HiBGT: High-Performance Bayesian Group Testing for COVID-19

Weicong Chen
Dept. of Computer and Data Sciences
Case Western Reserve University
Cleveland, USA
wxc326@case.edu

Curtis Tatsuoka
Dept. of Medicine
University of Pittsburgh
Pittsburgh, USA
cut4@pitt.edu

Xiaoyi Lu
Dept. of Computer Science and Engineering
University of California Merced
Merced, USA
xiaoyi.lu@ucmerced.edu

***Abstract*—The COVID-19 pandemic has necessitated disease surveillance using group testing. Novel Bayesian methods using lattice models were proposed, which offer substantial improvements in group testing efficiency by precisely quantifying uncertainty in diagnoses, acknowledging varying individual risk and dilution effects, and guiding optimally convergent sequential pooled test selections. Computationally, however, Bayesian group testing poses considerable challenges as computational complexity grows exponentially with sample size. HPC and big data stacks are needed for assessing computational and statistical performance across fluctuating prevalence levels at large scales. Here, we study how to design and optimize critical computational components of Bayesian group testing, including lattice model representation, test selection algorithms, and statistical analysis schemes, under the context of parallel computing. To realize this, we propose a high-performance Bayesian group testing framework named HiBGT, based on Apache Spark, which systematically explores the design space of Bayesian group testing and provides comprehensive heuristics on how to achieve high-performance, highly scalable Bayesian group testing. We show that HiBGT can perform large-scale test selections ($> 2^{50}$ state iterations) and accelerate statistical analyzes up to 15.9x (up to 363x with little trade-offs) through a varied selection of sophisticated parallel computing techniques while achieving near linear scalability using up to 924 CPU cores.**

***Index Terms*—Group testing, Bayesian, Lattice model, Apache Spark, COVID-19**

I. Introduction

Since the outbreak of COVID-19, there has been a renewed interest in group testing due to the dire need for widespread testing [1]–[4]. As new outbreaks emerge, large-scale and repeated testing will play an essential role in future surveillance. Efficiencies of scale in testing are needed, and group testing can provide massive gains.

Originated by Dorfman [5] in 1943, the intuitive group testing formulation is described as follows: if biomarker samples from N subjects are pooled, and if the prevalence is low, most likely the test results for the pooled sample will return negative, indicating that all N subjects are negative, using only one test. Otherwise, a positive result would indicate there is at least one positive sample present among the pool. Dorfman suggested that subsequent individual-level tests be conducted for each subject. The Dorfman group testing approach has found use in various applications, including COVID-19 testing. Nonetheless, this approach can be problematic as it fails to recognize the presence of testing error, e.g., dilution effect [6], in which pooled tests may fail to return positive results (false negatives) due to insignificant viral loads when pooled samples contain few positives relative to a large number of negatives. Another compelling aspect of COVID-19 is the constantly changing prevalence and individual risk levels, such as through seasonality, variable vaccination rates, and the emergence of new variants. Deploying Dorfman's group testing or even some of the latest group testing approaches [7], [8] under these fluctuating circumstances may result in inefficient classification (i.e., use more tests than testing individually) or failure (e.g., unacceptable false positive/negative rates).

A. Motivation

In comparison, our previous study proposed an innovative group testing approach by using lattice models and Bayesian analysis to guide test selections [9]. This framework allows consideration of potential sources of testing error, such as dilution effects, as well as modeling heterogeneous individual-level risk among samples. This approach is promising as it achieves accurate, large-scale, flexible, and efficient group testing. Figure 1 presents a high-level overview of our Bayesian group testing workflow, along with the motivation and computational challenges we recognize and address in this paper.

The workflow of Bayesian group testing can be summarized as follows: the possible diagnostic outcomes of subjects can be represented using a *lattice model*. Acknowledging varying local prevalence and individual risk levels are done in a Bayesian manner through prior probability specifications. By iteratively performing a set of *test selection algorithms*, i.e., *Bayesian Halving Algorithm (BHA)* and *k-step Lookahead Halving Algorithm (k-BHA)*, and updating posterior probability distributions on this lattice model, sequences of pooled test selections can be made, and tree-based *statistical analysis* of classification performance can be conducted.

Preliminary studies have shown that Bayesian group testing can achieve outstanding statistical performance under varied prevalence and individual risk levels while considering test errors, e.g., it constantly reaches over 99.5% correctness in identifying positive subjects with less than 0.1% false pos-

This work was supported in part by the NSF research grant DRL #1561716 and CCF #2132049.

2640-0316/22/$31.00 ©2022 IEEE
DOI 10.1109/HiPC56025.2022.00033

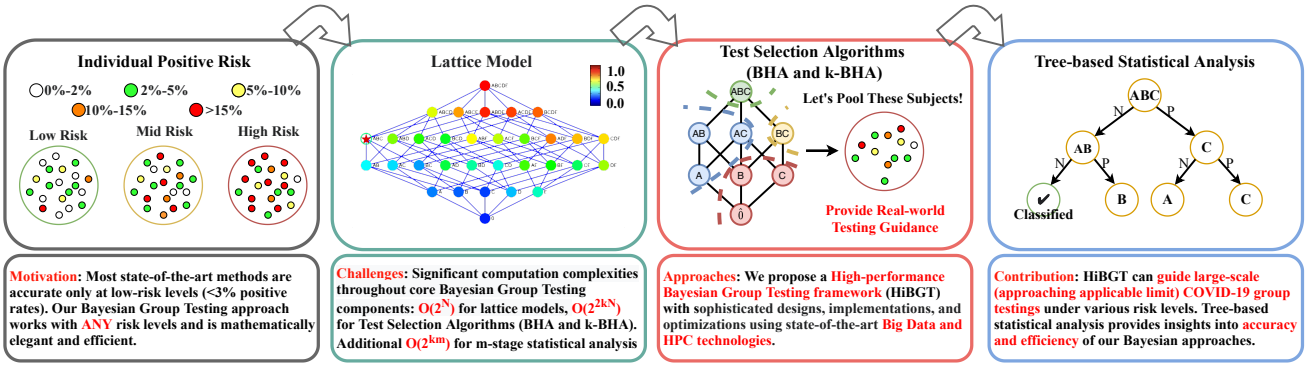


Fig. 1: Overview of Bayesian Group Testing Workflow, Motivation, Challenges, Our Approaches, and Contribution

itive/negative across different prevalence and individual risk levels, while saving up to 7x in numbers of expected tests compared to individual testing. However, this work suffers from significant computational challenges. Its modeling complexity, i.e., constructing lattice models, its test selection algorithms, i.e., BHA and k-BHA, and its tree-based exhaustive statistical analysis setting, will all grow exponentially as the number of subjects increases. We illustrate these data and algorithmic complexities using $N = 25$, which is approaching an upper limit for practical COVID-19 group testing application [10], [11] i.e., dilution effects were still manageable relative to pooled test accuracy. At $N = 25$, a corresponding lattice model will comprise 2^{25} (over 33 million) different states, with each state corresponding to a unique test selection. In order to find the best test selection, BHA needs to iterate through these states and perform computation for over 2^{50} (1.15×10^{15}) times; using k-BHA, k=2, the required iterations will immediately explode to 2^{100}. Moreover, the tree-based statistical analysis adds another layer of complexity on top of test selections, as it can potentially spawn millions of lattice models along branches, requiring millions of test selections. As can be seen, without leveraging state-of-the-art HPC stacks and sophisticated algorithmic optimizations, this Bayesian group testing approach will quickly become impossible to compute when N gets large.

