireless model. *Theoretical Computer Science*, 852:61–72, 2021.

[13] J. Czyzowicz, K. Georgiou, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, and S. Shende. Search on a line by byzantine robots. In *ISAAC 2016*, pages 27:1–27:12, Toronto, Canada, 2016. LIPIcs.

[14] J. Czyzowicz, E. Kranakis, D. Krizanc, L. Narayanan, and Opatrny J. Search on a line with faulty robots. In *PODC 2016*, pages 405–414, Chicago, Illinois, 2016. ACM.

[15] E.D. Demaine, S.P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361(2):342–355, 2006.

[16] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment. In *FOCS 1991*, pages 298–303. IEEE, 1991.

[17] B. Fristedt. Hide and seek in a subset of the real line. *International Journal of Game Theory*, 6(3):135–165, 1977.

[18] B. Fristedt and D. Heath. Searching for a particle on the real line. *Advances in Applied Probability*, 6(1):79–102, 1974.

[19] S. Gal. A general search game. *Israel Journal of Mathematics*, 12(1):32–45, 1972.

[20] F. Hoffmann, C. Icking, R. Klein, and K. Kriegel. The polygon exploration problem. *SIAM Journal on Computing*, 31(2):577–600, 2001.

[21] A. Kupavskii and E. Welzl. Lower bounds for searching robots, some faulty. In *PODC 2018*, pages 447–453, Egham, United Kingdom, 2018. ACM.

[22] X. Sun, Y. Sun, and J. Zhang. Better upper bounds for searching on a line with byzantine robots. In *Complexity and Approximation*, pages 151–171. Springer, 2020.

On the Request-Trip-Vehicle Assignment Problem

J. Carlos Martínez Mori* Samitha Samaranayake†

Abstract

The *request-trip-vehicle* assignment problem is at the heart of a popular decomposition strategy for online vehicle routing. In this framework, assignments are done in batches in order to exploit any shareability among vehicles and incoming travel requests. We study a natural ILP formulation and its LP relaxation. Our main result is an LP-based randomized rounding algorithm that, whenever the instance is feasible, leverages mild assumptions to return an assignment whose: *i*) expected cost is at most that of an optimal solution, and *ii*) expected fraction of unassigned requests is at most $1/e$. If trip-vehicle assignment costs are α-approximate, we pay an additional factor of α in the expected cost. We can relax the feasibility requirement by considering the penalty version of the problem, in which a penalty is paid for each unassigned request. We find that, whenever a request is repeatedly unassigned after a number of rounds, with high probability it is so in accordance with the sequence of LP solutions and not because of a rounding error. We additionally introduce a deterministic rounding heuristic inspired by our randomized technique. Our computational experiments show that our rounding algorithms achieve a performance similar to that of the ILP at a reduced computation time, far improving on our theoretical guarantee. The reason for this is that, although the assignment problem is hard in theory, the natural LP relaxation tends to be very tight in practice.

1 Introduction

In the *request-trip-vehicle* (RTV) assignment problem, we are given a set R of travel requests, a set T of candidate trips (i.e., a collection of subsets of R), and a set V of vehicles. Assigning a vehicle to a trip has an associated cost, typically representing distance traveled or incurred delays. The problem is to find a minimum cost set of trip-vehicle assignments such that: *i*) each request appears in exactly one trip-vehicle assignment, and *ii*) each vehicle is assigned to at most one trip.

The problem is at the heart of a decomposition strategy for online vehicle routing problems popularized by Alonso-Mora et al. [1] within the context of high-capacity ridesharing. Compared to traditional literature on vehicle routing [49], this *anytime* optimal framework decouples the routing and matching aspects of the problem, making it well-suited for parallel, online computation. The term *online* refers to the real-time nature of the system. Assignments are done in batches (e.g., every 5 to 30 seconds), rather than sequentially, to exploit any shareability [46] among vehicles and incoming travel requests. This style of solution approach is in fact used in practice by some well-known on-demand mobility service providers (for example, see [14]).

The framework exploits two key structural properties: *i*) there are tight quality of service constraints (e.g., maximum wait time, maximum travel time) [46, 21, 50], and *ii*) the feasible space is downward closed. In particular, a necessary condition for a potential trip-vehicle assignment to be feasible is that all of its sub-trip-vehicle assignments are feasible. This means T itself is downward closed. Together, these properties help prune a priori infeasible trips and vehicle assignments, thereby thwarting the combinatorial explosion.

1.1 Related Work.

Previous work around the RTV assignment problem has primarily focused on experimental performance. Ota et al. [40] give a greedy assignment algorithm based on a request indexing scheme. Simonetto et al. [48] solve the problem via linear assignments. Rather than tackling the problem all at once, they decompose it into a sequence of semi-matching linear programs in which each vehicle can be matched to up to one request. In their experiments, they achieve a system performance similar to that of [1] in about a fourth of the time. Lowalekar, Varakantham, and Jaillet [33] generate assignments at the *zone path* level. They group trips that have compatible pickup and drop off locations. Riley, Legrain, and Van Hentenryck [44] propose a column generation algorithm under soft constraints, where quality of service is enforced through a Lagrangian approach. They solve their pricing problem through an anytime optimal algorithm that, similar

*Center for Applied Mathematics, Cornell University. Email: jm2638@cornell.edu. Work partially supported by the Dwight David Eisenhower Transportation Fellowship Program under Award No. 693JJ32145020.

†School of Civil and Environmental Engineering, Cornell University. Email: samitha@cornell.edu. Work partially supported by the National Science Foundation under Grant No. CNS-1952011 and DMS-1839346.

to the trip generation step in [1], explores the feasible space in increasing order of trip size. Their algorithm takes exponential time in the worst case, but their experiments suggest soft constraints can reduce wait times and route deviations.

Bei and Zhang [6] show the problem is NP-hard even when no more than two requests can share a vehicle. They also give a 5/2-approximation algorithm for the total distance minimization version of this special case (under the additional assumption that there are *exactly* twice as many requests as there are vehicles). Li, Li, and Lee [29] match the 5/2 approximation guarantee while relaxing the latter assumption. Namely, they only require there being *at most* twice as many requests as there are vehicles. Luo and Spieksma [35] obtain a 2-approximation algorithm under the same assumptions as in [6]. They also obtain a 5/3-approximation when the objective is to minimize the total latency. Lowalekar, Varakantham, and Jaillet [34] study the competitive online version of the problem in the special case where vehicles return to their depot after serving a set of requests. If requests arrive in batches under a known adversarial arrival distribution, they obtain an algorithm whose expected competitive ratio is 0.3176 whenever vehicles have seating capacity 2. If vehicles have seating capacity $k > 0$, their expected competitive ratio is $\gamma > 0$, where γ is the solution to $\gamma = (1-\gamma)^{k+1}$.

1.2 Contributions.

1. We consider a natural integer linear programming (ILP) formulation that, in the worst case, has exponentially many variables. It is therefore not immediately clear whether one can always solve or even approximate its linear programming (LP) relaxation in polynomial time. To this end, we study the dual separation problem (i.e., column generation). The hope would be to approximately separate over the dual to approximately solve the primal, in the style of [8, 23, 18].

