Exploring the Landscape of Big Data Analytics Through Domain-Aware Algorithm Design

Sajal Dash

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Application

Wu-chun Feng, Chair
Christopher L North
Sharath Raghvendra
Scott C Leman
Ramamoorthy Anandakrishnan

June 25, 2020
Blacksburg, Virginia

Keywords: Big data analytics, High-performance computing, Algorithmic machine learning, Incremental algorithm, Approximate algorithm etc.

Exploring the Landscape of Big Data Analytics Through Domain-Aware Algorithm Design

Sajal Dash

(ABSTRACT)

Experimental and observational data emerging from various scientific domains necessitate fast, accurate, and low-cost analysis of the data. While exploring the landscape of big data analytics, multiple challenges arise from three characteristics of big data: the volume, the variety, and the velocity. High volume and velocity of the data warrant a large amount of storage, memory, and compute power while a large variety of data demands cognition across domains. Addressing domain-intrinsic properties of data can help us analyze the data efficiently through the frugal use of high-performance computing (HPC) resources. In this thesis, we present our exploration of the data analytics landscape with domain-aware approximate and incremental algorithm design. We propose three guidelines targeting three properties of big data for domain-aware big data analytics: (1) explore geometric and domain-specific properties of high dimensional data for succinct representation, which addresses the volume property, (2) design domain-aware algorithms through mapping of domain problems to computational problems, which addresses the variety property, and (3) leverage incremental arrival of data through incremental analysis and invention of problem-specific merging methodologies, which addresses the velocity property. We demonstrate these three guidelines through the solution approaches of three representative domain problems.

We present Claret, a fast and portable parallel weighted multi-dimensional scaling (WMDS) tool, to demonstrate the application of the first guideline. It combines algorithmic concepts extended from the stochastic force-based multi-dimensional scaling (SF-MDS) and Glimmer. Claret computes approximate weighted Euclidean distances by combining a novel data mapping called *stretching* and Johnson Lindestrauss' lemma

to reduce the complexity of WMDS from $O(f(n)d)$ to $O(f(n)\log d)$. In demonstrating the second guideline, we map the problem of identifying multi-hit combinations of genetic mutations responsible for cancers to weighted set cover (WSC) problem by leveraging the semantics of cancer genomic data obtained from cancer biology. Solving the mapped WSC with an approximate algorithm, we identified a set of multi-hit combinations that differentiate between tumor and normal tissue samples. To identify three- and four-hits, which require orders of magnitude larger computational power, we have scaled out the WSC algorithm on a hundred nodes of Summit supercomputer. In demonstrating the third guideline, we developed a tool iBLAST to perform an incremental sequence similarity search. Developing new statistics to combine search results over time makes incremental analysis feasible. iBLAST performs $(1+\delta)/\delta$ times faster than NCBI BLAST, where δ represents the fraction of database growth. We also explored various approaches to mitigate catastrophic forgetting in incremental training of deep learning models.

Exploring the Landscape of Big Data Analytics Through Domain-Aware Algorithm Design

Sajal Dash

(GENERAL AUDIENCE ABSTRACT)

Experimental and observational data emerging from various scientific domains necessitate fast, accurate, and low-cost analysis of the data. While exploring the landscape of big data analytics, multiple challenges arise from three characteristics of big data: the volume, the variety, and the velocity. Here volume represents the data's size, variety represents various sources and formats of the data, and velocity represents the data arrival rate. High volume and velocity of the data warrant a large amount of storage, memory, and computational power. In contrast, a large variety of data demands cognition across domains. Addressing domain-intrinsic properties of data can help us analyze the data efficiently through the frugal use of high-performance computing (HPC) resources. This thesis presents our exploration of the data analytics landscape with domain-aware approximate and incremental algorithm design. We propose three guidelines targeting three properties of big data for domain-aware big data analytics: (1) explore geometric (pair-wise distance and distribution-related) and domain-specific properties of high dimensional data for succinct representation, which addresses the volume property, (2) design domain-aware algorithms through mapping of domain problems to computational problems, which addresses the variety property, and (3) leverage incremental data arrival through incremental analysis and invention of problem-specific merging methodologies, which addresses the velocity property.

We demonstrate these three guidelines through the solution approaches of three representative domain problems. We demonstrate the application of the first guideline through the design and development of Claret. Claret is a fast and portable parallel weighted multi-dimensional scaling (WMDS) tool that can reduce the dimension of

high-dimensional data points. In demonstrating the second guideline, we identify combinations of cancer-causing gene mutations by mapping the problem to a well known computational problem known as the weighted set cover (WSC) problem. We have scaled out the WSC algorithm on a hundred nodes of Summit supercomputer to solve the problem in less than two hours instead of an estimated hundred years. In demonstrating the third guideline, we developed a tool iBLAST to perform an incremental sequence similarity search. This analysis was made possible by developing new statistics to combine search results over time. We also explored various approaches to mitigate the catastrophic forgetting of deep learning models, where a model forgets to perform machine learning tasks efficiently on older data in a streaming setting.

Dedication

To my parents for providing me the platform to pursue science

Acknowledgments

Pursuing a PhD in Computer Science was the fulfillment of my childhood dream of becoming a scientist. It was made possible by the support, guidance, and collaboration of many individuals. I like to thank them all.

First and foremost, I would like to thank my advisor, Dr. Wu-chun Feng, for his continuous research guidance. I am grateful to him for allowing me to explore challenging research avenues over the years. I also like to thank my committee members Drs. Chris North, Sharath Raghvendra, Scotland Leman, and Ramu Anandakrishnan for their guidance, suggestions, and support.

My gratitude extends to my research collaborators Sarthok Rasique Rahman, Qais Al Hajri, and Anshuman Verma. My thanks to Dr. Heather Hines, Dr. Junqi Yin, and Dr. Mallikarjun Shankar. My research was greatly benefited by their contribution, data, and mentor-ship.

I like to thank my past and current lab-mates and friends Anshuman Verma, Sharmi Banerjee, Vignesh Adhinarayanan, Sarunya Pumma, Frank Wanye, and Atharva Gondhalekar for their companionship, insightful conversations, and mental support.

While the PhD journey was intellectually rewarding, it was also mentally perilous. The completion of the journey would not have been possible without the mental support of my friends Bushra Tawfiq Chowdhury, Archi Dasgupta, Rubayet Elahi, Syeed Md Iskander, Nabil Nowak, Asifur Rahman, Sazzadur Rahaman, Tonmoy Roy, Farin Siddiquee, Munawwar M. Sohul, Ipsita Hamid Trisha, and Fahmida Shaheen Tulip. Thank you for bearing with me, listening to my complaints, and making me feel that I matter.

I like to thank my siblings Prosenjit Dash, Pankaj Dash, and Sampa Dash, for being

there for me from across the globe though I practically disappeared for the last few years. Thank you for always having faith in me.

I cannot thank my parents enough for all the sacrifices they have made for me. I would not be here without your unconditional support, love, and care. I hope I made you happy and proud. This dissertation is for you.

Funding Acknowledgment: My research was supported by NSF grant IIS-1447416, a grant from General Dynamics Mission Systems, a seed grant from the SEEC Center (supported by ICTAS at Virginia Tech), a grant from Oak Ridge National Laboratory, a VCOM Bradley Foundation grant, and VCOM REAP grant RA2019.

Declaration of Collaboration: In addition to my advisor Wu-chun Feng, the research presented in this dissertation benefited from several collaborators:

- Anshuman Verma (Microsoft), Michelle Dowling (VT), Scotland Leman (VT), and Chris North (VT) contributed to the work included in Chapter 2.
- Ramu Anandakrishnan (VCOM), Nicholas A. Kinney (VCOM), Robin T. Varghese (VCOM), Harold R. Garner (VCOM), and Qais Al Hajri (Microsoft) contributed to the work included in Chapter 3.
- Sarthok Rasique Rahman (Penn State) and Heather M. Hines (Penn State) contributed to the work included in Chapter 4.
- Junqi Yin and Mallikarjun Shankar contributed to the work included in Chapter 5.

Contents

List of Figures

List of Tables

List of Abbreviations

DL Deep Learning

GPU Graphics Processing Unit

HPC High-Performance Computing

MDS Multi-Dimensional Scaling

ML Machine Learning

MPI Message Passing Interface

WMDS Weighted Multi-Dimensional Scaling

Chapter 1

Introduction

With advancements in scientific equipment development, large-scale experiment design, and manufacturing of high capability sensors, various scientific domains amass a large volume of data in a short amount of time. For example, recent multi-national scientific collaborations resulted in several large-scale projects such as the Human Genome Project (HGP) [39, 41, 177], the Laser Interferometer Gravitational-Wave Observatory (LIGO) [5], and the Event Horizon Telescope (EHT) [113]. These projects produced some of the most advanced examples of technological marvels and achievements of recent history, and they gather and generate a large volume of data intending to make new scientific discoveries. Discovering knowledge from this data requires a tremendous amount of analytics effort. These data hold keys to many scientific discoveries, understanding of a market and economic dynamics, shading light to collective social behavior, and various other vital aspects of modern life. However, some dynamic systems evolve rapidly, making collected data obsolete pretty fast, while the knowledge from other data paves the path for the next level of advancement in knowledge discovery. For data analytics to be useful, our analysis methodology and computing resources must provide fast analysis at an affordable cost.

Research efforts in Machine learning and statistics-based analysis improved analytics methodology significantly. Advancements in deep learning models such as ResNet [78] and machine learning models such as XGBoost [35] enabled us with unprecedented classification and regression accuracy. Researchers in the high performance computing community developed pretty powerful computing platforms and software tools to

leverage these platforms. Oak Ridge National Lab recently developed the fastest supercomputer Summit [3] in the world with exascale computing capability and Junqi et al. [190] deployed state of the art machine learning and deep learning tools across thousands of compute nodes of Summit. A combination of these methodologies and computing platforms benefit data analytics a great deal. However, access to state of the art computing platform is not easy for the majority of the research institutes and individual researchers because they cost a great deal of money. Another concerning fact is that we produce data at a an exponential rate [2, 72] which has the potential to overwhelm our computing resource.

1.1 Characteristics of Big Data

Doug Laney [102] characterized big data using 3 V's, volume, variety, and velocity. While the community has proposed several other characteristics in recent times, these 3 Vs are widely used to define big data. For this thesis's scope, we characterize big data using volume, variety, and velocity and provide our interpretation of each of these characteristics.

Volume Volume is the required storage of the data; big data requires large storage. We define volume using two quantitative metrics, (1) the number of data points, and (2) the number of features of the data. Any data with a large value for any of these two metrics can be qualified as big data. Due to our increased observational capacity and digitization of modern life, we amass a vast number of data points from many application sources. With the improvement in various sensor technologies, widespread digital presence, the innovative marketplace, and increased complexity of social interactions, the amassed data also exhibits increased complexity. The complexity of the data can be quantified using the number of features. While many data points can be beneficial

if we can keep up with the compute-, memory-, and communication- requirement, a large number of features can be problematic. Data with a large number of features can be presented as points in high-dimensional space, and the curse of dimensionality can prevent us produce meaningful knowledge from big data.

Variety In most literature, variety is defined as the various formats and structures of the data. We expand this definition by referring to an observation, different formats, and structures of data come from different sources. Data can come from various scientific domains, social media, health monitoring, and service industries. Depending on how well defined the data, and well established the data collection method is, data can exhibit varying degrees of structures. For example, a weather forecasting entity collects many well-defined metrics, such as airspeed, temperature, and humidity. So, data compiled by that entity will be well structured. On the other hand, an epidemiologist trying to predict a developing pandemic's course has to rely on very unstructured data such as tweets, news articles, and local medical reports. The data source can present unique opportunities to organize the data into a structure by a clear interpretation of the semantic of the data and the underlying process.

Velocity Velocity is the rate at which the data arrives. The fast velocity of the data can characterize big data. Large scale scientific simulations can produce data at a rapid pace, which will require online learning. Traffic surveillance data, weather data during a developing storm, disease spread data during an ongoing pandemic represent varying velocity, which requires incremental and real-time learning.

1.2 A Case for Domain-Aware Algorithm Design

A closer understanding of data and the problem can reduce the burden on the computing platform by providing insights into algorithmic innovation. High dimensional space of the data and its volume reflects the complexity of the data. Data exhibits some counter-intuitive properties in high-dimensional space, and often projecting the data onto lower-dimensional space facilitates meaningful statistical analysis at significantly reduced computation cost. We can reduce the volume of data through the computation of coresets and geometric transformation at the expense of bounded approximation [7, 60]. Coreset is a summary set of representative points that approximately maintains the problem-specific properties of the original data points [7]. Feldman et al. [60] computed coresets for projective clustering through $O\left(\frac{k}{\epsilon}\right)$ rank approximation. This succinct representation helps with the efficient computation of PCA and *k-means* clustering.

While exploring geometric properties of data can help us with efficient analytics, understanding the domain background of the data can provide us with unique insights into how to formulate the problem in the more tractable form. For example, we must understand the relationship of this data with cancer biology and attribute appropriate semantics to the data to analyze genomic data obtained from cancer patients. This process will help us map the problem of cancer biology domain to the framework of rigorously studied computational problem families. Many bioinformatics problems such as constructing genome sequences from overlapping fragments, non-overlapping local alignments, and matching three-dimensional molecular structures have been mapped to well known computational problems *Clique-detection* by embedding the domain knowledge of each of these problems into mapping strategies and parameters [28].

These first two approaches can help us with efficient data representation and optimal problem formulation, respectively. Another source of optimization can be insight through the understanding of the domain-specific analytics process and practices. For

example, in their study of various types of omics, biologists perform sequence similarity searches against ever-growing publicly available sequence databases at regular intervals. This warrants for reusing of search efforts from the past interval and perform an incremental search against the additional data.

1.3 Three Guidelines for Efficient Big Data Analytics

In light of these observations, we propose three guidelines for data analytics and demonstrate the guidelines through solving approaches of three representative problems from various domains.

1. [**Volume guideline**] **Explore geometric and domain-specific properties of high dimensional data for succinct representation** Reduce the volume of data through its dimension reduction or coreset computation appropriate for the problem by bounded approximations by exploiting geometric properties of high-dimensional space and transformations across spaces of different dimensions.

2. [**Variety guideline**] **Design domain-aware algorithms through mapping of domain problems to computational problems** We postulate that understanding domain originated properties of data, and the underlying process they represent can provide us with fresh insights for developing streamlined analytics solutions. Proper semantics can help with a structured representation of the data and optimal mapping of the problem to a computational framework.

3. [**Velocity guideline**] **Leverage incremental arrival of data through incremental analysis and invention of problem-specific merging methodologies** data arrives in a stream, and data gets updated over time. The ability to

reuse past analysis and merge the past result with incremental analysis through new statistics and problem properties can save us a great deal of redundancy in computing.

1.4 Representative Applications to Demonstrate Three Guidelines

We choose three problems from various domains to demonstrate these guidelines' applications, and we adopt the guidelines into the solving approaches.

1.4.1 Developing a Parallel and Portable Weighted Multi-Dimensional Scaling Tool

Statistical methods for analyzing high-dimensional data points are computationally expensive, and drawing meaningful statistical inferences in high dimensional space is often-time problematic. So, it's advantageous to reduce the data dimension without losing much information required to solve the problem. Multi-dimensional scaling (*MDS*) and its extension, weighted MDS (*WMDS*), are popular approaches for dimension reduction.

Many scientific domains use MDS for succinct data representation and data exploration. Psychologists use MDS to study the relationship between different stimuli[49]; Biologists use MDS for applications such as sequence alignment, protein substructure search, and RNA microarray analysis [137]. Many visual analytics programs project high-dimensional data points onto two- or three-dimensional space, and sometimes, the users inject their domain knowledge through various interactions that can be implemented using WMDS.

We demonstrate the first guideline (volume guideline) by exploring geometric properties of high dimensional data and geometric transformation (Chapter 2).

1.4.2 Identifying Carcinogenic Multi-Hit Combinations of Genetic Mutations

Experimental studies and mathematical models suggest that carcinogenesis is likely due to different combinations of a small number of carcinogenic mutations (hits) [19, 20, 109, 114, 130, 173, 195]. Current computational approaches focused on identifying individual genes that are cancer drivers, cannot find the specific combinations of mutations responsible for individual instances of cancer.

We want to develop a method for identifying combinations of genetic mutations that are most likely responsible for individual instances of cancer. It is theoretically possible to search for combinations of individual mutations, but the problem becomes computationally intractable since most genes contain hundreds of somatic mutations.

Understanding the biological process behind carcinogenesis can give us insight on how to formulate this difficult combinatorial problem and so that we can solve it reasonably. We demonstrate the second guideline (variety guideline) by designing a domain-aware approximate algorithm to identify carcinogenic gene combinations (Chapter 3).

1.4.3 Incremental Sequence Similarity Search via Automated E-Value Correction

Many bioinformatics research investigating biological or structural functions of nucleotide or protein sequences utilize some sequence similarity search tool. *BLAST*, short for Basic Local Alignment Search Tool [12] is a widely used (75,905 citations,

February 2019) sequence alignment tool that is capable of conducting a sequence similarity search for a sequence of interest against a curated sequence database. BLAST uses a statistical threshold called an expect-value (i.e., e-value) to infer homologous sequences from a curated database.

Sequencing data stored in the NCBI database has expanded astronomically over the years, reportedly doubling in the number of bases submitted to GenBank [22] every year over the last three decades (1982-present).

BLAST is computationally expensive to run, with computational time impacted by the number of queries and reference database size. Furthermore, genome sequencing and annotation projects can be reasonably long-term projects that require updates mid-project, e.g., regular annotation updates [74, 132]. However, for such updates, sequence similarity search steps have to be executed from scratch as search results from BLAST use similarity scores and e-values that depend on the size of the database, which continues to increase. Thus, it is required to discard the results of prior searches and rerun the entire search, which translates to irredeemable time, money, and computational resources.

In practice, new sequences get added to the search database(s) of interest in two ways: temporally and spatially. *Temporal* addition occurs when new sequences are added to a database over time (e.g., a regular update to the nr database). *Spatial* addition happens when different databases are available for search simultaneously. We need to combine the search results against these databases as if the result was obtained by searching against a combination of all these databases. Thus, we must compose search results in both the temporal and spatial dimensions and answer the following question: *Can we develop statistics to compose temporal and spatial BLAST search results through an e-value correction?*. We answer this question using the third guideline (velocity guideline) (Chapter 4).

1.4.4 Mitigating Catastrophic Forgetting Using Historical Summary

With the unprecedented advancement in scientific equipment and adaptation of electronics in our everyday life, the scientific community and consumer technology produce large and complex data at a high rate. Making meaningful inferences in a reasonable amount of time and cost from this big data is difficult for limitation in computing power and storage. Incrementally trained models increasingly perform poorly on past data; this phenomenon is known as *catastrophic forgetting*. There are several approaches to alleviating this problem with varying degrees of success. We want to answer the question: *Can a small volume of historical summary mitigate catastrophic forgetting effectively?*

We answer this question using the third guideline (velocity guideline) and the first guideline (volume guideline) in Chapter 5 primarily.

1.5 Organization of the Dissertation

The remainder of this thesis is organized to demonstrate three guidelines through representative applications (Figure 1.1). In Chapter 2, we demonstrate the volume guideline through the fast computation of dimension reduction technique WMDS through a geometric transformation and parallel programming. We also demonstrate how the accelerated dimension reduction tool can be used in fast visual analytics of big data.

Chapter 3 describes an application of the variety guideline through the mapping of the problem of identifying multi-hit combinations of genetic mutations responsible for cancers that can be mapped to weighted set cover through transforming cancer biology knowledge to computing domain. We show scaling up of the WSC algorithm to iden-

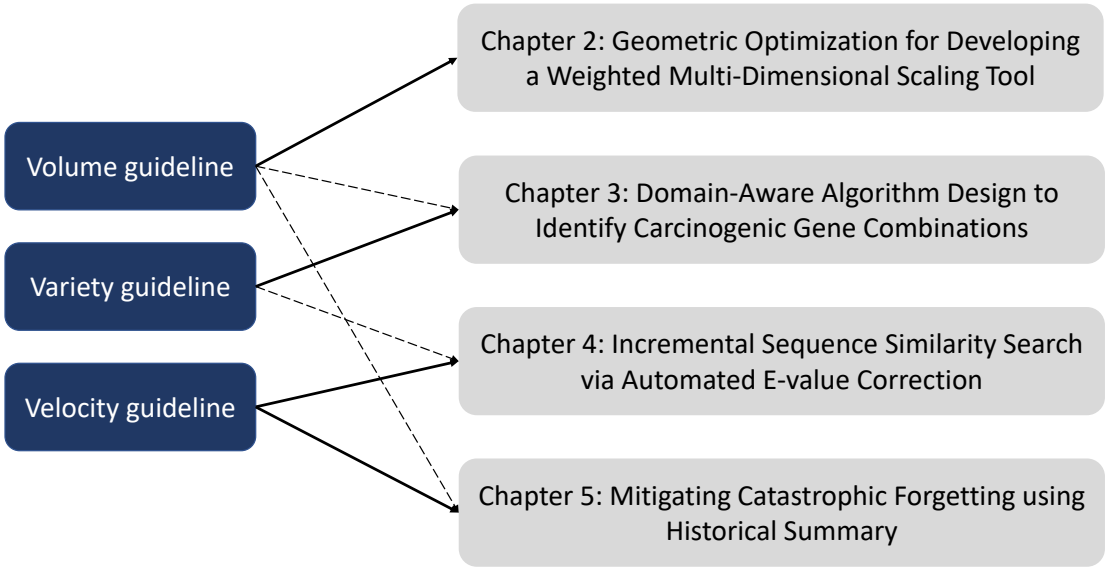


Figure 1.1: Organization of the dissertation's research contributions. The solid line between a guideline and a chapter indicates this chapter's primary goal is to demonstrate the guideline. The dashed line between a guideline and a chapter indicates that the guideline was also used to solve the problem addressed in that chapter.

tify three-hit combinations on a GPU using a bit-wise matrix representation of cancer genomic data (Section 3.4). Next, we show the scaling out of the WSC algorithm on the Summit supercomputer to identify four-hit combinations through some memory optimizations and equitable workload distribution (Section 3.6). We also make use of the volume guideline in scaling the approximate algorithm.

We demonstrate applications of the velocity guideline by solving two problems in Chapter 4 and Chapter 5. Chapter 4 describes our demonstration of the velocity guideline through the development of an incremental sequence similarity search tool iBLAST.

In Chapter 5, we explore various approaches to mitigate catastrophic forgetting in training deep learning models in a streaming setting. In Chapter 6, we present the conclusion, the thesis artifacts, and future work.

Chapter 2

Geometric Optimization for Developing a Weighted Multi-Dimensional Scaling Tool

2.1 Introduction

The representation of complex scientific data often materializes into points in high-dimensional space. Statistical methods for analyzing these high-dimensional data points are computationally expensive, rendering it unfeasible to draw any statistical inferences in a reasonable amount of time. Dimension reduction is an essential computational method for making the data comprehensible. Multi-dimensional scaling (*MDS*) and its extension, weighted MDS (*WMDS*), are popular approaches for dimension reduction. This chapter demonstrates how exploring geometric properties of data combined with parallel computation can generate a fast and portable dimension reduction tool (volume guideline).

MDS in science and visualization MDS is a tool of choice for many applications. Psychologists use MDS to study the relationship between different stimuli, where each stimulus is a multi-dimensional data point [49]. Biologists use MDS for many applications, including sequence alignment, protein substructure search, and RNA microarray

analysis [137]. For visual analytics, high-dimensional data points are projected onto two- or three-dimensional space so that scientists can more easily explore these points. Sometimes, the users inject their domain knowledge through various interactions. This domain knowledge is then utilized to refine the visualization.

WMDS introduces weights on different dimensions to enable users to explore a space of projections. For example, Leman et al. [107] used WMDS to create 2D-embedding by translating visual interactions into dimensional weights.

MDS for real-time visual analytics Inferring meaningful insights from the data is primarily offloaded to various complex mathematical, statistical, and machine learning algorithms. Sophisticated machine learning models, in many cases, are black-boxes and infer knowledge without offering much explanation to the underlying insights. Human domain experts can be integrated into the process to steer the analytics in a meaningful and explainable way.

Visual analytics of information plays a vital role in this regard by letting the user provide input to the visualization tool. It seeks to support sense-making on complex data through interactive visualizations without having to use complex mathematical representation. Semantic interaction is one such technique. Information visualization of high-dimensional data involves projecting complex high dimensional data onto a *two/three* dimensional space to work within the constraint of human visual cognition ability.

For interactive and real-time visual analytics, MDS must run in real-time on available computing devices. Python Scikit MDS uses the SMACOF[51] method, which requires computing $n^2/2$ pairwise distances. This approach is problematic because storing $n^2/2$ distances in memory can slow down overall system performance. For a dataset of size 683×9, Scikit-MDS[1] takes $30 - 50$ seconds. So, it is not suitable for real-time

interactive visualization. The same is true for virtually every sequential MDS method. While parallelized GPU implementations of MDS exist, they require NVIDIA GPU cards. We aim to develop a fast and portable MDS implementation that can run in parallel on available parallel hardware, such as multi-core CPUs, MICs, or GPU cards made by any vendor.

Dimension reduction of big data is a computationally expensive task. The considerable processing time in the dimension reduction phase makes information visualization of high-dimensional data and user interactions on big data impractical. We want to explore if a fast WMDS tool can expedite interactive visualization by integrating it into a family of visual analytics tool that uses Visual to Parametric Interaction (V2PI).

Chapter outline In Section 2.3.1, we present Claret, a parallelized and portable force-based WMDS. We ported and extended Chalmer's stochastic force-based MDS (SF-MDS)[33] to OpenCL, which runs on various platforms, including multi-core CPUs, GPUs, and FPGAs. To support the incremental nature of interactive visualization, we extended Glimmer's multi-level algorithm to use our OpenCL-based stochastic force calculation. To enable stable visualizations over time, in Section 2.3.3, we propose a method to quantify the quality of a layout, compare two embeddings quantitatively, and align embeddings.

For high-dimensional data points, the weighted Euclidean distance computation is a bottleneck even for parallel hardware. In Section 2.4.2, we prove that with a combination of a new mapping of data points (*Stretching*) and Johnson Lindenstrauss' lemma [88] that we can preserve weighted Euclidean distances and expedite distance computation.

In Section 2.5.3 and Section 2.6.2, we present our exploration to make one prominent family of interactive visual analytics tools (V2PI tools) fast through accelerated

dimension reduction, algebraic optimization, and incremental gradient computation. In Section 2.5.3), we describe the extensions of Claret to support V2PI interactions through accelerated forward WMDS and the integration of extended Claret into the Andromeda software ecosystem to facilitate visual analytics on big data. In Section 2.6, we show the algebraic optimizations and incremental computations to expedite inverse WMDS computation. In Section 2.5.4, we present a case study to demonstrate the usefulness of accelerated V2PI interaction in analyzing big data from cancer biology.

2.2 Background and Related Work

Dimension-reduction tools preserve some essence of the high-dimensional data such as the pairwise dissimilarity and variance. Popular dimension-reduction techniques include principal component analysis (PCA), multi-dimensional scaling (MDS), and linear discriminant analysis (LDA). PCA [124] reduces the dimension of the data by choosing directions along which the total variance of projected data points is maximized. MDS [174] preserves pairwise distances, which is a measure of dissimilarity. Weighted multi-dimensional scaling (WMDS) preserves pairwise weighted distances. LDA [83] finds a linear combination of features to reduce the dimension, and it preserves the class discrimination between points.

2.2.1 Mathematical Formulation

Assume there are n points in $\mathbb{R}^d$ and they are represented as a collection of n d-dimensional vectors. W is a d-dimensional weight vector. The projection of these high-dimensional data points onto $2D$ space, L is n two dimensional vectors. H, W, L

can be written as $\begin{pmatrix} \{h_{1,1} \dots h_{1,d}\} \\ \vdots \\ \{h_{n,1} \dots h_{n,d}\} \end{pmatrix}$, $\begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix}$, and $\begin{pmatrix} \{l_{1,1}, l_{1,2}\} \\ \vdots \\ \{l_{n,1}, l_{n,2}\} \end{pmatrix}$. MDS projects data by minimizing some variant of a stress measure, as depicted in equation 2.1.

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\sqrt{\sum_{k=1}^{d} (h_{i,k} - h_{j,k})^2} - \sqrt{\sum_{k=1}^{2} (l_{i,k} - l_{j,k})^2} \right)^2 \tag{2.1}$$

WMDS, on the other hand, preserves weighted high-dimensional distances between points in a low-dimensional space. It minimizes a slightly different stress function, which uses W while computing high-dimensional distances. Stress for WMDS is defined as follows:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \left(\sqrt{\sum_{k=1}^{d} (h_{i,k} - h_{j,k})^2 w_k} - \sqrt{\sum_{k=1}^{2} (l_{i,k} - l_{j,k})^2} \right)^2 \tag{2.2}$$

A formal definition of WMDS follows:

Definition 2.1 (WMDS)**.** Given high-dimensional data $\mathbf{H}$, and a dimensional weight vector W, find a two-dimensional embedding $\mathbf{L}$ that minimizes the stress in equation 2.2.

We present the state of the art of MDS tools, focusing mainly on GPU-based tools. We also briefly discuss random projection and layout matching.

2.2.2 Force-Directed Multi-Dimensional Scaling

We can view the projection of high-dimensional points onto low-dimensional space as a layout optimization problem. Minimizing the stress function narrows the difference between the high-dimensional distance and low-dimensional distance for all pairs. If we start with an initial random layout, i.e. *2D* projection, and guide the points to move

around while lowering the difference between distances in two spaces, it will eventually converge to the optimal layout.

In the n-body problem, every point exerts forces on all other $n-1$ points depending on some measure such as mass and distance in the gravitational force field and charge and distance in the electrostatic force field. In layout computation, a given point needs to move towards or further from any other point in the embedding. The amount of these movements depends on how closely their distance in the current layout matches with their high-dimensional counterpart. If we are to attach a force between these two points, it should be proportional to the difference between these two distances.

Thus, the layout computation problem maps to an n-body problem when we apply a force between two points i and j proportional to the measure, $LD(i,j) - HD(i,j)$. If $LD(i,j) - HD(i,j) < 0$, the force is repulsive, and it will move the two points apart. If $LD(i,j) - HD(i,j) > 0$, the force is attractive, and this will move the two points closer. For the i^{th} point, the total experienced force is

$$F_i = \sum_{j \neq i} K \times (LD(i,j) - HD(i,j))$$

Here K is a constant that we can tune for the dataset and simulation environment. Once we compute force for a point in *2D*, we can estimate acceleration, which is proportional to the force. This acceleration can be used to calculate current velocity, and in turn, the point's next position. Velocity is updated using $v = v_0 + a \times \delta t$, and the next position is updated using $x = x_0 + v \times \delta t$. Here, δt is the simulation step. $\langle v_0, x_0 \rangle$, and $\langle v, x \rangle$ are the velocity and position at the beginning and at the end of the current timestamp.

Force computation at each step is a $O(n^2)$ operation since we have to compute $O(n^2)$ distances in $2D$. This estimate also assumes that we pre-compute all the high-dimensional distances; otherwise, this computation becomes $O(n^2 \times d)$ operations.

We can compute forces, velocities, and positions of n points independently. Multiple implementations use spring-force simulation as a means to perform MDS. Since we can map MDS to an n-body problem, a well-studied and optimized problem in GPU computing, we take this approach as the core of Claret.

Stochastic force-based MDS (SF-MDS)

Force-based MDS can be computed in parallel using many cores; however, the computation workload per thread is still large. Assuming we have n parallel threads at our disposal, the i^{th} thread will compute F_i, which is a summation of forces exerted by the $n-1$ other points. For large n, each thread might take a long time to finish its computation.

Chalmers et al. [33] made an observation that we can perform force simulation with much less effort using two small representative sets from the $n-1$ points. In their algorithm, they maintain two sets, a "near set" of size s_n and a "random set" of size s_r. The near-set gradually converges to contain the nearest s_n points the while random set always picks s_r new points at every simulation step. The near-set expedites convergence by providing local structure information, and the random set helps the embedding by enforcing global structure. The sizes used by Chalmers' algorithm are 14 and 10, respectively, which were determined empirically. Though Chalmers' version is a sequential one, we can leverage the fact that n-body simulation can be performed in parallel, and reduce the per-thread workload to a constant amount. This observation also helps us with the stress computation. Instead of adding $O(n^2)$ differences, we can add $O(n)$ distances to approximate the stress.

Several realizations of GPU-based *MDS* exist. The field of bioinformatics has produced quite a few GPU-based MDS tools in CUDA, an NVIDIA-specific GPU programming language. Fester et al. [61] implemented a CUDA version of HiT-MDS by employing

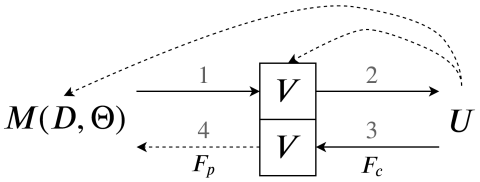
reduction to add a large group of numbers and computing multi-dimensional distances in parallel. CUDA-based fast multi-dimensional scaling (CFMDS) [137] dynamically decides whether to run MDS on the entire dataset or divide the data into chunks that can fit into the global memory of the NVIDIA GPU card depending on the input size.

Multi-level SF: Glimmer Glimmer by Ingram et al. [81] uses stochastic force as the base algorithm for their force-based MDS, and they implemented this on a GPU using OpenGL, primarily a graphics programming language. One major contribution of their work is that they optimize the layout at multiple levels. Glimmer divides the dataset into $log_b n$ levels, data ranges in these levels are $[0, max(MIN\_SIZE, \frac{n}{b^{\log_b n}})], \ldots, (\frac{n}{b^2}, \frac{n}{b}], (\frac{n}{b}, n]$, where b is a constant called the decimation factor. Glimmer uses three operators at each level: restrict, relax, and interpolate. The *restrict* operator samples points for that range. *Relax* runs stochastic force to find optimal embedding for all points up to the previous level. *Interpolate* uses all relaxed points up to the previous level to sample the near and random set to run stochastic force on the data at the current level. In the last level, relaxing all data produces the final embedding.

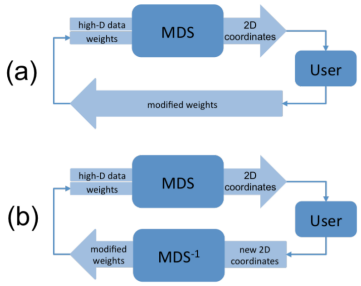
2.2.3 Application of Weighted Multi-Dimensional Scaling (WMDS) in Visual to Parametric Interaction (V2PI)

The choice of dimension reduction method largely depends on the type of visual interaction in use. Various interactive visual analytics tools use two categories of interactions [107]:

1. *Surface level interactions*: all interactions performed on visual media and limited to data manipulation in the visual domain are surface-level interactions. Selecting, dragging, and highlighting projected points are some examples of surface-level interactions.



(a) V2PI interaction pipeline. In step 1, mathematical model M creates a visualization V using input data D and model parameter Θ. In step 2, V is displayed to user. In step 3, the user interacts with the visualization on surface level (F_c) and in step 4, user interaction is parameterized (F_p) to update model parameter Θ. The user can also directly modify the Θ. (The image is redrawn from [107]).



(b) Andromeda pipeline. (a) parametric interaction, the user directly alters dimensional weights (Θ). (b) V2PI interaction, the user provides feedback through visual interaction, that feedback is parameterized to modify Θ. (Reproduced from [158])

Figure 2.1: V2PI interaction pipeline and pipeline for one of its realization: Web Andromeda.

2. *Parametric interactions*: direct interactions to change the mathematical model parameters Θ are parametric interactions. In the case of a visualization created by WMDS, changing dimension weights directly would be a parametric interaction.

[107] described a typical data visualization pipeline as an iterative process where a mathematical model summarizes the data through dimension reduction. The user discovers knowledge at the end of this process through observation. This pipeline can miss some useful visual representations when the predefined model parameters don't agree with or reflect expert knowledge. Visual to parametric interaction (*V2PI*) introduces a new type of interaction where users' surface-level interactions are quantified automatically to modify the model parameters (Figure 2.1a). This interaction alters model parameters on behalf of the user, thereby allowing them to interact with the system without needing in-depth knowledge of the underlying models themselves.

Notion of *forward* and *inverse* WMDS in the Context of V2PI Tools

Web Andromeda is one realization of V2PI, which supports both parametric interaction and visual to parametric interaction within a weighted multi-dimensional scaled (*WMDS*) projection of data onto a lower-dimensional place. In parametric interaction, the user can up-weight or down-weight every dimension of the data through sliders associated with them. These weights are WMDS' parameter Θ. In V2PI, the user moves around projected data points. The data points brought together by the user are deemed as similar, and the data points moved apart are interpreted as dissimilar. These interpreted similarities and dissimilarities are then quantified to modify the dimension weights Θ; this parameter is then used by WMDS to create a $2D$ projection. (Figure 2.1b).

In parametric interaction, the user is directly modifying the model parameter Θ, and the V2PI tool needs to use WMDS to project the data onto 2D, we call this WMDS as *forward* MDS.

Forward WMDS in Creating a Visualization from Parametric and Visual to Parametric Interactions

Through parametric interaction, the user provides the input model parameter, $\Theta = W = [w_1, w_2, \ldots, w_d]^T$. The mathematical model used here is WMDS, and $WMDS(H, \Theta)$ creates the visualization. When we have the input high dimensional data points H, dimensional weights $\Theta = W$, and we are tasked to create the low dimensional projections L, $WMDS(H, W)$ is called forward MDS.