B. Contribution

To address these challenges, this paper adopts a three-step research methodology illustrated in Figure 1 to explore the design space for the intricate interplay across Bayesian group testing components, including modeling, i.e., how to construct a lattice model efficiently; Bayesian-based test selection algorithms, i.e., how to optimize BHA and k-BHA by leveraging parallel computing; and tree-based statistical analysis, i.e., how to design tree structures for performance evaluation and to exploit Bayesian probabilities on lattice models for acceleration.

We realize our research by developing a high-performance Bayesian group testing framework based on Apache Spark [12], named HiBGT. The framework aims to deliver a high-performance, scalable, easy-to-use Bayesian group testing system and to show the effectiveness and efficiency of our optimized modeling, test selection algorithms, and statistical analysis workflow through large-scale parallel computing and is the first framework to do so for Bayesian group testing. In HiBGT, we propose numerous novel designs and optimizations to achieve this goal, comprehensively covering each step in our research methodology. For example, we propose three data and computation parallelisms that target different computation stages and allow a dynamic combination to achieve better computation efficiency and load balance. We propose algorithmic optimizations for k-BHA to drastically bring down its asymptotic complexity. We also propose three tree construction schemes for tree-based statistical analysis to adapt to different real-world analysis scenarios and requirements.

We systematically evaluate our artifacts on two HPC clusters, one is Intel-based with up to 924 CPU cores, and the other is AMD-based with up to 528 CPU cores. Based on evaluations, we show that HiBGT can ① handle large-scale test selections, e.g., $N = 25$ in 4 hours ($> 2^{50}$ state iterations) through sophisticated intra-lattice parallelism; ② accelerate statistical analyzes up to 15.9x, or up to 363x with little trade-offs by leveraging a varied selection of tree parallelization schemes compared to naive parallelization; and ③ achieve near-linear scaling efficiency in both test selections and statistical analyzes.

II. Lattice Model

In Bayesian group testing, the classification objective is to identify the "true" profile that characterizes positive and negative disease status among the individuals being considered for pooling. Given N subjects considered for pooling and classification, there are 2^N possible profiles of individual-level negative and positive diagnoses. A natural partial ordering arises among the states through inclusion, with states representing the subset of subjects that are negative. For example, the top element is the state reflecting that all subjects are negative, the bottom element is the state that all subjects are positive, and other states represent a mix of negatives and positives. Lattices are partially ordered sets that further assume

for any two elements, and there is a unique greatest lower bound and unique least upper bound. This structure is key to understanding how statistical discrimination between states can occur in Bayesian group testing. More details on lattice classification models are provided in [9], [13]–[16].

In a Bayesian setting, each state j in the lattice model is associated with a value $\pi(j)$, which reflects the posterior probability value that state j is the true profile. Here, a key concept in lattice models is the notion of an *up-set*, which is defined as follows:

Definition II.1 (Up-set of a State in a Lattice)**.** For a state s in the lattice, the up-set $\uparrow s$ is the subset of states within the lattice that are at least as great as (i.e. contain) s.

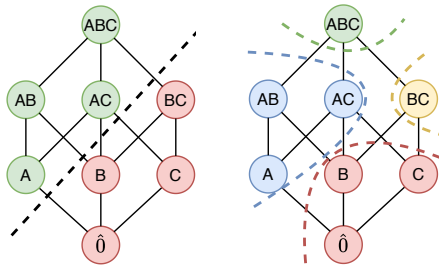
Ex II.1. As shown in Figure 2a, in a lattice model with 3 subjects, A, B, and C, $\uparrow A = \{A, AB, AC, BC, ABC\}$, $\uparrow BC = \{BC, ABC\}$, and for $\uparrow\hat{0}$ it is all 8 of the states.

A practical interpretation of an up-set in a lattice model under group testing is as follows: $\uparrow ABC$ is all the states for which a pool of A, B and C would contain only negative subjects, $\uparrow A$ is all the states such that A is negative. Conversely, the complement of the up-set of $\uparrow A^c = \{BC, B, C, \hat{0}\}$ represents the states for which a test sample from subject A is likely to be positive. Therefore, the up-set of a state and its complement generate a partition of the classification states based on whether the corresponding pooled test would contain all negatives or at least one positive. The posterior mass on these partitions will be the basis for test selection and for "halving" the lattice.

A lattice model can be efficiently constructed using bit-wise operations since the lattice model is a *powerset*, which is the set of all possible combinations of pooled subjects. The next task is finding an appropriate data structure to store state-wise posterior probability values. Such data structure should effectively accommodate critical operations, including insertion, deletion, and look-up in constant time for `<state, prob>` key-value pairs, which naturally drives us to choose a hash map. On the other hand, the high space complexity of a lattice model ($O(2^N)$) necessitates the design of a high-performance hash map rather than the Java default. Therefore, we choose the FNV1 hash function, which has been widely used in many places (e.g., Domain Name Servers) [17]. This hash function has an extremely low collision rate and high speed, which is a good fit for our requirements. Additionally, we use an up-set cache to speed up collecting the up-set of any given state. We design the caching process to interleave with lattice model generation using multithreading, which completely hides the caching overhead behind the generation process.

III. Test Selection Algorithms

This section first introduces core test selection algorithms used in Bayesian group testing: BHA and k-BHA. We then propose an approximation for k-BHA to drastically bring down its asymptotic complexity from $O(2^{2kN})$ to $O(k2^{2N})$. At the end of this section, we propose two types of parallelism for parallelizing test selections at different levels.



(a) BHA (b) 2-BHA

Fig. 2: Examples of Partitioning Lattice Model with BHA and k-BHA Using Up-set Information. In 2a, Model is Partitioned Through A using BHA; In 2b, Model is Partitioned Through A and BC using k-BHA, k = 2.

A. BHA: Bayesian Halving Algorithm

We propose BHA as an optimal strategy for test selection. The key objective of BHA is to systematically partition the lattice model based on the current posterior distribution on the lattice. After observation, by partitioning as close to half as possible, the posterior probability mass will increase in one of the two partitions and decrease in the other. This property systematically implies that the posterior probability mass will quickly accumulate to a single state. Importantly, purposeful and systematic test selections can be made based on observed test outcomes that eventually lead to the correct classification of the true state. In fact, BHA attains the optimal rates of convergence of the Bayesian posterior probability for the true state to 1 almost surely, regardless of the true state, and even under strong dilution effects. Critically, the convergence rates are exponential due to the discrete nature of the lattice model, which is reflected in real practice with short testing horizons needed for high accuracy.