Although our ILP formulation is more general, we focus on the typical case in which trip-vehicle assignment costs correspond to the distance traveled by a vehicle when serving the requests in a trip. In this case, we identify the core of the dual separation problem as an instance of the *net-worth maximization* version of the prize-collecting traveling salesman problem (TSP). We note there is a closely related inapproximability result by Feigenbaum, Papadimitriou, and Shenker [16] for the net-worth maximization version of the prize-collecting Steiner tree problem. Intuitively, we expect[1] their result to carry over to the tour version, but their gadget is not easily adaptable for this purpose. To the best of our knowledge, this is an open problem. We also make a note on incompatible statements made in passing in the literature, particularly around the applicability of existing approximation algorithms for a different version of the prize-collecting TSP.

This unfavorable prospect motivates us to assume the trip list T is pre-computed and polynomial-sized. While this seems a rather strong assumption at first glance, it becomes much more reasonable if we a priori prune T based on fixed vehicle seating capacities $k > 0$ and tight quality of service constraints. We emphasize this is a key factor that differentiates ridesharing applications of vehicle routing (e.g., in contrast to applications in logistics), and that pre-computing T is in fact what is done in practical approaches for this problem [46, 1, 14].

2. Our second and main contribution is a simple LP-based randomized rounding algorithm for the RTV assignment problem with provable performance guarantees. We summarize our result as follows.

THEOREM 1.1. *Suppose we have a feasible instance of the RTV assignment problem with T polynomial-sized. If trip-vehicle costs are oracle-given and monotonic increasing w.r.t. request inclusion[2], there is a randomized algorithm such that:*

- *The expected cost of the final solution is at most that of an optimal solution.*
- *The expected fraction of unassigned requests is at most $1/e$ (i.e., less than 36.8% of all requests).*

If trip-vehicle costs are α-approximate, we pay an additional factor of α in the expected cost.

For example, Theorem 1.1 is applicable when trip-vehicle assignment costs correspond to distance traveled. Our rounding technique is similar in style to that of Raghavan and Thompson [43] for the minimum capacity multi-commodity flow problem. In addition, we show that *i*) the bound of $1/e$ on the rejection rate is tight for our algorithm, and *ii*) the integrality gap of the natural LP relaxation is at least 2.

In practice, a feasible solution to the RTV assignment problem might not always exist (e.g., when vehicle demand exceeds supply). Therefore, in practice, we may instead need to consider the penalty version of the problem, in which a penalty is paid for each ignored

[1] The results for Steiner tree do not immediately translate into results for TSP since, in this version of the problem, the objective function is the difference of two terms.

[2] The cost of a trip-vehicle assignment cannot decrease by adding an extra request.

request. We show that we can still use our algorithm for this version of the problem, and make the following remarks:

- In practice, unassigned requests are carried over to the next round of batch assignments (e.g., 5 to 30 seconds later). We argue that for any request $r \in R$, with high probability over a number of rounds, our algorithm either assigns it to a vehicle or purposely ignores it, in accordance to the sequence of LP solutions. That is to say, if a request is ignored by our algorithm after various rounds, it is increasingly likely to be so because of the LP solutions and not because of a rounding error.

- When the penalty terms eclipse the assignment costs (e.g., routing costs), the penalty version of the problem is essentially an instance of the cardinality version of a maximum coverage problem with group budget constraints, for which an $1/e$-approximation algorithm is known [11], assuming T is given explicitly. Our algorithm matches this guarantee in the sense that the expected fraction of requests that could have been covered but were not is at most $1/e$, while it provides the additional benefit that the quality of the assignment is also optimized as part of the LP.

To the best of our knowledge, this is the first algorithmic result with provable performance guarantees for the RTV assignment problem in its full generality. In particular, we allow high-capacity ridesharing (more than two requests can share a vehicle) under the general class of monotonic cost coefficients. Our techniques generalize to a class of set-partitioning problems.

3. We complement our analysis with computational experiments. We generate 10 distinct sets of $1,440$ simulated instances of the penalty version of the RTV assignment problem, corresponding to different combinations of number of vehicles and vehicle capacities. In our experiments, we evaluate *i*) solving the ILP to optimality using a commercial solver, *ii*) our randomized rounding algorithm, and *iii*) a deterministic rounding heuristic inspired by our randomized technique.

We observe that the percentage of rejected requests incurred by either rounding method is consistently below one percentage point higher than it is by solving the ILP (our heuristic performs slightly better than our randomized algorithm). Moreover, our implementation of either rounding method is faster than solving the ILP, with the percent improvement for mean and median values generally ranging between $2-7\%$. The percent improvement for mean values is consistently higher than the percent improvement for median values, showing computation time improvements are skewed to the right. In other words, the percent improvement is higher for the worst case (i.e., slowest) instances, which are arguably the most critical for real-time applications.

The performance of our randomized rounding algorithm in practice is far better than the theoretical guarantee in Theorem 1.1, but admittedly the ILP can be solved much faster than anticipated. Therefore, we investigate this further and find that the overwhelming majority ($\sim 96\%$) of supported LP variables are in fact integral. We moreover observe that $\sim 62\%$ of non-integral LP variables are half-integral. Of course, commercial ILP solvers leverage the quality of these fractional solutions, and our rounding algorithms make little to no mistakes on these assignments.

Our experiments confirm what we know anecdotally from other researchers and practitioners: although the RTV assignment problem is hard in theory [6], the natural LP relaxation tends to be very tight.

1.3 Future Research. We introduce rounding algorithms that are robust to worst case fractional solutions. However, we also present empirical evidence suggesting the natural LP relaxation can be very tight in practice. Therefore, we believe it would be valuable to rigorously understand whether or not this is always the case in practice (e.g., using data sets from various cities, beyond the habitual NYC Taxi and Limousine Commission (TLC) data [39] we used in our experiments). If there are instances in which the LP is not as tight, they will likely be harder commercial ILP solvers, and so they might more strongly showcase the benefit of LP rounding. Lastly, we note that in practice the bottleneck remains to be the generation of the candidate trip list T. The difficulty of this step corresponds to our discussion of the dual separation problem and its approximability.

1.4 Organization. The remainder of this paper is organized as follows. In Section 2 we introduce our notation and our ILP formulation. In Section 3 we study its LP relaxation. In Section 4 we present and analyze our randomized rounding algorithm. In Section 5 we introduce the penalty version of the problem. Lastly, in Section 6 we introduce our deterministic rounding heuristic and share our computational experience.

2 Preliminaries

2.1 Notation and Assumptions. Let $T(r)$ be the set of trips that contain request $r \in R$. Let $V(t)$ be the set of vehicles that can serve trip $t \in T$ and $T(v)$ be the set of trips than can be served by vehicle $v \in V$. For ease of presentation, with the exception of Section 6, we

assume any vehicle can serve any trip. Then, $V(t) = V$ for each $t \in T$ and $T(v) = T$ for each $v \in V$. This simplifies the LP constraints, which further simplifies our study of the dual LP. Our algorithm does not rely on this assumption. For technical reasons, we further assume $\emptyset \in T$ and $V(\emptyset) = V$. Any cost involving the empty trip $\emptyset \in T$ represents the cost of serving the passengers currently inside the vehicle, which is zero if the vehicle is empty. We use OPT to denote the cost of an optimal solution to the RTV assignment problem and LP$(\cdot)$ to denote the objective value of a linear program. An α-approximation algorithm for a minimization problem returns a solution of cost no more than $\alpha \geq 1$ times that of an optimal solution. An α-approximation algorithm for a maximization problem returns a solution of value at least $0 \leq \alpha \leq 1$ times that of an optimal solution.