Inverse WMDS to quantify V2PI interactions

When a V2PI interaction occurs, the user makes some surface-level interaction to convey domain knowledge to the visualization. Andromeda interprets the semantics of this interaction to parameterize the model parameter. The user interactions amount to low dimensional coordinates of a subset of the points and Andromeda finds the dimensional weights by minimizing the same stress function as forward WMDS. We call it inverse WMDS and a formal definition of inverse WMDS follows:

Definition 2.2 (iWMDS)**.** Given high-dimensional data **H**, and a two-dimensional embedding **L**, find the dimensional weight vector W, that minimizes the stress in equation 2.2.

2.3 Claret: A Fast and Portable Multi-Dimensional Scaling (MDS) Tool

To develop Claret, we parallelize the sequential MDS algorithm SF-MDS and port the parallelized version into parallel hardware using OpenCL. Claret also uses Glimmer's multi-level approach to obtain faster convergence.

Continuing from sub-section 2.2.2, SF-MDS is a linear approximation of force-based MDS. Instead of computing force from all $n-1$ points, SF-MDS uses two small subsets to do so. At every iteration, the near set is updated by choosing the s_n nearest points (according to their high dimensional distances from the point under consideration) from $pivot\_size = s_n + s_r$ pivot points. We summarize SF-MDS in Algorithm 1, where $f()$ and $g()$ are linear functions to allow tuning simulation parameters.

We want to parallelize algorithm 1 for efficient implementation and fast execution on any OpenCL supported devices. In OpenCL architecture, we have two types of

Algorithm 1 Force-based MDS.

Require: $highD[n \times d]$, $lowD[n \times 2]$, $velocity[n \times 2]$, $force[n \times 2]$, $pivots[n][pivot\_size]$

```
1:  while converge() ≠ true do
2:      for i = 0 → n do
3:          near_set ← pivots[i][0 ... near_set_size]
4:          random_set ← randomindices
5:          my_pivots ← near_set ∪ random_set
6:          for j = 0 → pivot_size do
7:              k ← my_pivots[j]
8:              hDistance[k] ← dist(highD[i], highD[k])
9:              lDistance[k] ← dist(lowD[i], lowD[k])
10:             force+ = f(hDistance[k] − lDistance[k])
11:         sort my_pivots based on hDistance
12:         a ← g(force)
13:         velocity[i] ← velocity[i] + at
14:         lowD[i] ← lowD[i] + velocity[i] × t
15:         pivots[i] ← my_pivots
```

computing devices: host and device. The host is usually a CPU which can launch parallel programs into devices and act as the moderator and controller. The host has host memory, and the device has a hierarchy of memory consisting of global memory, constant memory, and local memory. We load the input data into host memory; the host program then launches parallel programs in SIMD fashion into computing units of one or more devices.

Given the sequential algorithm as depicted in algorithm 1, the goal is to develop a parallel program that achieves similar functionalities and can run on any computing device having the parallel computing architecture specified by the OpenCL standard.

2.3.1 Porting Stochastic Force-Based Multi-Dimensional Scaling (SF-MDS) to GPU Using OpenCL

Any n-body problem can be parallelized across n points. Every point experience force from all other (or in the case of SF-MDS, a subset of) points which are frozen in time

and space. At the beginning of every iteration, each point sees the same configuration ($\langle position,\ velocity,\ \ldots \rangle$) of points. In SF-MDS, every point experiences force from $pivot\_size = O(1)$ other points. So, we don't unroll the loop for iterations over time; instead, we unroll/parallelize the outer for loop in line 2 as every point can be processed independently in a given duration of δt.

So, every iteration for every point runs in $O(1) = O(n)/n$ time. The code block consisting from line 3 through line 15 computes the force and updates the position for a given point. We can take this block and put it inside a computing unit, namely thread, to take care of individual points in parallel. Algorithm 2 shows a high-level OpenCL kernel of Claret. At every iteration, n such kernels are launched to update the positions of n points in parallel.

There are some implementation/porting challenges which can cripple the performance on different accelerators in OpenCL programming paradigm. We address some of these issues in the remaining part of this section.

Memory and data management Solving an n-body problem for large data requires storing a significant amount of data in memory, efficient access to that memory, and minimal data transfer between the host and device.

We give an estimate of the in-memory data storage requirement during a single iteration in table (2.1). For the purpose of demonstration, we set $pivot\_size$ to 8.

Buffer	Purpose	Size	Type
highD	high dimensional data	$n \times d$	float
lowD	2D projection	$n \times 2$	float
pivot_indices	Near and Random index	$n \times 8$	unsigned int
hd_distances	HD distances to pivots	$n \times 8$	float
ld_distances	LD distances to pivots	$n \times 8$	float

Table 2.1: The comprehensive list of required memory.

Algorithm 2 Claret Kernel.

1: $gid \leftarrow get\_global\_id(0)$
2:
3: //copy data from global memory
4: $near\_set \leftarrow pivots[i][0 \ldots near\_set\_size]$
5: $random\_set \leftarrow randomindices$
6: $my\_pivots \leftarrow near\_set \cup random\_set$
7: $v_0 \leftarrow velocity[gid]$
8: $x_0 \leftarrow lowD[gid]$
9:
10: //compute force using pivot points
11: **for** $j = 0 \rightarrow pivot\_size$ **do**
12: $k \leftarrow my\_pivots[j]$
13: $hDistance[j] \leftarrow dist(highD[gid], highD[k])$
14: $lDistance[j] \leftarrow dist(lowD[gid], lowD[k])$
15: $\delta v \leftarrow v_0 - velocity[k]$
16: $force+ = f\big(hDistance[k] - lDistance[k], \delta v\big)$
17: sort my_pivots based on $hDistance$
18: globalSynchronization()
19:
20: //update velocity and position
21: $a \leftarrow g\big(force\big)$
22: $v \leftarrow v_0 + at$
23: $x \leftarrow x_0 + v \times t$
24: //copy data back to global memory
25: $velocity[gid] \leftarrow v$
26: $lowD[gid] \leftarrow x$
27: $pivots[gid] \leftarrow my\_pivots$
28: globalSynchronization()
29:
30: //Compute low dimensional distances
31: **for** $j = 0 \rightarrow pivot\_size$ **do**
32: $k \leftarrow my\_pivots[j]$
33: $lDistance[j] \leftarrow dist(lowD[gid], lowD[k])$

We have to store around $(22 + d) \times n$ floating point numbers in device memory. So, even for an input data as big as $10^6 \times 100$, the required memory is around $500MB$, which can easily fit in the device memory of modern hardware accelerators.

Since pre-computing $O(n^2)$ distances requires a large amount of device memory, we compute distances on the fly. That also helps avoid moving a great deal of data between the host and device.

Low latency in memory access We coalesced memory access so that whenever possible, the compiler can resort to vector operation. Before the first iteration, we pre-compute $n \times pivot\_size/2$ pivot indices in the range $[0, n)$ and offload the whole data into device's global memory. At every iteration, for every point we generate a random starting point as $si = f(global\_id,\ iteration)$ and access $pivot\_indices[si \ldots si + pivot\_size/2]$ as new random points.

Moving data back and forth between the host and device is a time-consuming task. So, we move almost the entirety of the data at the beginning of the first iteration to device global memory, and then between iterations, we only fetch a constant sized data from the device to host.

Global Synchronization and Kernel Fusion At any given iteration of force simulation, every point sees the same configuration, the same high and low-dimensional positions. We want to ensure consistent access to global memory shared by all threads.

In algorithm 2, during a given iteration (same time window), all threads access $lowD$ 4 times in lines 9, 14, 26, and 33. Except for the third access, all other accesses are read accesses. These accesses have a deterministic order, let's call them $R1$, $R2$, $W1$, and $R3$. $W1$ is a write access that can create a data race between threads if the threads do not synchronize before and after this step. So, we put two global synchronization

points in the kernel.

Unfortunately, OpenCL does not directly support global synchronization. Furthermore, the global synchronization mechanism offered by Xiao et al. [187] is not viable because OpenCL cannot globally synchronize across workgroups. Instead, we break the workflow of a single thread across three kernels at the points of required global synchronization. Between two kernel calls, control returns to the host, and thus, all threads get a chance to synchronize globally.

While this approach ensures the correctness of our parallelization, the overhead of coming back to CPU is non-trivial. So, we seam the kernels back together using kernel fusion through double buffering. We maintain two buffers for one array, and at any given step, all threads read from the same buffer and write to the other buffer. This method ensures $W1$ does not create any inconsistency in values read by different threads.

Computing distance in parallel Every thread needs to compute $pivot\_size$ pairwise distances, each of them is an $O(d)$ task if computed sequentially within the thread. Ideally, each pairwise distance computation comes down to reducing d values into 1 value. Each main thread is launching $(pivot\_size \times d)$ threads to reduce $pivot\_size$ values. This mechanism is known as dynamic parallelism, and only a handful of GPU cards support this. Since we do not want to restrict Claret to run on only a selected few accelerators, we solve this in software.

We can break the kernel into three segments for three tasks. Each thread will run in parallel to compute pivot indices and then they will sync. After that, we combine all threads' reduction jobs into one big reduction job. Here $(n \times pivot\_size \times d)$ values will be reduced to $(n \times pivot\_size)$ values by $(n \times pivot\_size \times d)$ reduction threads.

Once the reduction threads finish, all distances for all regular threads are completed and available. Now, each regular thread can resume computing forces and positions for

the points for which they are responsible.

Stress computation and convergence At every iteration, after every point's position is updated, we compute stress by a reduction in the accelerator. We smooth the stress curve by taking moving average, and we use Cauchy Convergence test for deciding termination. In each level, the stress starts from a high point and eventually plateaus.

2.3.2 Performance and Portability of Claret

The two main foci of this undertaking of implementing WMDS in OpenCL are:

1. to be able to run this on multiple accelerator types.
2. make the runtime fast enough for interactive visual analytics.

In this section, we will first start with demonstrating the correctness of our implementation by comparing the embedding for several datasets produced by Claret against that of Scikit-MDS and Glimmer. Next, we will show the runtime performance on various accelerators with different configurations.

We use a range of datasets from different sources.

1. **Cancer**: This dataset contains information regarding breast cancer patients. There are 683 patients each with 9 features.
2. **Shuttle**: This dataset is collected from NASA. It contains 43500 data points about shuttle turn correlation, and each data point has 9 features.
3. **Supreme Court Ruling**: We constructed two datasets of size 14000×20 and 14000×100 by running topic analysis on supreme court rulings.

4. **Artificial Data**: We generate artificial data from 10 20-variate normal distributions. These datasets will have 10 clusters with 20 dimensions.

Layout validation We get a similar output irrespective of our choice of the accelerator for running Claret. We compute embeddings for the same datasets using Claret on GPU, Scikit-MDS on CPU and Glimmer on GPU. As we can see in figure 2.2, the Claret's embedding is comparable with the Glimmer and Scikit-MDS's embedding. Though the later was only able to compute embedding for the smallest dataset. These embeddings are visually similar. We also compared stress of the embeddings, and the stresses are within ±5% of each other.

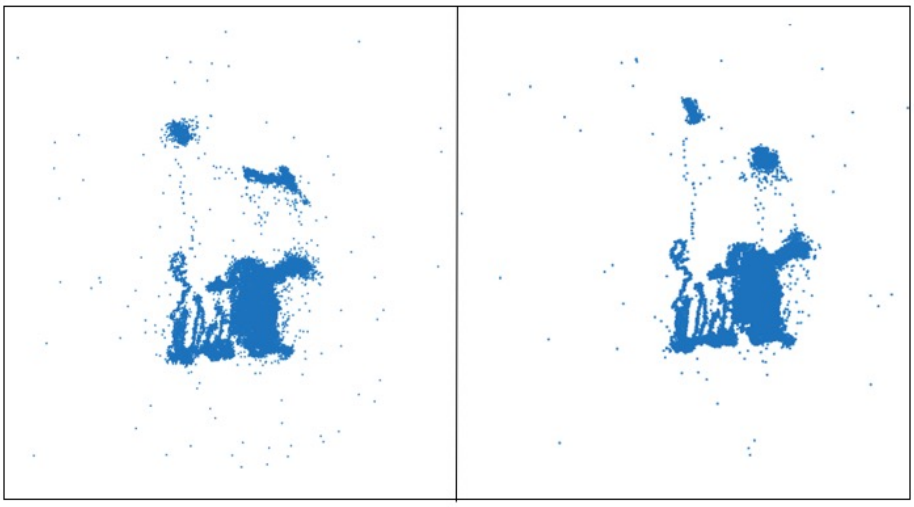


Figure 2.2: Embeddings produced by Claret (left) and Glimmer (right). The embeddings are visually similar as well as their stresses are within ±5% of each other.

Comparison against Other GPU-based MDS Tools We want to compare Claret's performance against a sequential MDS tool such as Scikit-MDS. As anticipated in earlier sections, Scikit-MDS can not compute embeddings for larger datasets. It ran out of memory for Shuttle and SC Ruling data even in a system with 8GB of memory. So, we compare Claret's performance against that of Scikit-MDS's on the Cancer dataset.

We could run Scikit-MDS only on Cancer dataset (683×9); it took 15s even with 4 parallel threads. Claret took 310ms. We ran both tools in Xeon-E5 with 4 CPU cores.

There are several GPU-based MDS tools; CFMDS and Hit-MDS are implemented in CUDA and Glimmer are implemented in OpenGL. CFMDS has a dependency on CULA which is discontinued, and Hit-MDS is implemented using very old version of CUDA. Osipyan et al. [133] reported that Glimmer is faster than CFMDS and Hit-MDS. So, we compare Claret's performance against Glimmer's performance on NVIDIA Titan X GPU card. From Figure 2.3, we see that Claret is $(3-9)X$ faster than Glimmer.

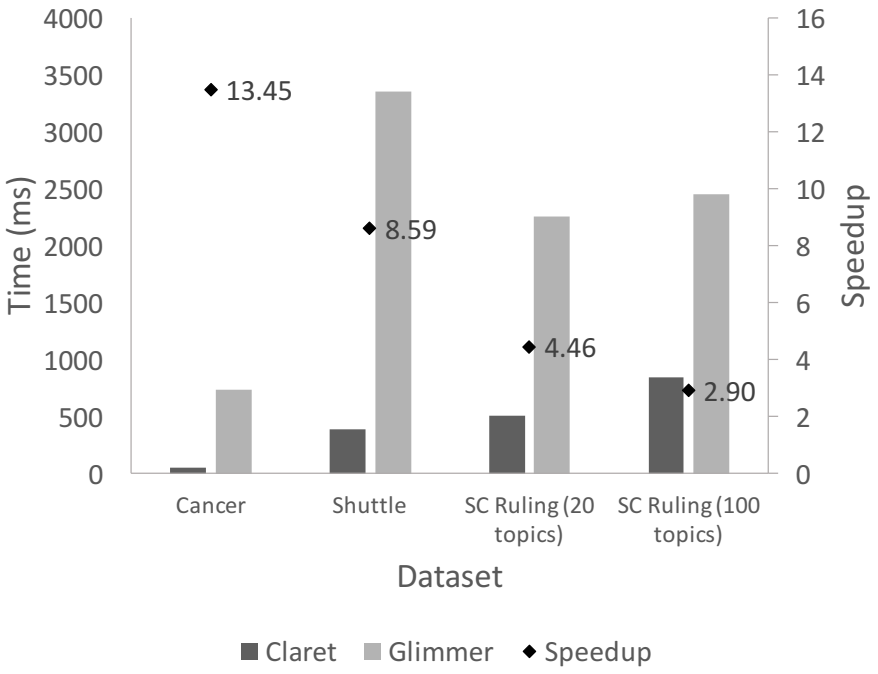


Figure 2.3: Performance comparison between Claret and Glimmer on four different datasets.

We also experimented with artificial data to see whether the performance is dependent on the values of n. Figure 2.4 shows the result. Claret's speedup compared to Glimmer is in the range $6.28X - 1.45X$. As the data size increases, the speedup decreases.

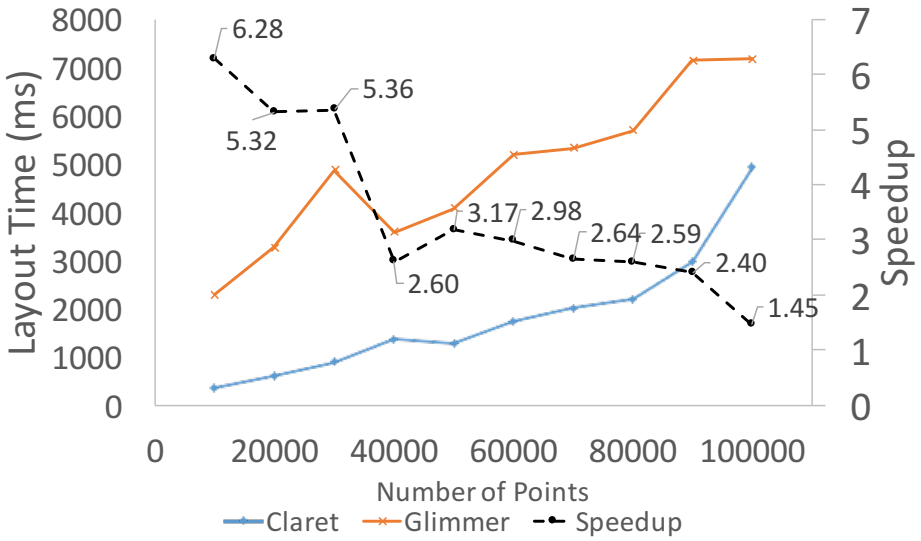


Figure 2.4: Layout computation time for Claret and Glimmer.

Running on various accelerators The crux of our motivation is portability – the ability to run on many different kinds of hardware. We show that Claret runs on 4 different accelerators including:

1. 22-core Xeon E5-2637 CPU @3.50GHz by Intel

2. Tesla K80(Kepler) GPU by NVIDIA with 2496 cores

3. Hawaii GPU by AMD

4. Xeon Phi accelerator by Intel with 61 cores

From Figure 2.5, CPU, and GPU performances are comparable because we have used a powerful CPU with 22 cores. The poorer performance on Xeon Phi in contrast to GPUs suggests that we should take individual core's parallelizability and computing power into consideration when designing parallel tasks.

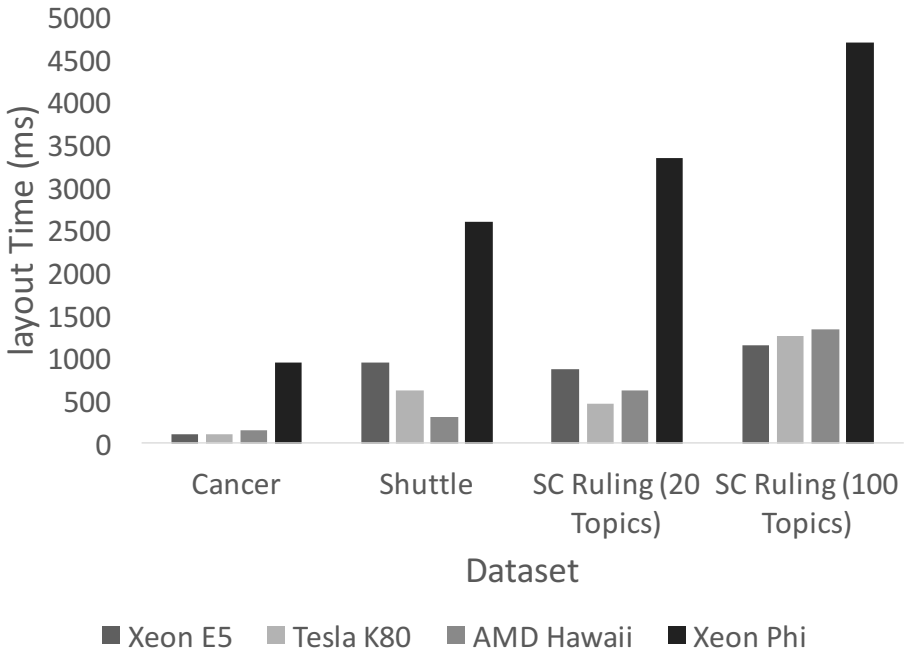


Figure 2.5: Layout computation time in different accelerators.

2.3.3 Quantifying Layout Similarity

Embeddings created by different MDS implementations or the same implementation in different phases might appear dissimilar. We want to investigate similarity between such embeddings.

To quantify the similarity between embeddings, we propose a method based on geometric shape matching. Let point set P consist of n points in d-dimensional space. Let L_1, L_2 be two projections created by MDS. We want to quantify the similarity between these two.

If two embeddings are similar under rotation and translation, we align them using center of mass and principal components. First, we compute centers of mass C_1, C_2 of L_1, L_2 respectively. Then, we apply $C_2 - C_1$ translation to L_1 so that their centers coincide. $L_1' = L_1' + C_2 - C_1$. We compute their first principal components $\vec{v_1}, \vec{v_2}$ for L_1', L_2 and the angle θ between $\vec{v_1}, \vec{v_2}$. Finally, we apply θ rotation to L_1' so that $\vec{v_1'}$ and $\vec{v_2}$ are

aligned.

To compute similarity between the layouts L_1, L_2, we then pick n corresponding point pairs (p_i, q_i) where $p_i \in L_1'$ and $q_i \in L_2$. We then compute n Euclidean distances between points in each pair, and take their average to get the final score. Formally,

$$similarity(L_1, L_2) = \sum_{\substack{i=1 \to n \\ p_i \in L_1', q_i \in L_2}} dist(p_i, q_i) \tag{2.3}$$

The alignment procedure can be used to stabilize $2D$ projections in different phases of interactive visualization.

2.4 Stretched Random Projection

In WMDS, we have to compute $O(n^2)$ weighted Euclidean distances in $\mathbb{R}^d$ which requires $O(n^2 d)$ operations. We propose a way to cut this computation.

2.4.1 Johnson-Lindenstrauss Lemma for Weighted Euclidean Distance

The compute kernels compute pairwise distances on the fly To reduce global memory usage. Each distance computation is a $O(d)$ task which can slow down the program for large $d(d \approx n)$.

We solve this problem using a result from geometry. Johnson–Lindenstrauss lemma [88] states that if we project a high dimensional dataset onto a randomly chosen subspace of much smaller dimensions, it preserves the Euclidean distances approximately. Since *WMDS* requires retaining weighted Euclidean distances, we extend JL lemma to prove that similar result can be achieved for weighted Euclidean distance as well.

Definition 2.3. Given a set of n points in $\mathbb{R}^d$, and a parameter $\epsilon > 0$, a projection of P onto a random k-dimensional linear subspace, a distance $||p - q||_2$ is ϵ-preserved if $(1 - \epsilon)||p - q||_2 \leq \sqrt{d/k}||f(p) - f(q)||_2 \leq (1 + \epsilon)||p - q||_2$.

Theorem 2.4. ***Johnson-Lindenstrauss Lemma*** *Let P be a set of n points in $\mathbb{R}^d$, let $\epsilon > 0$ be a parameter, and let $k = (1/\epsilon^2) \log n$. Let Q be the projection of P onto a random k-dimensional linear subspace. Then all pairwise Euclidean distances in P are ϵ-preserved by the corresponding pairwise Euclidean distances in Q with probability at least $1/2$.*

Definition 2.5. Given a point set P in $\mathbb{R}^d$ and a dimensional weight vector W, the weighted Euclidean distance between two points p, q in P is $||p-q||_{w2} = \left(\sum_{i=1}^{k} w_i (x_{pi} - x_{qi})^2 \right)^{1/2}$.

Definition 2.6. Stretching $p = [x_{p1}, x_{p2}, \ldots x_{pd}]^T$ by $W = [w_1, w_2, \ldots, w_d]^T$ is scaling p by w_i along i^{th} dimension for $i \in \{1, 2, \ldots, d\}$ so that $p' = [\sqrt{w_1} \times x_{p1}, \ldots, \sqrt{w_d} \times x_{pd}]^T$. Formally, $p' = p \circledcirc W = [\sqrt{w_1} \times x_{p1}, \ldots, \sqrt{w_d} \times x_{pd}]^T$. A point set P is stretched by W if every point $p \in P$ is stretched by W.

Theorem 2.7. ***Johnson-Lindenstrauss Lemma for weighted Euclidean distance*** *Let P be a set of n points in $\mathbb{R}^d$, W is d-dimensional weight vector, let $\epsilon > 0$ be a parameter, and let $k = (1/\epsilon^2) \log n$. Let Q be the projection of $P \circledcirc W$ onto a random k-dimension linear subspace. Then, all pairwise weighted Euclidean distances in P are ϵ-preserved by the corresponding pairwise Euclidean distances in Q with probability at least $1/2$.*

Proof Let, $p' = p \circledcirc W$ and $q' = q \circledcirc W$. Both, p' and q' are points in $\mathbb{R}^d$ and they can be mapped to their objects p and q. By Theorem 2.4, $||p' - q'||_2$ is ϵ-preserved after projecting $P \circledcirc W$ onto k-dimensional linear subspace.

$$
\begin{aligned}
||p' - q'||_2 &= ||p \circledcirc W - q \circledcirc W||_2 \\
&= \Big(\sum_{i=1}^{k} (\sqrt{w_i} x_{pi} - \sqrt{w_i} x_{qi})^2 \Big)^{1/2} \\
&= \Big(\sum_{i=1}^{k} w_i (x_{pi} - x_{qi})^2 \Big)^{1/2} \\
&= ||p - q||_{w2}
\end{aligned}
\tag{2.4}
$$

According to Theorem 2.4, $(1-\epsilon)||p'-q'||_2 < \sqrt{d/k}||f(p')-f(q')||_2 < (1+\epsilon)||p'-q'||_2$. We know from equation 2.4, $||p'-q'||_2 = ||p-q||_{w2}$.
So, $(1-\epsilon)||p-q||_{w2} \leq \sqrt{d/k}||f(p')-f(q')||_2 \leq (1+\epsilon)||p-q||_{w2}$. Hence, the weighted Euclidean distance between two points in P are preserved by the corresponding Euclidean distance in Q with probability at least 1/2.

2.4.2 Computing Weighted Euclidean Distance Using Random Projection

To compute pairwise weighted Euclidean distances in P, we first compute $P \circledcirc W$. This can be accomplished by multiplying $(n \times d)$- matrix P with a $(d \times d)$ diagonal matrix W_D, where $W_D[i,i] = w_i$. $P' = P \circledcirc W = P \times W_D$.

Then, we will project P' onto k-dimensional linear subspace by multiplying P' with a $d \times k$ dimensional random matrix R. So, the projected point set $Q = P' \times R = P \times W_D \times R$.

Once we have computed Q, we will compute pairwise Euclidean distances in Q, and they will be ϵ-approximation of the weighted Euclidean distances in P.

Pre-processing time Construction of W_D takes $O(d)$ time. $P \circledcirc W$ takes $O(nd)$ time since W_D is a sparse matrix. R can be a sparse matrix [6] with only 1/3 non-zero entries. So, the overall runtime for this pre-processing step is $O(n^2 + n\log n + n^2 \times logn)$ $= O(n^2 \log n)$.

For large d, we use this result to create $H' \in \mathbb{R}^{O(\log n)}$ and then run MDS on H'. The saved computation in MDS for the reduced dimension is enough to pay for this pre-processing step.

2.5 Accelerated Forward Weighted Multi-Dimensional Scaling for Visual to Parametric Interaction Tools

In our prior work [45], we developed a portable parallel tool called Claret to compute MDS/WMDS projections fast using parallel computing on available hardware accelerators. WMDS from Claret can project large volumes of high-dimensional data in less than a second while its sequential counterpart can take minutes. Integrating WMDS from Claret into V2PI analytics tools can make interactive visual analytics fast and provide a better user experience.

Claret MDS/WMDS implementation works with Euclidean distances only. A geometric transformation called *Stretched Random Projection* reduces high-dimensional (when $d \approx n$) data into an intermediate low ($\log n$) dimensional space as a pre-processing step when Euclidean and weighted Euclidean distance is concerned. However, in the current implementation of Claret, MDS/WMDS for non-Euclidean distance is not supported. Tools facilitating V2PI interactions need to be able to use non-Euclidean distances such as Jaccard similarity index, Cosine similarity, and Mahalanobis distance. For Claret to be beneficial for V2PI interactions, we need to extend Claret's functionalities to support these distances.

Claret was implemented in OpenCL to provide portability across accelerators. However, implementation in the vendor-specific language can speed up the computation significantly. In this work, extend Claret in two additional GPU programming languages: CUDA for NVIDIA GPUs and HIP for AMD GPUs. Claret runs an iterative force-based algorithm to optimize lower-dimensional projection by minimizing the stress as a measure of force between two points.

2.5.1 Supporting Non-Euclidean Distances

There are various types of non-Euclidean distances that V2PI tools can use to quantify the similarity between data points, depending on the context. We primarily consider three distances:

1. ***Jaccard index*** [148] is a measure of similarity between two sample sets. For data with binary features, a single data point can be considered as a set of features. If two data points are represented as A and B, the similarity between them is computed using $J_I(A,B) = \dfrac{A \cap B}{A \cup B}$. The distance between these two points is then computed as $d_{AB} = 1 - J_I(A,B)$.

2. ***Cosine similarity*** [162] measures similarity between two points by quantifying similarity between two representing non-zero vectors in high-dimensional space. This similarity is measured by taking the cosine of the angle between the two vectors. For two data points $A = [A_1, A_2, \ldots, A_d]$ and $B = [B_1, B_2, \ldots, B_d]$ is: $similarity(A,B) = \dfrac{\sum_{i=1}^{d} A_i B_i}{\sqrt{\sum_{i=1}^{d} A_i^2}\sqrt{\sum_{i=1}^{d} B_i^2}}$. Distance between these two points, $d_{AB} = 1 - similarity(A,B)$.

3. ***Manhattan distance*** [98] between two points is the sum of the absolute differences of their feature values. Manhattan distance between two data points $A = [A_1, A_2, \ldots, A_d]$ and $B = [B_1, B_2, \ldots, B_d]$, $d_{AB} = \sum_{i=1}^{d} |A_i - B_i|$.

In the claret::wmds kernel (Algorithm 3), we first compute weighted Euclidean distances between the current point and points in its near- and random- sets (Line 13).

We have modified the distance function (dist) to take one additional parameter (*distance_type*), and based on its value, this function now can compute three additional types of distances.

Algorithm 3 claret::wmds kernel.

1: $gid \leftarrow get\_global\_id(0)$
2:
3: //copy data from global memory
4: $near\_set \leftarrow pivots[i][0 \ldots near\_set\_size]$
5: $random\_set \leftarrow randomindices$
6: $my\_pivots \leftarrow near\_set \cup random\_set$
7: $v_0 \leftarrow velocity[gid]$
8: $x_0 \leftarrow lowD[gid]$
9:
10: //compute force using pivot points
11: **for** $j = 0 \rightarrow pivot\_size$ **do**
12: $k \leftarrow my\_pivots[j]$
13: $hDistance[j] \leftarrow dist(highD[gid], highD[k], W)$
14: $lDistance[j] \leftarrow dist(lowD[gid], lowD[k])$
15: $\delta v \leftarrow v_0 - velocity[k]$
16: $force+ = f\big(hDistance[k] - lDistance[k], \delta v\big)$
17: sort my_pivots based on $hDistance$
18: globalSynchronization()
19:
20: //update velocity and position
21: $a \leftarrow g\big(force\big)$
22: $v \leftarrow v_0 + at$
23: $x \leftarrow x_0 + v \times t$
24: //copy data back to global memory
25: $velocity[gid] \leftarrow v$
26: $lowD[gid] \leftarrow x$
27: $pivots[gid] \leftarrow my\_pivots$
28: globalSynchronization()
29:
30: //Compute low dimensional distances
31: **for** $j = 0 \rightarrow pivot\_size$ **do**
32: $k \leftarrow my\_pivots[j]$
33: $lDistance[j] \leftarrow dist(lowD[gid], lowD[k])$

2.5.2 Interface Between Claret and Web Andromeda

We developed Claret as a portable and parallel tool that can facilitate fast MDS and WMDS computation using various hardware accelerators. We implemented Claret using OpenCL [126] (*C++* extension), and through our distribution, we provide a Python package for the end-users (developers). There is a *C++* version, as well.

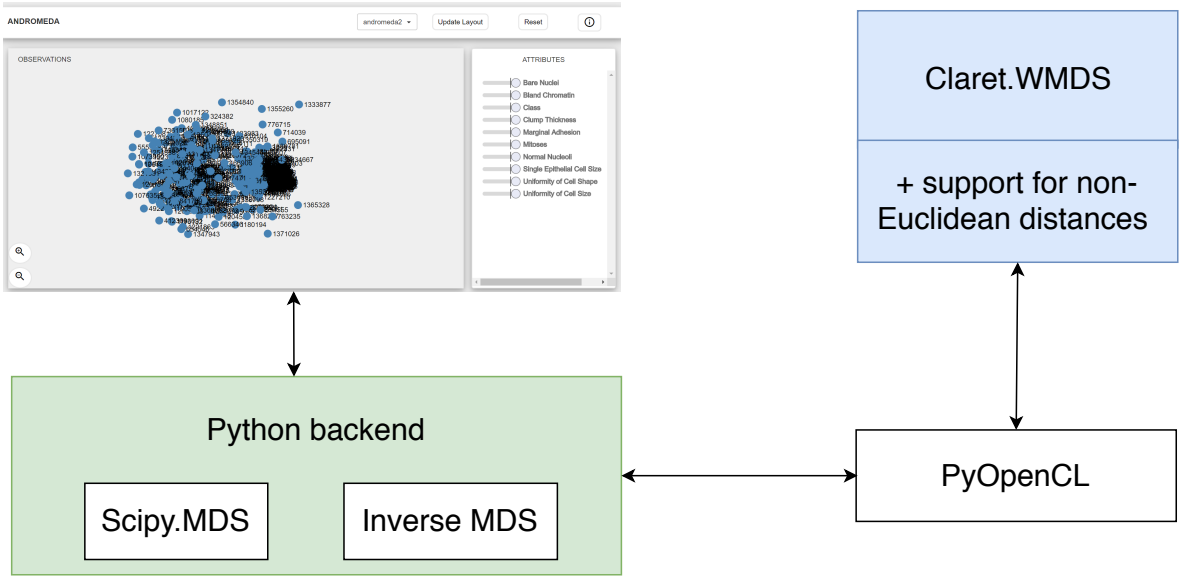


Figure 2.6: Integration of Claret.WMDS into Andromeda pipeline.

Web Andromeda has a Python backend, and it currently uses scipy [179]. MDS to project data onto 2D. It first computes all pairwise weighted Euclidean distances and then passes these metric distances to scipy's MDS routine. Claret MDS/WMDS has a host-side code in C++ and device-side code in OpenCL. We implemented a python host code using PyOpenCL [96] that can invoke the OpenCL device code (kernels). Then the Python backend of Andromeda can call Claret MDS/WMDS routines. In this implementation, we developed a Python distribution of Claret (*claret*) that has a module named *wmds* using PyOpenCL to interface between OpenCL and Python.

Dataset	size	claret::wmds time	scipy wmds time
Crescent tfidf.csv	41x275	0.29	0.11
Crescent topics	41x10	0.22	0.11
UK health	41x874	0.43	0.26
Animal data paper	13x13	0.25	0.06
Animal data square	13x13	0.22	0.06
Animal data study	49x72	0.24	0.21
Animal data transpose	72x49	0.26	0.25
STAT2004 subset noname	50x28	0.28	0.17
Patient data	22x91	0.25	0.07
UK health small	41x50	0.25	0.26
Automobiles	159x21	0.27	0.84
Breast cancer data	683x10	0.34	25.38
Congressional Voting	232x17	0.29	1.68
Shuttle turn corner (1K)	1000x9	0.44	65.11
Shuttle turn corner (2K)	2000x9	0.50	409.47
Shuttle turn corner (5K)	5000x9	0.68	-
Shuttle turn corner (43.5K)	43500x9	3.52	-
Supreme Court Ruling (20 topics)	14092x20	2.32	-
Supreme Court Ruling (100 topics)	14092x100	8.24	-

Table 2.2: Performance comparison between vanilla Andromeda and Andromeda with Claret. Using Claret is advantageous for big data.

2.5.3 Performance Comparison

Table 2.2 shows the performance comparison between Web Andromeda with scipy's MDS and Claret-embedded Andromeda. For small datasets, Claret does not help much with the speedup. For large datasets, Claret produces visualization within seconds while the sequential MDS cannot finish the computation in minutes to hours.

2.5.4 Case Study: Analyzing TCGA Mutation Data to Discover Cancer-Causing Genes Through the Incorporation of Domain Knowledge

The Cancer Genome Atlas (TCGA) [184] collects and hosts DNA sequence and gene expression data from tumor tissues and blood-derived normal tissues of thousands of cancer patients. By comparing the data from tumor samples against the normal samples, we can have insight into cancer-causing gene mutations and expression levels.

For this case study, we are particularly interested in DNA sequence data for breast cancer patients. The human genome has around 20000 genes, and any of these genes can host many mutations in its hundreds of locations. We choose each gene as a binary feature of the data, where a 0-value indicates this sample does not have a significant mutation in that gene, an 1-value indicates it does. For simplicity, we chose the top 1007 genes to use as features based on their ability to separate between tumor and normal samples. Previous studies have applied various mathematical, statistical, and machine learning models to discover these *driver* genes. Here, we use the Claret-enabled V2PI tool Andromeda to visually explore this 944×1007 (703 tumor samples, 241 normal samples) dataset to discover potential cancer-causing genes.