We describe BHA using mathematical formula as follows: BHA selects a pooled experiment comprised of the subjects depicted as negatives for a state s that satisfies

$$\min_{s} \quad |m(s) - \frac{1}{2}| \quad \text{s.t.} \quad m(s) = \sum_{j \in \uparrow s} \pi(j) \tag{1}$$

We illustrate partitioning using BHA in Ex III.1.

Ex III.1. Consider a lattice generated by subjects A, B, C, depicted as a Hasse diagram in Figure 2a, to illustrate the partitioning of a lattice model using BHA in order to find a desired pooled test. As shown in Figure 2a, the lattice model is partitioned through state A. States highlighted in green is hence $\uparrow A$ and states highlighted in red is $\uparrow A^c$. If the sum of posterior probabilities of all states highlighted in green is the closest to 0.5 among partitioning through any other states, then A is the desired test selected by BHA.

As we can see, BHA iterates through every state and compute its posterior probability mass to find the desired

test selection. The asymptotic complexity can be expressed as $\sum_{i=0}^{N} C_N^i \cdot 2^i$, which sums to $O(2^{2N})$.

B. k-BHA: k-test Look-ahead Bayesian Halving Algorithm

k-BHA is the high-throughput extension of BHA, which allows k tests to be selected simultaneously. High-throughput test selection is more attractive as it can reduce the back and forth in preparing pooled samples for testing. Formally, k-BHA selects k states $s_1, \ldots, s_k$ as follows:

$$\begin{aligned} \min_{s_1 \ldots s_k} \quad & |\sum_{j \in s_1 \cap s_2 \ldots s_k} \pi_n(j) - 1/2^k| \quad + \\ & |\sum_{j \in s_1 \cap s_2 \ldots s_k^c} \pi_n(j) - 1/2^k| \quad + \\ & \ldots \quad + \\ & |\sum_{j \in s_1^c \cap s_2^c \ldots s_k^c} \pi_n(j) - 1/2^k| \end{aligned} \tag{2}$$

We illustrate how to perform partitioning using 2-BHA in Ex III.2.

Ex III.2. Assume the model is partitioned through states A and BC, where $\uparrow A = \{A, AB, AC, ABC\}$, $\uparrow A^c = \{B, C, BC, \hat{0}\}$, $\uparrow BC = \{BC, ABC\}$ and $\uparrow BC^c = \{A, B, C, AB, AC, \hat{0}\}$. As shown in Figure 2b, states are hence partitioned into four groups, each containing states of $\uparrow A \cap \uparrow BC = ABC$, $\uparrow A \cap \uparrow BC^c = A, AB, AC$, $\uparrow A^c \cap \uparrow BC = \{BC\}$ and $\uparrow A^c \cap \uparrow BC^c = \{B, C, \hat{0}\}$ respectively. If the four partitions generated by halving through A and BC contain more evenly distributed posterior probability sums than any other two-state combinations, then A and BC are the desired tests selected by 2-BHA.

As we can see in Ex III.2, to find the two desired states using 2-BHA, a total of 64 combinations of tests need to be evaluated. In general, k-BHA has an asymptotic complexity of $O(2^{2kN})$.

C. An Approximation Algorithm for k-BHA

k-BHA is prohibitively costly to compute. Therefore, we strive to explore opportunities to optimize this computationally demanding algorithm. Here, we propose an approximation algorithm for k-BHA, which reduces the asymptotic complexity from $O(2^{2kN})$ to $O(k2^{2N})$. In the approximation algorithm for k-BHA, instead of selecting k tests simultaneously, we successively select each test. Given current posterior probability π_n at stage n, we first choose state e_1 using BHA, then choose state e_2 that minimizes Equation 2 given π_n and e_1. We keep doing so until all k tests are selected.

Ex III.3. We illustrate the procedure of this approximation algorithm using the lattice model depicted in Figure 2a with $k = 2$. Assuming the first selected test is A, then the lattice model is partitioned into two groups as shown in Figure 2a. Next, based on the two existing partitions, the approximation algorithm finds the second desired test by halving through every state and examining each partition's posterior probability sum. Figure 2b illustrates the partitioning status where the approximation algorithm performs a second halving through BC. By using the up-set information, the first group $\{A, AB, AC, ABC\}$ is further divided into $\{A, AB, AC\}$ and $\{ABC\}$, and the second group $\{B, C, BC, \hat{0}\}$ is divided into $\{B, C, \hat{0}\}$ and $\{BC\}$.

Compared to k-BHA, which simultaneously selects k tests from a lattice model, the approximated k-BHA drastically brings down the asymptotic complexity to $O(k2^{2N})$ by dividing test selection into k BHA-like procedures and computing each sequentially. In the rest of this paper, we refer to this approximation algorithm as k-BHA.

D. Parallelizing Test Selections

Besides reducing algorithmic complexity, parallel computing provides another promising passage toward speeding up test selection algorithms. In HiBGT, parallel test selection algorithms must accommodate real-world test selection requirements and other critical components used for Bayesian group testing, i.e., lattice modes and statistical analysis. After careful evaluation, we conclude that the design space for parallelizing test selection algorithms can be characterized using two pairs of contradicting adjectives.

Overlapped and Independent: test selection algorithms rely heavily on computation over the same, complete lattice model, which results in profound overlapping in memory access patterns, especially for distributed memory models such as Spark. Nonetheless, these computations can also be performed independently of each other at the state level, which exposes an excellent opportunity for computation parallelization.

Heterogeneous and Homogeneous: the heterogeneity represents the varying needs in real-world test selections, where group testing pools are divergent regarding the number of individual subjects and risk levels, resulting in significantly varied computation workloads among test selections. Furthermore, many such pools will often query HiBGT simultaneously, necessitating considerable efforts to calibrate their varied workloads in parallel. On the other hand, the homogeneity can be referred to in the sense that designing parallel computation for these algorithms falls into the same spectrum, as they share similar algorithmic coherency.

Based on these observations, we immediately realize that no one-size-fits-all solution exists in designing parallelized test selection. We thus propose two abstractions of parallelism: *intra-lattice parallelism* and *inter-lattice parallelism*.

Intra-lattice parallelism solves the overlap-independence contradiction by parallelizing test selection algorithms inside the lattice model. It also serves the purpose of speeding up test selection algorithms. Intra-lattice parallelism allows many computing processes to work on a single test selection task over a lattice model by first performing efficient data parallelization (the complete lattice models) over a distributed memory model. After that, computations of posterior probability masses can be computed in parallel for each state in the lattice model. Additionally, it provides coordination among

processes to achieve good load balance. More details about intra-lattice parallelism will be discussed in Section VI-B1.

Inter-lattice parallelism addresses the heterogeneity-homogeneity contradiction by parallelizing the test selection process among multiple lattice models. Importantly, inter-lattice parallelism focuses on orchestrating a fine-tuned schedule to achieve load balance for nondeterministic workload introduced by heterogeneous test selection tasks. It also serves an indispensable mechanism to seamlessly bridge test selections with HiBGT's parallel statistical analysis design. More details will be given in Section V-B.