2.2 Problem Formulation. Let $c_{tv} \geq 0$ be the cost of assigning trip $t \in T$ to vehicle $v \in V$. Again, we assume trip-vehicle costs are monotonic increasing with respect to request inclusion.

Typically, c_{tv} corresponds to the distance traveled by a single vehicle v when serving the requests in t (and the passengers currently inside v, if any). In the simplest idealized scenario, the single vehicle routing problem is an instance of the metric TSP [26, 3]. Then, the monotonicity assumption is met [47]. In reality, however, the single vehicle routing problem actually corresponds to some generalization of the TSP that includes service-specific constraints. For example, it may involve paths rather than tours [22], pickups and deliveries [45, 15], time windows [15, 4], capacity constraints [9], neighborhoods [19], and so on.

Now, consider the following ILP formulation which, in the worst case, has exponentially many variables due to the size of T. It moreover has polynomially many constraints (except for the binary constraints). Here, x_{tv} is set to 1 if trip $t \in T$ is assigned to vehicle $v \in V$.

$$
\begin{aligned}
\min \quad & \sum_{t \in T} \sum_{v \in V} c_{tv} x_{tv} \\
\text{s.t.} \quad & \\
& \sum_{(t,v) \in T(r) \times V} x_{tv} \geq 1, && \forall r \in R \\
& \sum_{t \in T} x_{tv} \leq 1, && \forall v \in V \\
& x_{tv} \in \{0,1\}, && \forall t \in T, v \in V
\end{aligned} \tag{2.1}
$$

Our objective is to minimize the total cost of trip-vehicle assignments. The first set of constraints ensure each request is served. We can introduce these as covering constraints since all sets are downward closed and costs are monotonic increasing, but we could have equivalently introduced them as set-partitioning constraints and required equality. The second set of constraints ensure each vehicle is assigned to at most one trip. Since we assume $\emptyset \in T$, we again could have required equality for these constraints.

3 On the LP Relaxation

We would like to use the LP relaxation of (2.1) as part of our algorithm. It is therefore natural to ask whether we can solve it in polynomial time under a succinct representation of T. We consider two approaches.

3.1 An Assumption on T. Suppose T is given explicitly as a polynomial-sized set. That is, $|T| = \text{poly}(|R|, |V|)$ and so we can directly write (and solve) the LP. We argue this assumption is not too bad. Since vehicles have a fixed seating capacity $k > 0$, the size of T is typically $O(n^k)$. This is not always the case since a vehicle that drops off passengers along the way may serve more than k requests. Nevertheless, tight quality of service constraints typically prevent large trips from being feasible (and hence can be excluded from T a priori, as in [1]).

Now, this assumption does not change the fact that we need to compute the coefficients c, which may be NP-hard. The following claim is easy to show.

LEMMA 3.1. *Suppose we have a feasible instance of the RTV assignment problem with T polynomial-sized. Given an α-approximation algorithm for c, we can α-approximate LP(2.1).*

Proof. Let x be an optimal solution for oracle-given coefficients c. Let x' be an optimal solution for α-approximate coefficients c'. Clearly $c'(x') \leq c'(x)$ and $c'(x) \leq \alpha \cdot c(x)$, which shows $c'(x') \leq \alpha \cdot c(x)$. ∎

For example, if the underlying routing problem were the metric TSP, we could 3/2-approximate c with Christofides' algorithm [12] (we can use this algorithm since, for most practical purposes in road networks, we may assume we operate on a symmetric metric space [36]). If the underlying graph were in addition planar, we could use Klein's polynomial time approximation scheme (PTAS) [25]. If the underlying routing problem were instead some generalization of the TSP, the argument would follow identically except we would use whatever approximation guarantee is available (which cannot improve on what is available for the TSP).

3.2 Dual Separation. Since the primal LP has exponentially many variables in the worst case, it is natural to consider its dual, which has polynomially many

variables but exponentially many constraints. While we remain unable to write down all dual constraints, we could in principle solve the dual LP using the ellipsoid method together with a dual separation algorithm.

Given a solution to a partially specified dual LP, a dual separation algorithm finds a violated dual constraint, if one exists. The polynomial time solvability of the dual separation problem implies the polynomial time solvability of the primal LP. Similarly, the polynomial time approximability of the dual separation problem could imply the polynomial time approximability of the primal LP, as in [8, 23, 18]. In any case, note that solving the dual separation problem (whether we can do it in polynomial time or not) generates dual constraints, which is equivalent to generating columns for the primal LP. Column generation is a popular strategy for solving large LPs, and is used within branch-and-price frameworks for solving large IPs [5]. In this section we explore this possibility.

Consider the dual linear program.

$$\begin{aligned} \max \quad & \sum_{r\in R} y_r - \sum_{v\in V} z_v \\ \text{s.t.} \quad & \\ & \sum_{r\in t} y_r - z_v \leq c_{tv}, \quad \forall t \in T, v \in V \\ & y_r \geq 0, \quad \forall r \in R \\ & z_v \geq 0, \quad \forall v \in V \end{aligned} \tag{3.2}$$

Given a polynomial time separation oracle for (3.2), we can use the ellipsoid method to solve it in polynomial time. Then, since the ellipsoid method only considers polynomially many constraints in (3.2), only polynomially many variables need to be written in the LP relaxation of (2.1). Now, (3.2) may be rewritten as follows.

$$\begin{aligned} \max \quad & \sum_{r\in R} y_r - \sum_{v\in V} z_v \\ \text{s.t.} \quad & \\ & (y, z_v) \in \mathcal{P}_v, \quad \forall v \in V \\ & y_r \geq 0, \quad \forall r \in R \\ & z_v \geq 0, \quad \forall v \in V \end{aligned} \tag{3.3}$$

Here, $\mathcal{P}_v$ is a polytope associated with vehicle $v \in V$ that is given by the constraints $\sum_{r\in t} y_r - z_v \leq c_{tv}$ for all $t \in T$ together with non-negativity constraints. Therefore, we may design a polynomial time separation oracle for (3.2) by designing a polynomial time separation oracle for $\mathcal{P}_v$ and iterating over all $v \in V$.

Consider any vehicle $v \in V$. We need a polynomial time subroutine that, given some (y, z_v), either certifies that $(y, z_v) \in \mathcal{P}_v$ or returns a violated constraint. In other words, we need to verify that $\sum_{r\in t} y_r - c_{tv} \leq z_v$ for all $t \in T$. The core of our separation problem for $\mathcal{P}_v$ is then

$$\max_{t\in T} \left\{ \sum_{r\in t} y_r - c_{tv} \right\}. \tag{3.4}$$

This is because this quantity is bounded by z_v if, and only if, $(y, z_v) \in \mathcal{P}_v$. We note that had we not made the simplifying assumption that any vehicle can serve any trip, the problem for vehicle $v \in V$ would need to optimize over $T(v) \subseteq T$. Presumably, $|T(v)| \ll |T|$ under tight quality of service constraints[3]. In fact, this structure is precisely exploited in [1, 44].