Visual to parametric interaction on breast cancer data from TCGA Multi-hit theory of carcinogenesis ([17, 19, 20, 109, 114, 173] suggests that combinations of $two - eight$ gene mutations cause cancer in humans. We want to exploit our knowledge of the samples regarding them being tumor or normal to discover the cancer-causing genes or the gene combinations. We name the tumor samples with a T-prefix and the normal samples with an N-prefix. We first project all the samples by assigning equal weights to all genes (Figure 2.7a). Different genes have different roles in various biological processes, so a central mass of projected data with no distinguishable pattern

is expected when all the genes are equally weighted.

Since we know which samples are tumor samples and which samples are normal, we can feed this knowledge to Andromeda through surface-level interaction. We separated 20 tumor samples and 20 normal samples from the concentrated mass randomly and moved these two groups far from each other (Figure 2.7b). Then we pressed the *Update Layout* button to request a V2PI interaction. Andromeda performed an inverse WMDS operation to parameterize the surface level visual interaction and then change the model parameters (i.e., weights of the features). As a result, we see that two genes (*ENSG00000149531, ENSG00000211896*) were significantly up-weighted compared to all other genes. This change in parameters also resulted in an updated visualization with 4 separate clusters, as seen in Figure 2.7c. Further examinations of these clusters might result in more insights.

Focusing on the two genes that got up-weighted simultaneously, we cross-referenced the role of these two genes and their combinations in causing breast cancer using the paper by [48]. This paper identified two-hit combinations of genes responsible for 17 cancer types. They reported eight two-hit combinations (Table 2.3) for breast cancer, and the combination (*ENSG00000149531, ENSG00000211896*) was reported as the second most frequent combination in tumor samples which is present in 38% of the tumor samples. This co-occurrence of the same combination in a more traditional cancer biology research and, in this case-study through V2PI interaction, suggests V2PI interactions have the potential in expediting scientific discoveries in various domains.

Time required for V2PI interactions Moving 40 points to two corners (Figure 2.7a) through visual interactions (dragging with mouse) was instantaneous and took nearly the time to decide on the data points to move. Parameterizing these interactions to update the feature weights and then creating the new visualization (Figure 2.7c) took $5 - 10$ seconds.

Rank	Gene 1	Gene 2	%Coverage
1	ENSG00000205277	ENSG00000184956	86%
2	ENSG00000149531	ENSG00000211896	38%
3	ENSG00000219481	ENSG00000173213	33%
4	ENSG00000185567	ENSG00000090512	9%
5	ENSG00000170471	ENSG00000205869	9%
6	ENSG00000178104	ENSG00000275113	5%
7	ENSG00000149531	ENSG00000084731	3%
8	ENSG00000137210	ENSG00000198888	1%

Table 2.3: Sample coverage by combinations for BRCA [48] .

2.6 Optimizing Inverse WMDS for Quantifying Visual to Parametric Interactions

Figure 2.1a shows the iterative process of *V2PI*. In step 1, the application creates a visualization using a mathematical model, M that depends on data D and model parameters Θ. The actual implementation uses weighted MDS as the dimension reduction method. A wrapper class on top of a readily available java implementation of MDS to facilitate the weighted version of MDS The tool presents visualization V to the user for interaction in step 2. The user interacts with the visualization in step 3 using standard interaction methods such as dragging the points further or closer, highlighting some points, etc. In step 4, user interaction is parameterized and fed back to the mathematical model.

We are interested in step 4 since it is the most time-consuming part. Converting visual interaction to update the parameters of the model requires solving an optimization problem in an ample search space From Section 2.6.2 we need to find a combination of weights for all the dimensions that minimize the stress function. We elaborate on the stress equation to give a comprehensive description of the inverse problem.

When the user interacts with the displayed visualization V and provides visual feedback in terms of a new visualization V', the application needs to convert the input into the parametric form. We can formulate this problem as, given high-dimensional data points H and corresponding *2D* projection, we need to find a d-dimensional weight vector that minimizes the stress described by equation 2.2.

Since H and L are given, searching for an optimal w that minimizes stress necessitates searching in a d-dimensional search-space. In its simplistic version, the tool runs the optimization for a predefined number of iterations. At every iteration, the current **W** is perturbed along one dimension after another. If the perturbed **W** yields lower stress, the perturbation is used in the next iteration. Otherwise, the original **W** is restored. At the end of the predefined number of iterations, the **W** that minimizes the stress function is retained and is fed back to the model.

Algorithm 5 describes the process of searching an optimal parameter to specify the user's visual interaction best.

Algorithm 5 Optimizing stress function.

```
1: Input: H, L
2: for i = 1 → maxIteration do
3:     for j = 1 → d do
4:         currentStress = computeStress(W)
5:         W' = changeWeight(j)
6:         newStress = computeStress(W')
7:         if newStress ≤ currentStress then
8:             accept the change
9:             W = W'
10:        else
11:            reject the change
```

Changing the weight of a particular dimension is tricky. Just choosing a random value is not enough, other dimensions' weights also need to be normalized. For the weight vector $\left[w_1, w_2, \ldots, w_p\right]^T$, we maintain that the vector is normalized at any phase of the algorithm, that is $\sum_{k=1}^{d} w_k = 1$ To maintain this invariant, whenever $k'th$ dimension's

value is changed to w'_k from w_k, the weight vector is normalized by dividing each dimension's weight by a normalizing factor $N = w_1 + w_2 + \cdots + w'_k + \cdots + w_d$, thereby changing W to $W' = \left[w_1/N, w_2/N, \ldots, w_k/N, \ldots, w_d/N\right]^T$.

With the help of this formula, the algorithm for changing a particular dimension weight is stated below:

Algorithm 6 Changing a dimension's weight.

```
1: Input: W, k
2: W' ← W
3: w'_d ← rand()
4: normalizationFactor ← 0
5: for i = 1 → d do
6:     normalizationFactor+ = w_i
7: for i = 1 → d do
8:     W'[i] ← W'[i]/normalizationFactor
```

From equation 2.2, computing stress is supposedly the most time-consuming step of this optimization process. The following algorithm shows how stress is computed.

Algorithm 7 Computing Stress.

```
Input: H, L, W
stress ← 0
for i = 1 → n do
    for j = i + 1 → n do
        weightedHighDistance ← 0
        for k = 1 → d do
            weightedHighDistance+ = (h_{i,k} − h_{j,k})^2 × w_k
return stress
```

2.6.1 Runtime Analysis

To analyze the runtime performance of backward computation, we need to identify the parameters determining the performance. Let, *maxIteration* = I. From algorithm 5, the outermost loop runs I times, and the inner loop runs p times To fully account for the runtime, now we need to look into algorithm 7 and algorithm 6, which is a step in

algorithm 1 There are three nested loops they run in n, $O(n)$, and d times respectively. So, the three parameters that determine the runtime are I, n, and d.

First we determine the runtime of algorithm 7 that computes stress. Outer loop runs n times and the inner loop runs $O(n)$ times. Innermost loop runs d times So, the total runtime for algorithm 7 comes to $O(n \times n \times d) = O(n^2 d)$. Changing weight in algorithm 5 is performed by algorithm 6. This algorithm runs in $O(d)$ time. Since we know the runtime for *computeStress* algorithm of algorithm 5, and *changeWeight* in line 4, we can complete our analysis for algorithm 5. The outermost loop runs I times. The inner loop starting at line 2 runs p times. Line 3 and 6-11 takes $O(1)$ time. Line 4 takes $O(d)$ time. Line 5 takes $O(n^2 d)$ times. So, the total runtime for algorithm 1 is $O(I \times p \times (O(1) + O(d) + O(n^2 d)) = O(I n^2 d^2)$.

So, the runtime performance of this algorithm is linear in the number of iterations and quadratic in both n and d. By keeping the number of iterations fixed, we concentrate on time complexity in terms of n and d.

2.6.2 Optimizing Runtime for Algorithm Parameters

Three algorithm parameters determine the runtime, the number of iterations I, the number of data points n, and the dimension of data points d. The baseline implementation deals with a fixed value for I, and its typical value is 500 to 1000. We don't focus on this parameter since our goal is not to come up with a different algorithm. Rather we want to produce the same result as the original implementation much faster. So, we focus on reducing runtime's dependency on parameters n and d by bringing it down from quadratic to linear.

Optimization for d The bottleneck for algorithm 1 is its stress computation part. Computing stress is a $O(n^2 d)$ operation. Computing stress is essentially a three-step

operation.

1. Compute pairwise distances between n points in $2D$.
2. Compute the pairwise weighted distance between n points in high dimensional space with the changed weight vector.
3. For each pair, take the absolute difference between their distances in high dimensional space and 2D space. The summation of these differences is stress.

For a given **H**, **L** pairwise distances between points in $2D$ is the same for all the iteration. So, this can be done just once, even before entering the outermost loop of algorithm 1 However, step 2 needs to be done in each iteration of the inner loop since the weight of a particular dimension is changing, and the normalization step effectively changes all the weights. Thus, this step affects all the pairwise distances. We can compute the weighted distance between two points in high dimensional space using equation 2.5

$$HD(i,j) = \sqrt{(H_{i,1} - H_{j,1})^2 \times w_1 + \cdots + (H_{i,d} - H_{j,d})^2 \times w_d} \tag{2.5}$$

For each of these $I \times d$ iterations, we have to recompute n^2 pairwise weighted distance, and computing one distance takes $O(d)$. Distance computation makes stress computation a $O(n^d)$ operation. Since a change in one dimension's weight affects all n^2 distances, we cannot do anything here in terms of n. We observed that though the weighted distances between two points in two successive iterations are different, they only differ in only one component, as seen in equation 2.5 To strengthen the case, we do some algebraic re-arrangement.

Let, weighted distance between points i and j in high dimensional space is $HD(i,j)$ at the end of current iteration. In the next iteration d^{th} dimension's weight is changed from w_k to w'_k. This changes $W = \left[w_1, w_2, \ldots, w_k, \ldots, w_d\right]^T$ to $W' = \left[w_1/N, w_2/N, \ldots, w_k/N, \ldots, w_d/N\right]^T$.

Changing weight is essentially picking a random weight for a given dimension. The rate at which a change in a dimension takes us to convergence determines how fast we need to change its value. we observed we could recompute the weighted distance between two points in constant time instead of $O(d)$ time if the value for the previous iteration is known. If we maintain a $2D$ $n \times n$ array of the weighted distances between points in high dimensional space, the sequential nature of two outermost loops guarantees that for every iteration, the value from the previous iteration is known. With this

observation, we claim that step 2 can be performed in $O(n^2)$ time instead of $O(n^2 d)$ time. We can perform step 3 in $O(n^2)$ time; this step can also be merged with step 2 As soon as one pairwise distance is recomputed, the difference from its lower dimensional counterpart can be computed and added to the stress.

The modified version of algorithm 5 will require *computeStress* to be replaced by *updateStress*. Algorithm 3 will provide the changed weight for the current dimension to be perturbed. The new *updateStress* algorithm will take the dimension index, and it's changed weight to compute new stress using the pairwise weighted distance between points in high dimensional space.

Algorithm 8 Updating stress.

Input: **HD**, **LD**, k, w_k, W'
$stress \leftarrow 0$
$normFactor \leftarrow 0$
for $i = 1 \rightarrow d$ **do**
    $normFactor+ = w_i$
    $normFactor+ = (w'_k - w_k)$
for $i = 1 \rightarrow n$ **do**
    for $j = i + 1 \rightarrow n$ **do**
        $newContribution \leftarrow (H_{i,k} - H_{j,k})^2 \times w'_k$
        $oldContribution \leftarrow (H_{i,k} - H_{j,k})^2 \times w_k$
        $currentValue \leftarrow HD^2_{i,j}$
        $HD_{i,j} \leftarrow \sqrt{\frac{currentValue - oldContribution + newContribution}{normFactor}}$
        $stress+ = |HD_{i,j} - LD_{i,j}|$
return *stress*

Optimization for n Equation 2.2 dictates that computing stress requires $O(n^2)$ operations, so there is no way reducing runtime for n while computing stress, so we should try to reduce the number of times we need to compute stress.

The purpose of this application is to help users to discover knowledge by transferring their domain-knowledge through visual interaction. We can consider two extremes. In one extreme, the user knows inter-relation between all the data points. In that case, the

user can move all data points in the $2D$ projection in one single pass. The application does not need to recompute the visualization since all the projected points are already in the correct place. In another extreme, the user knows nothing about the dataset. So, there is no way she can give any meaningful feedback to the initial visualization. It's safe to assume, user's knowledge would not be in any of these extremes, rather her expertise regarding the dataset would be partial, so at any pass, the user will only move a fraction of the data points. Besides, in the case of big data, it's not practical to interact will all the points.

As we have seen, moving a fraction of data points is inherently confusing. While the relative positions between moved points transfer user's knowledge about similarity/dissimilarity between those points, it does not say these points' relation with unmoved points. For example, in Figure 2.8, when point a is moved away from unmoved point b and close to another moved point c and unmoved point d. We can construe this movement in any of the following ways.

1. a is similar to c
2. a is similar to c and dissimilar to b
3. a is similar to c and similar to d
4. a is similar to c and d, and dissimilar to b

Each of these options offers a different kind of knowledge, and $V2PI$ does not have a way to decide conclusively which one of them the user intended. To address this problem, *Semantic Interaction* paper introduces highlighting options. That means, if the user wants to convey any knowledge about the unmoved points, she needs to highlight those points. With this highlighting options, now we know with certainty which points to consider for translating user knowledge into model parameters. Let f be the fraction of the points the user touches (moves and/or highlights). Only the touched points are

This process can be illustrated using following diagram.

Let, X be an element-wise binary operator that multiplies elements from two matrices at the same index and then collects the nonzero elements into a result matrix. The final stress is the summation of the components of that result matrix.

2.7 Conclusion and Discussion

In this chapter, we presented Claret, a tool for weighted MDS implemented in OpenCL, which outperforms Glimmer, the previously best performing method designed for GPUs. We show that Claret is indeed a write-once-run-anywhere tool and can run on a plethora of devices. We also presented a geometric result claiming that weighted Euclidean distances can be preserved through *stretched*-random projection. The proposed quantification of the quality of an embedding has the potential to make visual analytics consistent. We have successfully demonstrated Claret's portability across various accelerators.

Interactive visualization on big data is challenging due to limited computational re-

sources and a lack of parallel implementations of dimension reduction algorithms. We have extended a portable MDS/WMDS tool to be useful for dimension reduction and integrated it with a V2PI tool, Web Andromeda. The resulting speedup enables users to use Andromeda interactively without prohibitive computational latency. Through algebraic optimization and reusing incremental computation, we reduce the optimization task of parameterizing the visual interaction from $O(n^2d^2)$ to $O(nd)$. With the Python distribution, the extended Claret can be useful in other interactive visualization tools too.

Through the case study, we demonstrated that V2PI interactions, along with domain knowledge, can make scientific discoveries in various domains such as cancer biology. By separating a small number of tumor samples from a small number of normal samples through V2PI interaction, we could detect two potential carcinogenic genes that can be validated through other existing literature. The computational speed to make these interactions are significantly faster to enable a user to discover knowledge through visual interactions.

However, despite the computation speed, analyzing big data using the Andromeda interface presents a new problem: projecting thousand or more data points on a small screen makes it hard to differentiate between the data points. To reduce the user's cognitive overhead and create a tidier visualization, we can choose to show a smaller number of representative data points through various summarization techniques such as micro-clustering, coreset computation, and sampling.

Integration of accelerated forward WMDS from extended Claret, algebraic optimization, and incremental gradient computation in inverse WMDS into the Andromeda software ecosystem enabled the faster performance of V2PI interactions on big data, thereby facilitating scientific discoveries from various domain problems. By further modifying the visual interface and incorporating domain knowledge, this accelerated

V2PI interactions can play a vital role in knowledge discovery from a big and complex dataset.

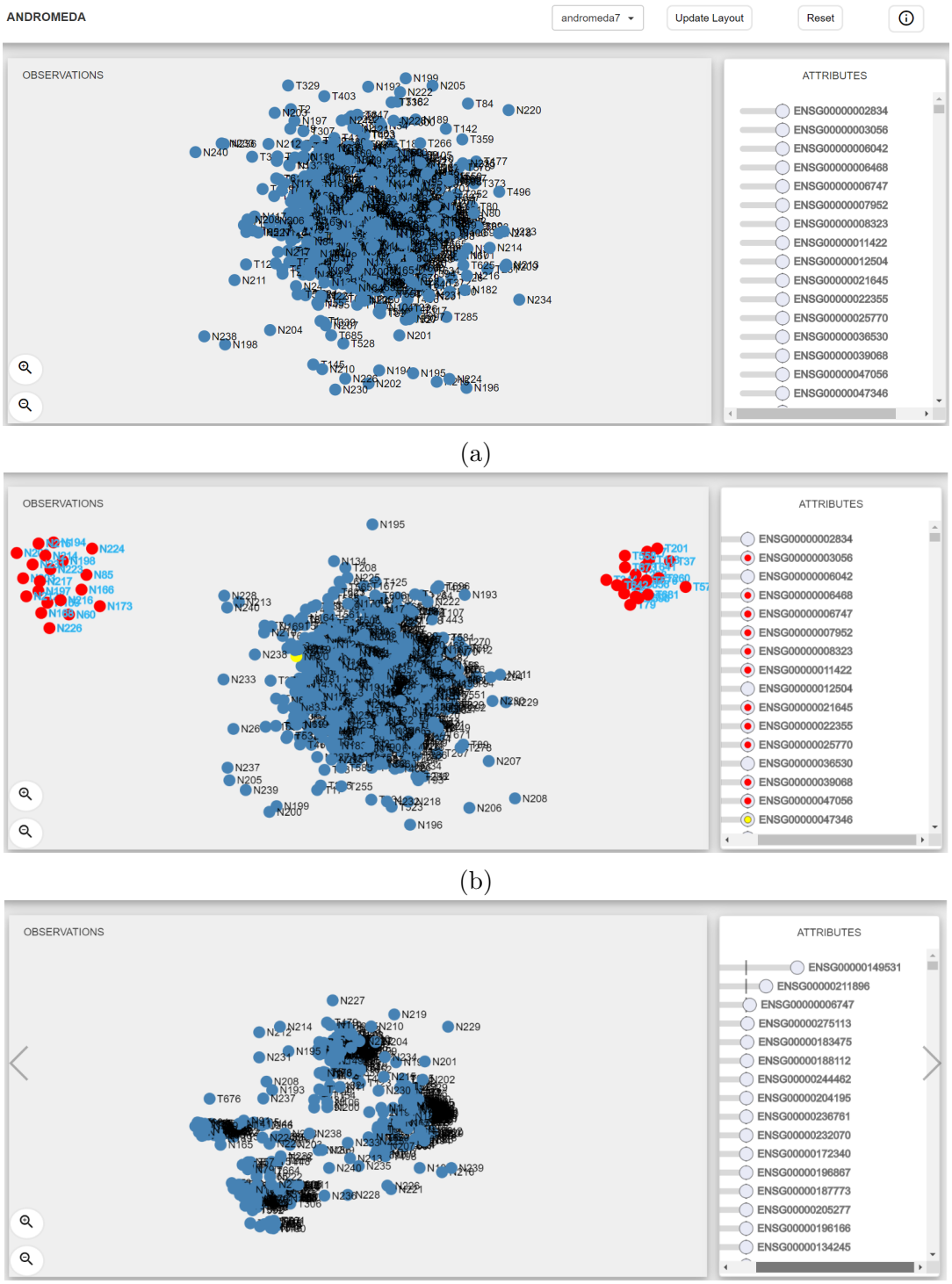


Figure 2.7: (a) Initial projection on *2D* space. All tumor and normal samples are projected together in a central mass. This projection suggests some genes might have more of a role than others in separating tumor samples from normal samples. (b) We moved 20 tumor samples and 20 normal samples from the center mass in opposite directions to tell Andromeda that tumor samples are very dissimilar from normal samples. (c) After moving the two sets of samples further from each other, we pressed the *Update Layout* button, which resulted in a conversion of the visual interaction to parametric interaction using *inverse* WMDS. The value of the feature weights changed, and a new visualization was created using *forward* WMDS.

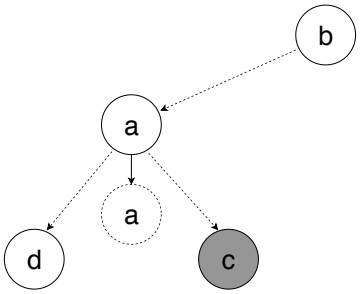


Figure 2.8: Point a is being moved from its original position (solid ball) to a new position (dashed ball). a is moved away from unmoved point b towards moved point c (gray ball) and unmoved point d (white ball). This movement can be interpreted in many ways in terms of pairwise similarity and dissimilarity.

Chapter 3

Domain-Aware Algorithm Design to Identify Carcinogenic Gene Combinations

3.1 Introduction

In this chapter, we demonstrate the second guideline related to variety through the solving process of a problem from the domain of cancer biology. Experimental studies and mathematical models suggest that carcinogenesis is likely a result of different combinations of a small number of carcinogenic mutations (hits) [19, 20, 109, 114, 130, 173, 195]. Mathematical models estimate that the number of such hits varies from two to eight [15, 19, 20, 109, 114, 130, 173, 195]. Yet, our collective computational and experimental efforts and the accumulation of cancer genomic data have failed to identify, for most cancers, the specific combinations of mutations triggering carcinogenesis. There are a large number of genes (20000) and hundreds of mutations in each of these genes. This presents us with a solution space of $\binom{20000}{h}$ candidate combinations and $2^{\binom{20000}{h}}$ candidate collections of combinations. Since, $h \in [2, 9]$, these numbers rise exponentially with increasing value of h.

Current computational efforts to find carcinogenic mutations generally focus on identifying individual "driver mutations", based on mutational frequency and signatures

[52, 100, 168, 172]. These driver mutations have been shown to be associated with an increased risk of cancer. However, they can not generally cause cancer by themselves. For example, 72% of women with an inherited BRCA1 mutation are likely to get cancer by age 80. However, even for women with the BRCA1 mutation, none are likely to get cancer before age 20, and 28% of them may never get cancer [99]. The Li Fraumeni syndrome is another example where germline P53 mutations is associated with early onset cancer predisposition (e.g. soft tissue and bone sarcomas). However, cancer penetrance is less than 20% for children while approaching 80% by age 70, indicating that multiple hits are required for carcinogenesis [13, 71, 117, 136].

The relationship between most other known genetic markers and increased cancer risk is far weaker [70, 94]. The limited early cancer incidence in individuals with germline mutations suggests that additional genetic defects acquired over an individual's lifetime are necessary for carcinogenesis. Therefore, current computational approaches focused on identifying individual genes that are cancer drivers, cannot find the specific combinations of mutations responsible for individual instances of cancer. Several factors, other than genetic mutations, have also been implicated in carcinogenesis, such as epigenetic modifications [167], tumor environment[157], and adaptive evolution [11]. However, carcinogenesis is primarily a result of genetic mutations [180].

The goal of this work is to develop a method for identifying combinations of genetic mutations that are most likely responsible for individual instances of cancer. This goal is fundamentally different from identifying the most frequent driver mutations, and represents the first computational study to specifically identify multi-hit combinations.

Our approach consists of first identifying likely combinations of genes with carcinogenic mutations. We then present a method, based on the mutational profile of these genes, for identifying likely carcinogenic mutations within these genes. Although it is theoretically possible to search for combinations of individual mutations using our

method, the problem becomes computationally intractable, since most genes contain hundreds of somatic mutations. In addition, in the much larger set of somatic mutation combinations many carcinogenic combinations will be rarely represented, further increasing the challenge of identifying these combinations. Therefore, we chose to first identify combinations of genes with somatic mutations, and then present an approach for identifying likely carcinogenic mutations within these genes. We also scale up and scale out our approach to identify combinations of length more than two.

3.1.1 Domain-Aware Algorithm Design

We mapped the problem of finding these combinations to the extensively studied weighted set cover (WSC) problem [37]. Finding the optimal solution to the corresponding WSC problem is computationally intractable due to the exponentially large number of possible sets of multi-hit combinations. However, there exist approximation algorithms for finding near-optimal solutions [37, 59]. We adapted one such algorithm to find a set multi-hit combinations that maximize the number of tumor samples that contain one of the multi-hit combinations while minimizing the number of normal samples that contain any of the combinations. The number of candidate set covers is an exponentially large quantity due to the large number of combinations.

We applied the above algorithm to find a set of two-hit combinations using somatic mutation data from the cancer genome atlas (TCGA). For the 17 cancer types with at least 200 matched tumor and blood-derived normal samples in TCGA, the algorithm identified a set of 197 2-hit combinations. For a separate set of Test samples, these combinations were able to differentiate between tumor and normal samples with 91% sensitivity (95% Confidence Interval (CI)=89-92%) and 93% specificity (95% CI=91-94%) on average, for the 17 cancer types. The results are consistent across different randomly selected Training and Test sets. Despite this high accuracy, our analysis

of the results shows that many of the two-hit combinations are likely to be two-gene subsets of three or more-gene combinations. So, the need for identifying combinations of higher length motivates us in finding a scalable solution to this problem.

Identifying gene combinations is important for two reasons. First, it brings us closer to the understanding of carcinogenesis and the complexity of cancer biology. Second, the identification of the specific combination responsible for a given instance of cancer can help us design more effective combination therapies for treating the disease. Combination therapies can be more effective than single target treatments; however, most current therapeutic combinations have been based on trial and error [10, 105]. Identifying the precise combination of genomic anomalies responsible for individual instances of cancer provides a more rational basis for designing combination therapies.

3.1.2 Parallelization of the Approximate Algorithm to Identify Three- and Four-Hit Combinations

The computational complexity of the approximate algorithm has a runtime complexity of $O(G^h \times C \times (N_t + N_n))$, which limits the combinations that can be practically identified to two-hit ($h = 2$) combinations, where $G \approx 20000$ is the number of genes with mutations in the input data, C is the number of combinations identified by the algorithm, N_t is the number of input tumor samples, N_n is the number of input normal samples, and h is the number of hits. For example, it took 39 minutes to calculate a set of two-hit combinations for breast cancer (BRCA) using 911 tumor samples from the cancer genome atlas (TCGA).[1] We estimate (as described in Section 3.5) that it will take 253 days to calculate a set of three-hit ($h = 3$) combinations for BRCA, without any additional optimization or parallelization. We aim to optimize the multi-

[1] Algorithm was run on an Intel Xeon E5-2630 2.1 GHz central processing unit (CPU) with 256 GB memory.

hit algorithm to identify combinations of more than two hits in a practical time frame ($<$ 1 month).

Achieving this level of speedup requires parallel execution across a large number of processors. Graphical processing units (GPUs) with thousands of processors are a natural choice for massively parallel processing [4]. However, GPUs have three key limitations that must be addressed to achieve significant speedup. (1) Speed of memory access is significantly slower on GPUs compared to CPUs, e.g. on the Intel Xeon E5-2630 CPU L1 and L2 cache access require 4 and 11 cycles respectively [84], compared to 28 and 193 cycles for the NVIDIA V100 GPU [86]. Therefore, speedup from parallelization will be offset by slower memory access for algorithms that require access to a large amount of data from memory. (2) GPUs have limited amount of accessible memory, e.g. 32GB for the NVIDIA V100, compared to 1.5TB for Intel Xeon E5-2630 [82]. (3) On NVIDIA GPUs, divergent branching during execution will result in unbalanced processor load, which also limits the achievable speedup from parallelization [16, 131, 153, 154, 156]. To address these GPU limitations, we employed two general strategies. (1) We used a compressed binary representation for the Gene-Sample Mutation matrix (described in Methods), which reduced memory requirement by 16-fold and resulted in an average 10 fold speedup (see Results). (2) We restructured and optimized the algorithm for parallel execution on a NVIDIA Tesla V100 PCIe graphical processing unit by minimizing divergent branching in addition to other optimizations described in the Methods section.

The compressed binary representation alone resulted in a 0.4–18 fold speedup for the two-hit algorithm, compared to the original integer matrix, depending on cancer type. This additional speedup, and the associated increase in software complexity, was not necessary for the identification of two-hit combinations, and insufficient by itself for the identification of three-hit combinations on the CPU. However, the optimized GPU implementation combined with the compressed binary representation was 0.7–224 times

faster than the original CPU based integer matrix implementation, for the two-hit algorithm, depending on cancer type. The three-hit algorithm was an estimated 29–33,690 times faster for the optimized GPU implementation compared to the original CPU implementation. For the breast cancer samples mentioned above, we were able to compute a set of three-hit combinations in 23 minutes with the optimized GPU implementation compared to the estimated 253 days for the original CPU implementation.

The set of three-hit combinations identified using a randomly partitioned training set was able to differentiate between tumor and normal samples in separate test data with overall sensitivity of 90% (95% confidence interval (CI) = 88–91%) and overall specificity of 93% (95% CI = 92–94%). Despite this relatively high accuracy, the multi-hit gene combinations identified by our algorithm may not represent cancer genes (see Discussion). Further experimental validation will be required to determine if mutations within these genes may play a role in cancer genesis or progression.

3.1.3 Scaling Out the Algorithm Using Hundreds of GPUs

The computational complexity of the algorithm scales exponentially with the number of hits, i.e. G^h, where $G \approx 20000$ is the number of genes and h is the number of hits. This exponential scaling limits the number of hits that can be practically identified to three-hit combinations, for sample size greater than 200, even with parallelization across thousands of processors on an NVIDIA V100 graphical processing unit (GPU) [9]. For example, the identification of two-hit combinations for breast cancer with 911 tumor samples, took 143 seconds on a single Intel Xeon E5-2630 CPU and 21 seconds on a single NVIDIA V100 GPU. The identification of three-hit combinations took 13860 minutes on a single CPU and 23 minutes on a single GPU [9]. We estimate that the identification of four-hit combinations would require over 500 years on a single CPU and over 40 days on a single GPU.

Previous studies have shown that most cancers require an estimated four – nine hits [17, 19, 20, 109, 114]. Therefore, two- and three-hit combinations will not be able to identify the specific combination of gene mutations responsible for individual instances of some cancers, which is the ultimate goal of this approach. Pathways affected by these specific genes can then be targeted by combination therapy for a more effective treatment. An added benefit is that, the additional genes in the combination offer additional therapeutic targets in cases where therapies do not exist for one or more of the genes.

To be able to identify combinations of more than three-hits, we restructured and optimized the algorithm for parallel execution across multiple GPUs on multiple nodes of the Summit supercomputer at the Oak Ridge National Laboratory.

Efficiently scaling the algorithm for parallel execution across hundreds of nodes and GPUs presented three key challenges: balancing the load across tens of thousands of processors, minimizing the use of slow global memory, and inter-processor and inter-node communication.

Outline of This Chapter

In what follows we present our approach for finding genes with mutations responsible for cancer. In Section 3.2 we describe the mapping of the problem to the weighted set cover (WSC) problem and the WSC approximation algorithm used to identify the multi-hit combinations. In Section 3.3, we show that our approach can identify a set of multi-hit combinations that can differentiate between tumor tissue and normal tissue samples with over 90% sensitivity and specificity. This result is robust to different randomly selected training and test sets. We discuss how these combinations can be used to distinguish carcinogenic and non-carcinogenic mutations within genes and to design targeted combination therapies.

Next, in Section 3.4, we describe the parallelization of the approximate algorithm, the compressed binary representation of the input matrix, the mapping of the algorithm to the GPU, and its optimization for parallel execution. In Section 3.5, we describe the speedup achieved by the optimized parallel implementation, the breakdown of the contribution of different optimizations, and the accuracy of the multi-hit combinations identified.

In Section 3.6, we describe our approaches for addressing challenges related to load-balancing across hundreds of GPUs and global memory access latency. This approach resulted in an average 455-fold speedup using 600 V100 GPUs (100 nodes), compared to the execution on a single GPU, for identifying four-hit combinations. We present our result on scaling efficiency and classification performance in Section 3.7.

In Section 3.8, we discuss how carcinogenic and non-carcinogenic mutations within the gene combinations can be distinguished. We also discuss how the multi-hit combinations can be used to develop targeted combination therapy. We also discuss various approaches to scale our algorithm further to identify multi-hit combinations beyond four-hits.

3.2 Mapping the Problem to Weighted Set Cover (WSC) Problem

Our approach for identifying sets of multi-hit combinations consists of two steps. (Figure 3.1). First, we identified somatic mutations from whole exome sequencing data for tumor and normal tissues with matched blood-derived normal samples from The Cancer Genome Atlas (TCGA). Somatic variants called from matched tumor tissue and blood-derived normal samples can detect low-frequency variants, which would not be detected when using tumor samples alone. Second, we use a weighted set cover

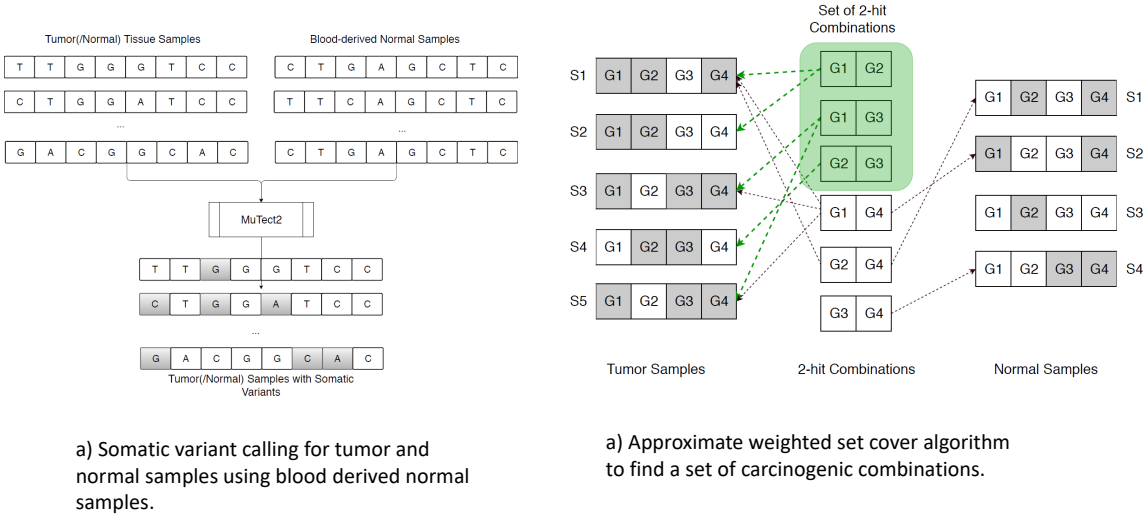


Figure 3.1: Approach for identifying multi-hit combinations. (a) Whole exome sequencing data from The Cancer Genome Atlas (TCGA) for tumor samples and normal tissue samples with matched blood-derived normal samples were used to identify somatic mutations. Somatic mutations were calculated using the Mutect2 variant caller and the Variant Effect Predictor (VEP). (b) The problem of identifying multi-hit combinations is mapped to the weighted set cover (WSC) problem. An approximate WSC algorithm was used to identify a set of multi-hit combinations that was able to differentiate between an independent set of tumor and normal tissue samples with over 90% sensitivity and specificity.

algorithm to identify multi-hit combinations that can differentiate between tumor and normal samples with high sensitivity and specificity. The problem of identifying a set of multi-hit combinations is computationally intractable; however, there exist algorithms for finding a near-optimal approximate solution. We used a variant of one such algorithm to identify a set of multi-hit combinations for each cancer type, using a randomly selected subset of the available tumor and normal tissue samples (the Training set). The accuracy (sensitivity and specificity) of the resulting multi-hit combinations was evaluated using the remaining tumor and normal tissue samples (the Test set).

3.2.1 Somatic Mutations Calculated from the Cancer Genome Atlas (TCGA) Data

The primary input to our algorithm is somatic mutation data for tumor and normal tissue samples. TCGA contains a set of such data for tumor tissue samples with matched blood-derived normal samples, in mutation annotation format (MAF) datasets [185]. These somatic mutations were identified using the commonly used and well documented Mutect2 software. For normal tissue samples we identified a set of 333 normal tissue samples with matched blood-derived normal samples. We calculated somatic mutations for these normal tissue samples using the same Mutect2 protocol used for the tumor tissue samples. We use the Variant Effect Predictor (VEP) to determine the location (intron, exon, UTR) and effect of these variants (synonymous, non-synonymous, missense, nonsense). The specific commands and parameters used are included in Supporting Information (SI). In our analysis we only consider protein-altering variants (non-synonymous, nonsense, and insertion/deletions in exons), as predicted by VEP. We found 6733 tumor samples with $\sim 10^7$ pre-calculated protein-altering somatic variants in the MAF files for the 17 cancer types with at least 200 matched tumor and blood-derived normal samples. In addition, we found 333 matched normal tissue samples in TCGA, in which we identified $\sim 10^6$ protein-altering somatic mutations using the Mutect2/VEP protocol detailed in SI.

The algorithm presented below is based on the somatic mutation data described above, which does not include possible germline mutations that may contribute to carcinogenesis. However, carcinogenic germline mutations are in general relatively rare. For example, BRCA1 is one such rare exception where it occurs as a germline mutation in $5-10\%$ of breast and ovarian cancer patients with a BRCA1 mutation [23, 42]. However, the other $90-95\%$ of cases with the BRCA1 mutations are somatic variants. Therefore, the following algorithm should still be able to identify mutations in such

genes as carcinogenic, although the possible presence of germline mutations may limit the accuracy of the algorithm.

3.2.2 Mapping the Problem of Finding Multi-Hit Combinations to a Weighted Set Cover (WSC) Problem

Our goal is to identify a set of multi-hit combinations of gene mutations, such that at least one combination occurs in each tumor sample while minimizing the number of normal samples containing any of the combinations. Identifying this set of carcinogenic multi-hit combinations can be mapped to the extensively studied weighted set cover (WSC) problem. The WSC problem can be described as follows. For a universal set of elements and a collection of weighted subsets of this universal set, find a minimum weight collection of subsets such that all elements of the universal set are covered. The problem of identifying a set of multi-hit combinations that optimally differentiates between tumor and normal samples can be mapped to the WSC problem as follows.