Additionally, given the homogeneity in the design space, we conclude that the two proposed parallelisms will offer the same effectiveness and efficiency for all discussed test selection algorithms. They also reciprocally adapt to HiBGT's other components without any design-wise modifications.

IV. Tree-based Statistical Analysis Schemes

HiBGT performs statistical analyses to evaluate the efficiency and effectiveness of performing Bayesian-based test selection algorithms over lattice models. This is achieved by constructing all possible testing sequences formulated into a tree structure. Statistical analysis is then conducted by exploring every tree branch, which comprises the probabilities of all testing sequences. The tree size will depend on two factors: the test selection algorithms used and the maximum test sequence length (stage) allowed. For example, in constructing a 24-stage tree using BHA as its test selection rule, BHA selects one test at the first stage, which can return two possible results: negative or positive. For the negative branch, after the root lattice model updates its posterior probability distribution based on the negative response, a second-stage BHA can be performed. This holds similarly for the positive branch at the first stage. Therefore, this tree can be spawned like a fully-grown binary tree up to 24 stages, generating a total of 2^{24} (over 16 million) branches (test sequences). For k-BHA, k selected tests in every stage correspond to 2^k potential testing results. Hence for m stages, the total branches will be 2^{km}. Since this tree can grow huge, we refer to it as *the big tree*. After the big tree is constructed, a statistical analysis evaluates the tree, which essentially examines and summarizes every branch's metadata given each state being assumed as true. As discussed in Section III, there are a total of 2^N possible true states, so this evaluation must be performed on the big tree for 2^N times. As we can see, the complete statistical analysis process adds extra complexity on top of lattice model generation and test selections computation, which motivates us to speed up this process.

To speed up statistical analysis using parallel computing, HiBGT offers a whole tree-parallelization scheme named the *multi-tree scheme* to split the tree-construction process into generating a sequence of smaller trees specified by true states, which allows a statistical analysis to parallelize the tree construction and the statistical evaluation processes at each tree level. On top of the multi-tree scheme, we propose two extra schemes for achieving more aggressive computation speed, each making trade-offs in generality and statistical accuracy.

A. The Multi-tree Scheme

The multi-tree scheme is motivated by the fact that for a given true state, most of the test sequences in the big tree are statistically non-significant and can hence be omitted. For example, assume the true state is ABC (subject A, B, and C are all negative), then the test sequence {<ABC, Positive>, <A, Positive>, <C, Positive>, <B, Positive>} is practically impossible. However, this sequence will potentially be enumerated by the big tree because it is considered highly possible for true state $\hat{0}$ (subject A, B, and C are all positive). Based on this observation, a tree can be structured specifically based on a given true state, where practically impossible branches (test sequences) can be pruned, which helps drastically reduce the complexity of this tree. Since each state is considered true at some point in the computations, multiple trees are required to evaluate all of the states, namely the multi-tree scheme.

A benefit of the multi-tree scheme is that it can generate true-state-specific trees independently, which exploits parallelization for both tree constructions and statistical evaluation, namely *tree-level parallelism*. Note that this parallelism is a derivation of inter-lattice parallelism with a stricter setting, where the heterogeneity lies at the tree level instead of at the lattice level. Nevertheless, the load balance technique remains similar (discussed in Section V-B).

Figure 3 depicts an architecture example for the multi-tree scheme with three nodes, where each node constructs one tree per given true state using depth-first construction (DFC). DFC exposed opportunities to enable branch pruning in a tree and reduce the computation workload in constructing trees by detecting low probability branches based on the given true state. After each tree is generated, the same process can immediately start statistical analysis for this tree.

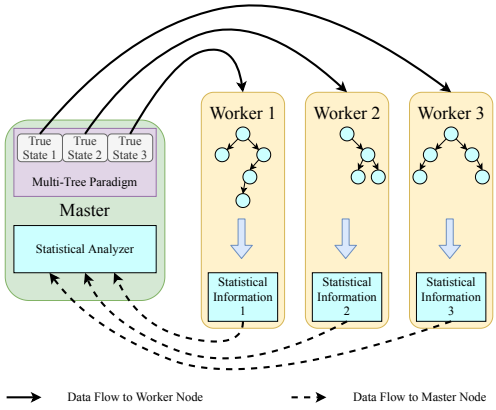


Fig. 3: Architecture Overview of Parallelizing Tree-based Statistical Analysis

B. Symmetry in Non-heterogeneity Pooling

While HiBGT is based on the prior probability information about individual risk levels, in practical use cases, this

information may not be easily available due to the lack of effective background tracing or privacy concerns. In this common scenario, homogeneity is assumed across all testing subjects, in which individual prior probability values p_0 are all set equal, and the constructed lattice model probability distribution becomes symmetric in the sense that states having an equal number of negatives will have the same prior probability value. Computational savings can be leveraged by avoiding redundant computations involving true states with the same number of negatives in the multi-tree scheme. We hence name this technique as *symmetric scheme*, which reduces the number of generated trees from 2^N to $N+1$, e.g., starting with a 3-subject pool, we only need to evaluate four true states: $\hat{0}$, A, AB, and ABC.

C. Trade-off between Accuracy and Complexity

In computing averages, metrics and statistical parameters such as the average number of tests and stages are computed at the state level, and the overall weighting is done by the respective prior probabilities of the states. When constructing the lattice model, some states will have substantially lower prior probabilities compared to others, which means contributions to the total statistical summary by these states can be relatively small. For example, assume a 3-subject pool where each subject has a probability of having COVID-19 is 0.02, then true state ABC will have a prior probability value of $0.98^3 \approx 0.941$ and the true state $\hat{0}$ will have prior probability value of $0.02^3 = 8 \times 10^-6$. So statistics evaluated from $\hat{0}$ will contributes much less than the one from ABC. We can hence skip assessing such less contributing states as true states when performing statistical analyses while maintaining acceptable statistical accuracy. For example, we can achieve 99% statistical accuracy by removing states with the lowest prior probability values that sum up to 1%, which, in this case, we name as *the 99% scheme*. One thing to note is that the effectiveness of this scheme will vary depending on the inputted individual risk. We will illustrate more in Section VI.

D. Shrink the Lattice Model by Removing Classified Subjects

Intuitively, if an individual subject is classified at some stage, it can be removed from the model. This occurs when the posterior mass on the up-set of the corresponding atom (profile only containing the individual) is greater or less than the respective thresholds for classifying as negative or positive. In a lattice model, it means removing all states containing the individual, which reduces lattice size by half. This strategy can lead to a significant reduction in computation for the remaining tree constructions. Note that to preserve statistical information for precise classification and for consistency in interpretation, posterior probabilities of the removed states are combined with those of corresponding states that represent the same individuals once the classified individual is removed. Consider the following example:

Ex IV.1. Denote the new posterior distribution after shrinking as π'_n, and consider the lattice model shown in Figure 2a. If B is classified at stage n, then $\tilde{B} = B, AB, BC, ABC$ will be removed and the posterior probability of B will grouped with its equivalent state $\hat{0}$, leading to $\pi'_n(\hat{0}) = \pi_n(\hat{0}) + \pi_n(B)$. Respectively, $\pi'_n(A) = \pi_n(A) + \pi_n(AB)$, $\pi'_n(C) = \pi_n(C) + \pi_n(BC)$ and $\pi'_n(AC) = \pi_n(AC) + \pi_n(ABC)$. As for the lattice model, we remove all states in the up-set of B, $\tilde{B}$, mapping the current lattice model into a powerset lattice of only A and C.