In some select cases, it may be possible to efficiently solve (3.4). For example, if we somehow knew T forms a matroid (e.g., a k-uniform matroid where $k > 0$ is a fixed vehicle seating capacity) and trip-vehicle assignment costs were additive with respect to request inclusion, we could use the greedy algorithm to optimally solve the problem [27]. However, by and large, instances of practical interest are far less structured.

Suppose the routing problem defining the cost coefficients c were the TSP. Then, we would identify (3.4) as an instance[4] of the *net-worth maximization* version of the prize-collecting TSP [24], a problem that is NP-hard, where the profits are given by the dual variables y. Therefore, one may ask if (3.4) can be approximated.

The closest related inapproximability result is that of Feigenbaum, Papadimitriou, and Shenker [16] for the *directed* prize-collecting Steiner tree problem. They show, via a gap reduction from SAT, that it is NP-hard to approximate the problem within any constant factor. Intuitively, one would expect their result to carry over to the directed prize-collecting TSP, but it seems hard to modify the gadget in [16] for this purpose. The difficulty is that adding the edges necessary to allow tours may enable unanticipated interactions between variables and their negations, which the tree structure avoided. We believe this is an open problem.

The reader may notice that some literature (for example, [13, 10, 38, 42]) mention the result of Feigenbaum et al. [16] as an inapproximability result for the *undirected* prize-collecting Steiner tree problem. Although not explicitly mentioned by Feigenbaum et al. [16], their gadget can be easily updated to work on undirected graphs [41]. The only change needed is to set the profit

[3] Assuming $T(v)$ is polynomial-sized is the same as assuming T is polynomial-sized, and so we have already considered this in Section 3.1. This is because there are only polynomially many vehicles and $\bigcup_{v\in V} T(v) = T$ assuming our instance is feasible.

[4] Strictly speaking, we require $T = 2^R$ so that maximizing over $t \in T$ is the same as maximizing over $t \subseteq R$. This corresponds to the setting with lax quality of service constraints, which is what may cause T to be exponential-sized to begin with.

of clause nodes to $2K$ and the cost of edges between literals and clauses to K. The reader may also notice that [49, 17] associate constant factor approximation results with the net-worth maximization version of the prize-collecting TSP. However, the references cited therein [7, 20] actually correspond to a different version of the prize-collecting TSP, namely the one in which a non-negative penalty is paid for each node excluded from the tour. The simple transformation between the two versions of the problem given in Section 10.2.2 of [49], while valid for optimal solutions, is not approximation preserving. See [2] for some tractable special cases and [24] for a catalogue of different versions of the prize-collecting Steiner tree problem.

Since any generalization of the TSP includes the TSP as a special case, an inapproximability result for the directed (or undirected) prize-collecting TSP would also hold for any of the more realistic directed (or undirected) routing problems, outlined in the beginning of this section, that could determine the coefficients c. This would already avert the design of an approximate separation oracle[5] to obtain a constant factor approximation for the LP relaxation of (2.1) in the style of [8, 23, 18]. Note that this would be in addition to the inherent difficulty with the mixed-sign objective of (3.2).

In practice, problem (3.4) appears as the pricing problem within branch-and-price frameworks for vehicle routing, possibly with additional service specific constraints. Pricing problems are typically solved via a variety of exact methods. See [49, 17] for surveys, where they refer it as the *profitable tour problem* and [44] for a particularly relevant example. See also [32, 28] for exact branch-and-cut and dual ascent-based branch-and-bound methods for the closely-related net-worth maximizing Steiner tree problem, respectively, and [31] for a survey on recent developments. We emphasize that, for this particular problem, these methods take exponential time in the worst case.

The unfavorable prospect outlined in this section motivates us to assume the candidate trip list T is precomputed and polynomial-sized whenever we formally consider the LP relaxation of (2.1).

[5] We say a separation oracle for $\mathcal{P}_v$ is α-approximate if, given some (y, z_v), it either certifies $(y, \frac{z_v}{\alpha}) \in \mathcal{P}_v$ or returns a violated constraint. An α-approximation algorithm for (3.4) would yield α-approximate separation oracle for $\mathcal{P}_v$. To see this, let z^* be the value obtained by an α-approximation algorithm for (3.4) and let t^* be a trip $t \in T$ achieving z^*. If $z^* > z_v$, we know t^* induces a violated constraint. Otherwise, for any $t \in T$ we have $\sum_{r \in t} y_r - c_{tv} \leq \frac{1}{\alpha}\left(\sum_{r \in t^*} y_r - c_{t^* v}\right) = \frac{z^*}{\alpha} \leq \frac{z_v}{\alpha}$ and so $(y, \frac{z_v}{\alpha}) \in \mathcal{P}_v$.

4 Randomized Rounding

We now turn to our randomized rounding algorithm. Consider a feasible instance of the problem and the version of (2.1) *with* equality constraints. We assume we can solve its LP relaxation (e.g., assuming $|T| = \text{poly}(|R|, |V|)$ as a formality) to obtain a fractional solution x of value LP(2.1). We further assume the cost coefficients $c \geq 0$ are oracle-given. α-approximate cost coefficients can be accounted for by Lemma 3.1. We consider two simple randomized rounding techniques, the second of which overcomes a deal-breaking shortcoming of the first.

In both cases, let $X_{tv} \in \{0, 1\}$ be a random variable indicating the assignment of vehicle $v \in V$ to trip $t \in T$. Let X be the set of X_{tv} random variables for all $(t, v) \in T \times V$. Let $X_v \subseteq X$ be the subset of X involving vehicle $v \in V$ and $X_r \subseteq X$ be the subset of X involving trips $t \in T$ such that $r \in t$.

4.1 Independent Rounding. Consider setting each $X_{tv} \in X$ independently to 1 with probability $0 \leq x_{tv} \leq 1$ and to 0 otherwise. Clearly $\mathbb{E}[c(X)] = \sum_{v \in V} \sum_{t \in T} c_{tv} x_{tv} = \text{LP}(2.1)$. However, the rounded solution may be infeasible.

One advantage of this technique is that we can easily upper bound the probability of a fixed request $r \in R$ being left unassigned. This is given by

$$\begin{aligned}\prod_{(t,v) \in T(r) \times V} (1 - \Pr[X_{tv} = 1]) &= \prod_{(t,v) \in T(r) \times V} (1 - x_{tv}) \\ &\leq e^{-\sum_{(t,v) \in T(r) \times V} x_{tv}} \\ &= e^{-1},\end{aligned}$$

where the last equality holds by the LP constraints. We will later see how we can handle the event in which a request is over-assigned. The bigger issue is that a vehicle $v \in V$ is over-assigned with non-zero probability (i.e., there are two or more trips $t \in T$ such that $X_{tv} = 1$). This is problematic because, even if we can upper bound the probability of this event, there is always the possibility that we over-commit our fleet. We could in principle re-sample all random variables, potentially many times, until we reach an assignment with no over-commitments. Alternatively, we can bypass this issue altogether through a dependent rounding technique.