1. Let, $T = \{t_1, t_2, \ldots, t_{N_t}\}$ be a set of N_t tumor samples, and $N = \{n_1, n_2, \ldots, n_{N_n}\}$ be a set of N_n normal samples. We consider T as the universal set in the WSC problem. N will be used in computing weights.

2. Let $C = \{c_1, c_2, \ldots, c_M\}$ be a set of M possible combinations. We construct a subset for each of these combinations by taking the tumor samples containing that combination. T^{c_i} represents the subset associated with combination c_i, i.e. $T^{c_i} = \{t_1^{c_i}, t_2^{c_i}, \ldots\}$, where all tumor samples in T^{c_i} contain the combination c_i. Union of all the subsets T^{c_i} constructs the universal set T.

3. Assign a weight w_i to each combination c_i (subset T^{c_i} in the WSC problem) such that the weight represents the inverse likelihood of the combination being

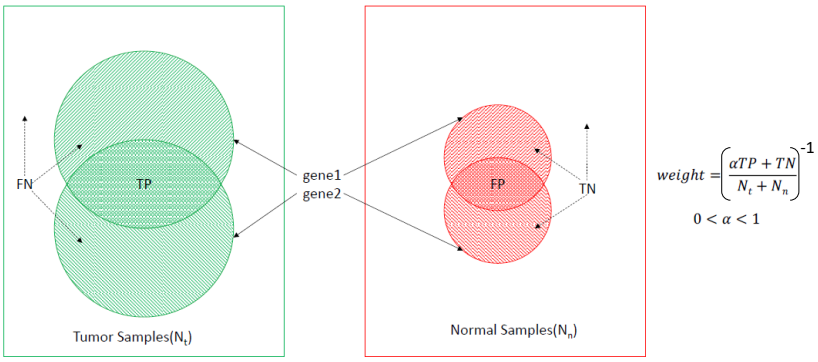


Figure 3.2: Weight computation for a combination of two genes $\langle gene1, gene2 \rangle$. Tumor samples covered by both genes are true positives (TP), tumor samples not covered by one or both genes are false negatives (FN), normal samples covered by both genes are false positives (FP), and normal samples not covered by one or both genes are true negatives (TN). The scaling factor α is used to balance the relative importance of sensitivity and specificity.

carcinogenic. w_i is described below. Combinations with lower weights have higher likelihood to be carcinogenic.

4. Find a set of combinations $C^* = \{c_1^*, c_2^*, \dots\}$ such that all the samples in T are covered and the total weight $W = \sum w_j^*$ is minimized .

The goal of the algorithm is to maximize sensitivity TP/N_t and specificity TN/N_n, where TP is the number of true positives, TN is the number of true negatives, N_t is the number of tumor samples, N_n is the number of normal samples (Fig. 3.2). Therefore, we assign a weight to each combination as the inverse of the accuracy metric, $w_i = \left(\frac{\alpha TP+TN}{N_t+N_n}\right)^{-1}$, where $0 \leq \alpha \leq 1$ is a scaling factor. The scaling factor is used to balance the optimization of sensitivity and specificity simultaneously. We use the scaling factor 0.1 to reflect the fact that the WSC solution for the Training set always has a true positive rate of 1.0, i.e. every tumor sample in the Training set contains at least one combination.

3.2.3 Algorithm for Finding an Approximate Solution to the Weighted Set Cover Problem

The computational complexity for finding an optimal solution to the WSC problem scales exponentially with problem size, making it computationally intractable. For the problem of finding a set of multi-hit combinations, let $G = 20000$ be the number of genes and $h = 8$ be the maximum number of hits. Then, the number of possible combinations $M = \sum_{c=2}^{h} \binom{G}{c} \approx 6 \times 10^{29}$. The number of possible subsets of these combinations is 2^M. The optimal solution would be a subset of combinations with the minimum weight. Though a brute-force search could find the optimal solution, the size of the search-space makes the task computationally impossible. However, many approximate algorithms have been developed and analyzed for solving set cover and weighted set cover problems. The algorithm iterates through the following three steps until all tumor samples have been excluded, as illustrated in Fig. 3.3.

1. Compute a weighted accuracy metric F_i for all $i = [1, H]$ possible h-hit combinations, where H is the number of possible combinations. F_i is a combined measure of the specificity and sensitivity with which each combination can differentiate between tumor and normal samples in a training set.

$$F_i = \frac{\alpha TP_i + TN_i}{N_t + N_n} \tag{3.1}$$

   where, for a given combination i, TP_i is the number of true positives (tumor samples with mutations in the gene combination i), TN_i is the number of true negatives (normal samples without mutations in the gene combination i), N_t is the total number of tumor samples, N_n is the total number of normal samples and $\alpha = 0.1$ is a weighting factor to balance the contribution of sensitivity and specificity to the metric.

2. Select the combination of hits with the maximum F_i value, and add it to the list of selected multi-hit combinations.

3. Exclude all tumor samples that contain mutations in this combination of genes, from further consideration.

The computational complexity of the approximate algorithm is $O(G^h \times C \times (N_t + N_n))$ where G is the number of genes and C is the number of combinations selected. The input to the algorithm are two Gene-Sample Mutation matrices, a tumor mutation matrix $(M^t_{ij}) \in \{0,1\}^{G \times N_t}$ and a normal mutation matrix $(M^n_{ij}) \in \{0,1\}^{G \times N_n}$. Non zero values in these binary matrices represent mutations in gene g_i, $i = [1, G]$ within sample s_j, $j = [1, N_t]$ for tumor samples and $j = [1, N_n]$ for normal samples (Fig. 3.3). In addition, these are sparse matrices with only 2% of the elements having a non-zero values. To take advantage of these characteristics of the input matrices, we considered two possible alternatives to the matrix representation: indexed array and compressed binary representations, as described below.

3.3 Classification Performance and Quality of Identified Two-Hit Gene Combinations

We implemented a weighted set cover algorithm to identify 2-hit combinations of cancer causing genes with mutations using a randomly selected Training set of tumor and normal tissue samples (see *Methods*). The set of combinations distinguish between tumor and normal tissue samples with over 90% sensitivity and specificity. This result is robust to different Training and Test set partitions of the available tumor and normal tissue samples. Although the identified combinations contain many genes previously implicated in cancer, our approach has also identified several potentially novel cancer

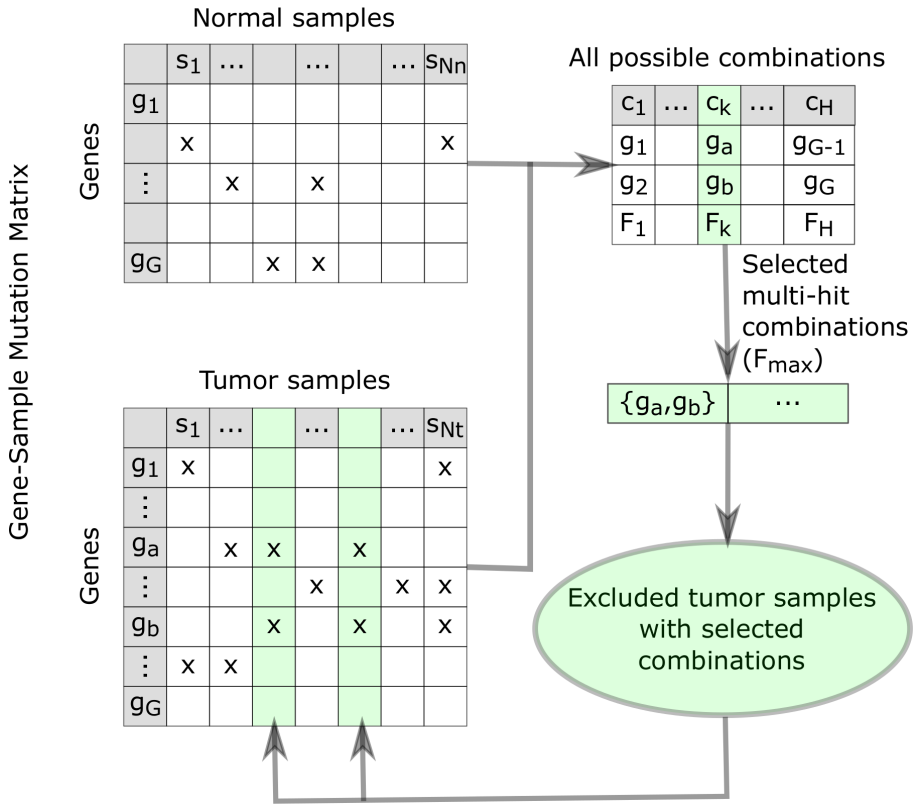


Figure 3.3: Algorithm for finding multi-hit combinations, illustrated for two-hit combinations. The cells marked with x in the Gene-Sample Mutation matrices represent samples with mutations in the corresponding gene. There are $H = G(G-1)/2$ possible two-hit combinations involving two different genes. The algorithm iterates through three steps. (1) Equation (3.1) is used to calculate F_i for each combination. (2) The combination (g_a and g_b in this example) with the maximum value of F_i, (F_k in this example) is added to the list of selected multi-hit combinations. (3) Tumor samples containing mutation in the selected combination of genes are excluded from consideration in subsequent iterations of the algorithm. The green shaded columns in the Tumor Gene-Sample Mutation matrix represent excluded samples in the iteration shown. The algorithm terminates when all tumor samples have been excluded, i.e. "covered" by the set of multi-hit combinations.

genes. Our results suggest that some of the combinations identified are 2-hit subsets of 3+ hit combinations.

3.3.1 Differentiation Capability Between Tumor and Normal Tissue Samples With High Accuracy via a Set of Two-Hit Combinations

We implemented the weighted set cover algorithm described in *Methods*, for identifying a set of 2-hit combinations with the goal of maximizing accuracy (sensitivity and specificity) in differentiating between tumor and normal samples. Using a randomly selected Training set (see *Methods*), we identified a set of two-hit combinations for each of the seventeen cancer types with at least two hundred matched tumor and blood-derived normal samples.

When tested against a separate randomly selected Test set, the identified set of combinations were able to differentiate between tumor tissue samples and normal tissue samples, for their respective cancer types, with greater than 90% specificity and sensitivity on average. Table 3.1 shows the sample sizes, sensitivity, and specificity for the Training and Test sets for each of the seventeen cancer types. Sensitivity varies from 83% to 100% and specificity varies from 86% to 100%, depending on cancer type.

The number of combinations identified varies from 8-20 for the 17 cancer types (Table 3.1). In total, 197 combinations were identified (Tables S2 - S18). The top three 2-hit combinations are summarized in Fig. 3.4. The combinations include 256 unique genes with 138 genes occurring in more than one combination.

3.3.2 Robustness of Two-Hit Combinations to Different Training and Test Sets

To test the robustness of the above results, we randomly re-partitioned the available samples into two more alternative Training and Test sets. Figure 3.5 shows specificity

		Discovery Set								Validation Set								
		Tumor Samples				Normal Samples				Tumor Samples					Normal Samples				
Cancer Type	#Combinations	True Positives	False Negatives	Total	Sensitivity	True Negatives	False Positives	Total	Specificity	True Positives	False Negatives	Total	Sensitivity	95% CI	True Negatives	False Positives	Total	Specificity	95% CI
Bladder Urothelial Carcinoma (BLCA)	18	267	0	267	100%	245	2	247	99%	89	12	101	88%	80-93%	74	12	86	86%	76-92%
Breast invasive carcinoma (BRCA)	8	703	0	703	100%	236	11	247	96%	207	1	208	100%	97-99%	82	4	86	95%	88-98%
Cervical squamous cell carcinoma and endocervical adenocarcinoma (CESC)	9	217	0	217	100%	247	0	247	100%	52	5	57	91%	80-97%	84	2	86	98%	91-99%
Colon adenocarcinoma (COAD)	9	291	0	291	100%	245	2	247	99%	85	9	94	90%	82-95%	83	3	86	97%	90-99%
Glioblastoma multiforme (GBM)	10	253	0	253	100%	247	0	247	100%	72	6	78	92%	84-97%	78	8	86	91%	82-95%
Head and Neck squamous cell carcinoma (HNSC)	13	347	0	347	100%	245	2	247	99%	102	21	123	83%	75-89%	81	5	86	94%	86-98%
Kidney renal papillary cell carcinoma (KIRP)	11	175	0	175	100%	246	1	247	100%	50	3	53	94%	84-98%	86	0	86	100%	95-100%
Brain Lower Grade Glioma (LGG)	9	356	0	356	100%	245	2	247	99%	111	12	123	90%	83-94%	80	6	86	93%	85-97%
Liver hepatocellular carcinoma (LIHC)	9	233	0	233	100%	246	1	247	100%	78	1	79	99%	93-99%	79	7	86	92%	83-96%
Lung adenocarcinoma (LUAD)	13	318	0	318	100%	245	2	247	99%	83	8	91	91%	83-96%	79	7	86	92%	83-96%
Lung squamous cell carcinoma (LUSC)	12	224	0	224	100%	246	1	247	100%	68	13	81	84%	74-91%	82	4	86	95%	88-98%
Ovarian serous cystadenocarcinoma (OV)	8	235	0	235	100%	246	1	247	100%	75	7	82	91%	83-96%	83	3	86	97%	90-99%
Prostate adenocarcinoma (PRAD)	20	327	0	327	100%	245	2	247	99%	83	11	94	88%	80-94%	68	18	86	79%	68-87%
Sarcoma (SARC)	6	167	0	167	100%	247	0	247	100%	47	5	52	90%	78-96%	86	0	86	1.00	95-100%
Stomach adenocarcinoma (STAD)	19	306	0	306	100%	247	0	247	100%	72	10	82	88%	78-93%	77	9	86	90%	81-95%
Thyroid carcinoma (THCA)	13	314	0	314	100%	245	2	247	99%	94	13	107	88%	80-93%	78	8	86	91%	82-95%
Uterine Corpus Endometrial Carcinoma (UCEC)	10	368	0	368	100%	247	0	247	100%	121	6	127	95%	90-98%	81	5	86	94%	86-98%
Total	197	5101	0	5101	100%	4170	29	4199	99%	1489	143	1632	91%	89-92%	1361	101	1462	93%	91-94%

Table 3.1: Two-hit combinations can differentiate between tumor and normal tissue samples with over 90% sensitivity and specificity. The combinations were identified using a randomly selected 75% subset (training set) of the available matched tumor and blood-derived normal samples for each cancer type with at least 200 matched samples in TCGA. See Tables A2-A18 for the list of gene combinations for each cancer type. The resulting combinations were then tested against the remaining samples (test set).

and sensitivity of the algorithm across the seventeen cancer types considered here, for three different sets of partitions. The average difference in sensitivity between any two pairs of train-test partitions is less than 4.2% and the average difference in

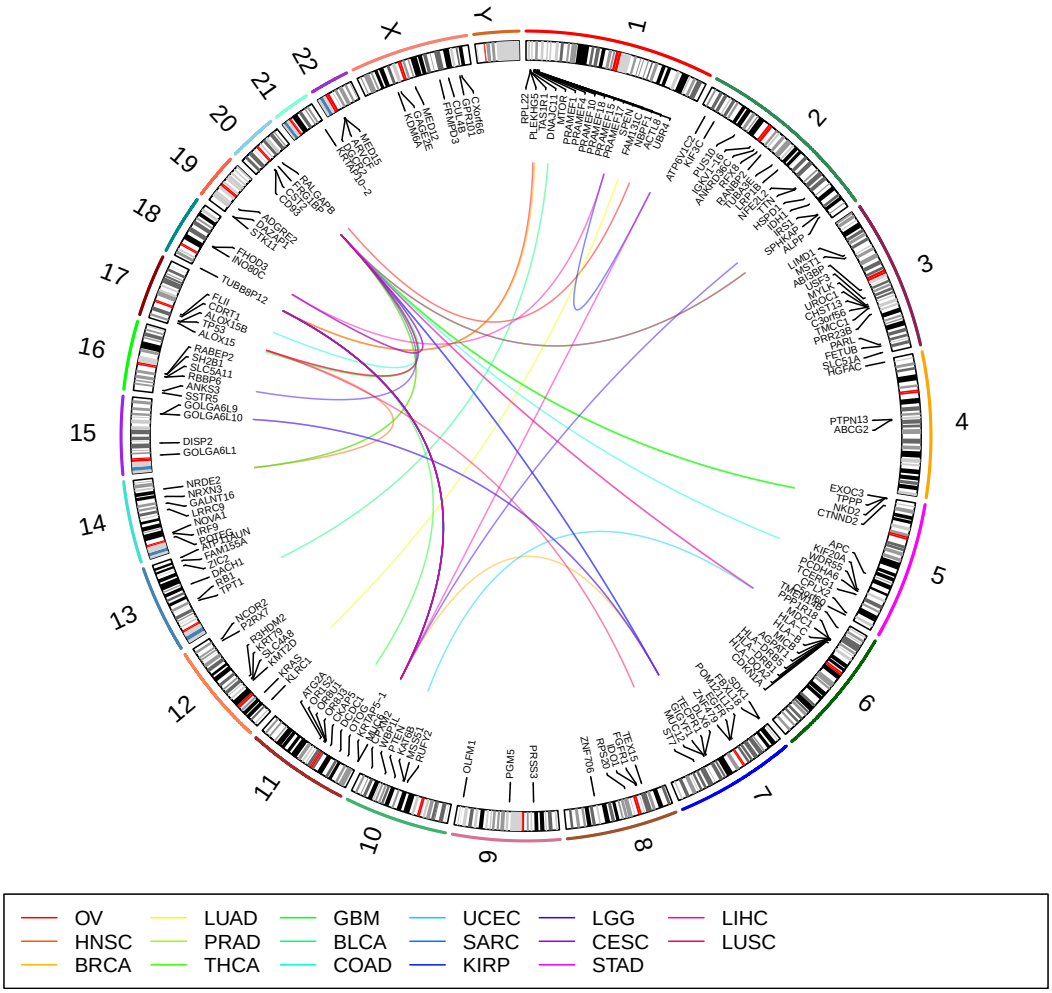


Figure 3.4: Top three two-hit combinations for 17 cancer types. See Table 3.1 for abbreviations for cancer types. Each line in the center of the Circos plot connects the two genes in a two-hit combination. This plot was generated using RCircos [192].

specificity is less than 4.1%. The largest difference in sensitivity is 12% (BLCA) and the largest difference in specificity is 13% (KIRP). In addition, the most frequently occurring combinations in the tumor samples were the same between any two train-test partitions for 14 of 17 cancer types, representing 65% of tumor samples (Fig. 3.6). However, there were significant differences between the less frequently occurring combinations with only 39 common combinations, out of 197 total combinations, across

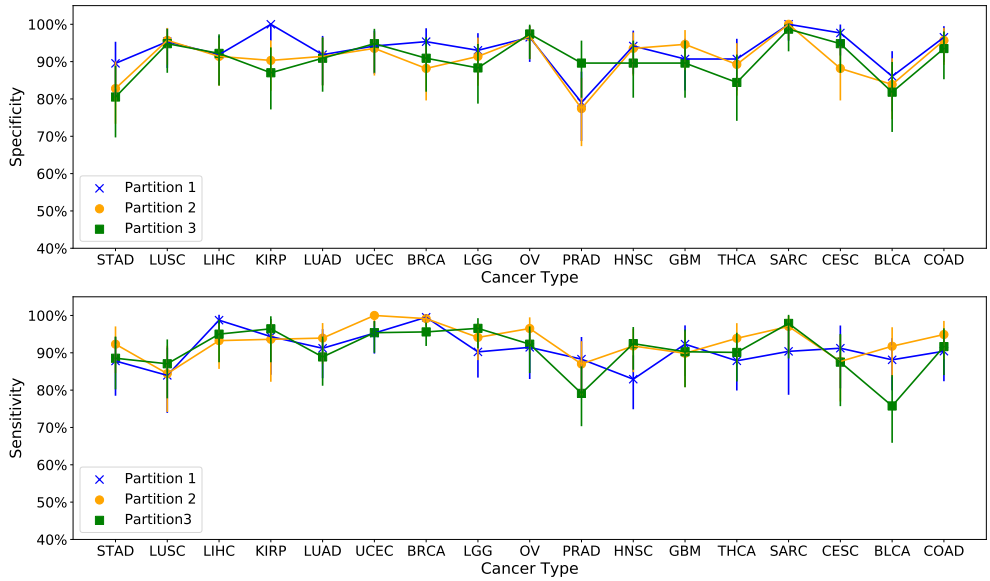


Figure 3.5: Sensitivity and specificity is robust across three different random training-test partitions of available samples. The average difference between any two pairs of partitionings is less than 4.2% for both sensitivity and specificity across all 17 cancer types. Error bars represent 95% confidence intervals. The vertical lines represent 95% confidence intervals.

the three sets of combinations for the three train-test partitions (Figure A2). Clearly, the samples included in the Training set affect the set of combinations identified. This is to be expected since 42% of the combinations occur in less than 5% of the samples for each cancer type (Figure A4). Different partitions of the tumor samples will result in different sets of these rare combinations being included in the Training set, resulting in different combinations being identified. In addition, since the approximation algorithm used here identifies a near-optimal solution, changes in the Training set can result in different near-optimal combinations being selected by the algorithm.

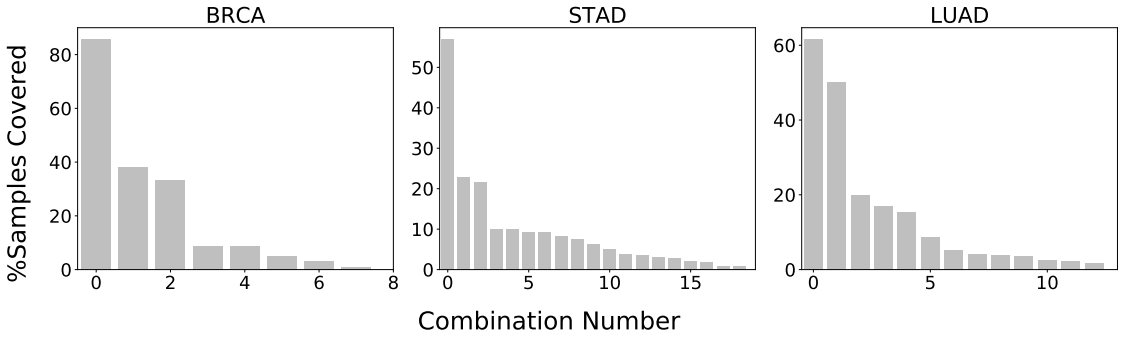


Figure 3.6: Occurrence of the two-hit combinations identified in tumor samples, for three representative cancer types. Figure A3 shows the distribution for all 17 cancer types. The top combination occurs in 65% of tumor samples, on average, while 42% of the combinations occur in less than 5% of the samples. Total percentage exceeds 100% because samples can contain multiple combinations.

3.3.3 Properties of Identified Genes and Combinations

The combinations identified include novel cancer genes The genes comprising the two-hit combinations identified above fall into three categories. (1) Confirmed cancer genes based on the Catalog of Somatic Mutations in Cancer (COSMIC) database [144]. (2) Non-COSMIC genes that have been implicated in cancer based on experimental evidence. (3) Genes that have not been experimentally implicated in cancer. Table 3.2 summarizes, from Tables S2 - S18, the 31 genes that comprise the top three most frequently occurring two-hit combinations for each of the cancer types studied. Of these genes, nine are confirmed cancer genes (e.g. APC, IDH1, KRAS, PTEN, RB1, and TP53), thirteen have been experimentally implicated in cancer (e.g. HLA-C, IGHG1, and KCNB1), and nine have not previously been implicated in cancer (e.g. TUBBP12).

The genes in the last category have not been extensively studied, and represent potentially novel cancer genes. For example, TUBB8P12 (Tubulin Beta 8 Pseudogene 12) occurs in the top three two-hit combinations in 15 of the 17 cancer types. However, TUBB8P12 has not been previously identified as frequently mutated in cancers.

There are two possible reasons why we have identified TUBB8P12 as a potential cancer gene while previous bioinformatics studies have not. The first reason is that, we considered low frequency somatic mutations, identified using matched tumor and blood derived normal samples, that were not included in many of the previous studies [100, 166, 172, 186]. Biopsy specimens contain a mix of tumor and normal tissue cells, tumor infiltrating lymphocytes, and stromal cells. In addition, tumor cells themselves can be genetically diverse. Therefore many somatic mutations are likely to be present at very low frequencies [152, 166]. Studies that use masked open-access TCGA data will exclude many such low-frequency mutations. The second reason is that, those studies that do use controlled-access TCGA data that include these low-frequency mutations, do not use matched normal tissue and blood-derived normal samples to quantify the differential mutation frequency between tumor and normal samples[52, 100, 168, 172]. By comparing somatic mutation frequency in matched tumor tissue samples to mutation frequency in matched normal tissue samples, we are able to identify genes that are significantly more frequently mutated in tumor samples relative to normal samples, while excluding genes that may be highly mutated in both tumor and normal samples.

The two-hit combinations may represent subsets of a larger number of hits

Due to practical limitations of computational resources, it is not practical to search for more than two-hit combinations using the current version of the algorithm presented (see *Methods*). The computer run times for identifying two-hit combinations were $\approx$ *two* hours, compared to estimated run times of over one year for three-hit combinations. Mathematical models predict that the likely number of hits required for carcinogenesis ranges from two to eight. Therefore, it is likely that the two-hit combinations identified here are different subsets of three or more hits In fact, we find that 65% of the samples contain multiple combinations (Fig. 3.7), and 138 of the 256 genes in these combinations occur in more than one combination, suggesting that the genes in the different two-hit

Table 3.2: Genes in the top three most frequently occurring two-hit combinations. Genes are color coded to identify those that are confirmed cancer genes, experimentally implicated in cancer, and potentially novel cancer genes. The numbers in the table (1, 2, and/or 3) indicate which of the top three two-hit combinations the gene belongs to.

	Confirmed Cancer Gene	1 = Gene is part of the most frequent comb
	Experimentally implicated in cancer	2 = Gene is part of the 2nd most frequent comb
	Potentially novel cancer gene	3 = Gene is part of the 3rd most frequent comb

Gene	Cancer Type																
	BLCA	BRCA	CESC	COAD	GBM	HNSC	KIRP	LGG	LIHC	LUAD	LUSC	OV	PRAD	SARC	STAD	THCA	UCEC
ALOX15[116]					3												
ALPP[21]													3		3	3	
APC[165]				1													
CACNA1E[127]										3							
CCDC30[149]	3																
CCDC43				3													
CTNND2 [182]	2												2			2	
DPP6[169]											3						
FHOD3[140]							2		1,3								
FRG1BP[14]	2	2	2	1,3	2,3	3	2	3	1	2	2	2	2,3	1	2,3	2,3	1
GOLGA6L10			3														
GOLGA6L9														3			
HLA-C[177]						3									2		
HLA-DRB1[125]																	3
HRNR [62]							3										
IDH1[54]								1									
IGHG1[135]		2			2	2				2							
KCNB1[40]												1					
KRAS[112]										3							
LCE1A									3								
MUC12[121]		1	2,3											1,3			1
MUC6[178]	1	1,3	1	2		1	1	1,2	2	1	1	1	1	2	1	1	2
NBPF1[147]		3															
OR2T7							3		2								
OR8U1					1												
PRAMF15												3					
PTEN[189]																	3
RB1[18]	3																
SLC5A11[175]								3									
TP53[79]						2					2,3	2					
TUBB8P12	1		1	2	1	1	1	2		1	1	3	1	2	1	1	2

combinations within a sample may instead represent a single combination consisting of more than two-hits. Therefore, the two hit combinations may produce some false positives in normal samples containing mutations in only two genes of a *three+*-hit combination. Therefore, searching for three or more hits may further improve the accuracy of our results.

Genes within combinations are not correlated Analysis of genes within each combination shows that they are not correlated. For each of the genes in a combination

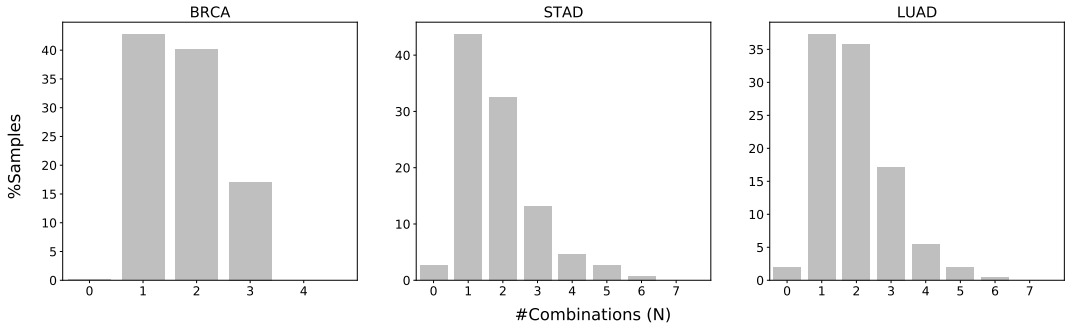


Figure 3.7: Distribution of overlapping combinations for three representative cancer types. Figure A5 shows the distribution for all 17 cancer types. 64.5% tumor samples contain multiple combinations, suggesting that the two-hit combinations might represent subsets of three or more hits.

we construct a vector of 0's and 1's. The length of the vector is equal to the number of normal samples, and the value in the i^{th} position of that vector represents whether the i^{th} normal sample has a protein-altering mutation (as determined my VEP) in that location or not. Then we computed Pearson's correlation coefficient[141] using *stats.pearsonr* routine from python module *scipy.stats* between two vectors representing two different genes. The Pearson correlation coefficient is less than 0.25 for the gene pairs within each combination (Figure A1). If the genes within a combination were correlated it would have suggested that the combination is a result of some common underlying cause, such as being a passenger mutation or due to structural chromosomal modification, and unlikely to be causative. We also examined the chromosomal location of genes within each combination (Fig. 3.8). Only two of the 197 combinations contain genes within the same chromosome, suggesting that the genes within combinations are not due to a chromosomal abnormality that may affect multiple genes within a chromosome.

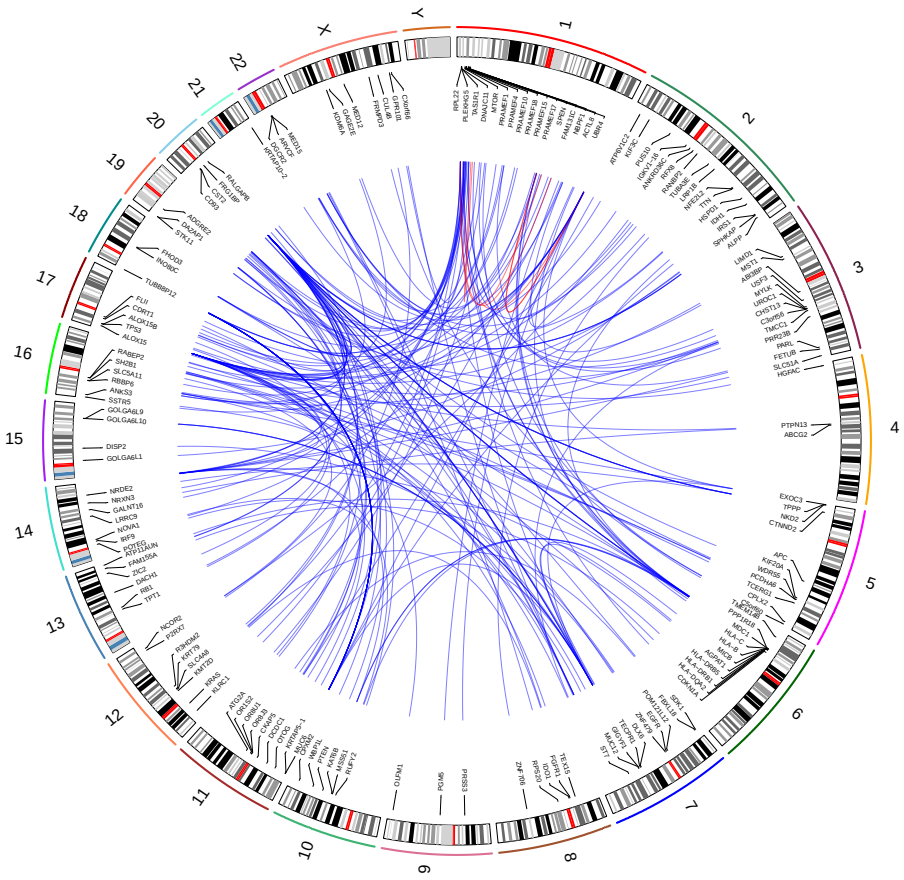


Figure 3.8: Chromosomal location of gene combinations. Each connecting line represents a two-hit combination. Blue lines represent gene combinations across different chromosomes. Red lines represent gene combinations within the same chromosome. Circos plot was generated using RCircos package [192].

3.4 Scaling Up the Approximate Algorithm to Identify Three-Hit Gene Combinations

The multi-hit algorithm presented in Section 3.2.3 for identifying combinations of genes with mutations that may represent the potential cause for individual instances of cancer.

Due to its computational complexity the algorithm was limited to identifying combinations of two hits. To identify combinations of more than two hits, the algorithm was restructured and optimized for parallel execution through some data-structure optimizations and algorithmic re-structuring.

3.4.1 Representation of Gene-Sample Matrix

Indexed array data structure One option for speeding up the above algorithm, by reducing the number of arithmetic operations, is to replace the Gene-Sample Mutation matrices with an indexed array data structure. The data structure consists of two arrays. One is a samples array where samples with mutations within each gene are listed sequentially. The second is a gene index array, which contains the starting index into the samples array for each gene. With the indexed array representation, the algorithm would only examine samples with mutations in the genes being considered, instead of all samples. It is more efficient than the original matrix representation since samples that do not contain mutations in a gene are not evaluated. On average only 2% of samples have mutations for a given gene, therefore, we expected a significant speedup with an indexed list representation However, due to an increase in the number of instructions and divergent branches, the speedup using this data structure was less than what was achieved using the compressed binary representation described below.

Compressed binary representation The binary values in the Gene-Sample Mutation matrices permits a reduction in memory requirement using a compressed binary representation. In addition, bitwise operations can be used with the compressed binary representation to reduce computational cost and divergent branching (discussed in section 5.5.2). Figure 3.9 illustrates how mutations in a group of four samples can be compressed into four bits. In the original implementation, each Gene-Sample value

was represented as a single 16-bit short integer [48]. For this implementation, we represent groups of 64 samples as a single 64-bit unsigned integer, which requires 4-fold fewer vector operations compared to the 16-bit unsigned integer representation. The resulting speedup was confirmed experimentally (results not shown). The compressed binary representation also results in 16-fold reduction in memory since each word of memory stores data for 16 samples, compared to one sample per word in the original integer matrix representation.

The number of samples with mutations in a combination of genes can then be efficiently determined by a bitwise *AND* operation followed by a count of the non-zero bits, as illustrated in Fig. 3.9. To count the number of non-zero bits for the CPU code, we implemented Brian Kernighan's algorithm [92]. For the GPU implementation, we used the built-in `popcll()` function to count the number of bits set to 1 in the 64 bit unsigned integer. This function was faster than our own implementation of Brian Kernighan's algorithm (e.g. runtime for optimized GPU implementation for the three-hit algorithm for BRCA using `popcll()`, is 451 sec faster than the runtime using Kernighan's algorithm).

For the computation of two-hit combinations for breast cancer (BRCA), the compressed binary representation resulted in a 16-fold speedup on the CPU compared to the original matrix representation, as shown in the Results. Since this speedup was considerably larger than the corresponding four-fold speedup for the indexed array representation described above, we did not consider the indexed array structure any further.

3.4.2 Mapping to the NVIDIA Tesla V100 PCIe Graphical Processing Unit (GPU)

To further speed up the multi-hit algorithm, we restructured the CPU code for parallel execution on one GPU, specifically the NVIDIA Tesla V100 PCIe GPU [4]. The V100

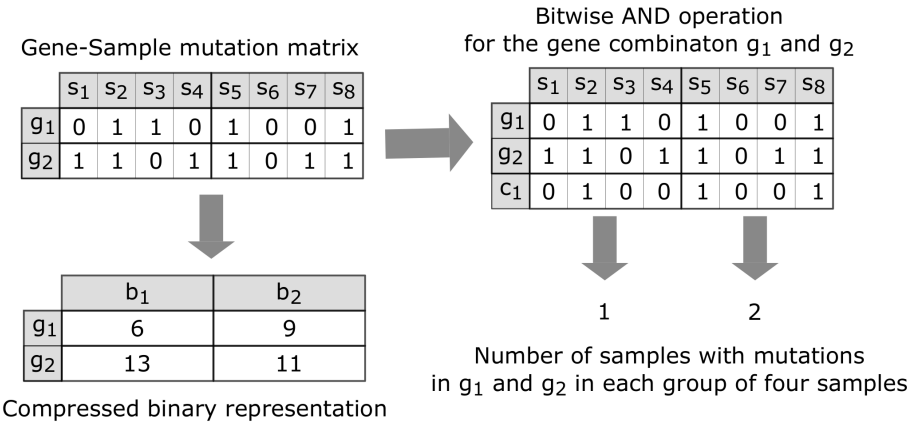


Figure 3.9: Compressed binary representation and bitwise operation for determining the number of samples with mutations in a combination of two genes. *Left:* Compressed binary representation of Gene-Sample Mutation matrices, illustrated for a four-bit unsigned integer. s_i represents the normal or tumor samples shown in Fig. 3.3. Elements with 0 in the matrix indicate that the sample does not contain mutations in the corresponding gene, while 1 indicates that the sample does contain a mutation in the corresponding gene. Mutations in four samples can be represented by a single four-bit unsigned integer. *Right:* Given any two genes g_i, g_j, the number of samples containing mutations in both these genes is determined by a bitwise `AND` operation for each of the integers representing mutations in g_i with the corresponding set of integers for g_j, and then counting the number of non-zero bits.

consists of 5376 32-bit floating point cores, 5376 32-bit integer cores, 2688 64-bit floating point cores, 672 tensor cores and 336 texture units. The cores are partitioned into 84 streaming multiprocessors (SM) with 128 KB of shared memory per SM, 6 MB of *L2* cache and 32 GB of global memory for the GPU. Each SM is further partitioned into four single instruction multiple thread (SIMT) *warps*, i.e. the same instruction is executed on all cores within a warp, with each core running a different thread [4]. For parallel execution of the algorithm, we partition the computation of F_{max} across multiple threads, where each thread computes the maximum value of F for a subset of combinations (F^i_{max}) where $i \in [1, G(G-1)/2]$. For two-hit combinations, each thread processes a single combination, therefore $F^i_{max} = F^i$ for the combination i, say g_a and g_b where $a < b \leq G$. For three-hit combinations F^i_{max} is the maximum value for all three-hit combinations with two of the hits corresponding to two-hit combination i, i.e. g_a, g_b and g_c where $a < b < c < G$, as illustrated in Fig. 3.10. The maximum value

F_{max} across all F^i_{max} is then calculated using parallel reduction [76].