V. Implementation and Optimization

A. Spark Implementation

HiBGT provides two basic operations for updating lattice models during statistical analyses: `test` $\leftarrow$ `partition(rule)` and `update(test, outcome)`, where `(test, outcome)` is a `<String, Boolean>` tuple indicating the selected test with its observed testing outcome; and `rule` is a string representing which test selection rule is used for finding the partitions of the lattice model associated with tests. For BHA and k-BHA, one test and k tests will be outputted respectively by calling `partition(rule,` and `update(test, outcome)` must be called for the corresponding times to completely update the posterior probabilities and metadata of the lattice model. To aid this process, the lattice model generation process includes generating and maintaining state-wise metadata such as up-sets, classification status, history of conducted tests, etc. This metadata is used for accelerating test selections and statistical analyses.

The multi-tree scheme leverages the true-state-level parallelism to construct many small-scale trees synchronously. The master first schedules each worker node to generate the same initial lattice model using the `mapPartition()` call. At the same time, asynchronously prepare true states in separate threads, which allow computation overlap between the master and workers and avoid data parallelization using the `broadcast()` call. Meanwhile, the master establishes the next-stage tasks by parallelizing all true states into a *resilient distributed dataset (RDD)* [18] right after worker nodes finish generating lattice models. Workers can hence start constructing many trees in parallel. At each worker, the simulation tree construction is performed as a Spark transformation using the `map()` call, and the statistics data is collected and aggregated as a Spark action through the `reduce()` call. Finally, statistical data are sent back to the master for statistical analysis. Overall, the computation workflow of the multi-tree scheme comprises a single round of one-to-all + all-to-one communications.

B. Optimizations

1) Towards Load Balance: In Section IV-D, we introduce shrinking the lattice size during tree construction to diminish the computation workload on generating statistical analysis trees dramatically. However, from the load balance perspective, this design introduces skewed data and computation in both tree construction schemes, which can cause stragglers and crash the expected speed up. To mitigate stragglers, for the tree-level parallelism (inter-lattice parallelism with a stricter

setting) used in the multi-tree scheme, we adopt a fine-grained task scheduler for true-state parallelism such that workers will acquire tasks in a first-come-first-serve style. In Spark, this scheduling is achieved by increasing the number of RDD partitions containing the true state. On the other hand, finer-grained tasks usually bring more scheduling overhead as more inter-node communications are required. Since our task is computation-intensive, especially for larger N, and each task only requires a small piece of RDD partition (a string containing up to N characters) to be sent out, the scheduling overhead can hence be amortized. Note that intra-lattice parallelism used in speeding up test selection algorithms can also benefit from finer-grained scheduling but can suffer more from the scheduling overhead due to less computational workload compared to statistical analysis. We hence propose an exclusive optimization in the following subsection for the intra-lattice parallelism to mitigate this overhead.

2) Multithreading on Spark: Inspired by the MPI + X hybrid architecture commonly found in the HPC field [19]–[21], we propose to deploy multithreading on top of Spark to reduce communication overhead, memory consumption, and further improve load balance for intra-lattice parallelism for speeding up test selection algorithms. We use a customized fork-join pool that uses the famous work-stealing algorithm [22], which allows a thread to "steal" unfinished tasks from other threads, which is ideal for executing dynamic computations such as our test selection algorithms. Note that the tree-level parallelism used for statistical analysis does not explicitly benefit from multithreading on Spark. In the two derived schemes: the symmetry scheme and the 99% scheme, multithreading on Spark can help augment their degree of parallelism, which is crucial in improving their scaling performance. We will illustrate this in more detail in Section VI.

VI. Performance Evaluation

A. Experimental Testbed

Table I details the specification of the clusters used for evaluating HiBGT.

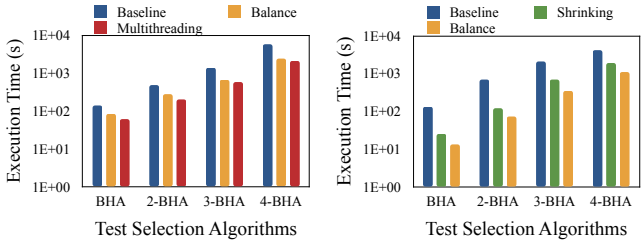
TABLE I: Specification of HPC Clusters

Cluster	Intel-Cluster	AMD-Cluster
Processor	Intel Xeon Gold 6132	AMD EPYC 7302P
No. of Cores	28	16
Clock Speed	2.6 GHz	3.0 GHz
RAM	192 GB	128 GB
Interconnect	IB-EDR (100 Gbps)	25 Gbps Ethernet
Storage	SAS-HDD (128GB)	SATA-SSD (500GB)
Spark Version	Spark-3.2.1	Spark-3.2.1
Scale	upto 33 nodes/924 cores	upto 33 nodes/528 cores

To systematically evaluate the performance of test selection algorithms and statistical analysis schemes in HiBGT, we propose comprehensive group testing scenarios to mimic the real-world group testing scenarios. We first define three individual risk patterns to reflect typical COVID-19 prevalence rates: 1) *low risk*, where all individual risks are 2%; 2) *mid risk*, where four individuals have risk levels equal to 20%, and the rest are 2%; and 3) *high risk*, where all individual risks are 20%. We choose a mid-range of $N = 15$, which is suitable for differentiating performance metrics while not putting too much burden on the cluster. For test selections, we choose to evaluate four algorithms: BHA, 2-BHA, 3-BHA, and 4-BHA. We also evaluate three tree schemes for statistical analysis: the multi-tree scheme, the 99% scheme, and the symmetry scheme. We threshold trees at 24, 16, 12, and 10 stages, respectively, for evaluated test selection algorithms.

B. Performance Evaluation

1) Evaluating Optimizations: In this subsection, we evaluate our proposed optimization techniques discussed in Section V-B for test selection algorithms and statistical analyzes using parallel computing. We illustrate the quantification of performance gains by sequentially adopting these optimizations in Figure 4. These optimization evaluations are conducted using the Intel Cluster with 112 worker cores (140 total cores). As depicted in Figure 4a, for test selection algorithms at $N = 20$, tuning load balance improves the performance by up to 1.67-2.37x, while leveraging multithreading on Spark further enlarges the performance gain to 2.33x-2.74x compared to the baseline. As depicted in Figure 4b, by utilizing lattice size shrinking, we speed up the statistical analysis by 2.3x-5.7x compared to the baseline (no optimization) for the multi-tree scheme. By further tuning for load balancing, we achieve an overall performance gain ranging from 3.8x to 10x.