4.2 Dependent Rounding. Recall each vehicle $v \in V$ satisfies $\sum_{t \in T} x_{tv} = 1$ by the LP constraints. We interpret this as a vehicle-specific probability distribution over the trips. Therefore, independently for each vehicle $v \in V$, assign it to a randomly chosen trip, where the probability of assigning it to trip $t \in T$ is given by $0 \leq x_{tv} \leq 1$. Let X be the set of random variables

corresponding to this step and observe that they are no longer independent.

Note that a request $r \in R$ may appear in multiple trip-vehicle assignments. Allow r to pick one arbitrarily. Here we are crucially using the fact that T is downward closed. Further, by the monotonicity of the cost coefficients $c \geq 0$ with respect to request inclusion, doing this cannot increase the cost of our solution. Define X' analogously to X, except this time it corresponds to the final output after the multiplicity correction step. The following result is immediate.

LEMMA 4.1. *The expected cost of the final RTV assignment X' is at most LP(2.1).*

Clearly, our procedure ensures each vehicle is assigned to exactly one trip (resolving our previous issue). However, it may still leave some unassigned requests. Nevertheless, we can still bound the probability of this event for any fixed request $r \in R$.

LEMMA 4.2. *For any request $r \in R$, the probability of it being left unassigned in the final RTV assignment X' is at most $1/e$.*

Proof. Since r is assigned in X if and only if it is assigned in X', we can focus on the former. Let Y_v be a binary random variable indicating whether vehicle $v \in V$ is assigned, in X, to a trip $t \in T$ such that $r \in t$. Then, $\Pr[Y_v = 1] = \sum_{t \in T(r)} x_{tv}$.

Note that r is left unassigned in X if and only if $Y_v = 0$ for all $v \in V$. Now, since the rounding is done independently at the vehicle level, the variables Y_v for $v \in V$ are independent, and so r is left unassigned with probability

$$\prod_{v \in V} (1 - \Pr[Y_v = 1]) \leq e^{-\sum_{v \in V} \Pr[Y_v = 1]} = e^{-\sum_{(t,v) \in T(r) \times V} x_{tv}} = e^{-1},$$

where, as before, the last equality holds by the LP constraints. ∎

COROLLARY 4.1. *The expected number of requests left unassigned in the final RTV assignment is at most $\frac{|R|}{e}$ (i.e., less than 36.8% of all requests).*

Proof. By linearity of expectation. ∎

Lemma 4.1 and Corollary 4.1, together with the fact that LP(2.1) $\leq$ OPT, imply Theorem 1.1.

For any fixed request $r \in R$ we also show that, in the multiplicity correction step, it is unlikely to have too many trips/vehicles to choose from. That is, we bound the probability of $\sum_{(t,v) \in X_r} X_{tv}$ deviating above its unit mean by $\delta \in \mathbb{Z}_{\geq 1}$-many trips. The proof uses a multiplicative Chernoff bound (see for example [37]).

LEMMA 4.3. *For any request $r \in R$, the probability of it being over-assigned by $\delta \in \mathbb{Z}_{\geq 1}$-many trips in the preliminary RTV assignment X is at most $\frac{e^\delta}{(1+\delta)^{1+\delta}}$.*

We note that we could have similarly used a multiplicative Chernoff bound to produce an alternate proof of Lemma 4.2. Lastly, although we did not need this in our analysis, we note that the sets of random variables X_v for $v \in V$, X_r for $r \in R$, and even X itself, are each negatively associated.

We end this section by introducing a class of instances showing that *i*) the integrality gap of the LP relaxation of (2.1) is at least 2, and *ii*) the upper bound of $1/e$ on the rejection rate is tight for our algorithm.

We are given two vehicles of capacity $k > 0$ together with $k + 1$ requests. Any trip of size $\leq k$ can be assigned to either vehicle at unit cost (e.g., all requests go from the same origin to the same destination). Figure 1 depicts the shareability graph of our instance when $k = 2$. In the kth instance, an optimal integer solution has cost 2. Meanwhile, an optimal fractional solution assigns fractional value $1/\binom{k+1-1}{k-1}$ to each of the $\binom{k+1}{k}$ distinct k-passenger trips, each at unit cost. The integrality gap of the kth instance is then

$$\frac{2}{\binom{k+1}{k} \cdot \frac{1+(k-1)\epsilon}{\binom{k+1-1}{k-1}}} = 2 \cdot \frac{k}{k+1}.$$

Since $\lim_{k \to \infty} 2 \cdot \frac{k}{k+1} = 2$, the integrality gap of the LP relaxation of (2.1) is at least 2. The intuitive reason for this phenomenon is that there is no real distinction between the vehicles. Then, although these instances are trivial in a practical sense, the generality of (2.1) (which is meant to model distinct vehicles) causes it to miss the obvious structure.

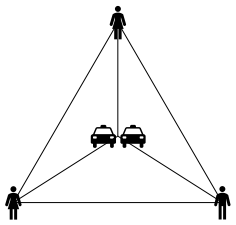


Figure 1: In this instance, any trip of size 2 or smaller can be served by either vehicle at unit cost. An optimal integer solution has cost 2, whereas an optimal fractional solution will be half-integral and have cost 3/2. The integrality gap of this instance is 4/3.

Now we slightly modify our instance class to show the upper bound of $1/e$ on the rejection rate is tight for

our algorithm. In the kth instance, rather than having 2 vehicles, we instead have $\binom{k+1}{k} = k+1$ vehicles. Assignment costs do not change, and so the costs of optimal integer or fractional solutions do not change either. However, for fractional solutions, we may reach an (admittedly pathological) optimal solution in which each vehicle is assigned to exactly one of the $k+1$ distinct k-passenger trips with fractional value $1/k$ (and to the empty trip with fractional value $\frac{k-1}{k}$ at zero cost). Then, any fixed request $r \in R$ has uniform fractional support on k distinct vehicles, and so the probability of it being left unassigned by our rounding algorithm is exactly $\left(\frac{k-1}{k}\right)^k$. Since $\lim_{k\to\infty}\left(\frac{k-1}{k}\right)^k = \frac{1}{e}$, Lemma 4.2 is tight in the limit[6].

5 Penalty Version and Multiple Rounds

The main shortcoming of our algorithm is that it may leave some unassigned requests. Typically, a remedy to this is to repeat the procedure to cover any missed assignments (at the expense of augmenting the expected objective value). However, vehicles cannot be assigned to more than one trip and so we cannot do this in general. Nevertheless, in practice and by design, any unassigned requests are carried over to the next round of batch assignments (e.g., 5 to 30 seconds later). In this section, we study the effects of this design feature on the subsequent assignment of initially unassigned requests.