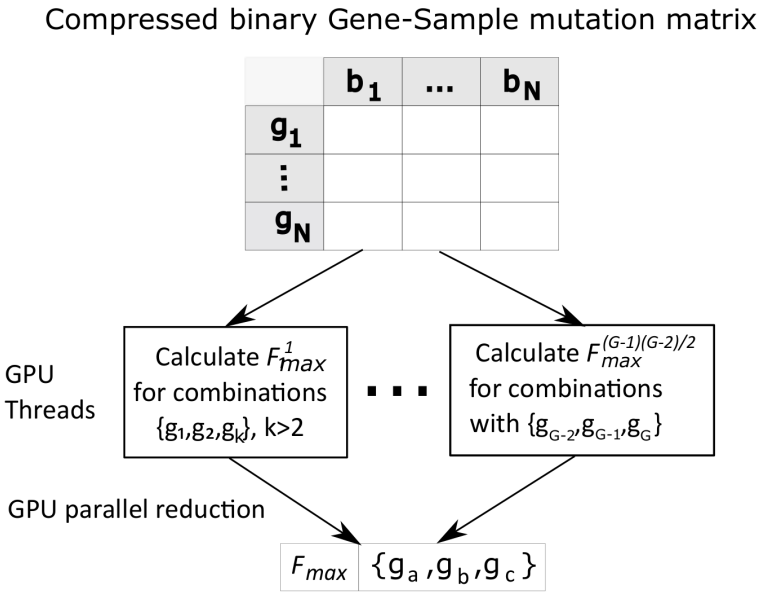


Figure 3.10: Mapping the multi-hit CPU algorithm to the GPU, illustrated for the three-hit algorithm with the compressed binary representation (Fig. 3.9). Each GPU thread computes F^i_{max} for a subset of all possible combinations. The results of each thread is stored in GPU global memory. F_{max} across all subsets of combinations is calculated using parallel reduction [76].

The sequential implementation of the above algorithm for three-hit combinations is illustrated in Algorithm 9. The `for` loops in lines 7, 8, and 9, in the sequential algorithm iterates through all possible $\binom{G}{3}$ three-hit combinations. Lines 10-14 compute F for one such combination. Lines 16-18 compute overall best combination along with it's F_{max} value. The remaining part of the algorithm updates excluded samples.

To run on parallel compute units of a GPU, we modified the above sequential algorithm as illustrated in Algorithm 10. We combined the two outer `for` loops into a single one, which iterates $\binom{G}{2}$ times (Line 7). Each iteration of this combined `for` loop can run in parallel on a different GPU compute units. For each of these parallel tasks, indexed by λ, there is a sequential `for` loop (Line 11) which computes the best combination among the three-hit combinations that start with i, j corresponding to the λ (3.2). The mapping of λ to i and j in lines 7 and 8 is described below under "Minimizing divergent

Algorithm 9 Sequential algorithm to compute three-hit combinations.

Require: *tumor-sets*, *normal-sets*, *tumor-samples*, *normal-samples*, α

1: $\textit{covered-samples} \leftarrow \Phi$
2: $\textit{combinations} \leftarrow \Phi$
3: $N_t \leftarrow \left|\textit{tumor-samples}\right|$
4: $N_n \leftarrow \left|\textit{normal-samples}\right|$
5: **while** $\textit{covered-samples} \neq \textit{tumor-samples}$ **do**
6: $\quad F_{max} \leftarrow 0.0$
7: $\quad$ **for** $i = 1 \rightarrow G$ **do**
8: $\quad\quad$ **for** $j = i + 1 \rightarrow G$ **do**
9: $\quad\quad\quad$ **for** $k = j + 1 \rightarrow G$ **do**
10: $\quad\quad\quad\quad TP \leftarrow \left|\textit{tumor-sets}[i]) \cap \textit{tumor-sets}[j]) \cap \textit{tumor-sets}[k])\right|$
11: $\quad\quad\quad\quad FP \leftarrow \left|\textit{normal-sets}[i]) \cap \textit{normal-sets}[j]) \cap \textit{normal-sets}[k])\right|$
12: $\quad\quad\quad\quad TN \leftarrow N_n - FP$
13: $\quad\quad\quad\quad FN \leftarrow N_t - TP$
14: $\quad\quad\quad\quad F \leftarrow \left(\dfrac{\alpha \times TP + TN}{N_t + N_n}\right)$
15:
16: $\quad\quad\quad\quad$ **if** $F \geq F_{max}$ **then**
17: $\quad\quad\quad\quad\quad F_{max} \leftarrow F$
18: $\quad\quad\quad\quad\quad \textit{best-combination} \leftarrow \langle i, j, k \rangle$
19:
20: $\quad \textit{combinations.add}(\textit{best-combination})$
21: $\quad i, j, k \leftarrow \textit{best-combination.extract}()$
22: $\quad \textit{covered-samples.add}(\textit{tumor-sets}[i]) \cap \textit{tumor-sets}[j] \cap \textit{tumor-sets}[k])$
23: **return** *combinations*

branches". At the end of this outer `for` loop, our parallel algorithm performs a parallel reduction (Line 22) to compute the best combination [76]. Then the tumor samples covered by this best combination are added to the covered samples.

This parallelization allows us to run $\binom{G}{2}$ parallel tasks with load $O\big(G(N_t+N_n)\big)$ instead of $\binom{G}{3}$ sequential tasks with load $O\big(N_t + N_n\big)$.

GPU optimization

The differences between the CPU and GPU architectures make certain coding techniques, that may be appropriate for serial execution on a CPU, sub-optimal or even

incorrect for parallel execution on a GPU. Three critical considerations for minimizing processor latency are: synchronizing update access to memory locations shared by multiple processors, divergent branching in a single instruction multiple thread (SIMT) architecture, and relative speed of shared memory vs. global memory.

Minimizing GPU synchronization A straightforward implementation would have a single global memory location for F_{max} which could be updated by all thread. However, such an implementation would require a synchronization or locking protocol to maintain cache coherence [161]. Synchronization of GPU threads to ensure correct results introduced significant processor latency and resulted in the GPU implementation running slower than the CPU version. We therefore allocate a separate memory location for each thread i to store its F^i_{max} value (Fig. 3.10), avoiding the need for synchronized memory access. We then use parallel reduction to efficiently calculate the global F_{max} value [76].

Minimizing divergent branches In the SIMT warps used by the GPU within streaming multiprocessors (SM), divergent branches introduce significant processor latency [16, 131, 153, 154, 156]. Divergent branches are `IF-ELSE` and `LOOP` control statements that cause execution along different paths depending on conditional values. Within a warp, all possible execution paths are serialized and evaluated [4, 154, 156]. Thus, significant latency is introduced due to the execution of instruction within branches that are not used. In the original integer matrix CPU implementation, conditional statements are required to count the number of samples with mutations in a gene combination. In the compressed binary implementation, these conditional statements are replaced by a set of bitwise operations, as described above (Fig. 3.9). In addition, the CPU implementation calculates a bound on the maximum possible value for F^i for a given combination i. If this value is less than the intermediate value for F_{max}

(Equation (3.1), subsequent processing for the combination is skipped. Although this strategy improved performance on the CPU, eliminating this branch and bound logic on the GPU resulted in an additional 6% average speedup for the two-hit algorithm and three-hit algorithm (Supplementary Tables S3 and S5).

The multi-hit algorithm only considers combinations represented by the upper triangular matrix, i.e. combinations of g_i and g_j where $i < j$. In the CPU implementation, processing is limited to the upper triangular matrix by loop control conditions. To eliminate these conditional branches, using the formulation from Ref [128], modified for upper triangular matrix instead of lower triangular matrix, we map the thread index λ to the upper triangular matrix $i < j$ as follows:

$$\begin{aligned} j &= \left\lfloor \sqrt{1/4 + 2\lambda} + 1/2 \right\rfloor \\ i &= \lambda - j(j-1)/2 \end{aligned} \tag{3.2}$$

Using shared memory for parallel reduction The F^i_{max} values calculated by each GPU thread i is stored in global memory (Fig. 3.10). The maximum value F_{max} across all these threads can be calculated using parallel reduction, directly in global memory. However, accessing global memory is significantly slower than accessing shared memory [86]. Therefore, we divide global memory data into blocks which are copied into shared memory. Parallel reduction is performed within each block using shared memory to compute the F^j_{max} for block j. The result is copied back to a new allocation in global memory. This new allocation is 1024 (the number of virtual threads per block) times smaller than the original allocation. This process is repeated with the newly allocated values until the single F_{max} values has been calculated. The above approach reduced the total global memory used by approximately 50%, e.g. 2.87 GB for BRCA compared to 5.75 GB without this approach.

3.4.3 Speedup and Accuracy Calculation

Speedup was calculated as t_{ref}/t_{new} where t_{new} and t_{ref} are the runtimes for the new code and the baseline reference, respectively. runtime was determined using the Linux `time` command, with runtime = `sys time + user time`. For identifying two-hit combinations, the runtime for the original CPU implementation [48] was used as the baseline reference t_{ref}. However, this was not practical for three-hit combinations for cancer types where the runtime using the original matrix code was over 30 days. Therefore we estimated the runtime for cancer types taking over 30 days, based on the actual runtimes for cancer types requiring less than 30 days. We assumed that the average ratio of two-hit vs. three-hit speedup for the compressed binary CPU implementation compared to the original matrix CPU implementation is the same for both categories (runtime $<$ 30 days and runtime $>$ 30 days). For cancer types with runtime $<$ 30 days, we calculated the average ratio $R = \text{Avg}(S_3/S_2)$, where S_3 is the three-hit speedup and S_2 is the two-hit speedup, for the CPU compressed binary implementation compared to the CPU original matrix implementation. For cancer types with three-hit runtime $>$ 30 days, we estimated the runtime as $R \cdot S_2 \cdot t_{3cb}$, where t_{3cb} is the three-hit runtime for the compressed binary CPU implementation. See Supplementary Table S4 for a list of actual and estimated three-hit runtimes.

All available mutation data was randomly partitioned into two subsets, with 75% of the data (Training set) used to identify the multi-hit combination using the above algorithm. The remaining data (Test set) was used to calculate the sensitivity, specificity and 95% confidence interval for the identified set of combinations' ability to differentiate between tumor and normal samples. Sensitivity was calculated as TP/N_t, where TP is the number of true positives (number of tumor samples containing one of the identified combinations) and N_t is the number of tumor samples. Specificity was calculated as TN/N_n, where TN is the number of true negatives (number of normal samples

without any of the identified combinations) and N_n is the number of normal samples. 95% confidence interval was calculated using the "exact" Clopper-Pearson method [38]. Overall sensitivity and specificity is calculated from the total count of true positives, true negatives, tumor samples and normal samples for all cancer types, using the randomly selected 25% test data set. However, it is important to keep in mind that these multi-hit gene combinations may not represent cancer genes. Additional experimental validation is required to determine if mutations within these genes may play role in cancer.

3.5 Classification and Runtime Performance of the Parallel Algorithm

Cancer is estimated to be caused by a combinations of a small number of (two to eight) genetic mutations (hits) [17, 19, 20, 109, 114, 130, 173, 195]. We had previously developed an algorithm for identifying a set of two-hit combinations of genes with mutations, that was able to differentiate between tumor and normal samples with high sensitivity and specificity [48]. Due to its computational complexity the algorithm is impractical for identifying more than two hits [48].

To identify combinations of more than two hits, we restructured and optimized the algorithm for parallel execution on a GPU, as described in the Methods section. These modifications can be grouped into two broad categories: compressed binary matrix representation and GPU parallelization.

The compressed binary matrix optimization and GPU parallelization resulted in an average speedup of 12,144x for the three-hit algorithm, relative to the original integer matrix based CPU implementation. With this speedup, we were able to identify three-hit combinations for the 32 cancer types for which data was available in TCGA. In

addition, we were able to identify four-hit combinations for 14 cancer types for which the runtime was less than 15 days. The accuracy of the three-hit combinations was found to be comparable to the two-hit combinations, with overall sensitivity of 90% (95% CI = 88–91%) and average specificity of 93% (95% CI = 92–94%).

3.5.1 Optimization and Parallelization Reduces Runtime for the Two-Hit Algorithm

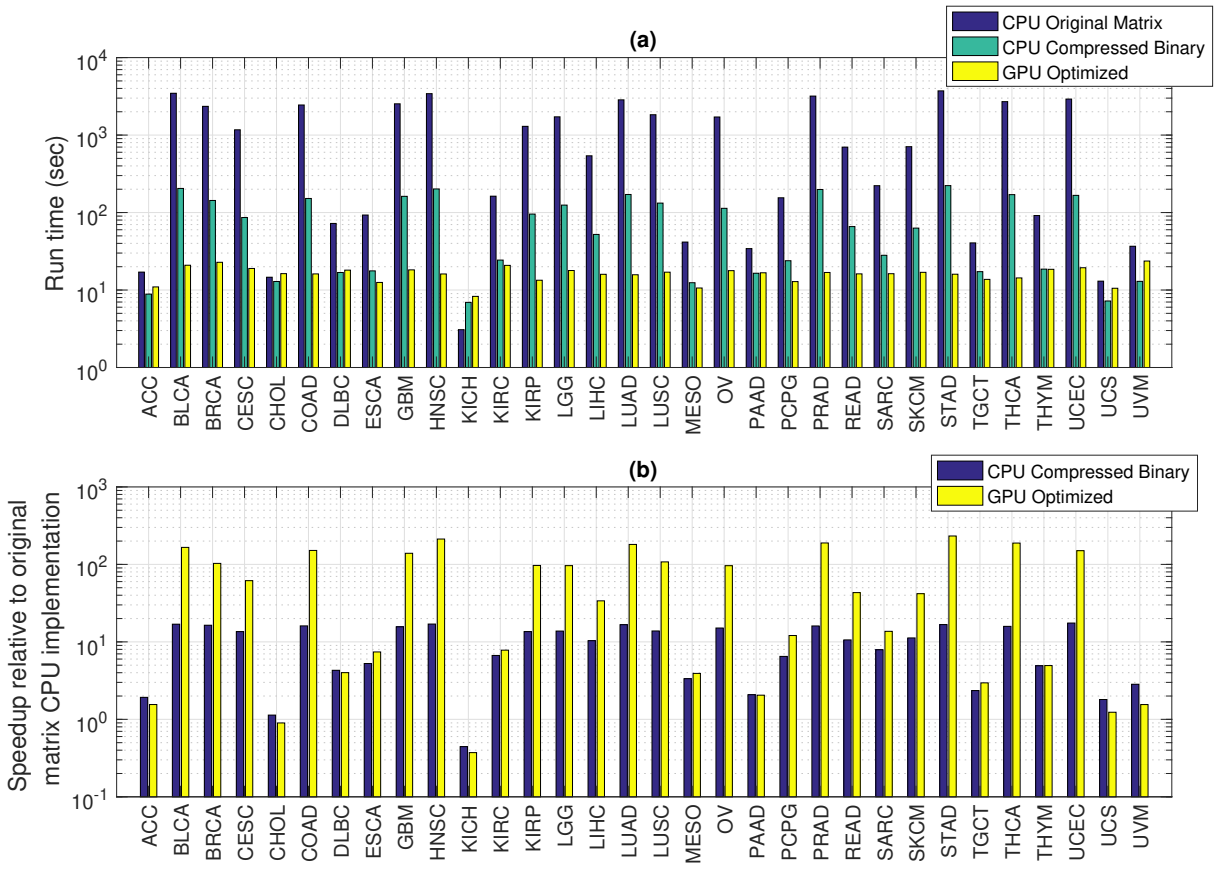


Figure 3.11: Comparison of different implementations of the multi-hit algorithm for identifying two-hit combinations. (a) runtime for the original matrix implementation on the CPU ranges from 3–3723 sec compared to 7-223 sec for the compressed binary CPU implementation and 5–33 sec for the optimized GPU implementation. (b) Speedup is on average 10-fold for the compressed binary CPU implementation and 68-fold for the optimized GPU implementation compared to the original matrix CPU implementation. Names for the cancer types shown along the x-axis are listed in Table S1.

Figure 3.11(a) shows that the runtime for identifying two-hit combinations ranges from 5–33 sec for the optimized GPU implementation compared to 7–223 sec for the compressed binary CPU implementation and 3–3,723 sec for the original matrix CPU implementation. The optimized GPU implementation of the two-hit algorithm is on average 68 times faster than the original CPU implementation, with the speedup ranging from 0.7–224x (Fig. 3.11(b)). However, due to the relatively large fixed data load time, these speedup numbers understate the effect of the optimization and parallelization described in the Methods. On average, the data load time for the two-hit optimized GPU implementation is 85% of the total runtime. The speedup values for the three-hit algorithm, where the above average data load times are 14% of total runtime for the optimized GPU implementation, is more closely representative of the effect of optimization and parallelization. Detailed runtimes for each cancer type, with a breakdown for data load time, for different implementations of the two-hit algorithm are shown in Supplementary Table S2.

3.5.2 Runtime Reduction Permits Identification of Three-Hit Combinations

Figure 3.12(a) shows that the runtime for identifying three-hit combinations ranges from 4 sec to 23 min for the optimized GPU implementation compared to 46 sec to 10 days for the compressed binary CPU implementation. For the original integer matrix CPU implementation the runtime ranges from 110 sec to an estimated 282 days. The optimized three-hit algorithm on the GPU results in an estimated 29 –33,690 fold speedup compared to the estimated time for the original matrix based CPU implementation (Fig. 3.12(b)), with an average 12,144 fold estimated speedup. Detailed runtimes and speedup for each cancer type for different implementations of the three-hit algorithm are shown in Supplementary Information Tables S4 and S5.

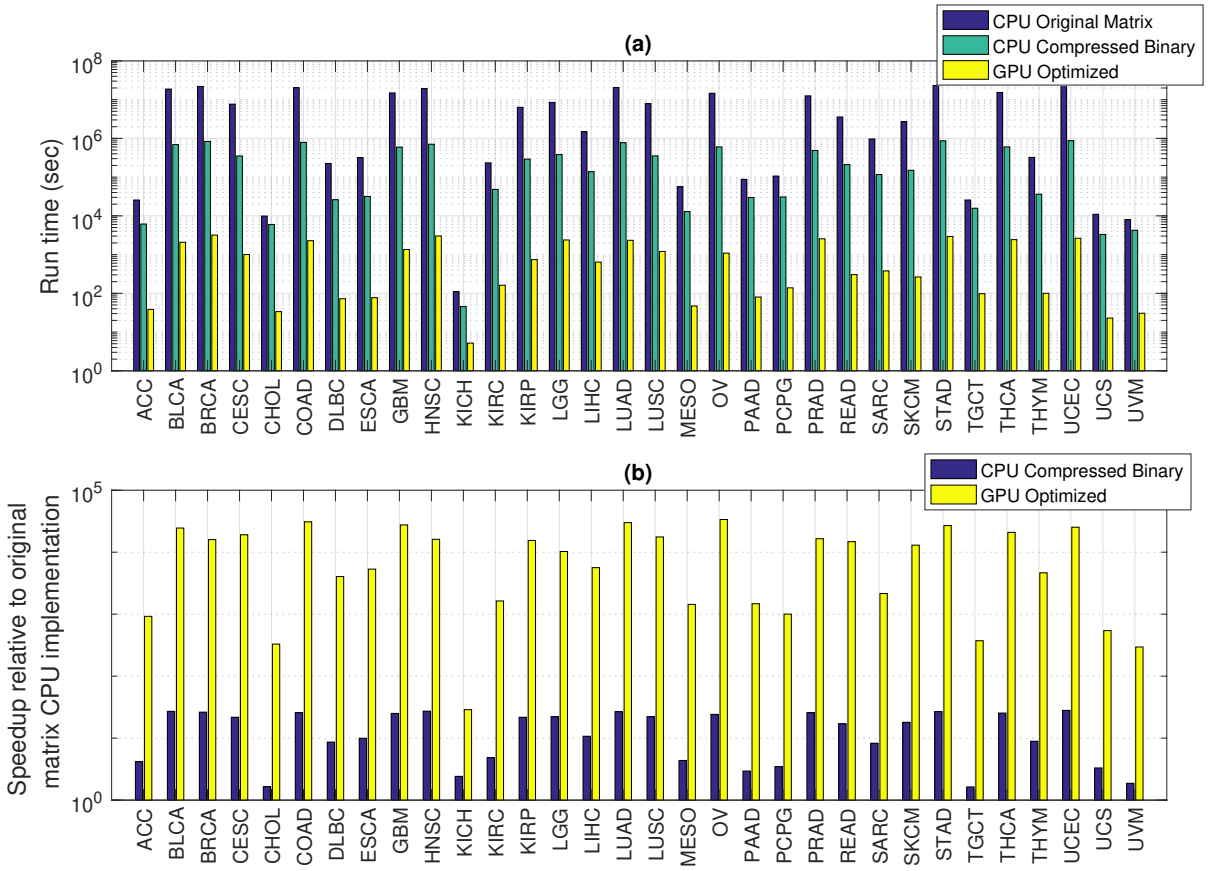


Figure 3.12: Comparison of different implementations of the multi-hit algorithm for identifying three-hit combinations. (a) runtime for the original matrix CPU implementation ranges from 110 seconds to an estimated 282 days, compared to 46 seconds to 10 days for the compressed binary CPU implementation and four seconds to 23 minutes for the optimized GPU implementation. Runtimes for the original matrix CPU implementation requiring over 30 days were estimated as described in Methods. (b) Speedup for the compressed binary CPU implementation ranged from 2x–28x, and from 29x–33,690x for the optimized GPU implementation. Names for the cancer types shown along the x-axis are listed in Table S1.

3.5.3 Runtime Reduction Permits Identification of Some Four-Hit Combinations

With the reduction in runtime resulting from the optimization and parallelization described in the Methods section below, we were able to identify four-hit combinations for some cancer types. For cancer types where the number of genes with mutations

$G < 19000$ it takes less than 15 days to identify four-hit combinations. Detailed runtimes for these cancer types are shown in Supplementary Information Table S6. To identify four-hit combinations for all cancer types will required additional optimization and parallelization across multiple GPUs, which will be presented in a separate forthcoming study.

3.5.4 Contribution of Optimization Techniques to Overall Speedup

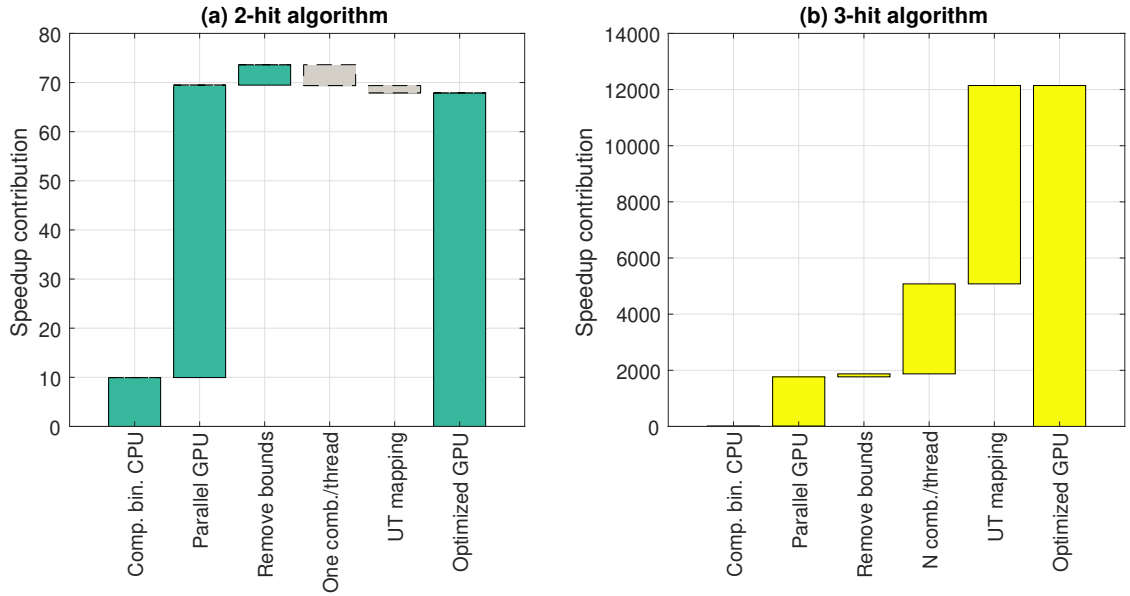


Figure 3.13: Average contribution of optimizations and parallelization to speedup. Breakdown of contributions due to compressed binary representation, GPU parallelization, removal of branch and bound logic, single two-gene combination per thread, and mapping of upper triangular (UT) gene combination to a sequential thread ID. (a) Breakdown of two-hit speedup. (b) Breakdown of three-hit combinations. Contribution due to compressed binary representation is 15x for three-hits which is not visible in the scale of the figure.

The speedup reported above results from five key enhancements: compressed binary representation of the Gene-Sample Mutation matrices, parallel execution across multiple GPU cores, removal of branch and bound logic, computation of a single two-gene combination per thread, and mapping upper triangular matrix of two-gene combinations to thread index. See Methods section below. The breakdown of the contribution

due to each of these enhancements is shown in Fig. 3.13. The speedup contribution of each enhancement is calculated as the difference in average speedup for the implementation of each enhancement compared to the original matrix CPU implementation See Supplementary Tables S3 and S5. On, average, the largest contribution to speedup for the two-hit algorithm is due to GPU parallelization (Fig.3.13(a)). The largest contribution for the three-hit algorithm is due to mapping GPU threads to the upper triangular matrix of two-gene combinations (Fig. 3.13(b)). The contribution due to the first three factors – compressed binary representation, GPU parallelization and removal of branch and bound logic – is roughly consistent between the two-hit and three-hit algorithms. However, the enhancements for a single two-gene combination per thread and upper triangular thread mapping slow down the two-hit algorithm. This is because, for the two-hit algorithm, speedup due to higher processor utilization from these enhancements are offset by the additional operations and global memory access required to implement these modifications.

3.5.5 Multi-Hit Combinations Differentiate Between Tumor and Normal Samples with High Accuracy

The three-hit combinations identified using a 75% randomly selected Training set identified an average of seven combinations per cancer type with a total of 335 unique genes, compared to eight combinations per cancer type with a total of 310 unique genes for the two-hit combinations. The identified combinations are listed in Supplementary Tables S7–S9. The three-hit combinations were able to differentiate between tumor and normal samples in a separate Test set with overall sensitivity of 90% (95% CI = 88 – 91%) and overall specificity of 93% (95% CI= 92 – 94%), as shown in Fig. 3.14(b). This was comparable to the overall sensitivity and specificity for two-hit combinations with sensitivity = 90% (95% CI = 89 – 92%) and specificity = 94% (95% CI = 93 –

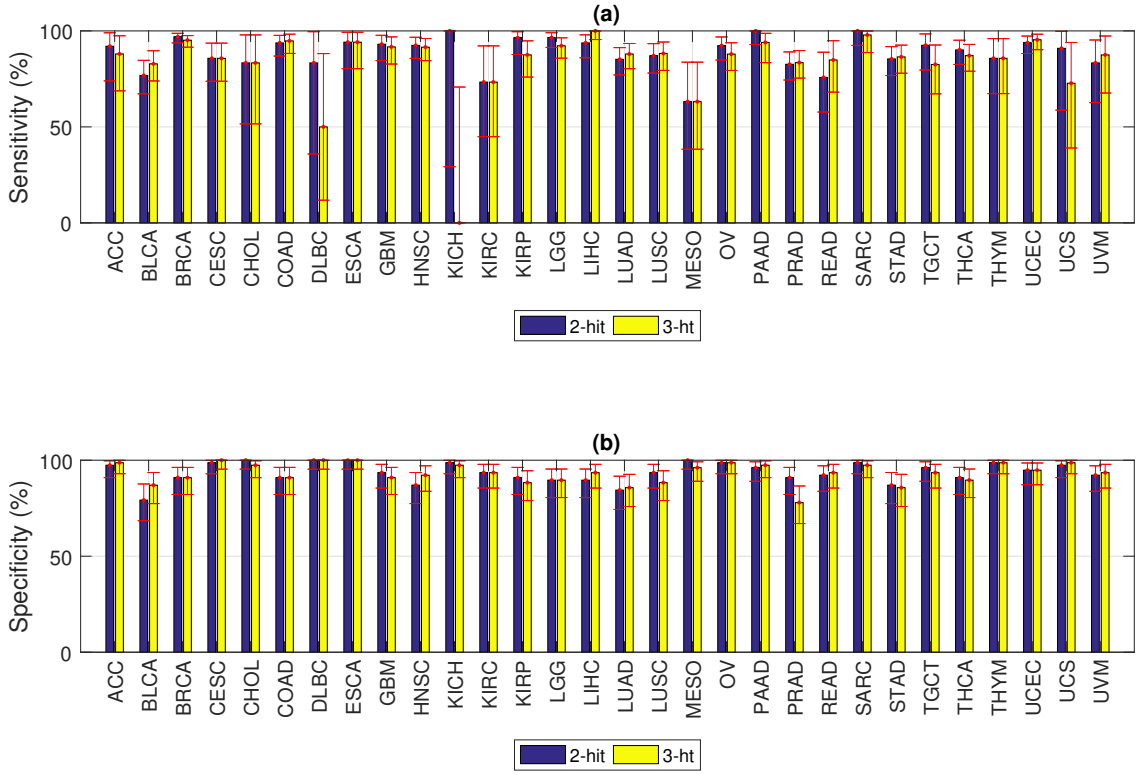


Figure 3.14: Accuracy of two- and three-hit combinations. (a) Sensitivity varies from 63–100% for two-hit combinations, and from 50–100% for three-hit combinations, excluding KICH for which there were only a total of nine tumor samples. (b) Specificity varies from 79–100% for two-hit combinations, and from 78–100% for three-hit combinations. Sensitivity and specificity were calculated on a randomly selected 25% Test data set. Error bars represent 95% CI. Cancer types with relatively large 95% CI (CHOL, DLBC, KICH, KIRP, MESO and UCS) are due to small sample size (total of 44, 43, 9, 88, 69, and 46 samples respectively).

95%), as shown in Fig. 3.14(a). The difference in average sensitivity and specificity between 2- and three-hit combinations was -6% (95% CI = -13.5 – +1.5%) and -1% (95% CI = -3.6 – +1.6%) respectively, with corresponding p-values of 0.12 and 0.44 respectively. Accuracy values are listed in Supplementary Tables S10 and S11. Since we did not see any improvement in accuracy for three-hit combinations compared to two-hit combinations, we speculate that additional accuracy improvement will require examining individual mutations within genes, as discussed below.

3.6 Distributing Large Combinatorial Workload Across Many GPUs

The approximate algorithm for identifying h-hit combinations iterates over all possible $\binom{G}{h}$ combinations to find the best combination (the combination with the maximum value of F). To compute F for $\binom{G}{h}$ gene combinations, the sequential algorithm iterates through h nested `for` loops (Algorithm 11). Computing F-value for each of these combinations is embarrassingly parallel, and in principle we can flatten these nested loops of depth h into a single `for` loop and compute the value of F for each combination in parallel, i.e. we can launch $\binom{G}{h}$ parallel threads to compute F-values. Then, the combination with the maximum value for F can be calculated by a parallel reduction in $\log\binom{G}{h}$ steps. However, the memory required to store the results from each of the $\binom{G}{h}$ threads for $h > 2$, exceeds the available memory on the V100 GPU. We address this issue by flattening only the top two `for` loops and launch $\binom{G}{2}$ threads, and assign $O(\binom{G}{h-2})$ workload to each thread (Algorithm 12). This implementation is described in detail in Ref [9].

Despite the above optimizations, four-hit combinations for most cancer types could not be identified in a practical time-frame (< 15 days), on a single GPU. Therefore, in this work we scaled out the approximate algorithm by restructuring and optimizing computational steps of the algorithm and by developing a scheduler for balanced workload distribution across GPUs and nodes of the Summit supercomputer at the Oak Ridge National Laboratory (ORNL).

3.6.1 Reducing Global Memory Access

In Algorithm 12, i, j, k, l correspond to genes in the gene-sample matrices, which reside in global memory. Reading data for these genes from global memory can stall

the threads while waiting for data transfer. To reduce the number of global memory accesses, we implemented three different memory optimizations:

1. **MemOpt1:** prefetch memories corresponding to genes i
2. **MemOpt2:** prefetch memories corresponding to genes j
3. **BitSplicing:** splice out covered samples from the tumor gene-sample matrix, after every iteration of the algorithm

Each thread in Algorithm 12 corresponds to a single unique value of λ which corresponds to a single unique combination of i and j (Algorithm 10). Although the values of k and l vary within a thread, with $i < j < k < l < G$, the values for i and j are fixed for a thread. So, instead of repeatedly accessing matrix rows corresponding to genes i and j within the loop from slower global memory, we prefetch those rows into the thread's faster local memory. Pre-fetching data for i and j (MemOpt1 and MemOpt2) reduces the global memory access during computation, and the potential for processor stall while waiting for data to be retrieved from global memory.

In each iteration of the algorithm (`while` loop in line 4 of Algorithm 12), covered tumor samples (samples that contain the combination with maximum value for F) are excluded from further consideration (line 13 of Algorithm 12). These tumor samples can be spliced out of the gene-sample matrix, reducing the size of the matrix and eliminating unnecessary memory accesses (Fig. 3.15). Combinations identified in earlier iterations tend to exclude a large number of tumor samples, so, BitSplicing can reduce the number of columns in the gene sample matrix. With every 64 samples excluded, the number of bitwise `AND` operations are reduced by three. Reduced column width of gene sample matrix effectively reduces the number of bitwise `AND` operations linearly. In the later iterations, when only a handful of samples gets excluded by each iteration's best combination, BitSplicing reduces the matrix size at a slower rate.

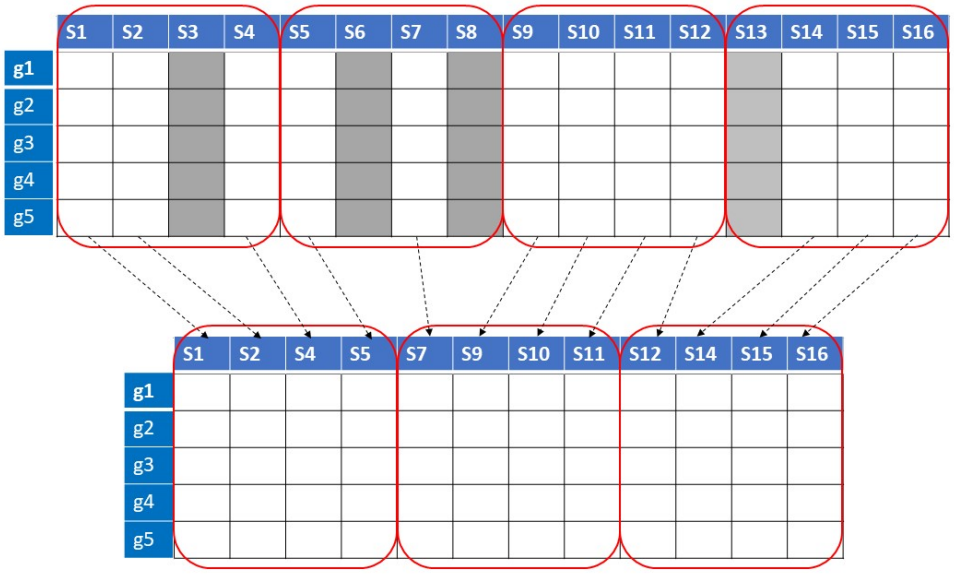


Figure 3.15: Illustrative example of BitSplicing for a simplified case of 16 samples (S1-S16) and five genes (g1-g5), where four samples are grouped together and represented by a single integer requiring a total of four integers in our compressed binary representation. Assume, the best combination identified in an iteration excludes samples S3, S6, S8, and S13. BitSplicing will splice out these bits and re-compress the gene sample matrix using only three integers per gene for next iteration. In actual implementation, we compress 64 samples into a single *unsigned long long int* variable.