(a) Test Selection Algorithms (b) Statistical Analyzes

Fig. 4: Performance Evaluation of Optimization Techniques for Test Selection Algorithms using the Intra-lattice Parallelism at $N = 20$ and Statistical Analyses using the Multi-tree Scheme at $N = 15$ with High-risk Pattern.

2) Evaluating Test Selection Throughput: Compared to statistical analyses, which are usually conducted offline to give insights into how efficient and effective the Bayesian group testing can address real-world group testing scenarios, a test selection offers more practical usage in guiding a pooling decision and should be considered as an online operation. Therefore, this necessitates the evaluation of test selection throughput. As discussed in Section III-D, we propose a novel intra-lattice parallelism for speeding up test selection algorithms. We thus assess its performance in the throughput test to see how big we can push the scale. For this evaluation, we use the AMD cluster with 512 worker cores. Based on its asymptotic complexity, we focus on testing the throughput

of BHA, which offers a much better test selection throughput than other test selection algorithms. The evaluation result is illustrated in Figure 5, where we use intra-lattice parallelism to perform parallel BHA starting at $N = 15$. As we can see, the throughput reaches over 9,100 test selections per hour (2.54 per second) at $N = 15$. When $N \leq 18$ (still achieves around one selection per second, which we consider as a real-time response), we see the throughput drop slower than BHA's asymptotic complexity ($O(2^{2N})$), meaning the primary computation bottleneck at this scale is Spark's scheduling overhead. After that, we see an accelerated throughput drop that is faster than BHA's asymptotic complexity, meaning the primary bottleneck at this scale is shifting to intra-lattice parallelism's data parallelization overhead. At $N = 20$, the throughput is 452 test selections per hour, or 8s per selection. We consider that this throughput can meet the real-world rapid testing requirement. When N reaches 25, which is approaching the upper limit of the suggested COVID-19 sample size, one test selection takes 14218s (3 hours 57 minutes), which is selecting the desired test out of over 33 million states from the lattice model by computing and iterating through these states for over 1.1 quadrillions (10^{15}) times.

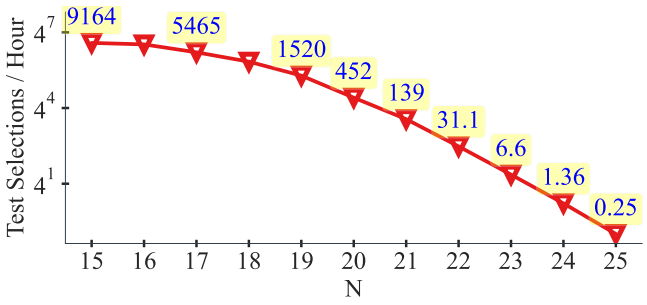


Fig. 5: Test Selection Throughput using BHA at Different Scale

3) Evaluating Statistical Analysis Schemes: In this subsection, we evaluate our proposed statistical analysis schemes discussed in Section IV. An overall picture of statistical analysis performance is given in Figure 6. These scheme evaluations are conducted on the Intel Cluster with 448 worker cores (476 total cores). We first set up the baseline, which is by constructing *the big tree*. To allow fair comparison, we enable *intra-lattice parallelism* for test selections during the big tree construction. We first compare the execution time between the multi-tree scheme and the baseline, which is shown that the multi-tree scheme outperforms the baseline by 2.1 - 15.9x, 2.1 - 8x, 2.9 - 3.3x, and 2.2 - 4.2x for BHA, 2-BHA, 3-BHA, and 4-BHA, respectively. As we can see, the multi-tree scheme is significantly faster than generating the big tree across all testing scenarios, which proves its effectiveness and sophistication in accelerating statistical analysis. When comparing the baseline with the 99% scheme, we see that the speed ups enlarge to 92 - 363x, 21 - 224x, 28 - 97x, and 26 - 96x for respective test selection algorithms. When comparing the baseline with the symmetry scheme, we see that the speed-ups reach 104 - 342x, 50 - 128x, 67 - 77x, and 52 - 108x for respective test selection algorithms. Note that the symmetry scheme does not apply to the mid-risk pattern since the individual risk levels are heterogeneous.

When we characterize the execution time for different test selection algorithms with different COVID-19 prevalence levels, we observe that the execution time will significantly increase due to the test selection complexity. Comparing BHA to 4-BHA shows up to a 20.3x increase in execution time. On the other hand, we also observe the execution time increase along with COVID-19 prevalence levels. When comparing high-risk and low-risk patterns, we see execution time increase up to 6x due to the construction of more complex tree structures. We also notice that the two auxiliary schemes adapt differently to varied prevalence levels. In the low-risk pattern, the 99% scheme is generally faster than the symmetry scheme, while in the high-risk pattern, the symmetry scheme outperforms the 99% scheme. This is because prevalence levels can alternate the effectiveness of reducing true states for the 99% scheme, where a higher prevalence level will generally result in less reduction.

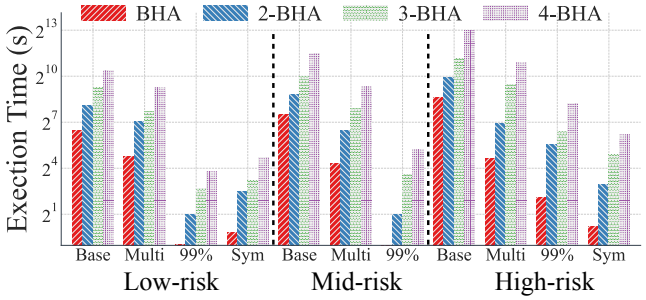


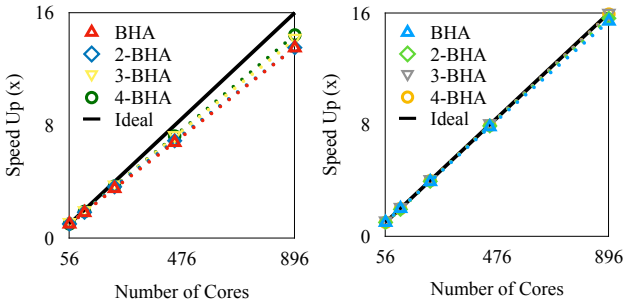
Fig. 6: Performance Evaluation of Three Statistical Analysis Schemes at $N = 15$. Base(line) is Constructing *the Big Tree* using *Intra-lattice Parallelism* for Test Selections.