Pick any request $r \in R$. For now, consider the probability of r being left unassigned after n *feasible* rounds. Let A_i be the event that r is unassigned after i feasible rounds. Then,

$$\begin{aligned}\Pr(A_n) &= \Pr(A_n|A_1)\Pr(A_1) + \Pr(A_n|A_1^c)\Pr(A_1^c)\\ &= \Pr(A_n|A_1)\Pr(A_1) \le \frac{1}{e}\Pr(A_n|A_1),\end{aligned}$$

where the second equality holds since $\Pr(A_n|A_1^c) = 0$ and the inequality holds by Lemma 4.2. Likewise, we can write

$$\begin{aligned}\Pr(A_n|A_1) &= \Pr(A_n|A_2,A_1)\Pr(A_2|A_1)\\ &\quad + \Pr(A_n|A_2^c,A_1)\Pr(A_2^c|A_1)\\ &= \Pr(A_n|A_2,A_1)\Pr(A_2|A_1)\\ &\le \frac{1}{e}\Pr(A_n|A_2,A_1),\end{aligned}$$

where the second equality holds since $\Pr(A_n|A_2^c,A_1) = 0$ and the inequality holds since, given that r is unassigned in the first feasible round, in the second feasible round we attempt to assign it but fail to do so with probability at most $1/e$ by Lemma 4.2. We can extend this argument to n feasible rounds, where $\Pr(A_n|A_n,\cdots,A_1) = 1$, to obtain the following.

[6]We note that if a request $r \in R$ has not necessarily uniform fractional support on k distinct vehicles, we could have used this limit argument together with the inequality of arithmetic and geometric means to obtain yet another proof of Lemma 4.2.

COROLLARY 5.1. *For any request $r \in R$, the probability of it being left unassigned after n feasible rounds of batch assignments is at most $(1/e)^n$.*

As in [1], we are implicitly accounting for requests that were already picked up by ensuring each term c_{tv} includes the cost of routing passengers inside the vehicle, if any, until the round in which they are dropped off.

Admittedly, there does not always exist a feasible solution to the RTV assignment problem. For example, this may occur if vehicle supply is low relative to travel demand. To formally handle this, one can introduce a dummy vehicle v_r for each request $r \in R$. This vehicle can only be assigned to trip $\{r\} \in T$, in which case we incur a large cost $\kappa_r \ge 0$ representing the penalty for ignoring r, or the empty trip $\emptyset \in T$ at zero cost. This is in fact the strategy followed in practice.

By doing this, we actually ensure we meet the feasibility assumption required by Theorem 1.1. We moreover preserve the monotonicity assumption required on the cost coefficients. Therefore, we can again use our randomized algorithm, although our performance guarantee is now with respect to the modified objective function which now includes the penalty terms. Still, a request $r \in R$ is unassigned (in terms of ILP feasibility) with probability at most $1/e$. In this case, we can formally cover r by assigning it to its dummy vehicle v_r and paying the corresponding penalty κ_r. This produces a feasible ILP solution yielding the following corollary.

COROLLARY 5.2. *Suppose T is polynomial-sized. If trip-vehicle costs are oracle-given and monotonic increasing w.r.t. request inclusion, and there is a penalty $\kappa_r \ge 0$ for ignoring request $r \in R$, there is a polynomial time randomized algorithm yielding a solution of expected cost at most $OPT + \frac{1}{e}\sum_{r\in R}\kappa_r$, where OPT is the optimal feasible sum of trip-vehicle assignment costs and penalties. If trip-vehicle assignment costs are α-approximate, we pay an additional factor of α in the first term of the expected cost.*

In practice, if there is enough vehicle supply to clear demand on every round (intuitively, the average number of vehicle seats available is greater than or equal to the request arrival rate times the average time spent in the system [30]), one can prioritize unassigned requests by augmenting their penalty terms in subsequent rounds of batch assignments[7].

We now interpret Corollary 5.1 within the context of the penalty version of the problem.

REMARK 5.1. *Corollary 5.1 is applicable to the penalty version of the problem in the following sense: For any request $r \in R$, with high probability over a number of rounds, the algorithm is not forced to ignore r, but rather assigns it to a vehicle or purposely ignores it, in accordance with the sequence of LP solutions.*

[7]If there is not enough vehicle supply, unassigned request queue up and quality of service is unmet, leading to reneging customers.

That is to say, if a request is ignored by our algorithm after a number of rounds, it is increasingly likely to be so because of the LP solutions and not because of a rounding error.

Lastly, we note than when the penalty terms eclipse the assignment costs (e.g., routing costs), the penalty version of the problem is essentially equivalent to a maximum coverage version of the problem, where the objective is to serve as many requests as possible given the available fleet. This maximum coverage problem is still subject to the quality of service constraints that are implicit in the RTV graph, and therefore can be posed as an instance of the cardinality version of the maximum coverage problem with group budget constraints, for which a $1/e$-approximation algorithm is known [11], assuming T is given explicitly. Our randomized algorithm matches this guarantee in the sense that the expected fraction of requests that could have been covered but were not is at most $1/e$. If assignment costs are non-zero, our method provides the additional benefit that the quality of the assignment (e.g., in terms of the routing costs of assigned requests) is also optimized as part of the LP.

6 Computational Experiments

We now share our computational experience. To evaluate our algorithm, we first run a number of simulations of a day in the operations of a high-capacity ridesharing system. We use publicly available NYC Taxi and Limousine Commission (TLC) data [39] and an implementation of the Alonso-Mora et al. framework [1]. In each simulation, we solve the penalty version of the distance-minimizing RTV assignment problem using a commercial ILP solver. Next, we gather all instances of the RTV assignment problem solved throughout our simulations and solve them once again using our techniques. Note that we are *not* comparing simulation paths to evaluate system-level performance. Rather, we use simulation paths to produce sets of test instances. This way we can have a side-by-side comparison of our algorithm against the use of a commercial ILP solver (the same we use to solve our LP relaxations). Each simulation produces $1,440$ instances.

We run a total of 10 simulations, one for each combination[8] of 500 or 1000 vehicles and vehicles of capacity $k = 2, 3, 4, 5, 6$. For simulations with 500 vehicles, the mean number of requests per instance is around 370 requests[9]. Likewise, for simulations with 1000 vehicles, the mean number of requests per instance is around 560 requests.

In addition, we evaluate a deterministic rounding heuristic inspired by our randomized rounding algorithm. The only difference between the two is that, in our heuristic, each vehicle $v \in V$ is deterministically assigned to the trip $t^* \in T$ with largest fractional value (i.e., vehicle $v \in V$ is assigned to trip $t^* = \arg\max_{t \in T}\{x_{tv}\}$). Note that we may still need to execute the multiplicity correction step.

[8] For vehicle capacities $k \geq 4$, our simulations employ time-outs in the trip generation step.

[9] Even with a fixed number of vehicles, different capacities k yield different simulation paths (e.g., with different service/reneging rates), and so the instances are not identical.

Table 1 presents statistics on the percentage of rejected requests for each method. As expected, solving the ILP leads to the lowest rate of request rejection. Moreover, the rejection rate is monotonic decreasing in the number of vehicles and in the vehicle capacity k. Surprisingly, our rounding algorithms are worse than the ILP on this metric by no more than one percentage point, and often much less. For our randomized rounding algorithm, this is far better than the theoretical guarantee provided in Theorem 1.1. We explore this further toward the end of this section. We also note that our deterministic heuristic slightly outperforms our randomized algorithm on this metric. Since the rejection rates are so close, the assignment costs (in terms of vehicle kilometers traveled) is comparable for all three methods. Solving the ILP leads to a slightly higher number of vehicle kilometers traveled since slightly less requests are rejected.