3.6.2 Distributing Workload Across Nodes and GPUs

Each Summit node has two IBM Power9 CPUs and six NVIDIA V100 GPUs. For simplicity, we abstract each node as having one CPU core that uses six V100 GPU devices, and each GPU device can serve thousands of threads (Figure 3.16). The entire workload ($\binom{G}{2}$ threads) is distributed across hundreds of Summit nodes using the message passing interface (MPI) where each node serves one MPI process.

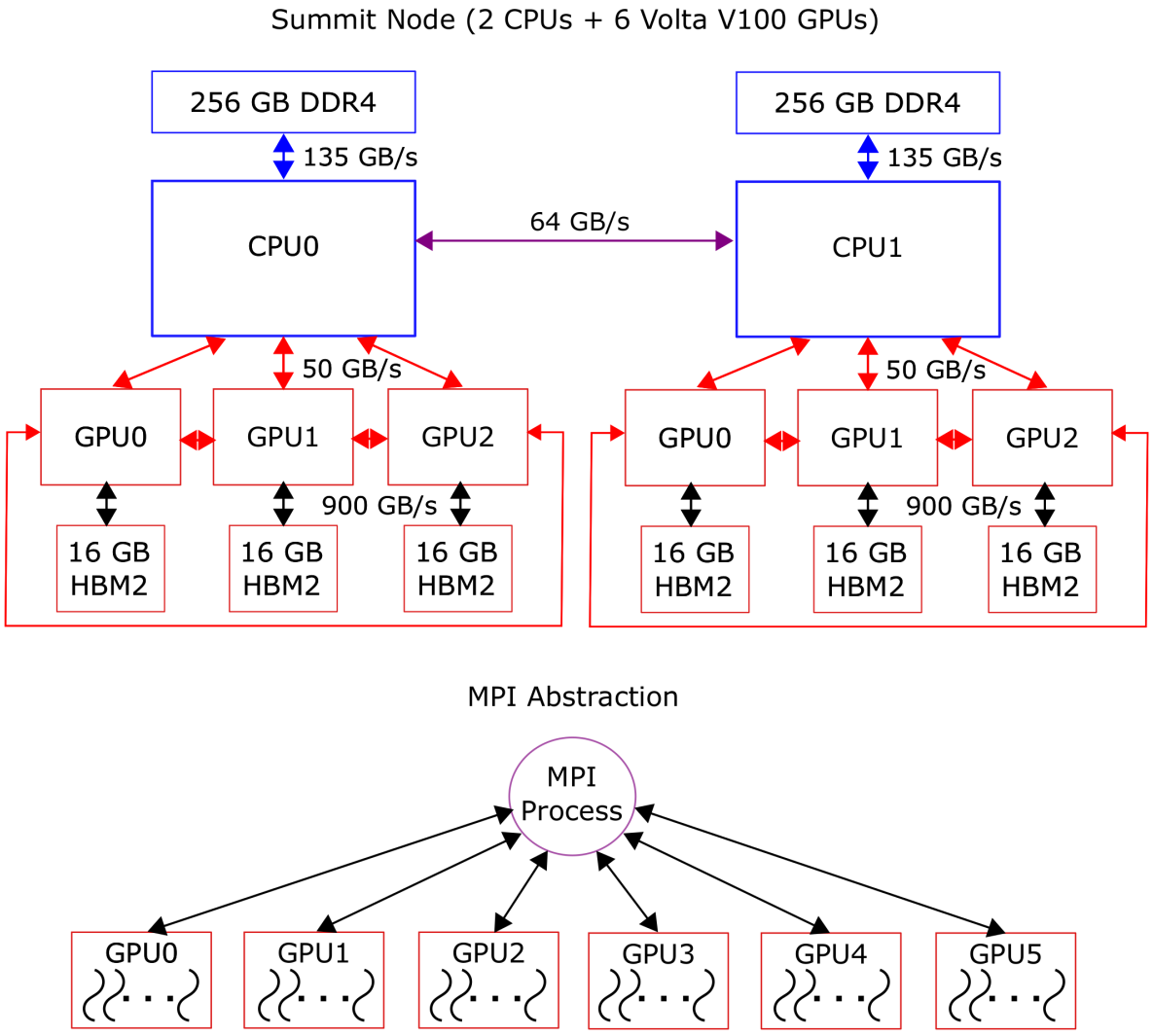


Figure 3.16: Summit node as a computational unit and its abstraction with a single MPI process per node. Top: Each Summit node consists of two IBM Power9 CPUs and six NVIDIA V100 GPUs. Bottom: Each Summit node is assigned to a single MPI process along with a range of threads (curved lines) that are in turn assigned to individual processors within the GPUs.

In Algorithm 12, different threads (λ) have different amounts of workload, e.g. the thread for $i = 1,\ j = 2$ will process $\binom{G-2}{2}$ combinations while the thread for $i = G-3,\ j = G-2$ will process only 1 combination. A naive implementation assigns equal number of threads to each node (and each GPU), which we refer to as equi-distance (ED) scheduling. A close inspection of the workload ($\binom{G-j}{2}$) vs λ shows an exponential curve with reducing amount of workload with increasing global thread id λ (Figure 3.17(a)). The area under this curve for each partition shows the total work per node. From Figure 3.17(a), we can see that areas under these different curves are very different. This will create significant load imbalance among the MPI processes.

To balance the workload we developed an alternate scheduling approach, which we refer to as equi-area (EA) scheduling. We partition the workload across the MPI processes and their GPUs based on the area under each partition's curve (Figure 3.17(b)). Assuming the workload function represented by the curve is continuous, a definite integral of the function in the domain determined by a GPU's partition will approximate the total workload assigned to that GPU.

Using the equi-area scheduler, we assign threads in the interval $[\lambda_n^{start}, \lambda_n^{end}]$ to node n. Each GPU across the nodes is assigned roughly same amount of workload. Each MPI process computes F-values for their assigned combinations and then perform a local parallel reduction to find out the combination with highest F-value. Each MPI process then sends their local best combination to the MPI process with rank 0. MPI process with rank 0 identifies the global best combination for the current iteration from the received local best combinations, then it broadcasts the global best combination to all MPI processes so that all MPI processes can apply BitSplicing to the gene sample matrices before next iteration starts.

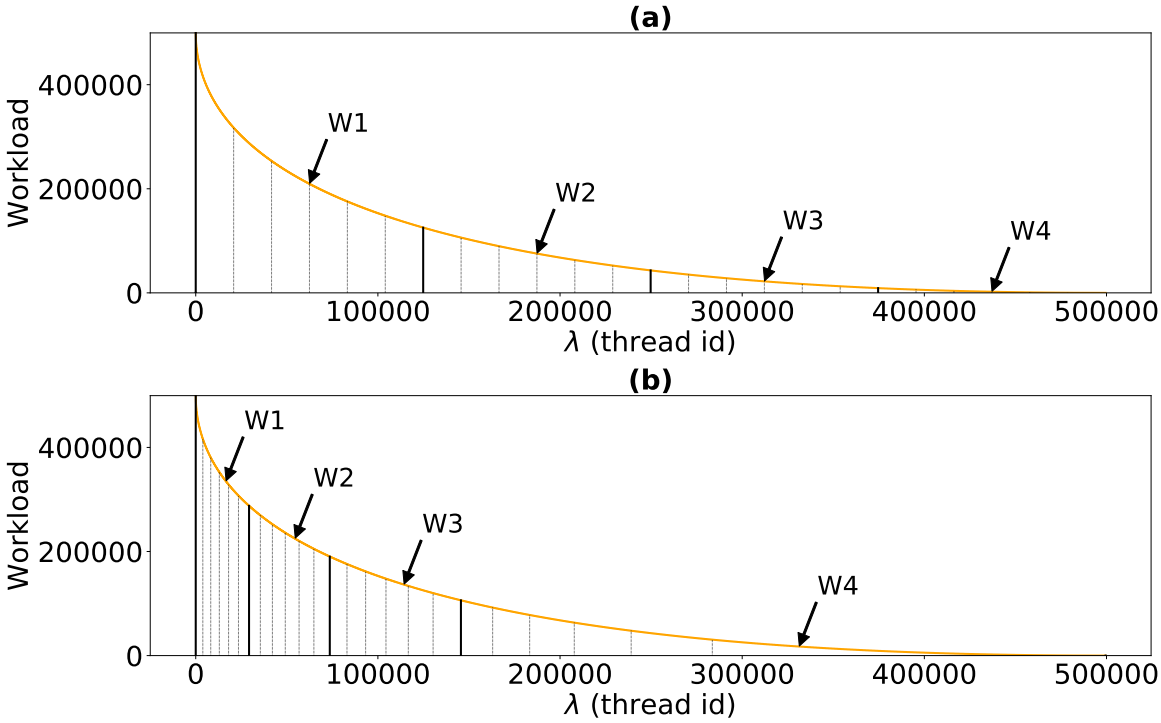


Figure 3.17: Workload distribution per node (and per GPU) for G = 1000 and four nodes. The y-axis shows the workload (number of combinations) processed by each thread. The vertical solid lines indicate the partitioning of threads (λ) across nodes and vertical dashed lines indicate partitioning of threads (λ) across GPUs. The area under the curve represents the workload Wi for each node i. (a) Partitioning for equi-distance scheduling where equal number of threads are assigned to each nodes, and equal number of threads are assigned to each GPU. (b) Partitioning for equi-area scheduling where threads are assigned to nodes so that each node and GPU have equal areas under curve.

3.7 Classification Performance and Scaling Efficiency of the Scaled-Out Algorithm

We ran the multi-hit algorithm on 100 Summit nodes (600 GPUs) to identify four-hit combinations for the 31 cancer types considered here. With the optimizations described above, we achieve an average 455-fold speedup on 600 GPUs compared to the runtime on a single GPU. The four-hit combinations were identified using a training set consisting of 75% of the sample set. See Artifact Description Appendix for details of the data used. The four-hit combinations identified by the algorithm were able to differentiate between

tumor and normal samples in the remaining 25% test set with 82% sensitivity (95% Confidence Interval (CI) = $66 - 91\%$) and 93% specificity (95% Confidence Interval (CI) = $85 - 97\%$) on average. The overall sensitivity is 84% (95% Confidence Interval (CI) = $83 - 86\%$) and the overall specificity is 93% (95% Confidence Interval (CI) = $92 - 94\%$) for all cancer types.

3.7.1 Scaling Out to 100 Nodes

The runtimes for 31 cancer types on 100 nodes are shown in Fig. 3.18(a). With 100 nodes, we can compute four-hit combinations for each of the 31 cancer types in less than two hours. These runtimes represent an average 455-fold speedup compared to single GPU runtimes (Fig. 3.18(b)). We calculated speedup using the single GPU runtime from the results reported in Ref. [9] as the baseline. However for 18 of the 31 cancer types, the single GPU computation was estimated to take more than 15 days. In these 18 cases we estimated the single GPU runtime using the average ratio, R, of actual runtimes for the four-hit algorithm versus the three-hit algorithm for the 13 cancer types that did complete in less than 15 days. This average ratio was then used to estimate runtimes for the four-hit algorithm from the corresponding three-hit algorithm runtime for the 18 cancer types that did not complete in 15 days, as $T_4 = R \times T_3$, where T_4 and T_3 are four-hit and three-hit runtimes.

For a given number of hits, the run time varies by cancer type (Fig. 3.18(a)). This difference in runtime between cancer types is determined by two main factors: the number of tumor samples and the number of multi-hit combinations identified. To calculate the value of F in Equation 3.1, the algorithm must read each of the samples to determine the number of samples that contain the combination (TP, True Positives). The Pearson correlation coefficient between runtime and the number of samples is 0.79 (Fig. 3.19(a)). The number of combinations required to cover the set of samples determines

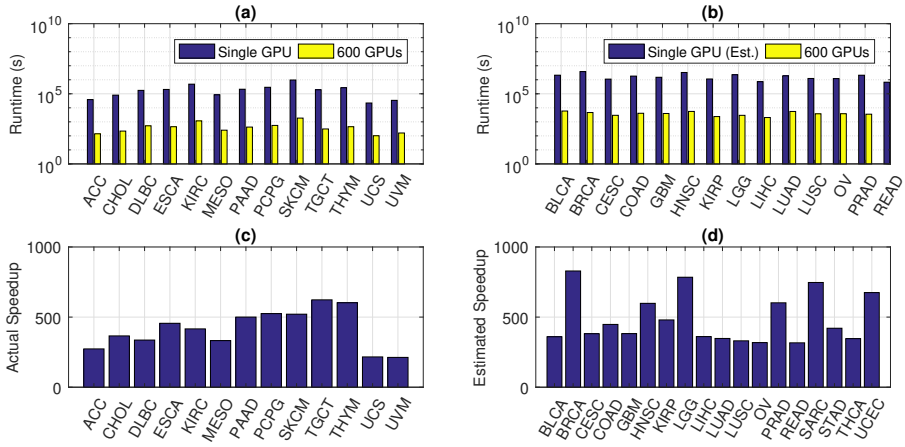


Figure 3.18: Runtime and speedup for four-hit algorithm. (a) Single GPU and 600 GPU runtimes for the 13 cancer types for which the computation finished in 15 days. (b) Runtimes for the 18 cancer types for which the computation did not finish in 15 days. The single GPU runtime was estimated from the average actual ratio of runtimes for the four-hit algorithm compared to the three-hit algorithm. (c) Actual speedup for 600 GPUs compared to single GPU runtime for the 13 cancer types in (a). (d) Estimated speedup for the 18 cancer types in (b). Names of the 31 cancer types are listed in the Artifact Description Appendix.

the number of iterations of the `while` loop in line 4 of Algorithm 12. Therefore, runtime can be longer in some cases even if the number of samples is smaller. For example, BRCA has the largest dataset (911 samples), however BLCA with 368 samples has a longer runtime (Fig. 3.18(a)). The Pearson correlation coefficient between runtime and the number of combinations is 0.92 (Fig. 3.19(b)).

Strong scaling from 50 to 100 nodes for breast invasive carcinoma (BRCA)

To minimize the effect of fixed costs on scaling efficiency calculation, we chose the cancer type with a long runtime, BRCA, for this analysis. The Summit resource allocation system limits the maximum runtime to two hours when less than 100 nodes are used. Therefore, we were restricted to a minimum of 50 nodes to complete the execution of the algorithm, which we used as the baseline for calculating scaling efficiency. With less than 50 nodes, the runtime exceeded 2 hours. We collected runtimes from 50 to 100

nodes at 10 node increments. The Scaling efficiency is 77% for 100 nodes compared to the runtime for 50 nodes (Fig. 3.20). The reason for this relatively low scaling efficiency is analyzed in Section 3.7.2 below.

Weak scaling

The strong scaling described above represents the effect of increasing the number of resources (GPUs), on runtime, for a fixed total workload. Here, we show the effect of increasing the number of GPUs, on runtime, for a fixed workload per processor. The equi-area scheduler assigns an equal amount of work to each processor. However, to ensure a fixed workload per processor we limited the runs to the first iteration (Algorithm 12), since depending on the number of nodes used, the later iterations produce varying amount of workload. The average weak scaling efficiency for BRCA is 80.6% for 1 to 100 nodes (Figure 3.21).

Effect of memory optimization on runtime

We evaluated the effect of the three memory optimization strategies, described in Section 3.6.1, using the BRCA dataset for the three-hit algorithm running on a single GPU. Together, prefetching data for samples associated with gene i (MemOpt1), prefetching data for samples associated with gene j (MemOpt2) and splicing out data for samples associated with covered gene combinations (BitSplicing), result in a 3-fold speedup (Fig. 3.22).

Comparison between runtimes of two schedulers

Based on a test of the four-hit algorithm for two cancer types, breast invasive carcinoma (BRCA) and esophageal carcinoma (ESCA), equi-area scheduler (EA) achieves a 3-4x

speedup over equi-distance scheduler (ED) (Table 3.3), by virtue of its more balanced workload, as described in Section 3.6.2.

Table 3.3: Runtime comparison between two scheduling approaches. Test of four-hit algorithm for two cancer types breast invasive carcinoma (BRCA) and esophageal carcinoma (ESCA). The equi-area (EA) scheduler is three-four times faster than the equi-distance (ED) scheduler.

Cancer Type	ED Time	EA Time	Speedup
BRCA	13942.7	4606.71	3.02
ESCA	1573.06	448.82	3.50

Effect of different block-sizes

To test the effect of block-size on runtime, we varied the block-size from 64 to 512 threads, for identifying a single four-hit combination for BRCA. The best runtime was obtained for a block-size of 128 threads, however the block-size did not significantly effect the runtime (Table 3.4).

Table 3.4: The effect of block size on runtime is not significant. Test to identify a single four-hit combination for BRCA shows that the best runtime was for a block-size of 128 threads. Average runtime = 1088 with standard deviation = 14.0.

Block-size (threads)	64	128	256	512
Runtime (seconds)	1092.24	1072.48	1082.99	1106.14

3.7.2 Compute Utilization and Analysis of Its Variance Across GPUs

The equi-area scheduler distributes approximately the same amount of workload, as measured by the number of combinations processed, across MPI processes, which are served by different nodes. The workload distributed among GPUs within each node for different MPI processes is also approximately the same. However, individual threads within a GPU and across GPUs will have different workloads. This combined with

different memory access patterns for different GPUs can result in a less than ideal strong scaling efficiency (77% for 100 nodes compared to 50 nodes, Fig. 3.20).

We analyzed compute utilization for the 600 GPUs in a 100-node run for the cancer type Adenoid cystic carcinoma (ACC) (Fig. 3.23), to identify the reason for lower utilization. Choosing ACC enabled us collecting performance metrics within the time-limit constraint. We used metrics on DRAM read/write throughput and instruction issue efficiency to analyze the variance of compute utilization cross GPUs. In general, utilization decreases with the increasing GPU index, with spikes in utilization around GPU #372, #504, and #560. GPUs with lower utilization represent processors that have completed their assigned work faster and are idle while the first GPU, with 100% utilization, is still running. Our analysis shows that compute utilization primarily depends on memory read/write throughput.

DRAM read/write throughput

Figures 3.23(a) and (b) show that Compute utilization up to GPU #500 is inversely correlated with DRAM read/write throughput. Although each GPU is assigned approximately the same workload, the range of memory accessed by threads within those GPUs decreases exponentially. For example, the thread with thread-id $\lambda = 0$ accesses $G \approx 20000$ different memory locations, while the thread with $\lambda = \binom{G}{2} - 2$ accesses only three memory locations. DRAM read/write thoughput is an indication of the number of cycles required for successful memory accesses. Above GPU #500 read/write throughput continues to increase without a corresponding decrease in utilization, indicating a transition of processor bottleneck from being memory bound to being compute bound. During this transition read/write throughput still affects utilization but to a smaller extent. This is reflected in the smaller spikes in utilization corresponding to spikes in read/write throughput for GPU index > 500.

Instruction issue efficiency

To further understand how DRAM read/write throughput affects compute utilization, we analyzed instruction issue efficiency. Thread blocks assigned to a GPU are assigned to its streaming multiprocessors (SM) and their execution is scheduled in groups of 32 threads (warp). The execution of these warps can be stalled if all necessary resources are not available or if all dependencies have not been satisfied. A breakdown of the stalled cycles shows three major contributors: memory dependency, memory throttle, and execution dependency (Fig. 3.23(c)). Stalls due to memory dependency indicate that resources required for load/store from memory are not available. Stalls due to memory throttle indicate that excessive pending memory operations are preventing further execution. Stalls due to execution dependency indicate that input data required for the instruction is not yet available. DRAM read/write throughput affects all three of these factors resulting in reduced compute utilization.

3.7.3 Classification Performance of the Identified Four-Hit Combinations

We identified 314 four-hit combinations for 31 cancer types, using a 75% training dataset. THCA had the largest number of combinations (20) and THYM, PAAD, and ESCA had the smallest (3). Figure 3.24 shows the top three four-hit combinations identified for low grade glioma (LGG) by our algorithm. The figure shows the location of each gene within the chromosome in which it is located, with each four-hit combination connected by curved lines of the same color. Names of the 31 cancer types and all of the four-hit combinations identified are listed in the Artifact Description Appendix.

To evaluate the quality of the identified four-hit combinations, we built a classifier per cancer type. The classifier measures the accuracy (sensitivity and specificity) with

which the four-hit combinations for each cancer type can differentiate between tumor and normal samples. For a given cancer type, let the set of combinations be $c_1, c_2, \ldots, c_p$. The classifier will classify a sample as a tumor sample if that sample has mutations in all the genes of any one of the combinations c_i $(1 \leq i \leq p)$. If there is no such combination, the sample will be classified as a normal sample. Using these per-cancer classifiers, we evaluate the classifiers' performance on the test dataset for each of the 31 cancer types. These classifiers achieve 82% sensitivity (95% Confidence Interval (CI) $= 66 - 91\%$) and 93% specificity (95% Confidence Interval (CI) $= 85 - 97\%$) on average (Fig. 3.25). Most existing approaches do not report comparable classification performance, since the focus is generally on discovery and characterization of cancer genes and driver mutations. We did find one study (ContrastRank) that reported classification performance for three cancer types [172], which are summarized in Table 3.5. However, ContrastRank identifies individual driver genes, but not the combinations. Table 3.5 also lists corresponding sensitivity and specificity for four-hit, three-hit and two-hit combinations from this study and Refs [48] and [9], respectively.

Table 3.5: Comparison of classification performance of the multi-hit algorithm and the ContrastRank method.

	Sensitivity/Specificity (%)			
Cancer Type	Four-hit Comb	Three-hit Comb	Two-hit Comb	Contrast Rank
COAD	96/91	95/91	90/97	86/97
LUAD	83/91	83/86	91/92	96/97
PRAD	77/86	83/78	88/79	91/92

3.8 Conclusion and Discussion

Cancer is many different diseases, although the symptoms may be similar. These different diseases are a result of different combinations of genetic defects (hits). In this study

we have developed a method for identifying combinations of genes with mutations that may be responsible for different instances of cancer. Our method is fundamentally different from current approaches which identify individual genes, instead of combinations of genes, in which mutations increase the likelihood of carcinogenesis.

The problem of identifying a set of multi-hit combinations that can differentiate between tumor and normal samples was mapped to the extensively studied weighted set cover (WSC) problem. We adapted a WSC algorithm to the problem of identifying multi-hit combinations. The algorithm was applied to a training set of somatic mutation data from the cancer genome atlas (TCGA) to identify a set of two-hit combinations for the 17 cancer types with at least 200 matched tumor tissue and blood-derived normal samples.

The resulting two-hit combinations were able to differentiate between tumor and normal tissue samples in a separate test set with over 90% sensitivity and specificity on average. Accuracy of the results were robust to different random partitionings of the available data between training and test sets. The resulting set of combinations include potential novel cancer genes, not previously implicated in cancer.

Next, we presented optimization and parallelization techniques that allowed us to extend the algorithm to identify three-hit combinations, and some four-hit combinations. The three-hit combinations are able to differentiate between tumor and normal samples with overall 90% sensitivity (95% CI = 88–91%) and 93% specificity (95% CI = 92–94%). We illustrate how the distribution of somatic mutations in these genes can be used to identify potential driver mutations for further investigation. For example, we identified two protein-altering somatic mutations in the KCNB1 gene which occur significantly more frequently in TCGA ovarian cancer samples compared to normal samples (p-value$<$0.0001), suggesting that these mutations may be positively selected for in ovarian cancer. However, further experimental validation is required to deter-

mine if these mutations represent novel cancer driver mutations, or are simply passenger mutations.

Most cancers however require an estimated four – nine hits. To be able to identify combinations of more than three hits, we restructured and optimized the algorithm for parallel execution across multiple GPUs on multiple nodes of the Summit supercomputer at the Oak Ridge National Laboratory. Parallel execution of the optimized algorithm across 100 Summit nodes (600 GPUs) resulted in an average 455-fold speedup compared to the runtime on a single GPU, allowing us to identify four-hit combinations for the 31 cancer types considered here.

The declining scaling efficiency with the number of nodes indicates that extending the algorithm beyond four-hit combinations will require further restructuring and optimization. In addition, analysis of the multi-hit combinations identified, shows that many of the genes contain passenger mutations rather than driver mutations, indicating a need for reexamining some details of the algorithm design. Specifically, the algorithm should be modified to search for combinations of individual mutations within genes, instead of combinations of genes with mutations.

With twenty times as many protein-altering mutations as there are genes, and the exponential computational complexity of the algorithm, significant additional scaling up of the algorithm will be required. Despite these limitations, the multi-hit combinations identified do include known cancer genes, suggesting that with further refinement, the algorithm has the potential for identifying combinations of cancer causing mutations.

Here we discuss how the multi-hit combinations identified can be used to identify carcinogenic (driver) and non-carcinogenic (passenger) mutations within genes. We also illustrate how these combinations may be used to design a combination therapy targeting the specific genetic mutations responsible for individual instances of cancer.

3.8.1 Distinguishing Between Driver and Passenger Mutations

The method used to identify multi-hit combinations uses a mutation frequency based approach to preferentially select driver genes instead of passenger genes, since the selected genes have a significantly higher mutation frequency in tumor samples compared to normal samples. For each gene, the mutation frequency in normal samples is considered to be approximately representative of the background mutation frequency for the gene. However, within these genes not all mutations are carcinogenic.

The two-, three-, and four-hit combinations found provide a starting point for examining a smaller subset of genes more closely to identify specific carcinogenic mutations within these genes. In identifying the multi-hit combinations, we did not take into consideration the location of mutations within genes. Clearly there are locations within a gene where certain mutations are unlikely to affect the function of the gene product. Such mutations can result in false positives and contribute to the large number (65%) of tumor samples containing multiple combinations (Fig. 3.7). Consider for example, the two-hit combination of IDH1 and MUC6 in brain lower grade glioma (LGG) tumor samples.

Of the 479 LGG tumor samples, 134 (28%) contain mutations in both IDH1 and MUC6, while 5 (1.5%) of 333 normal tissue samples contain a mutation in both these genes (Fig. 3.26). Comparing the mutations within these genes for normal and tumor samples may reveal which are carcinogenic and which are not. In this example, every one of the tumor samples contains a missense mutation at R132 in IDH1 and no other mutations, while the normal samples do not contain any mutations at this position (Fig. 3.26). Mutations at R132 in IDH1 have previously been implicated in cancer [111].

On the other hand, the IDH1 mutations seen in the normal samples are unlikely to be carcinogenic. Similarly, mutations at F1989 of MUC6, which occur most frequently in both tumor and normal samples are unlikely to be carcinogenic (Fig. 3.26). Excluding

such non-carcinogenic mutations can reduce the number of false positives and further increase accuracy of our algorithm. In our future work we will develop an automated method to compare and contrast the individual gene loci, so that all of these mutations within genes can be identified. To further improve accuracy of our algorithm, variants that are likely to be carcinogenic can be weighted higher than those that are unlikely to be carcinogenic.

Some of the genes identified by our approach may not be causative (passenger mutations) even though they may be correlated to cancer incidence. Functional analysis can be used to identify genes in the above set of combinations that are unlikely to be driver genes, even though they may be frequently mutated in tumors.[24, 52, 122] For example, the affect of specific mutations on gene expression levels can be analyzed to determine if the mutation is likely to have a functional effect. In addition we can analyze the pathways affected by the gene combinations (Table A.7). Studies show that combinations of driver gene mutations generally affect mutually exclusive pathways.[106] One of the genes in a multi-hit combination affecting the same pathway may include passenger mutations. Although in most cases multiple different pathways are affected by the gene combinations, Table A.7 shows that in some cases (e.g. MUC6 and MUC12) the same pathway is affected by both genes in the combination. Further analysis would be required to determine if the mutations within one of these genes are passenger mutations.

The search algorithm can be run iteratively to incrementally refine the list of multi-hit combinations by excluding these passenger mutations. The input to our algorithm is a list of genes with mutations for each sample. Genes with only passenger mutations can be excluded from this list to minimize the inclusion of passenger mutations in the resulting multi-hit combinations.

3.8.2 A Rational Basis for Combination Therapy

The two-, three-, and four-hit combinations identified, with further refinement and clinical validation, may represent a more rational basis for targeted combination therapy, instead of the current "marriages of convenience" [105] with limited biological rationale [10]. A more rational strategy may also reduce the risk of expensive failures such as the phase III trial of imfinzi plus tremelimumab. The combination of therapies for a given patient could be designed to target specific carcinogenic combinations of gene mutations found in the patient. Although only 30 of the 256 genes in the combinations identified above were formally identified as "cancer genes" in the catalog of somatic mutations in cancer (COSMIC), many of the other genes were previously implicated in cancer (Table 3.2).

Therapies that target many of the genes in both these categories may be available or under development. For example, the combination of mutations in TP53 and IGHG1 occur in 41% of HNSC tumor samples in TCGA. Therapies targeting TP53 (Styrylquinazoline) Several drugs that can restore TP53 function, deplete mutant TP53 or affect downstream targets are currently in pre-clinical development [139]. siRNA targeted silencing of IGHG1 has been shown to inhibit cell viability and promote apoptosis, which might therefore act as a potential target in cancer gene therapy [135, 188]. For patients with this combination of mutations, a combination therapy targeting both these genes may be more effective in combination, than separately.

3.8.3 Identifying Combinations of Gene Mutations From the Identified Gene Combinations

Not all mutations within a cancer gene are oncogenic [36, 100, 122, 146]. However to make the problem of identifying multi-hit combinations tractable, the algorithm

searched through all possible gene combinations, instead of all possible combinations of mutations. In the tumor sample data used, there were over 400,000 unique somatic mutations across ∼20,000 genes. It is theoretically possible to search all possible combinations of 400,000 protein altering somatic mutations instead of combinations of 20,000 genes with somatic mutations. However, searching all possible combinations of 400,000 mutations would increase the computational complexity for identifying three-hit combinations by over six orders of magnitude, making the problem computationally intractable. In addition, since there can be multiple different carcinogenic mutations within a gene, combinations of individual mutations will occur less frequently than combinations of genes with mutations, further increasing the challenge of identifying carcinogenic combinations within this much larger set of possible combinations. Therefore, we chose to first focus on combinations of genes with mutations. Mutations within these gene combinations can then be examined to identify potential driver mutations for further investigation, as illustrated below.

Consider for example, the two- and three-hit combinations identified for ovarian serous cystadenocarcinoma (OV) (Figs. 3.27 and 3.28). The most commonly occurring two- and three-hit combination are TP53+KCNB1 and TP53+KCNB1+TTN respectively. Mutations in TP53 and KCNB1 occur in 279 of 317 OV tumor samples and mutations in TP53, KCNB1 and TTN occur in 271 of 317 OV tumor samples. The distribution of protein altering somatic mutations in TP53, KCNB1 and TTN for the 271 OV tumor samples containing mutations in all three genes are shown in Figs. 3.29(a), 3.30(a), and 3.31(a), respectively. The distribution of protein altering somatic mutations in TP53, KCNB1 and TTN for 333 normal samples are shown in Figs. 3.29(b), 3.30(b) and 3.31(b), respectively. The difference in the frequency of individual mutations between tumor and normal samples may suggest potential driver mutations for further investigation.

The TP53 gene codes for the Tumor Protein P53. Mutations in TP53, a tumor suppres-

sor gene, have been extensively implicated in many cancers, including OV [8, 56, 57, 66, 75, 155]. In the 271 OV tumor samples containing the TP53+KCNB1+TTN three-hit combination, TP53 contains on average 1.8 protein altering somatic mutations per sample, compared to 0.15 mutations per sample in normal samples, with clear differences in the distribution of these mutations (Fig. 3.29). The three most frequently occurring mutations in the tumor samples (amino acid positions R248, R273, and R175) are potential driver mutations, since they rarely occur in normal tissue (Fig. 3.29). The mutation frequency at R248, R273 and R175 are 0.08, 0.07 and 0.06 per tumor sample, compared to 0.00 per normal sample (p-value < 0.0001 for the difference in proportions). In fact, previous studies have shown that the R248W, R273H and R175H mutations not only cause a loss of P53-based tumor suppressor activity, but also result in genomic instability causing gain of oncogenic activity [34, 110, 164]. On the other hand the two most frequently mutated amino acid positions in normal samples, T377 and T378, are likely to be passenger mutations. Normal tissue mutation frequencies of 0.07 and 0.05 per normal sample are comparable to tumor tissue mutation frequencies of 0.04 and 0.05 per tumor sample for T377 and T378, respectively (Fig. 3.29).

The KBNB1 gene codes for the Potassium Voltage-Gated Channel Subfamily B Member 1 protein. KCNB1 has been previously identified as a prognostic factor in gliomas due to its tumor suppressor function [181]. It contains on average 2.14 protein altering somatic mutations per tumor sample in the 271 OV samples containing the TP53+KCNB1+TTN three-hit combination, compared to 0.03 mutations per normal sample (p-value < 0.0001) (Fig. 3.30). The two most frequently occurring mutations at K776 and R736 are potential driver mutations worthy of further investigation. The mutation frequencies at these positions are 1.37 and 0.41 per tumor sample compared to 0.00 and 0.003 per normal sample, respectively (Fig. 3.30). Although KCNB1 has been extensively studied, primarily in the context of epilepsy [29, 103, 118, 123, 151, 171], these studies do not include either of the two mutations identified here. These two

mutations occur in the unstructured C-terminus cytoplasmic tail region of this transmembrane potassium channel protein [118, 151]. Further *in vitro* investigation will be required to understand how these mutations may affect the expression, structure or function of this protein, to determine if these could be driver mutations.

The TTN gene codes for the Titin protein of striated muscle. TTN expression level has been previously identified as prognostic marker for Ewing's sarcoma [53], and TTN mutations have been associated with several myopathies [43, 85, 90, 93, 191]. Titin is a large protein consisting of 34,350 amino acids, with a correspondingly large number of mutations, 15.37 protein altering somatic mutations per tumor sample and 3.98 mutations per normal sample, on average (Fig. 3.31). Three of the most frequent mutations in TTN in tumor samples, C21862G, E1656G and T2963P, occur more frequently in tumor samples compared to normal samples, suggesting that these may be potential driver mutations that should be investigated further. The mutation frequencies at these amino acid positions are 0.17, 0.20 and 0.20 per tumor sample, compared to 0.06, 0.003, and 0.03 per normal tissue sample, respectively (Fig. 3.31). Although TTN mutations have been extensively studied, primarily in the context of myopathies [43, 85, 90, 191], these studies do not include any of the three mutations identified here.

The above only provide a starting point for further investigation. The positive selection implied by the higher mutation frequencies seen above are confounded by several factors, including tumor microenvironment, tissue and cell type, epigenetic modifications, gene expression and co-expression, etc. [11, 104, 119, 120, 157, 167, 180]. A more detailed analysis of the potential driver mutations identified above using available literature, gene expression data, copy number variation, associated pathways, functional annotation, protein localization, etc., could provide additional evidence to either support or reject the mutation as a driver mutation. This information can be iteratively incorporated into the search algorithm described in Methods. We expect that excluding likely passenger mutations will reduce the number of false positives and prioritizing

likely driver mutations will reduce false negatives, improving the accuracy of the combinations identified. However, this could also potentially limit the discovery of novel genes.

Note that these somatic mutations were calculated using *protected* whole exome sequencing data from tumor samples with matched blood-derived normal samples. For tumor samples, protected somatic mutation data (MAF files) were downloaded from the cancer genome atlas (TCGA) with permission. Somatic mutations for normal tissue samples with matched blood-derived normal samples were called using the same protocol used by TCGA, as described in the methods. Variants called using matched blood-derived normal data identifies significantly more mutations than the number of variants called without matched blood-derived normal samples, for the following reasons [152, 166]. Biopsy specimens contain a mix of tumor and normal tissue cells, tumor-infiltrating lymphocytes, and stromal cells. In addition, tumor cells themselves can be genetically diverse. As a result, mutations in a subset of the cell population will present at a relatively low frequency. Using blood derived normal samples as a reference allows for the identification of such low frequency variants. Variants that could potentially lead to de-identification of donors ($\sim$80 million variants) are considered "protected" data in TCGA, and are not accessible by tools such as cBioPortal and TCGA queries that are based on "open" access data ($\sim$3 million variants). For example, the protected TCGA MAF files contain 617 protein-altering somatic mutations in TP53 in 317 OV samples, compared to the 276 somatic mutations reported by cBioPortal using open access data [32].

3.8.4 Beyond Four-Hit Gene Combinations

Although our algorithm may be able to identify mutations that contribute to cancer progression, many of the mutations in the gene combinations identified are likely to be

passenger mutations. To identify combinations of true oncogenic mutations will require searching for specific combinations of mutations within genes instead of combinations of genes with mutations.

Based on the computational complexity of the multi-hit algorithm , extending the four-hit algorithm from combinations of $\sim 2 \times 10^4$ genes to combinations of $\sim 4 \times 10^5$ protein-altering mutations will require a computational speedup of $\sim 10^5$ relative to the estimated single GPU runtime for the optimized code presented above. In addition, identifying combination of each additional number of hits will require an additional speedup of $\sim 4 \times 10^5$, e.g. going from four-hit combinations to 5-hit combinations. An additional challenge presented by mutation level combinations is that the input mutation-sample matrices are 20 times larger than gene-sample matrices, representing increased latency due to additional global memory access requirements. Following are four of many possible strategies that can help address these challenges. (1) Parallelize execution across the 27,648 GPUs on the Summit supercomputer. (2) Reduce workload imbalance by linear mapping of threads to the upper tetrahedral domain of mutation combinations. (3) Balance compute utilization through an on-demand scheduler addressing memory accesses (4) Minimize memory latency by distributing only the required subset of mutation-sample matrices to each GPU. (5) Limit combinations to the most probable oncogenic mutations. Following is a brief description of each of these strategies

(1) Scaling of the current implementation has been evaluated for up to 100 Summit nodes (600 GPUs). The following additional optimizations should also be tested on a limited number of nodes, to ensure that scaling efficiency does not drop off to the point where additional nodes do not provide significant additional speedup. The goal would be to ensure high scaling efficiency up to the 27,648 available GPUs on Summit.