C. Scalability Evaluation

1) Evaluating Strong Scaling Efficiency: As illustrated in Figure 7, we conduct the strong scaling evaluation for three statistical analysis schemes and the intra-lattice parallelism for test selections using our Intel cluster using 56, 112, 224, 448, and 896 worker cores. As shown in Figure 7b. The multi-tree scheme achieves near-perfect scaling efficiency, ranging from 96.3% to 99.5%. This is primarily benefited by the efficient parallelism nature of the multi-tree scheme discussed in Section IV-A, where each tree can be constructed and evaluated independently without data shuffle. Also, the load balance techniques discussed in Section V-B1 also significantly contribute to achieving these strong numbers. On the other hand, in Figure 7c, it is shown that the 99% scheme achieves strong scaling efficiencies of 75.7%, 83.1%, 93.1%, and 93.9% for BHA, 2-BHA, 3-BHA, and 4-BHA, respectively. The reduced efficiency compared to the multi-tree scheme is likely due to 1) reduced number of tasks due to a reduced number of true states, hence resulting in more

skew and worker dangling, and 2) multithreading on Spark causes lead to side effects such as thread contention in the multi-tree-like schemes. We also expect similar or even worse scaling efficiency in the symmetry scheme, as it performs even fewer tasks. In Figure 7d, we observe this trend till using 448 cores, which maintain strong scaling efficiency between 72.2% to 77.4%. However, at 896 cores, we notice an efficiency stall. By investigating this abnormality, we conclude that this is due to an insufficient degree of parallelism in using 896 cores for the symmetry scheme as 448 cores (16 worker nodes) have saturated 16 tree-construction tasks at $N = 15$ (with multithreading on Spark applied).

Figure 7a shows the strong scaling efficiency of test selection algorithm using the intra-lattice parallelism at $N = 22$. With the help of multiple load balance optimizations, we achieve 84.4%, 84.5%, 88%, and 90.1% scaling efficiency for BHA, 2-BHA, 3-BHA, and 4-BHA, respectively. The increased efficiency along more complex test selection algorithms reflects the increasingly amortized data communication and task scheduling overheads in larger workloads.

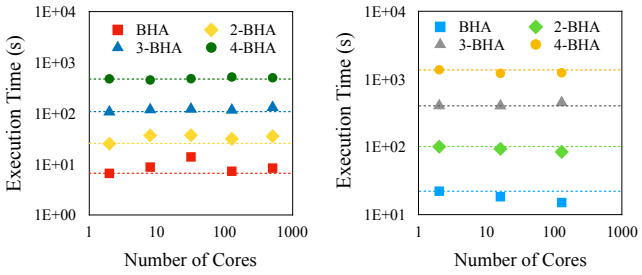


(a) Intra-lattice Parallelism (b) Multi-tree Scheme

(c) 99% Scheme (d) Symmetry Scheme

Fig. 7: Strong Scaling Efficiency of Test Selection Algorithms using the Intra-lattice Parallelism at $N = 20$ and Three Statistical Analysis Schemes at $N = 15$ Note that Values for Statistical Analysis Schemes are averaged by 3 Risk Patterns.

2) Evaluating Weak Scaling Efficiency: Figure 8 illustrates our weak scaling studies using the AMD cluster. To evaluate test selection algorithms using the intra-lattice parallelism, we evaluate N ranges from 16 to 20, using 2, 8, 32, 128, and 512 cores, respectively. The weak scaling efficiency is shown in Figure 8a, where the scaling curve is overall leveled, meaning the weak scaling performance generally follows the algorithms' asymptotic complexities. Next, we evaluate statistical analyzes using the multi-tree scheme. As discussed in Section IV-A, the structure and complexity of each small tree are closely related to the testing pool and given true state, which is opaque and hard to predicatively quantify. On the other hand, the asymptotic complexity of test selection algorithms and the number of true states is known, which can be represented as a cubic function. Therefore, we choose to evaluate N = 13, 14, and 15 with 2, 16, and 128 cores, respectively, using pools with low risk pattern. As illustrated in Figures 8b, we see that statistical analyzes using all four test selection algorithms can hold the weak scaling curve.



(a) Intra-lattice Parallelism (b) Multi-tree Scheme

Fig. 8: Weak Scaling of Test Selection Algorithms using the Intra-lattice Parallelism and Statistical Analyzes using the Multi-tree Scheme with Low Risk Pattern.

VII. Other Related Work

Recently, it has been of interest to develop one-stage group testing methods. High-throughput and one-stage pooled test designs have recently been proposed in [7], [8], and for instance, rely on error-correcting techniques. These state-of-the-art group testing approaches suggested for COVID-19 are only effective for the low-risk scenarios with low prevalence rate (like $< 1 - 4\%$), however, and do not necessarily disentangle the confounding from positive pooled test results. For the sake of consistent accuracy in light of dynamic prevalence levels, multiple-stage testing is needed.

Scalable epidemiological tasks that are enabled with HPC, including forecasting, planning, response, etc., have garnered interest since Covid-19 [23]. A rich body of work focuses on accelerating Bayesian-based algorithms through parallel computing has been extensively studied in multiple forms and has been widely adopted, such as with parallel Markov chain Monte Carlo (MCMC) methods [24], parallel Naive Bayes text classification [25] and parallel Bayesian Networks for genome-scale gene networks [26]. Implementations based on different parallelism frameworks include OpenMP [27], MPI [28], Hadoop [25], Spark [29] and GPU computing [30]. We do not conduct iterative Bayesian simulation methods for estimation of posterior distributions, as in MCMC, but rather simulate from specified probability distributions based on the realistic dilution effect models that are assumed. In practice, these distributions would be estimated from data. In group testing, a halving algorithm that uses prior probabilities

to pool samples to split aggregate risk is described in [31]. The approach studied here differs in that posterior probability updating is conducted here, which serves as the basis for pool selection and stopping.

VIII. Conclusion

The Bayesian approach to group testing is appealing as it can directly reflect variable levels of individual risk in the classification process and is accurate and efficient even when accounting for dilution effects. However, the required precision level comes with the cost of being computationally challenging. We hence propose a high-performance Bayesian group testing framework named HiBGT, which allows for accurate, scalable, flexible, and efficient guidance for group testing and the statistical analysis of performance. Through systematic evaluations of various designs and optimizations on Spark-based implementations, we show that our proposed HiBGT can perform large-scale test selections and significantly accelerate statistical analyses while achieving near-linear scalability in multiple designs up to 924 CPU cores.

References

[1] S. Lohse, T. Pfuhl, B. Berkó-Göttel, J. Rissland, T. Geißler, B. Gärtner, S. L. Becker, S. Schneitler, and S. Smola, "Pooling of Samples for Testing for SARS-CoV-2 in Asymptomatic People," *The Lancet Infectious Diseases*, vol. 20, no. 11, p. 1231–1232, 2020.

[2] F. Majid, S. B. Omer, and A. I. Khwaja, "Optimising SARS-CoV-2 Pooled Testing for Low-Resource Settings," *The Lancet Microbe*, vol. 1, no. 3, 2020.

[3] C. A. Hogan, M. K. Sahoo, and B. A. Pinsky, "Sample Pooling as a Strategy to Detect Community Transmission of SARS-CoV-2," *Jama*, vol. 323, no. 19, p. 1967, 2020.