Table 1: Request rejection.

500 Veh.		Requests Rejected [%]		
k		ILP	Rand. Rnd.	Det. Rnd.
2	Mean	27.09	27.37	27.32
	Med	31.50	31.55	31.56
3	Mean	20.31	20.73	20.65
	Med	23.83	23.98	24.00
4	Mean	15.41	15.84	15.77
	Med	17.82	18.12	18.12
5	Mean	12.69	13.14	13.07
	Med	14.50	14.88	14.80
6	Mean	11.51	11.93	11.85
	Med	12.91	13.29	13.20

1000 Veh.		Requests Rejected [%]		
k		ILP	Rand. Rnd.	Det. Rnd.
2	Mean	13.06	13.58	13.53
	Med	13.33	13.76	13.69
3	Mean	8.84	9.74	9.57
	Med	8.36	9.24	9.09
4	Mean	6.12	7.11	6.93
	Med	5.39	6.43	6.24
5	Mean	4.69	5.61	5.46
	Med	3.95	4.88	4.77
6	Mean	4.15	5.02	4.90
	Med	3.44	4.44	4.31

Table 2 summarizes the performance of our implementation of all three methods in terms of computation time. We observe that both rounding methods are faster than solving the ILP to optimality, with the percent improvement for mean and median values ranging between $2 - 7\%$. Importantly, we observe that the percent improvement for mean values is consistently higher than the percent improvement for median values. This shows the computation time distributions are skewed to the right, and so the percent improvement is higher for the worst case (i.e., slowest) instances. Arguably, these are the critical instances in real-time decision making, and so LP rounding methods provide a way to improve on the worst case computation time performance.

Lastly, we address the question of why the rejection rate of our rounding algorithms is so close to that achieved by

Table 2: Computation time.

500 Veh.		ILP	Rand. Rnd.		Det. Rnd.	
k		Time [s]	Time [s]	% Diff	Time [s]	% Diff
2	Mean	0.152	0.154	0.97	0.144	-5.01
	Med	0.154	0.158	2.59	0.150	-2.91
3	Mean	0.229	0.222	-2.99	0.215	-6.26
	Med	0.172	0.175	2.12	0.168	-2.03
4	Mean	0.319	0.303	-4.77	0.297	-6.91
	Med	0.199	0.197	-1.19	0.189	-5.42
5	Mean	0.324	0.309	-4.66	0.302	-6.86
	Med	0.215	0.212	-1.80	0.203	-5.69
6	Mean	0.310	0.297	-4.26	0.289	-6.85
	Med	0.217	0.211	-2.90	0.203	-6.64

1000 Veh.		ILP	Rand. Rnd.		Det. Rnd.	
k		Time [s]	Time [s]	% Diff	Time [s]	% Diff
2	Mean	0.731	0.710	-2.82	0.697	-4.64
	Med	0.761	0.756	-0.63	0.739	-2.90
3	Mean	1.259	1.188	-5.65	1.178	-6.47
	Med	0.996	0.959	-3.76	0.951	-4.50
4	Mean	2.075	1.942	-6.39	1.935	-6.74
	Med	1.472	1.411	-4.14	1.406	-4.47
5	Mean	2.212	2.073	-6.26	2.067	-6.54
	Med	1.708	1.640	-3.96	1.625	-4.88
6	Mean	2.124	1.994	-6.11	1.984	-6.60
	Med	1.710	1.639	-4.15	1.631	-4.61

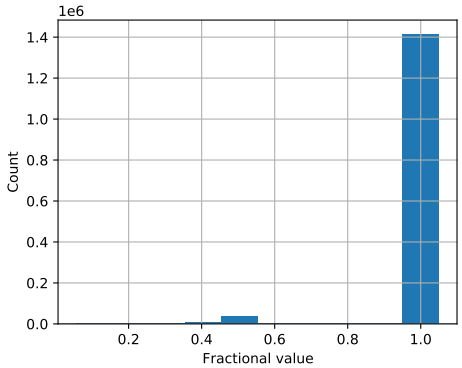


Figure 2: Histogram of LP support.

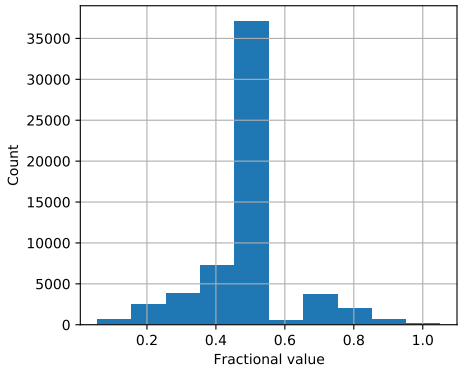


Figure 3: Histogram of non-integral LP support.

the ILP. Figure 2 shows a histogram of the fractional values supported on the LP solutions across all 1,440 instances of a simulation with 1000 vehicles of capacity $k = 4$. We see that the overwhelming majority ($\sim$ 96%) of non-zero fractional values are in fact integral. Of course, the rounding algorithms make no mistakes on these assignments. We extend this observation in Figure 3, which shows a histogram of the fractional values strictly between 0 and 1. We observe that whenever an assignment is not integral, it is likely to be half-integral ($\sim$ 62% of non-integral variables are half-integral). In half-integral cases, we can a posteriori tighten the probability that our randomized rounding algorithms ignores a fixed request $r \in R$ to be $\leq 1/4$.

We have been unable to concretely characterize reasons why the LP relaxation of (2.1) is so tight. We can produce toy examples as the one in Figure 1 (and more generally the class of instances described in Section 4.2). However, these examples are neither realistic nor exhaustive, and so we believe tackling this problem (e.g., in a data-driven way, with data sets besides the NYC TLC data [39]) is an interesting research direction. In particular, we do not know whether the LP is *always* tight in practice. If there are practical instances in which the LP is not as tight, they will likely be harder for an ILP solver, and so they might more strongly showcase the advantage of LP rounding.

Acknowledgments

We thank Matthew Zalesak for sharing his implementation of the Alonso-Mora et al. [1] framework with us, and the anonymous reviewers for helpful feedback.

References

[1] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, *On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment*, Proceedings of the National Academy of Sciences, 114 (2017), pp. 462–467.

[2] E. Angelelli, C. Bazgan, M. G. Speranza, and Z. Tuza, *Complexity and approximation for traveling salesman problems with profits*, Theoretical Computer Science, 531 (2014), pp. 54–65.

[3] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*, Princeton university press, 2006.

[4] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, *Approximation algorithms for deadline-tsp and vehicle routing with time-windows*, in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, 2004, pp. 166–174.

[5] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, *Branch-and-price: Column generation for solving huge integer programs*, Operations research, 46 (1998), pp. 316–329.