(2) When computing the value of the objective function F, the multi-hit algorithm

only needs to consider mutation combinations $\{m_i, m_j, \ldots, m_h\}, 1 < i < j < \cdots \leq N$, where m_i is mutation i, h is the number of hits and N is the number of all possible mutations. In Algorithm 10 each thread was uniquely mapped to combinations of two genes. Although this mapping is near-optimal for two-hit combinations, it is not optimal for three or more hits. Navarro et al. [129] derived a mapping of the tetrahedral domain, where $i < j < k \leq N$, to a linear thread index. This mapping can improve processor utilization.

(3) The variability in memory access patterns across nodes limits node utilization. By breaking the total work into finer grained workloads by considering the memory access patterns, and assigning them to idle nodes, we can balance the compute utilization across nodes and improve scaling efficiency. Memory coalescing and tiling while breaking down the total work into fine-grained workloads can benefit this on-demand scheduler in achieving high processor utilization.

(4) A range of threads (λ) process a limited range of combination, as seen in Algorithm 10. Therefore, only this limited subset of the mutation-sample matrix is need by these threads. The subset corresponding to the range of threads associated with each GPU can be calculated and only that subset of data copied to GPU global memory. The reduced input matrix size can reducing memory latency.

(5) Mutations in many genes are unlikely to be oncogenic, e.g. pseudogenes, genes not expressed in the cell of origin, known olfactory receptor genes, etc. Such domain knowledge can be used to limit the number of mutations N. However, orders of magnitude larger reductions in the number of mutations will be required with increasing number of hits. Bayesian variable selection can be used to filter out genes that are highly unlikely to be carcinogenic. Weghorn and Sunyaev have previously developed a hierarchical Bayesian framework for identifying likely cancer genes from germline mutation data [183]. Their software can be adapted to rank and limit mutations to the ones most

likely to be oncogenic.

The speedup from these optimizations can allow the identification of combinations of four or more oncogenic mutations.

Additional Information Additional information is available in Appendix A and the companion file $Dash_{SS}upplementaryTablesS1 - S112020.xlsx$.

Algorithm 10 Parallel algorithm to compute three-hit combinations.

Require: *tumor-sets*, *normal-sets*, *tumor-samples*, *normal-samples*, α

covered-samples $\leftarrow \Phi$
combinations $\leftarrow \Phi$
$N_t \leftarrow \left|tumor\text{-}samples\right|$
$N_n \leftarrow \left|normal\text{-}samples\right|$
while *covered-samples* $\neq$ *tumor-samples* **do**
    $F_{max}[1 \ldots \binom{G}{2}] \leftarrow [-\infty \cdots -\infty]$
    for parallel: $\lambda = 1 \rightarrow \binom{G}{2}$ **do**
        $F_{max}[\lambda] \leftarrow -\infty$
        $j \leftarrow \left\lfloor \sqrt{1/4 + 2\lambda} + 1/2 \right\rfloor$
        $i \leftarrow \lambda - j(j-1)/2$
        for $k = j + 1 \rightarrow G$ **do**
            $TP \leftarrow \left|tumor\text{-}sets[i]) \cap tumor\text{-}sets[j]) \cap tumor\text{-}sets[k])\right|$
            $FP \leftarrow \left|normal\text{-}sets[i]) \cap normal\text{-}sets[j]) \cap normal\text{-}sets[k])\right|$
            $TN \leftarrow N_n - FP$
            $FN \leftarrow N_t - TP$
            $F \leftarrow \left(\frac{\alpha \times TP + TN}{N_t + N_n} \right)$

            if $F \geq F_{max}[\lambda]$ **then**
                $F_{max}[\lambda] \leftarrow F$
                best-combinations$[\lambda] \leftarrow \langle i, j, k \rangle$

    best-combination, $F_{max} \leftarrow$ *parallel-reduction*(*best-combinations*$[1 \ldots \binom{G}{2}], F_{max}[1 \ldots \binom{G}{2}]$)

    combinations.add(*best-combination*)
    $i, j, k \leftarrow$ *best-combination.extract*()
    covered-samples.add(*tumor-sets*$[i]) \cap$ *tumor-sets*$[j] \cap$ *tumor-sets*$[k]$)
return *combinations*

Algorithm 11 Computing four-hit combinations with depth-4 nested for loops.

Require: *tumor-samples*, *normal-samples*, α

```
1:  covered-samples ← Φ
2:  answer ← Φ
3:
4:  while covered-samples ≠ tumor-samples do
5:      combinations ← Φ
6:      for i = 1 → G do
7:          for j = i + 1 → G do
8:              for k = j + 1 → G do
9:                  for l = k + 1 → G do
10:                     combinations.add(Combination(i, j, k, l))
11:
12:     answer.add(best(combination))
13:     covered-samples.update(tumor-samples, best(Combination))
14: return answer
```

Algorithm 12 Computing four-hit combinations with depth-3 nested for loops.

Require: *tumor-samples*, *normal-samples*, α

```
1:  covered-samples ← Φ
2:  answer ← Φ
3:
4:  while covered-samples ≠ tumor-samples do
5:      combinations ← Φ
6:      for λ = 1 → (G choose 2) do
7:          i, j ← unpack(λ)
8:          for k = j + 1 → G do
9:              for l = k + 1 → G do
10:                 combinations.add(Combination(i, j, k, l))
11:
12:     answer.add(best(combinations)
13:     covered-samples.update(tumor-samples, best(combinations))
14: return answer
```

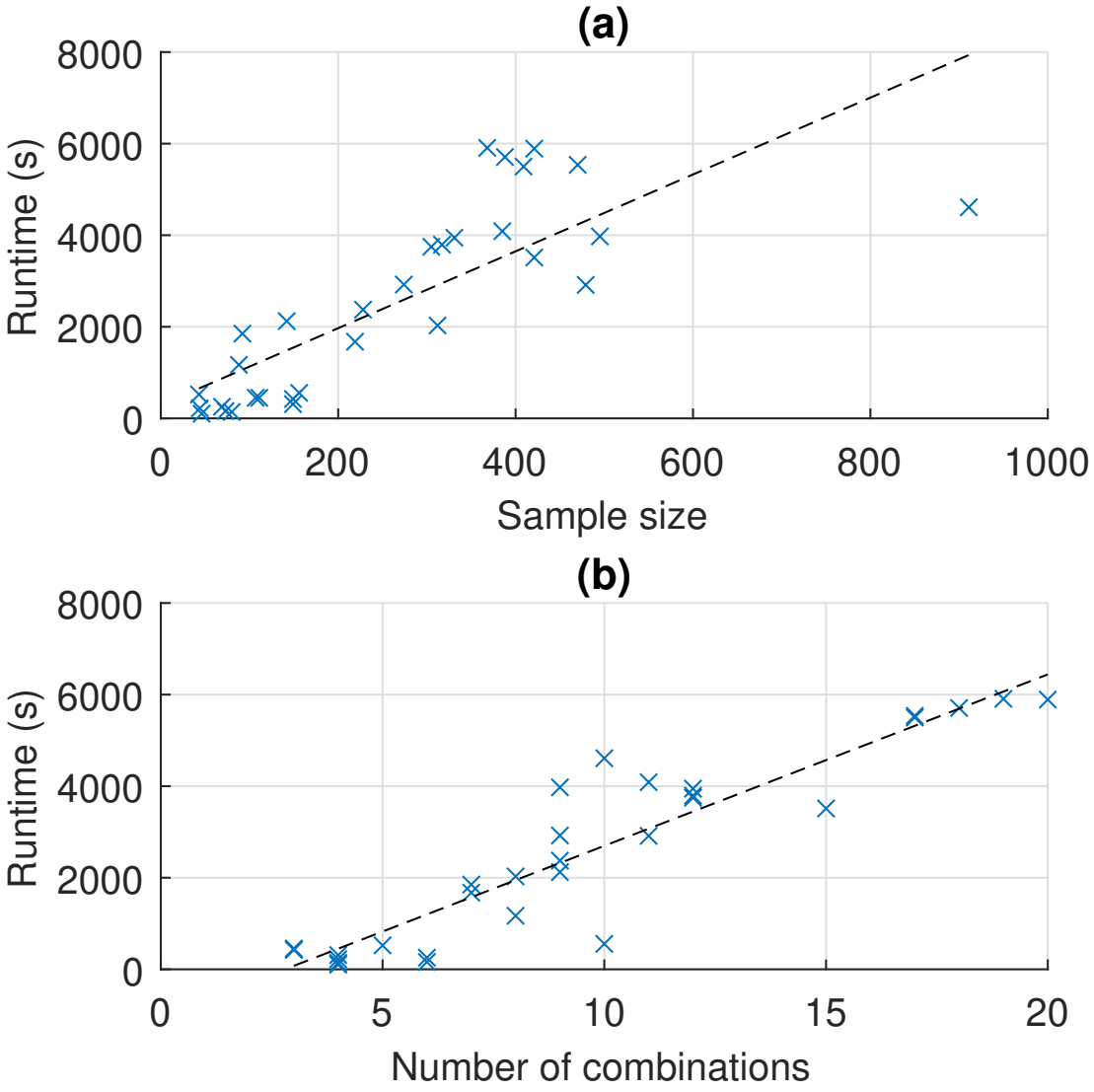


Figure 3.19: Runtime of different cancer types is highly correlated with sample size and number of combinations. (a) Correlation coefficient = 0.79 for sample size. (b) Correlation coefficient = 0.92 for number of combinations.

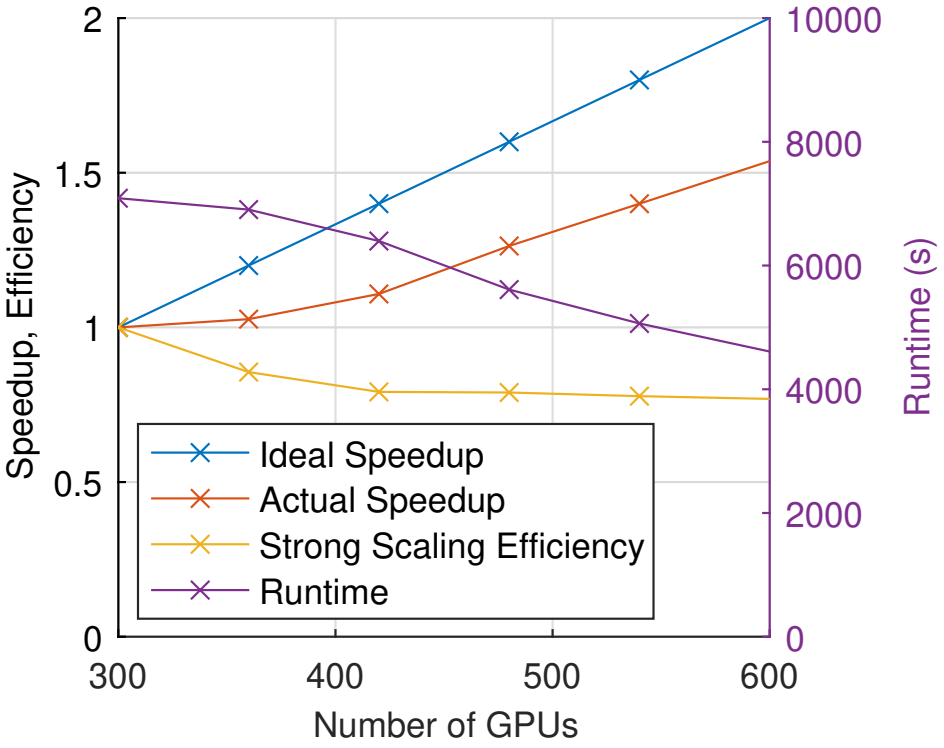


Figure 3.20: Strong scaling from 50 nodes (300 GPUs) to 100 nodes (600 GPUs) for the BRCA dataset. Scaling efficiency is 0.77 for 100 nodes relative to a baseline of 50 nodes.

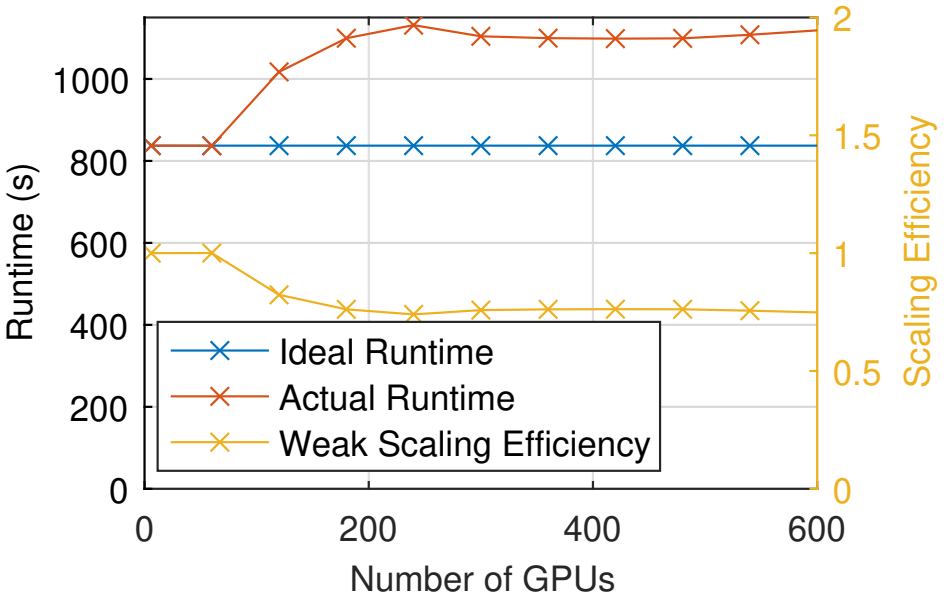


Figure 3.21: Weak scaling from six GPUs (one node) to 600 GPUs (100 nodes). The scaling efficiency is 80.6%. The runtime starting from 30 nodes remains almost unchanged.

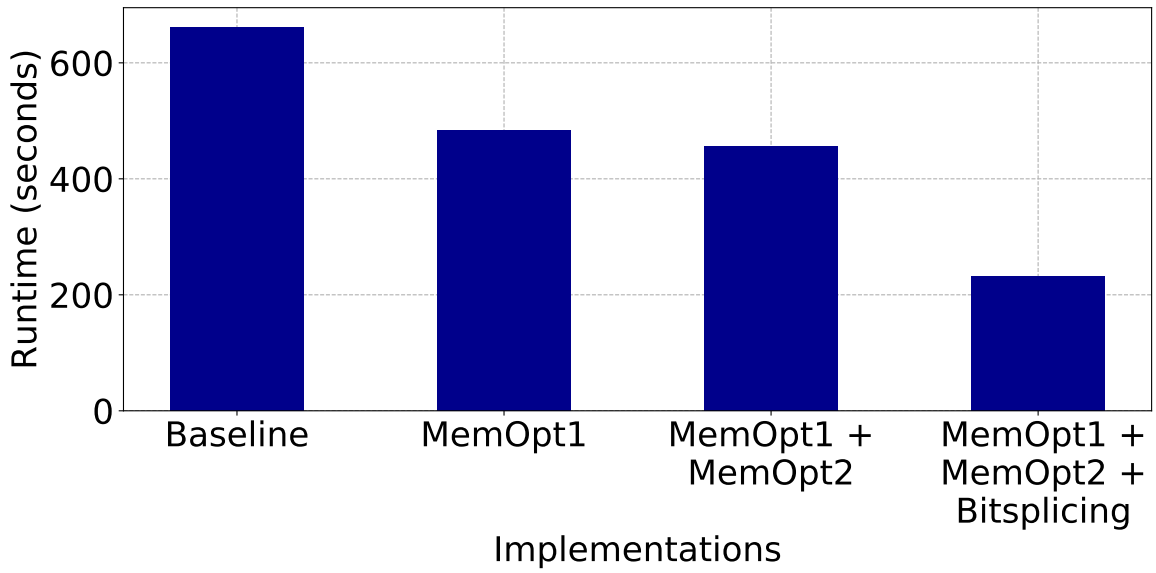


Figure 3.22: Effect of three memory optimizations on runtime. Tested on the three-hit algorithm running on a single GPU, for the breast invasive carcinoma (BRCA) dataset.

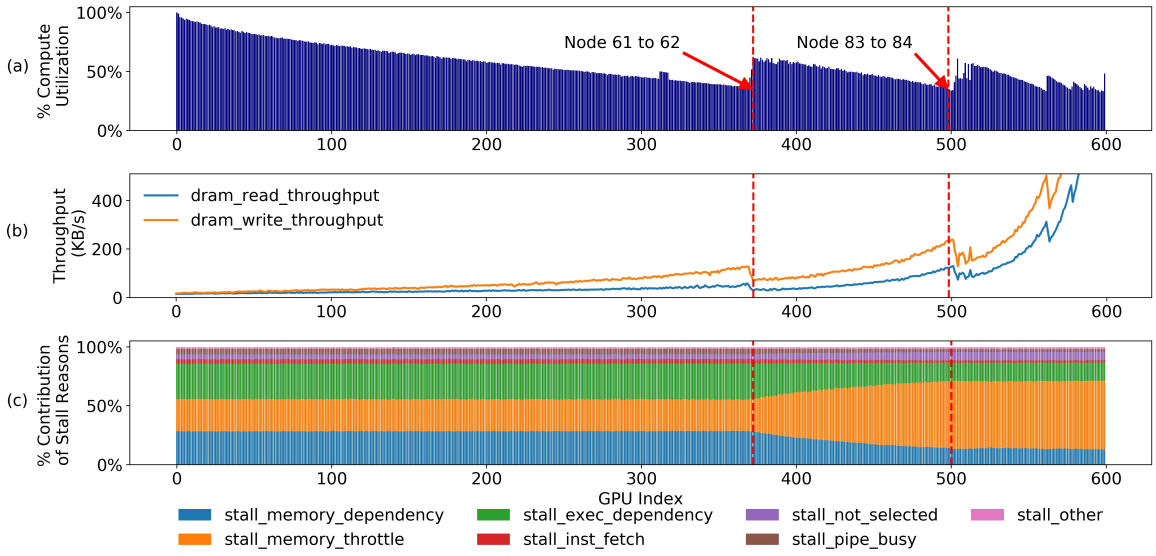


Figure 3.23: (a) Compute utilization is inversely correlated with DRAM read/write throughput up to GPU #500 (b). Above GPU #500, read/write throughput increases and the processor transitions from being memory bound to being compute bound. (c) Low read/write throughput stalls warp execution while data from memory is accessed.

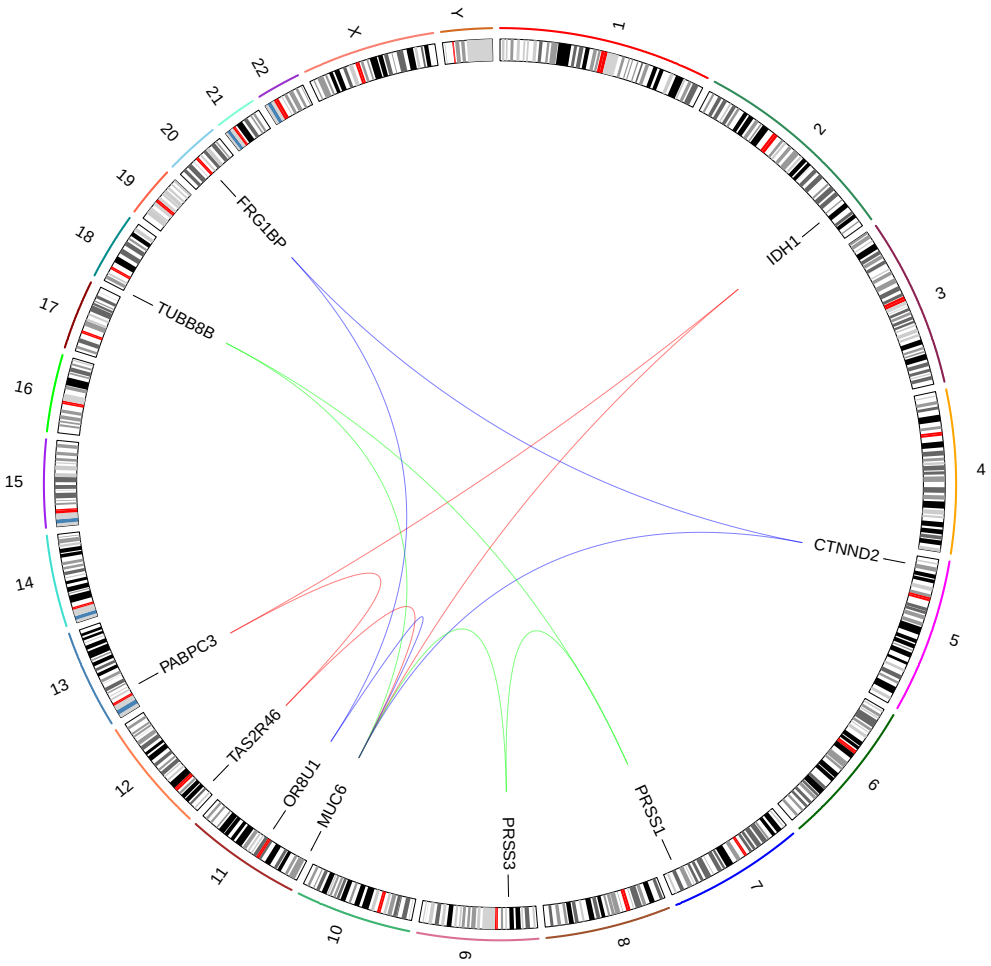


Figure 3.24: Top three four-hit combinations for low grade glioma (LGG). Each four-hit combination is represented by four curves of the same color connecting the four genes in the combination. The outer circle shows individual chromosomes with corresponding ideograms shown in the inner circle. Genes that comprise four-hit combinations are labeled inside the circle.

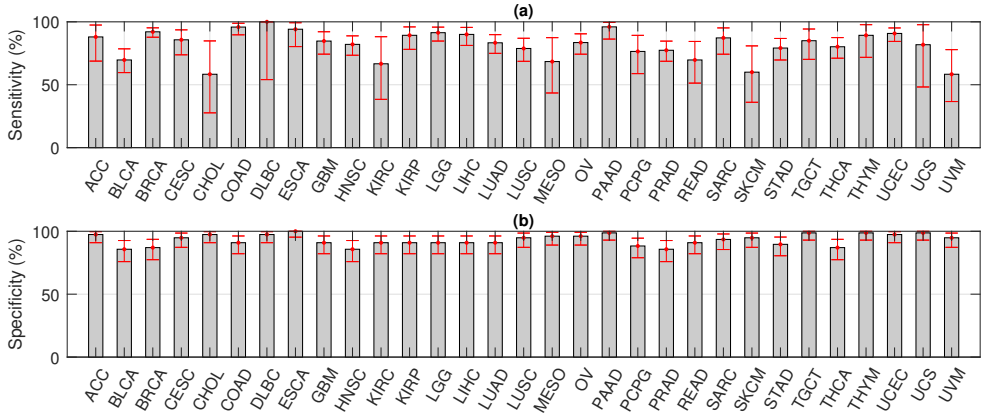


Figure 3.25: Classification performance of the identified combinations. Four-hit combinations were identified using a training dataset consisting of randomly selected 75% of the available tumor and normal samples. Classification performance measured by sensitivity (a) and specificity (b) was based on the remaining 25% test dataset. For the 31 cancer types considered here, average sensitivity and specificity were 82% and 93% respectively. Error bars represent 95% confidence interval (CI).

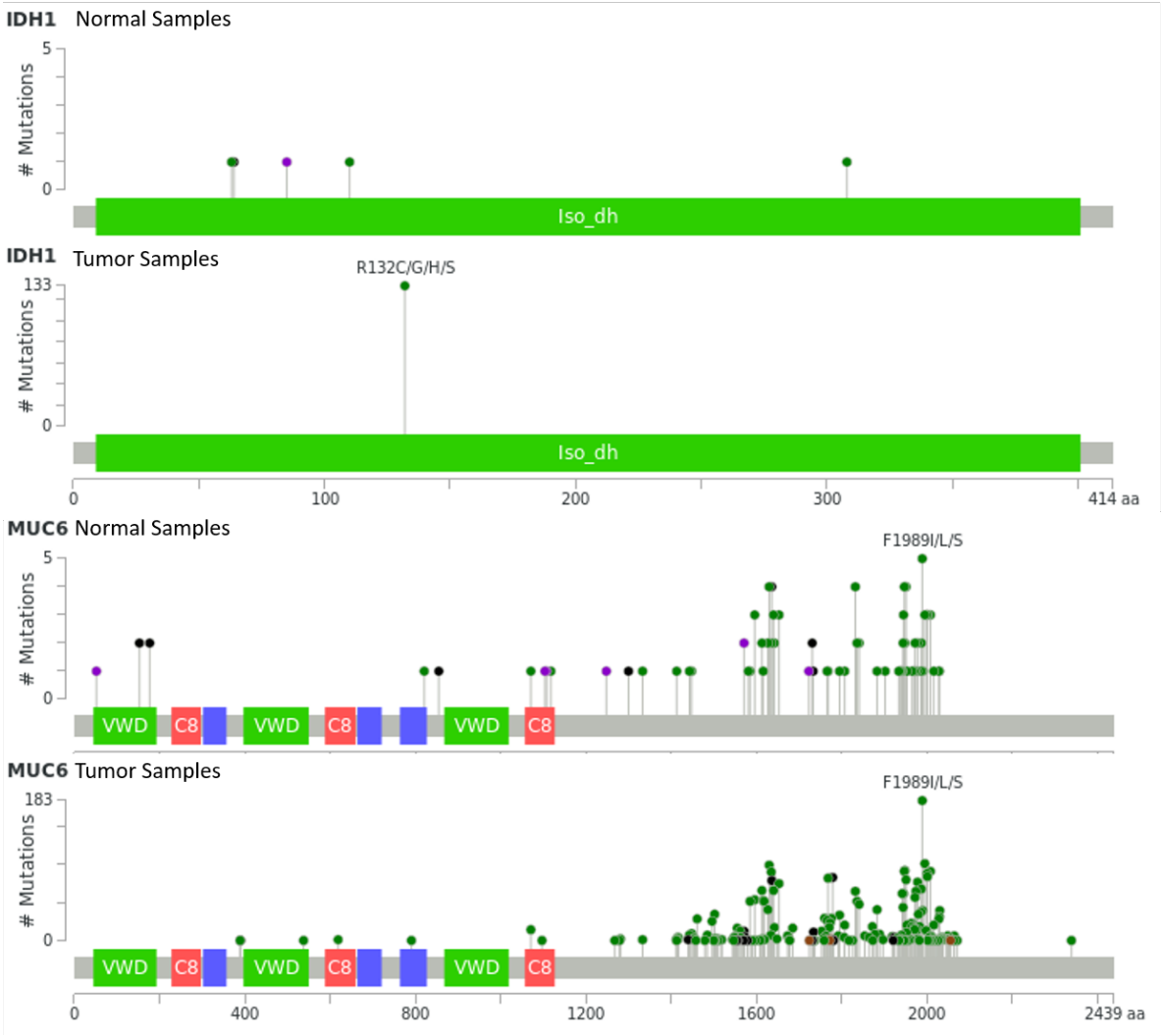


Figure 3.26: Mutations in normal and lower grade glioma (LGG) tumor samples with mutations in both IDH1 and MUC6. The difference in mutations between normal and tumor samples for the same two-hit combination can be used to further refine the search algorithm. In the above examples, a missense mutation at R132 in IDH1 is likely to be carcinogenic, whereas mutations at F1989 in MUC6 are unlikely to be carcinogenic. Colored bars represent known functional protein domains. Grey bars represent regions of unknown function. Green dots represent missense mutations, black dots represent truncating mutations and purple dots represent other protein-altering mutations. Figure generated using cBioPortal [32, 63].

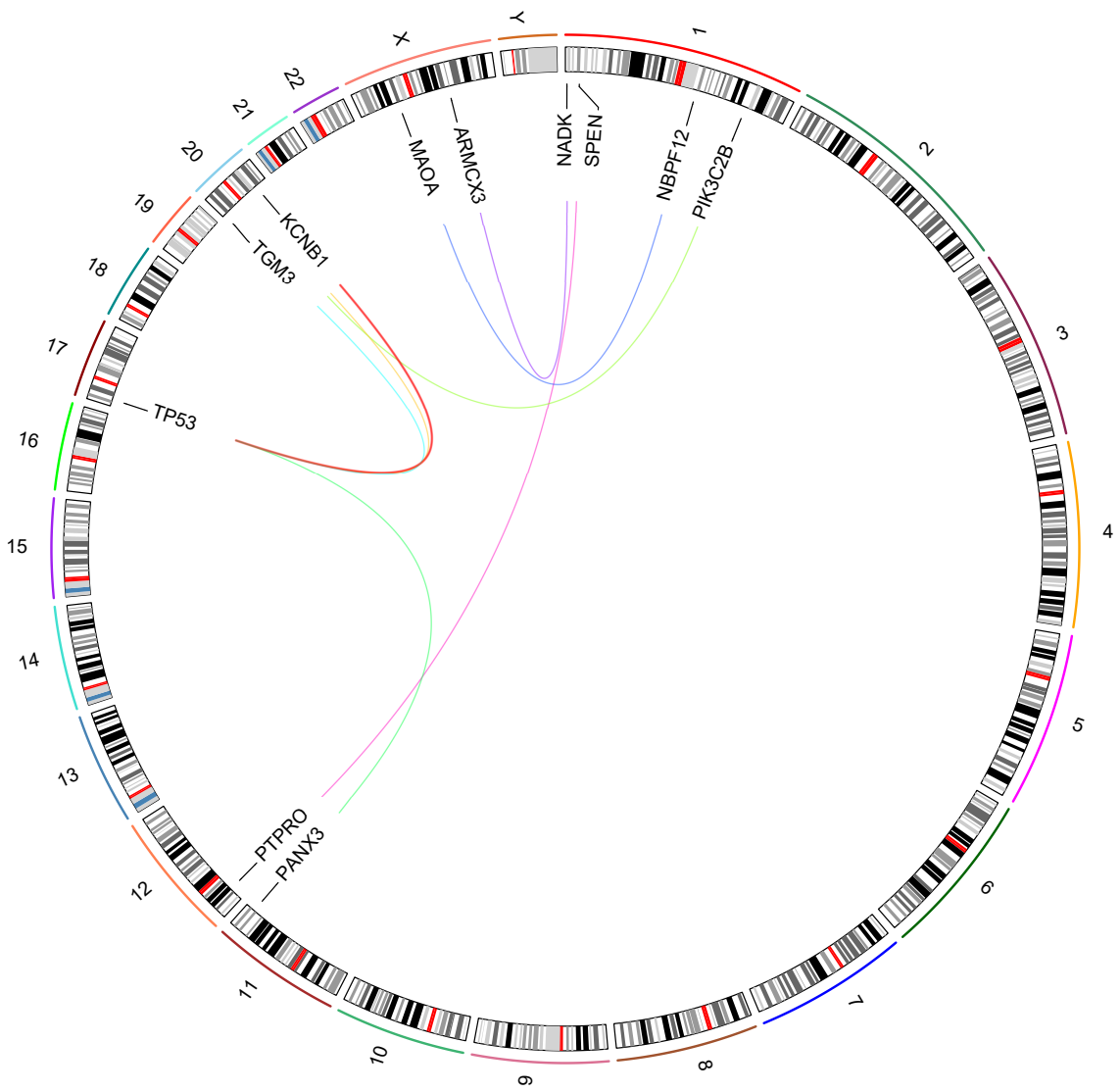


Figure 3.27: Two-hit combinations identified for ovarian serous cystadenocarcinoma (OV). The outer circle shows individual chromosomes with corresponding ideograms shown in the inner circle. Genes that comprise two-hit combinations are labeled inside the circle. Each two-hit combination is identified by differently colored lines connecting two genes. The red line represents the gene combination discussed in further detail. Images generated using RCircos [192].

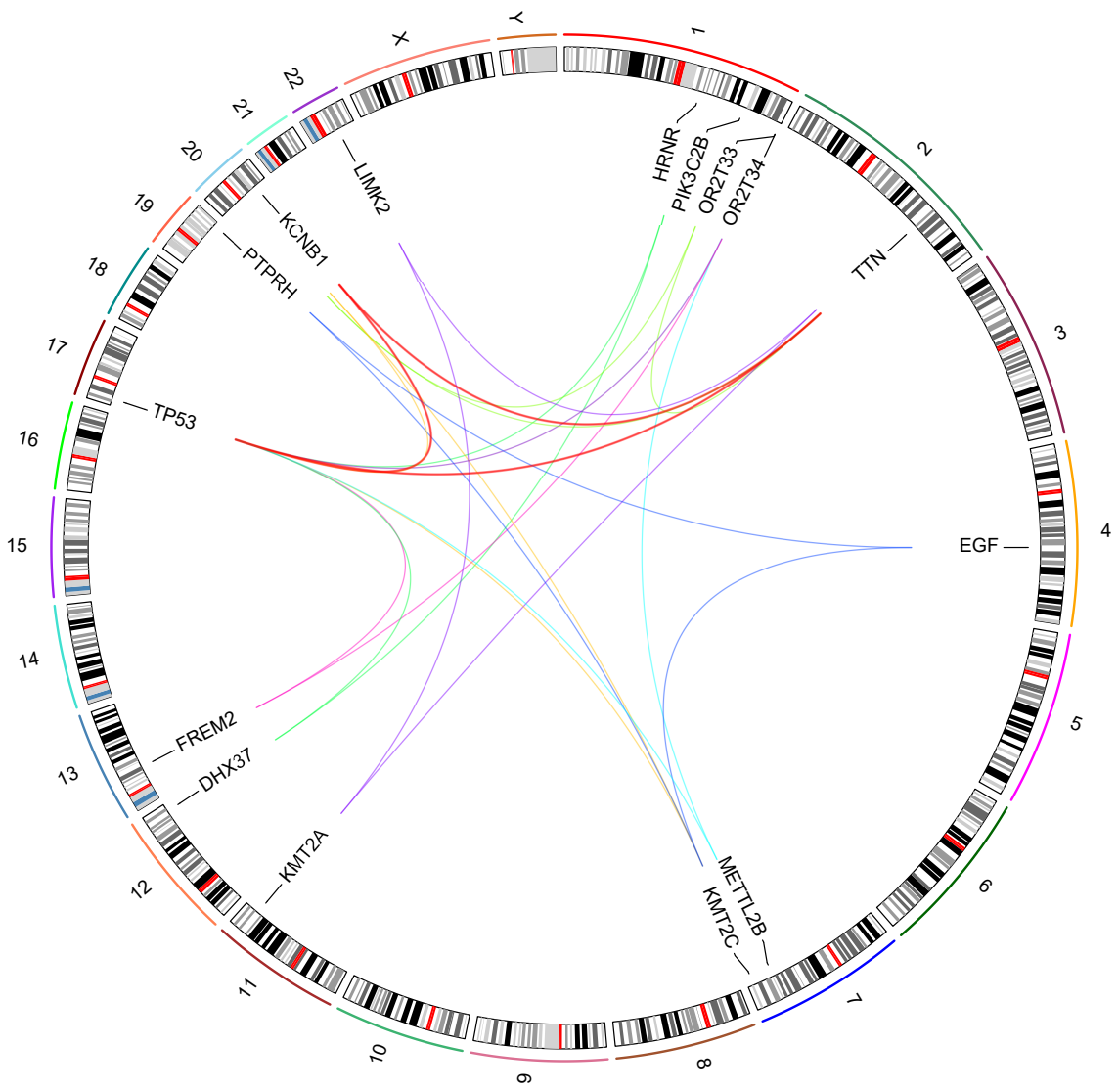


Figure 3.28: Three-hit combinations identified for ovarian serous cystadenocarcinoma (OV). The outer circle shows individual chromosomes with corresponding ideograms shown in the inner circle. Genes that comprise three-hit combinations are labeled inside the circle. Each three-hit combination is identified by differently colored lines connecting three genes. The red line represents the gene combination discussed in further detail. Images were generated using RCircos [192].