[4] D. Donoho, M. Lofti, and B. Ozturkler. (2020) The Mathematics of Mass Testing for COVID-19. [Online]. Available: https://sinews.siam.org/Details-Page/the-mathematics-of-mass-testing-for-covid-19

[5] R. Dorfman, "The Detection of Defective Members of Large Populations," *The Annals of Mathematical Statistics*, vol. 14, no. 4, pp. 436–440, 1943. [Online]. Available: http://www.jstor.org/stable/2235930

[6] A. C. Bateman, S. Mueller, K. Guenther, and P. Shult, "Assessing the Dilution Effect of Specimen Pooling on the Sensitivity of SARS-CoV-2 PCR Tests," *Journal of medical virology*, vol. 93, no. 3, p. 1568—1572, March 2021.

[7] N. Shental, S. Levy, V. Wuvshet, S. Skorniakov, B. Shalem, A. Ottolenghi, Y. Greenshpan, R. Steinberg, A. Edri, R. Gillis, M. Goldhirsh, K. Moscovici, S. Sachren, L. M. Friedman, L. Nesher, Y. Shemer-Avni, A. Porgador, and T. Hertz, "Efficient High-throughput SARS-CoV-2 Testing to Detect Asymptomatic Carriers," *Science Advances*, vol. 6, no. 37, 2020.

[8] S. Ghosh, A. Rajwade, S. Krishna, N. Gopalkrishnan, T. E. Schaus, A. Chakravarthy, S. Varahan, V. Appu, R. Ramakrishnan, S. Ch, M. Jindal, V. Bhupathi, A. Gupta, A. Jain, R. Agarwal, S. Pathak, M. A. Rehan, S. Consul, Y. Gupta, N. Gupta, P. Agarwal, R. Goyal, V. Sagar, U. Ramakrishnan, S. Krishna, P. Yin, D. Palakodeti, and M. Gopalkrishnan, "Tapestry: A Single-Round Smart Pooling Technique for COVID-19 Testing," *medRxiv*, 2020.

[9] C. Tatsuoka, W. Chen, and X. Lu, "Bayesian Group Testing with Dilution Effects," *Biostatistics*, p. kxac004, Apr. 2022. [Online]. Available: https://doi.org/10.1093/biostatistics/kxac004

[10] S. Lohse, T. Pfuhl, B. Berkó-Göttel, J. Rissland, T. Geißler, B. Gärtner, S. L. Becker, S. Schneitler, and S. Smola, "Pooling of Samples for Testing for SARS-CoV-2 in Asymptomatic People," *The Lancet Infectious Diseases*, vol. 20, no. 11, pp. 1231–1232, 2020.

[11] J. Yu, Y. Huang, and Z.-J. Shen, "Optimizing and Evaluating PCR-based Pooled Screening during COVID-19 Pandemics," *Scientific reports*, vol. 11, no. 1, pp. 1–14, 2021.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. Boston, MA: USENIX Association, Jun. 2010.

[13] C. Tatsuoka and T. Ferguson, "Sequential classification on partially ordered sets," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 65, no. 1, p. 143–157, 2003.

[14] T. S. Ferguson and C. Tatsuoka, "An Optimal Strategy for Sequential Classification on Partially Ordered Sets," *Statistics Probability Letters*, vol. 68, no. 2, p. 161–168, 2004.

[15] C. Tatsuoka, "Optimal Sequencing of Experiments in Bayesian Group Testing," *Journal of Statistical Planning and Inference*, vol. 133, no. 2, pp. 479 – 488, 2005.

[16] ——, "Sequential Classification on Lattices with Experiment-specific Response Distributions," *Sequential Analysis*, vol. 33, no. 3, pp. 400–420, 2014.

[17] G. Fowler, L. C. Noll, K.-P. Vo, D. Eastlake, and T. Hansen, "The FNV Non-cryptographic Hash Algorithm," *IETF-draft: Fremont, CA, USA*, 2011.

[18] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX Association, Apr. 2012, pp. 15–28.

[19] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009, pp. 427–436.

[20] E. Lusk and A. Chan, "Early Experiments with the OpenMP/MPI Hybrid Programming Model," in *OpenMP in a New Era of Parallelism*, R. Eigenmann and B. R. de Supinski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 36–47.

[21] M. J. Chorley and D. W. Walker, "Performance Analysis of a hybrid MPI/OpenMP Application on Multi-core Clusters," *Journal of Computational Science*, vol. 1, no. 3, pp. 168–174, 2010.

[22] R. D. Blumofe and C. E. Leiserson, "Scheduling Multithreaded Computations by Work Stealing," *J. ACM*, vol. 46, no. 5, p. 720–748, sep 1999.

[23] D. Machi, P. Bhattacharya, S. Hoops, J. Chen, H. Mortveit, S. Venkatramanan, B. Lewis, M. Wilson, A. Fadikar, T. Maiden, C. L. Barrett, and M. V. Marathe, "Scalable Epidemiological Workflows to Support COVID-19 Planning and Response," *medRxiv*, 2021.

[24] X. Feng, D. A. Buell, J. R. Rose, and P. J. Waddell, "Parallel Algorithms for Bayesian Phylogenetic Inference," *Journal of Parallel and Distributed Computing*, vol. 63, no. 7, pp. 707–718, 2003, special Issue on High-Performance Computational Biology.

[25] H. Amazal, M. Ramdani, and M. Kissi, "A Text Classification Approach using Parallel Naive Bayes in Big Data Context," 10 2018, pp. 1–6.

[26] S. Misra, M. Vasimuddin, K. Pamnany, S. P. Chockalingam, Y. Dong, M. Xie, M. R. Aluru, and S. Aluru, "Parallel Bayesian Network Structure Learning for Genome-Scale Gene Networks," in *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 461–472.

[27] V. K. Namasivayam and V. K. Prasanna, "Scalable Parallel Implementation of Exact Inference in Bayesian Networks," in *12th International Conference on Parallel and Distributed Systems - (ICPADS'06)*, vol. 1, 2006, pp. 8 pp.–.

[28] A. L. Madsen, F. Jensen, A. Salmerón, H. Langseth, and T. D. Nielsen, "A Parallel Algorithm for Bayesian Network Structure Learning from Large Data Sets," *Knowledge-Based Systems*, vol. 117, pp. 46–55, 2017, volume, Variety and Velocity in Data Science.

[29] Z. Tang, W. Xiao, B. Lu, Y. Zuo, Y. Zhou, and K. Li, "A Parallel Algorithm for Bayesian Text Classification Based on Noise Elimination and Dimension Reduction in Spark Computing Environment," in *Cloud Computing – CLOUD 2019*, D. Da Silva, Q. Wang, and L.-J. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 222–239.

[30] A. Munawar, M. Wahib, M. Munetomo, and K. Akama, "Theoretical and Empirical Analysis of a GPU Based Parallel Bayesian Optimization Algorithm," in *2009 International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2009, pp. 457–462.

[31] M. S. Black, C. R. Bilder, and J. M. Tebbs, "Group Testing in Heterogeneous Populations by Using Halving Algorithms," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 61, no. 2, pp. 277–290, 2012.