[6] X. Bei and S. Zhang, *Algorithms for trip-vehicle assignment in ride-sharing.*, in AAAI, vol. 18, 2018, pp. 3–9.

[7] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson, *A note on the prize collecting*

traveling salesman problem, Mathematical programming, 59 (1993), pp. 413–420.

[8] R. Carr and S. Vempala, *Randomized metarounding*, in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000, pp. 58–62.

[9] P. Chalasani and R. Motwani, *Approximating capacitated routing and delivery problems*, SIAM Journal on Computing, 28 (1999), pp. 2133–2149.

[10] O. Chapovska and A. P. Punnen, *Variations of the prize-collecting steiner tree problem*, Networks: An International Journal, 47 (2006), pp. 199–205.

[11] C. Chekuri and A. Kumar, *Maximum coverage problem with group budget constraints and applications*, in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Springer, 2004, pp. 72–83.

[12] N. Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[13] A. Costa, J.-F. Cordeau, and G. Laporte, *Steiner tree problems with profits*, INFOR: information systems and operational research, 44 (2006), pp. 99–115.

[14] Data Driven NYC, *Reengineering urban transit // Daniel Ramot and Saar Golde, Via (Firstmark's data driven)*. [YouTube video], April 2017. Accessed Oct. 28, 2020.

[15] Y. Dumas, J. Desrosiers, and F. Soumis, *The pickup and delivery problem with time windows*, European journal of operational research, 54 (1991), pp. 7–22.

[16] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker, *Sharing the cost of multicast transmissions*, Journal of computer and system sciences, 63 (2001), pp. 21–41.

[17] D. Feillet, P. Dejax, and M. Gendreau, *Traveling salesman problems with profits*, Transportation science, 39 (2005), pp. 188–205.

[18] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, *Tight approximation algorithms for maximum separable assignment problems*, Mathematics of Operations Research, 36 (2011), pp. 416–431.

[19] N. Garg, G. Konjevod, and R. Ravi, *A polylogarithmic approximation algorithm for the group steiner tree problem*, Journal of Algorithms, 37 (2000), pp. 66–84.

[20] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM Journal on Computing, 24 (1995), pp. 296–317.

[21] X. Guo, Y. Liu, and S. Samaranayake, *Solving the school bus routing problem at scale via a compressed shareability network*, in 2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2018, pp. 1900–1907.

[22] J. Hoogeveen, *Analysis of christofides' heuristic: Some paths are more difficult than cycles*, Operations Research Letters, 10 (1991), pp. 291–295.

[23] K. Jain, M. Mahdian, and M. R. Salavatipour, *Packing steiner trees*, in SODA, vol. 3, 2003, pp. 266–274.

[24] D. S. Johnson, M. Minkoff, and S. Phillips, *The prize collecting steiner tree problem: theory and practice*, in Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2000, pp. 760–769.

[25] P. N. Klein, *A linear-time approximation scheme for planar weighted tsp*, in 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05), IEEE, 2005, pp. 647–656.

[26] E. Lawler, J. K. Lenstra, A. Rinnooy Kan, and D. B. Shmoys, *The traveling salesman problem: a guided tour of combinatorial optimization*, Wiley-Interscience Series in Discrete Mathematics, 1985.

[27] E. L. Lawler, *Combinatorial optimization: networks and matroids*, Courier Corporation, 2001.

[28] M. Leitner, I. Ljubić, M. Luipersbeck, and M. Sinnl, *A dual ascent-based branch-and-bound framework for the prize-collecting steiner tree and related problems*, INFORMS Journal on Computing, 30 (2018), pp. 402–420.

[29] S. Li, M. Li, and V. C. Lee, *Trip-vehicle assignment algorithms for ride-sharing*, in International Conference on Combinatorial Optimization and Applications, Springer, 2020, pp. 681–696.

[30] J. D. Little, *A proof for the queuing formula: $L= \lambda w$*, Operations research, 9 (1961), pp. 383–387.

[31] I. Ljubić, *Solving steiner trees: Recent advances, challenges, and perspectives*, Networks, 77 (2021), pp. 177–204.

[32] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti, *An algorithmic framework for the exact solution of the prize-collecting steiner tree problem*, Mathematical programming, 105 (2006), pp. 427–449.

[33] M. Lowalekar, P. Varakantham, and P. Jaillet, *Zac: A zone path construction approach for effective real-time ridesharing*, in Proceedings of the International Conference on Automated Planning and Scheduling, vol. 29, 2019, pp. 528–538.

[34] ———, *Competitive ratios for online multi-capacity ridesharing*, in Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 2020, pp. 771–779.

[35] K. Luo and F. C. Spieksma, *Approximation algorithms for car-sharing problems*, in International Computing and Combinatorics Conference, Springer, 2020, pp. 262–273.

[36] J. C. Martínez Mori and S. Samaranayake, *Bounded asymmetry in road networks*, Scientific reports, 9 (2019), pp. 1–9.

[37] R. Motwani and P. Raghavan, *Randomized algorithms*, Cambridge university press, 1995.

[38] C. Nagarajan, Y. Sharma, and D. P. Williamson, *Approximation algorithms for prize-collecting network design problems with general connectivity requirements*,

in International Workshop on Approximation and Online Algorithms, Springer, 2008, pp. 174–187.
[39] NYC Taxi and Limousine Commission, *TLC trip record data*. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed: February 27, 2021.
[40] M. Ota, H. Vo, C. Silva, and J. Freire, *Stars: Simulating taxi ride sharing at scale*, IEEE Transactions on Big Data, 3 (2016), pp. 349–361.
[41] A. Paul and D. Freund. Personal communication, November 2020.
[42] A. Paul, D. Freund, A. Ferber, D. B. Shmoys, and D. P. Williamson, *Budgeted prize-collecting traveling salesman and minimum spanning tree problems*, Mathematics of Operations Research, 45 (2020), pp. 576–590.
[43] P. Raghavan and C. D. Thompson, *Randomized rounding: a technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[44] C. Riley, A. Legrain, and P. Van Hentenryck, *Column generation for real-time ride-sharing operations*, in International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2019, pp. 472–487.
[45] K. Ruland and E. Rodin, *The pickup and delivery problem: Faces and branch-and-cut algorithm*, Computers & mathematics with applications, 33 (1997), pp. 1–13.
[46] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, *Quantifying the benefits of vehicle pooling with shareability networks*, Proceedings of the National Academy of Sciences, 111 (2014), pp. 13290–13294.
[47] D. B. Shmoys and D. P. Williamson, *Analyzing the held-karp tsp bound: A monotonicity property with application*, Information Processing Letters, 35 (1990), pp. 281–285.
[48] A. Simonetto, J. Monteil, and C. Gambella, *Real-time city-scale ridesharing via linear assignment problems*, Transportation Research Part C: Emerging Technologies, 101 (2019), pp. 208–232.
[49] P. Toth and D. Vigo, *Vehicle routing: problems, methods, and applications*, SIAM, 2014.
[50] Y. Xu, J. Qi, R. Borovica-Gajic, and L. Kulik, *Geoprune: Efficiently matching trips in ride-sharing through geometric properties*, in 32nd International Conference on Scientific and Statistical Database Management, 2020, pp. 1–12.