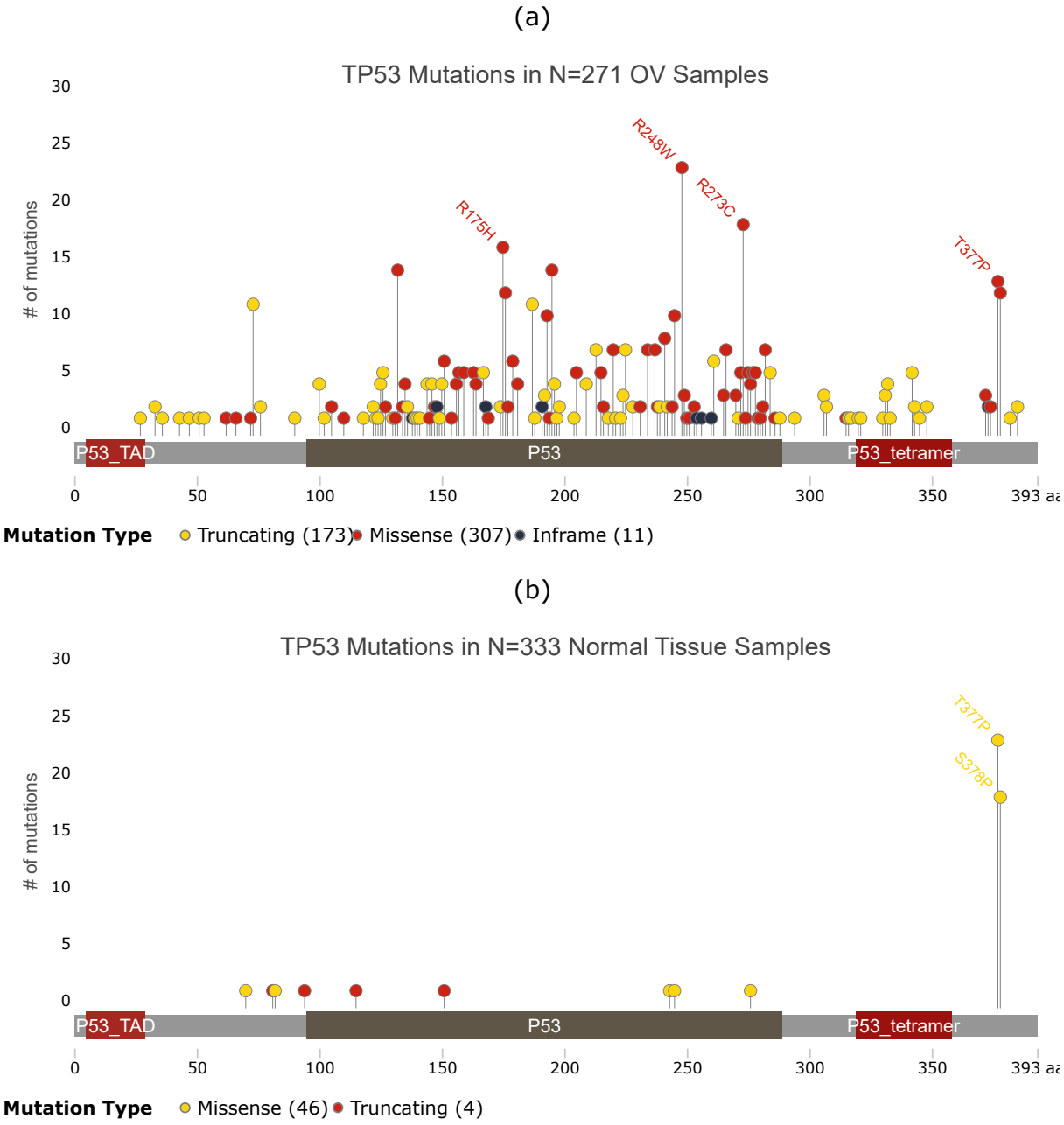


Figure 3.29: Distribution of somatic mutations in TP53 in ovarian tumor samples and normal samples. The horizontal bar shows amino acid position within the protein, with labels showing known functional domains. Vertical lines show the number of samples with protein altering mutations at each amino acid position. The most frequently mutated sites for each gene in (a) tumor and (b) normal samples are labeled for comparison. Image generated using g3viz [73].

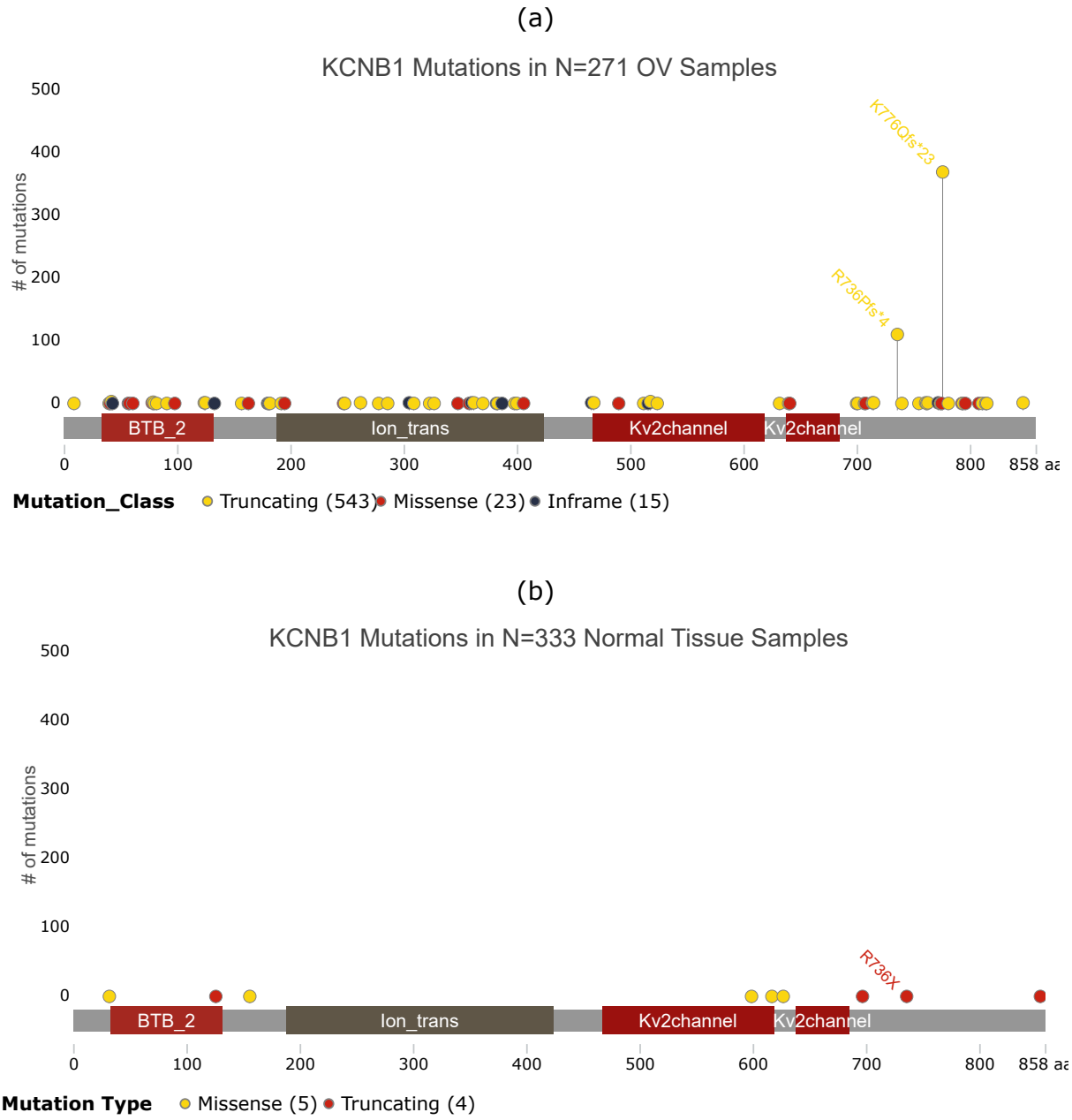


Figure 3.30: Distribution of somatic mutations in KCNB1 in ovarian tumor samples and normal samples. The horizontal bar shows amino acid position within the protein, with labels showing known functional domains. Vertical lines show the number of samples with protein altering mutations at each amino acid position. The most frequently mutated sites for each gene in (a) tumor and (b) normal samples are labeled for comparison. Image generated using g3viz [73].

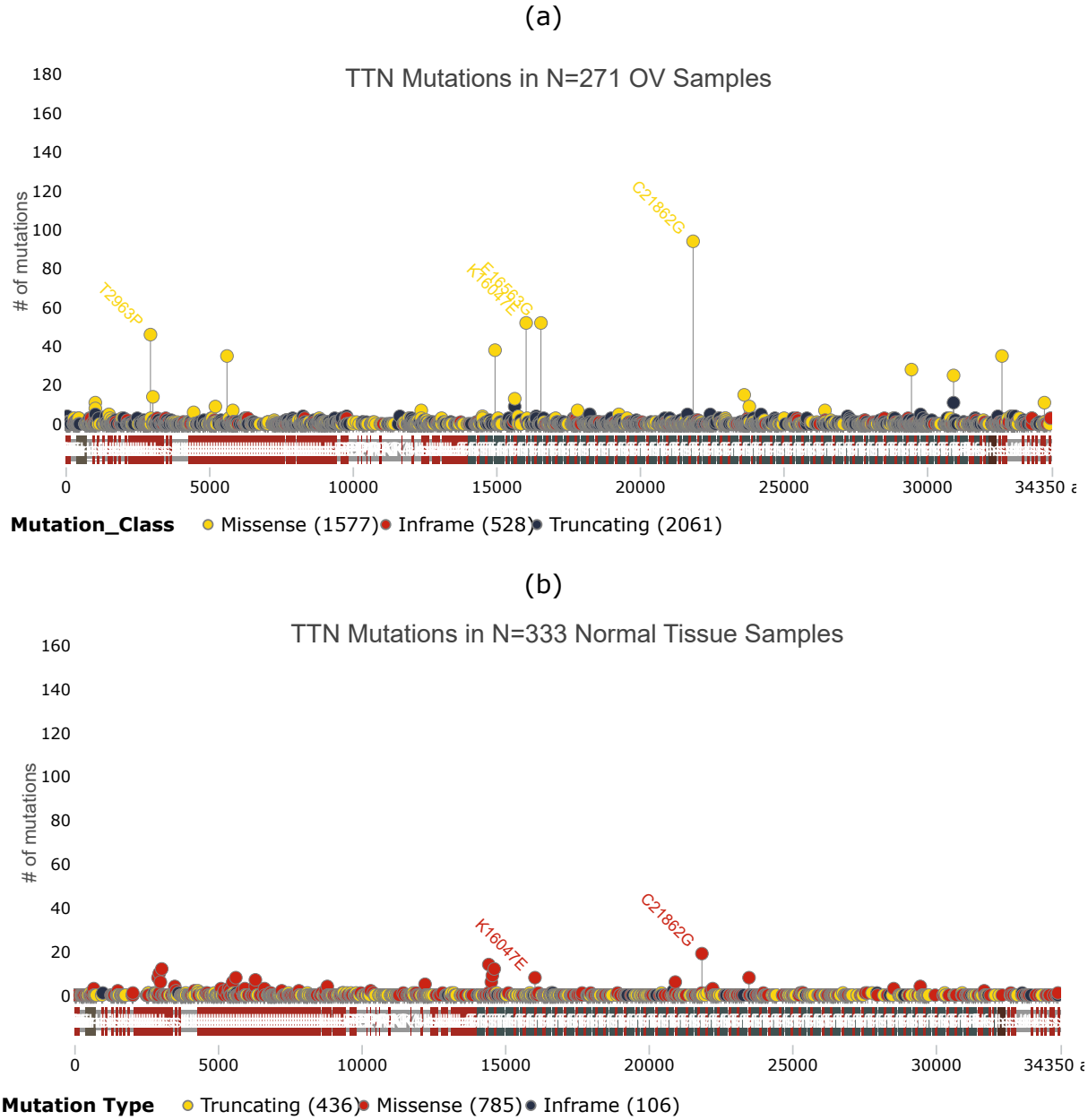


Figure 3.31: Distribution of somatic mutations in TTN in ovarian tumor samples and normal samples. The horizontal bar shows amino acid position within the protein, with labels showing known functional domains. Vertical lines show the number of samples with protein altering mutations at each amino acid position. The most frequently mutated sites for each gene in (a) tumor and (b) normal samples are labeled for comparison. Image generated using g3viz [73].

Chapter 4

Incremental Sequence Similarity Search via Automated E-Value Correction

4.1 Introduction

Utilization of a sequence similarity search tool is a central step in most bioinformatics research investigating biological or structural functions of nucleotide or protein sequences. BLAST*, short for Basic Local Alignment Search Tool [12] is a widely used (*$75,905$ *citations, February* 2019*) sequence alignment tool that is capable of conducting a sequence similarity search for a sequence of interest against a curated sequence database. BLAST relies on a heuristic approach for searching and provides results based on the identification of regions through seed-and-extend based local alignment, which can either be implemented using a web interface [87] or a set of standalone command-line tools maintained by the National Center for Biotechnology (*NCBI*) [30]. It uses a statistical threshold called an expect value (i.e., E-value) to infer homologous sequences from a curated database. BLAST is extensively used for the identification of unknown sequences, the detection of candidate genes, and during the annotation of assembled genomes and transcriptomes.*

Sequencing data stored in the NCBI database has expanded astronomically over the

years, reportedly doubling in the number of bases submitted to GenBank [22] every year over the last three decades (1982-present), as shown in Figure 4.1. Cheaper sequencing technology [68], as shown in Figure 4.1, the democratization of high-performance computing (HPC) through commercial cloud platforms [134], availability of efficient genomic/transcriptomic assemblers (e.g., StringTie [142], Trinity [69], ABySS [160], SOAPdenovo2 [115]), and annotation pipelines (e.g., NCBI prokaryotic genome annotation pipeline [170], MAKER [31]) and efforts to sequence many new taxa (e.g., Earth BioGenome Project [108], The i5k Initiative [80], BAT 1K [163], and The Genome 10K Project [97]) are some of the major reasons facilitating this growth. Fast-accumulating sequences in NCBI-curated databases have a profound impact on the computational efforts required to perform sequence similarity search.

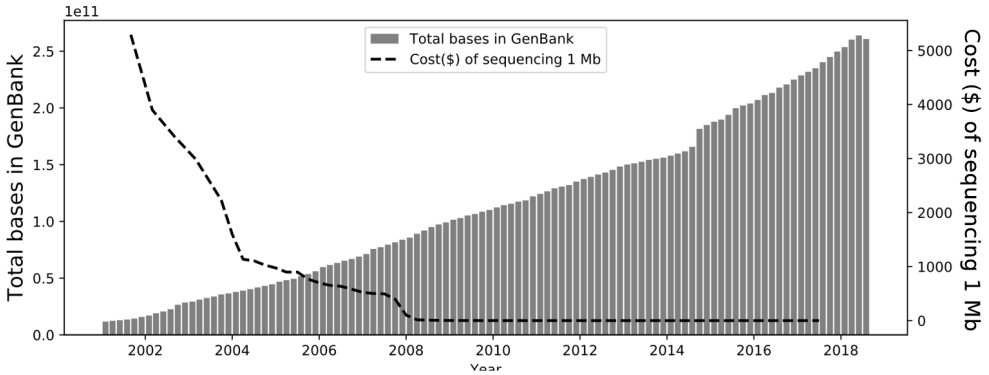


Figure 4.1: Increasing GenBank database size (available at `https://www.ncbi.nlm.nih.gov/genbank/statistics/`, accessed on September 15, 2018) follows a decreasing trend in sequencing cost (available at `https://github.com/TransDecoder/TransDecoder/wiki`, accessed on September 15, 2018) .

Providing fast and biologically valuable sequence alignment tools has been an active area of bioinformatics research, particularly in the context of ever-growing databases. Some sequence alignment programs have attempted to make algorithmic improvements (HMMER [55], DIAMOND [27]) while others have focused on improving parallelization and taking advantage of new HPC platforms and programming paradigms, including cuBLASTP [193], muBLASTP [194], mpiBLAST [44], and SparkBLAST [50] BLAST

tools. All of these tools provide similar output to NCBI BLAST at an improved computational speed. Other BLAST tools provide convenience factors of BLAST usage such as NOBLAST [101], which offers new options to limit the number of sequences to search, and JAMBLAST [101], which provides visualization tools for NOBLAST output.

BLAST is computationally expensive to run, with computational time impacted by the number of queries and reference database size. Furthermore, genome sequencing and annotation projects can be fairly long-term projects that require updates mid-project, e.g., regular annotation updates [74, 132]. However, for such updates, sequence similarity search steps have to be executed from scratch as search results from BLAST use similarity scores and E-values that depend on the size of the database, which continues to increase. For this reason, it is required to discard the results of prior searches and to rerun the entire search, which translates to irredeemable time, money, and computational resources, as shown in Figure 4.2(A).

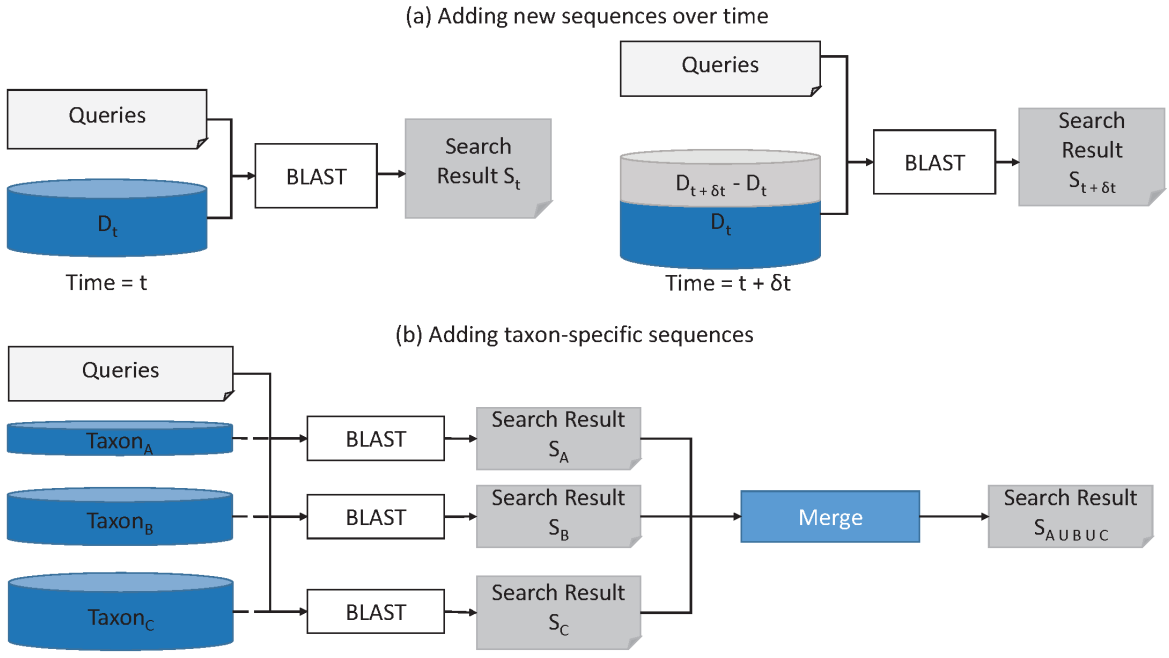


Figure 4.2: Incremental addition of new sequences.
(a) BLAST search when new sequences are added to the database. At time t, the database is D_t. In next δt interval, new sequences $D_{t+\delta t} - D_t$ are added, and the database becomes $D_{t+\delta t}$. With the traditional approach, the existing search result cannot be reused, and we have to perform an entire BLAST search against entire $D_{t+\delta t}$.
(b) BLAST search when several taxon specific databases are present and we want to get a result against the combined database. For three taxa, A, B, and C, we can perform individual BLAST searches against the databases D_A, D_B, D_C, respectively. If we want to obtain a search result against the combined database $D_{A \cup B \cup C}$, we need to merge the search results in a way that their E-values reflect the combined database size.

For bioinformatics projects requiring large-scale sequence alignment, such as those involving many transcriptomes from many taxa, the computational burden can be especially prohibitive. While this problem could be solved through performing iterative taxon-specific searches rather than conducting BLAST on the entire non-redundant database, such an approach has been historically difficult as one would need to stan-

dardize E-values when iteratively adding new databases to find the optimal identity of each query (Figure 4.2(B)).

In practice, new sequences get added to the search database(s) of interest in two ways: temporally and spatially. Temporal *addition occurs when new sequences are added to a database over time (e.g., a regular update to the nr database).* Spatial *addition occurs when different databases are available for search simultaneously, and we need to combine the search results against these databases as if the result was obtained by searching against a combination of all these databases. Thus, we must be able to compose search results in both the temporal and spatial dimensions and answer the following question:* Can we develop statistics to compose temporal and spatial BLAST search results through E-value correction?.

The short answer is yes, which we prove mathematically and then realize in software via our tool called iBLAST, short for incremental BLAST. iBLAST provides an efficient solution to E-value correction and allows the merging of results from two or more separate databases, thus allowing recycling of previous results, which subsequently saves both time and money. It also enables taxon-specific BLAST search, including incremental addition of specific biologically-relevant taxa to BLAST databases with subsequent merging. Thus, our approach will save significant time in large-scale projects that require pairwise sequence search; it will also better facilitate the sorting of hits by taxon. The iBLAST tool consists of the extension of NCBI BLAST program and a few key Python modules and supports the different versions of the NCBI BLAST command-line tools. We demonstrate application of guideline 3 through the design and implementation of this tool.

4.2 Background and Related Work

In this section, we present the core concepts that underlie BLAST, including the statistics for E-value computation and existing methods for correcting E-value computation when the size of the database is perceived to have changed.

4.2.1 Core Concepts of BLAST Result

When we perform a BLAST search against a sequence database with a query sequence, the BLAST program returns the best matching sequences from the target database. These best matching sequences are called hits. *Between the query and a hit sequence, there exist many pairwise local matches, which are called high scoring pairs or* HSPs. *One hit could consist of many HSPs. HSPs are scored using some statistical metrics by comparing aligned symbols. The score for a hit is the score of the highest scoring HSP that belongs to that hit. The E-value for an HSP is computed using the score, the database size, and other statistical parameters. The reported E-value of a hit is the E-value of the HSP with the lowest E-value.*

4.2.2 BLAST Statistics for E-Value Computation

BLAST programs use two different kinds of statistics for E-value computation: Karlin-Altschul statistics and Spouge statistics.

Karlin-Altschul statistics

Gumbel extreme value distribution (EVD) is often used to approximate the distributions of the maxima of long sequences of random variables, in this case, the distributions of the HSPs. The Gumbel EVD states that the probability of a score x greater than

or equal to S is $p(x \geq S) = 1 - e^{-\lambda(S-\mu)}$. Here λ is the scale parameter, and μ is the location parameter. Karlin and Altschul established a statistical theory about local alignment statistics using Gumbel EVD under certain assumptions to derive the formula for E-value (E),

$$E = e^{-\lambda(S-\mu)} = Kmne^{-\lambda S} \tag{4.1}$$

which is the renowned Karlin-Altschul equation [12].

Edge effect *Karlin-Altschul derives E-value statistics under the assumption that the sequence lengths are infinite, which does not hold with the introduction of the new length parameters m and n. The parameters m and n are called the effective length of the query and the database, respectively. They are introduced to compensate for the edge effect of alignments, which occurs at the end of the query sequence or the database sequence, where there may not be enough sequence space to construct an optimal alignment. So, the effective lengths are computed using a length adjustment l. Here $m = m_a - l, n = n_a - N \times l$, and $l = \ln(K \times m \times n)/H$ while N is the number of sequences in the database, m_a is the actual length of the query, n_a is the actual length of the database, and H is the entropy of the scoring system.The length adjustment l satisfies Equation (4.2).*

$$l = \frac{\alpha}{\lambda} \ln(K(m_a - l)(n_a - Nl)) + \beta \tag{4.2}$$

Here, α, β, λ are Karlin-Altschul parameters.

Spouge statistics

Spouge statistics [138] is developed on the Karlin-Altschul formula. Instead of computing the length adjustment l and then using it to compute the effective length of the database and query, this statistics applies a finite-size correction (FSC).

Finite-size correction [138] was introduced in command-line BLAST version 2.2.26. *Instead of estimating l, FSC estimates*

$$area = E[m - L_I(y)]^+[n - L_J(y)]^+ \tag{4.3}$$

as a measure of $(m-l)(n-Nl)$. Here I and J are two sequences to be compared. $L_I(y)$ is the distribution of the length required to attain a score of y or more. Equation (4.3) is practically computed by approximating the distribution of $\langle L_I(y), L_J(y)\rangle$.

There exists a range of statistical parameters that are used to compute the area. These parameters do not depend on the length of the database or the query. However, in actual BLAST implementations using Spouge statistics, the formula is modified to include a database scale factor. The database scale factor is calculated using the following formula:

$$db\_scale\_factor = \frac{n}{m} \tag{4.4}$$

For a given HSP with score S, the E-value E is calculated as

$$E = area \times Ke^{-\lambda S} \times db\_scale\_factor \tag{4.5}$$

Different statistics used by BLAST programs

Table 4.1 summarizes the statistics used by various BLAST programs.

BLAST Program	Function	E-value Statistics
blastn	Search nucleotide sequences with a nucleotide query	Karlin-Altschul
blastp	Search protein database with a protein query	Spouge
blastx	Search protein database with a nucleotide query translated to protein	Spouge
tblastx	Search a translated nucleotide database with a translated nucleotide query	Karlin-Altschul
tblastn	Search a translated nucleotide database with a protein sequence	Spouge

Table 4.1: Both Spouge and Karlin-Altschul statistics are used by various NCBI BLAST programs.

4.2.3 Existing E-Value Correction Software and Their Features

Here we discuss the different approaches for correcting E-value scores when the search database is partitioned.

mpiBLAST

mpiBLAST [44] is a parallel implementation of NCBI BLAST on a cluster. It partitions the database and performs BLAST searches against these partitions in parallel. For accurate E-value correction, mpiBLAST requires prior knowledge of the entire database.

We provide a detailed explanation of mpiBLAST in Section B.1.

NOBLAST

NOBLAST [101] provides new options for NCBI BLAST. It offers a way to correct E-values when split databases are in use and results need to be aggregated. E-value

computation requires knowledge about the entire database size, the number of sequences in the whole database N, and the total length of the database n. This tool does not address E-value corrections for the BLAST programs that use Spouge statistics.

We provide a detailed explanation of NOBLAST in Section B.1.

Comparison of mpiBLAST and NOBLAST to iBLAST

While the two aforementioned BLAST tools can provide exact E-value statistics for Karlin-Altschul statistics when the knowledge about the entire database is available a priori, they are not useful when the database keeps changing or two different search results against two different instances of similar databases needs to be aggregated. Table 4.2 provides a high-level comparison between mpiBLAST, NOBLAST, and our iBLAST.

Feature	mpiBLAST	NOBLAST	iBLAST
E-value correction for Karlin-Altschul statistics	✓	✓	✓
E-value correction for Spouge statistics	✗	✗	✓
Aggregate search results against pre-planned database segments	✓	✓	✓
Aggregate search results against arbitrary database instances	✗	✗	✓
Reuse existing search results	✗	✗	✓

Table 4.2: Comparison of three different BLAST tools that explicitly deal with E-value statistics correction. iBLAST supports E-value correction across time *and* space without requiring prior knowledge of the entire database. The other tools can correct E-values in limited scenarios.

4.3 Methods

To perform an incremental BLAST search temporally *as shown in Figure 4.3(A), we only consider the newly arrived sequences in the interval* δt *and perform a BLAST search against these to get the result* $S_{\delta t}$*. We correct E-values for the incremental result* $S_{\delta t}$ *and the past result* S_t *by using the size of the database* $D_{t+\delta t} = D_t + D_{\delta t}$*. To perform an incremental search* spatially*, as shown in Figure 4.3(B), we go through the search results from different databases and correct their E-values using the size of the combined database* $D_{A \cup B \cup C} = D_A \cup D_B \cup D_C$*. Then, we merge these search results with corrected E-values to obtain the final search result* $S_{A \cup B \cup C}$*.*

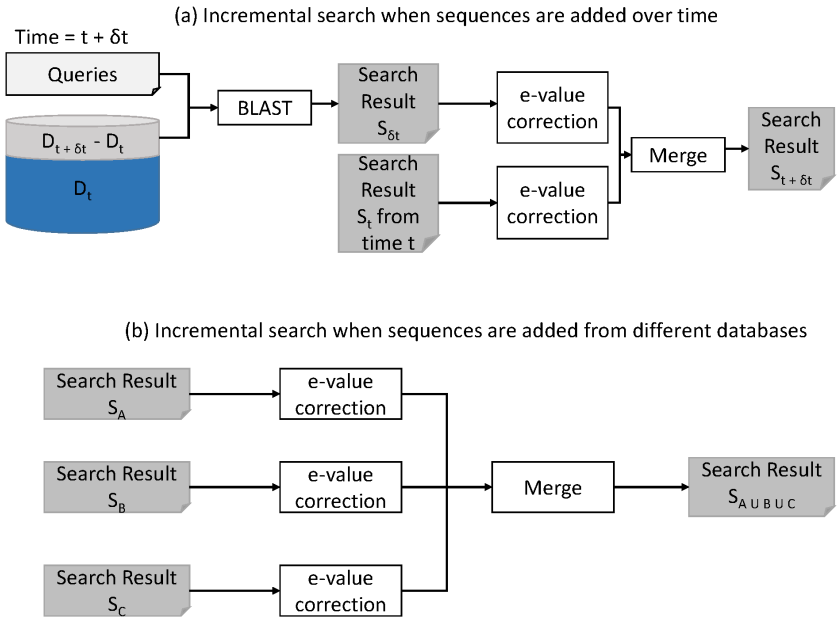


Figure 4.3: (a) Incremental search when new sequences get added to the database over time. We perform a BLAST search against the incremental database and combine the result with past results after E-value correction. (b) Incremental search when search results from different databases are available. Different search results are corrected for E-value against the combined database size; the corrected results are then merged together.

In the remaining part of this section, we present our E-value correction methods in detail, the implementation details of iBLAST, and the fidelity and efficacy of iBLAST

over NCBI BLAST via three case studies.

4.3.1 E-Value Correction in an Incremental Setting

Correct E-value computation requires the actual database length (total number of bases/residues) in both Spouge statistics and Karlin-Altschul statistics. While database-partitioning parallel BLAST applications like mpiBLAST and NOBLAST have prior knowledge about the total database length, iBLAST leverages the partial knowledge from a previous BLAST search and combines it with the new incremental additions to the database to infer the total database length and compute the proper E-value. The mpiBLAST and NOBLAST tools pass the actual database length to each of their parallel jobs, thus forcing the statistics module to compute correct E-values from the beginning. For the incremental iBLAST search, whenever new data arrives in the database, the pairwise sequence search is automatically refined in two steps. First, the search is only run on the new *sequences that have been added to the database. Second, the results generated from searching the new sequences in the database are then merged with the saved results from the previous BLAST search. This merging requires a re-evaluation of the E-values for all hits and their corresponding high scoring pairs (HSPs) using the new total length of the database.*

E-value correction for Karlin-Altschul statistics

Let n_c represent the current database length *and n_d represent the* length *of the newly arrived sequences for the database. Also let N_c be the* number *of sequences in the current database and N_d be the* number *of sequences in the newly arrived part of the database. Then, we have*

Actual length of the updated database: $n_t = n_c + n_d$.

Total number of sequences in updated database: $N_t = N_c + N_d$.

The actual query length m *does not change with the change in database. However, we do need to recompute the effective length* l *by solving the fixed-point equation for the new database length using Equation (4.6).*

$$l = \frac{\alpha}{\lambda} \ln \left(K(m-l)((n_c + n_d) - (N_c + N_d) \times l) \right) + \beta \tag{4.6}$$

Now, with the updated length adjustment l*, we can either recompute the E-values for all the matches or correct the E-values. To recompute all the E-values from scratch, we use Equation (4.7).*

$$E = e^{-\lambda(S-\mu)} = K(m-l)((n_c + n_d) - l)e^{-\lambda S} \tag{4.7}$$

Alternatively, we can correct the E-values from the current values. First, we use l *to recompute the value of the effective search space. We then use the newly computed effective search space to re-calibrate the E-values for all the reported HSPs from the current and delta search results. Assuming that* D_{part} *is the partial effective search space and that* D_{total} *is the total effective search space, then the corrected E-value is given by*

$$E_{total} = E_{part} + Ke^{-\lambda S} \times (D_{total} - D_{part}) \tag{4.8}$$

While both approaches require a constant number of arithmetic operations, the former approach, i.e., recomputing all the E-values from scratch, requires fewer arithmetic operations.

E-value correction for Spouge statistics

For Spouge statistics, the value of area *(i.e., Equation (4.3) described in Section 4.2.2) does not change since it is a function of the query length, sequence length, and Gumbel parameters. However, the database scale factor does change, and thus, we need to account for it. If the actual database lengths for the newly added part of the database and the total database are n_{part} and n_{total}, respectively, then*

$$E_{part} = area \times e^{-\lambda S} \times \frac{n_{part}}{m} \tag{4.9}$$

and

$$E_{total} = area \times e^{-\lambda S} \times \frac{n_{total}}{m} \tag{4.10}$$

So,

$$E_{total} = E_{part} \times \frac{n_{total}}{n_{part}} \tag{4.11}$$

Therefore, based on this derivation, we only have to re-scale the E-values instead of using Spouge's E-value computational methods.[1]

4.3.2 Merging Two Search Results with Correct E-Value Statistics

The hits reported by both searches are statistically significant. Once we correct E-values for both current search result and the new search result, we merge the hits into a single sorted list. Because iBLAST reports some better scoring hits which NCBI BLAST misses, reporting only max_target_seqs hits will result in missing some of the lower scoring hits from NCBI BLAST. So, we store and report $2 \times max\_target\_seqs$ hits.

[1]Caveat for re-scaling E-values that have been previously (and imprecisely) rounded to 0.0 by NCBI BLAST: Re-scaling an E-value that was previously (and imprecisely) rounded to 0.0 by NCBI BLAST obviously results in an incorrect 0.0 value. Thus, in the less than 0.1% occurrences of an extremely small but non-zero E-value, we ensure that this imprecise rounding does not occur.

Algorithm 13 documents the procedure to merge the hits from two results for the same query. All the statistical parameters dependent on total database size is recomputed to recompute or re-scale the E-values. The hits are selected in the ascending order of their E-values (descending order of their scores).

Algorithm 13 Merging results for Karlin-Altschul/Spouge statistics.

```
1: Input: result1, result2
2: merged_result ← Φ
3: recompute/re-scale E-values
4: m, n ← 0
5: for i = 1 → num_of_hits do
6:     e-value1, score1 ← min(result1.alignment[m].hsps)
7:     e-value2, score2 ← min(result2.alignment[n].hsps)
8:     if (e-value1 < e-value2) or e-value1 == e-value2 and score1 > score2) then
9:         merged_result.add(result1.alignments[m])
10:        increment m
11:    else
12:        merged_result.add(result2.alignments[n])
13:        increment n
14: return merged_result
```

Additional details on recomputing and re-scaling E-values is provided in Section B.2.

4.3.3 iBLAST Implementation

We develop iBLAST for performing incremental BLAST search as an extension to the NCBI BLAST code. It consists of Python wrapper scripts around the extended BLAST code and uses NCBI BLAST programs as black-box routines. Figure 4.4 shows the

software stack of iBLAST. The software stack consists of three major components: user interface, incremental logic, and record database. These modules interact with BLAST databases through the BLAST+ programs.

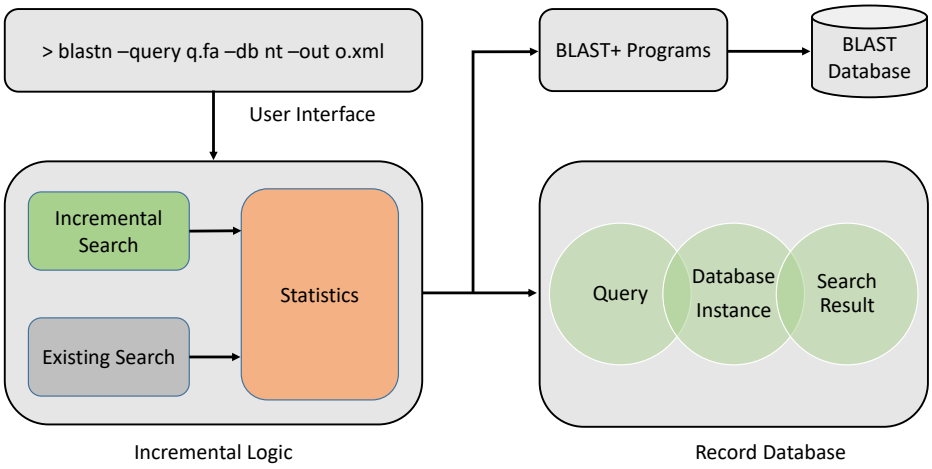


Figure 4.4: Software stack of iBLAST. The user can initiate a search using the user interface. The search parameters are then passed to the "Incremental Logic" module. After performing an incremental search, the backend of this module corrects the E-value statistics and merges the result. The "Incremental Logic" module looks into an external lightweight database module called the (*Record Database*) to decide whether and how to perform the incremental search. For the actual search and incremental database creation, we use NCBI BLAST tools such as blastdbcmd, blastdbalias, blastp, and blastn.

Command-Line User Interface *In our current version, we provide a command-line interface for iBLAST, which provides NCBI BLAST-like search options.*

Incremental Logic *This module decides whether to perform a new BLAST search based on existing results. Whenever the user requests a new BLAST search, this module checks for any pre-existing search result. If it does not find any pre-existing result, it performs a regular NCBI BLAST; but if there is a pre-existing result, the module first*

compares the database instance from the time of the past search with that of the present search. If there is any difference in the database size, this module builds a delta database that consists of the difference of these two database instances. It then performs a new BLAST search only against the delta database and merges the previous result with the new incremental result after statistical correction for E-values. Figure 4.5 shows the different components of this module. This module allows multiple updates to current searches with little extra time investment. The "Incremental logic" module contains three sub-modules: Existing Search*,* Incremental Search*, and* Merge*.*

1. **Existing Search.** *This sub-module looks for an existing search result with the help of the record database.*

2. **Incremental Search.** *This sub-module constructs an incremental database by comparing with the past instance of the database and performs a BLAST search on the incremental database.*

3. **Statistics.** *This sub-module reads the past search result and the new incremental search result, re-evaluates the E-values in both, and merges them according to their recomputed/re-scaled E-values.*

The details of these three sub-modules are illustrated in Figure 4.5.

Record Database for storing incremental search results *Every time a BLAST search is performed, we save the instance of the database along with the search result in a lightweight SQLite database. We save a minimalist index structure and size information that requires only a few bytes of storage. We keep the search parameters along with the search results as well.*

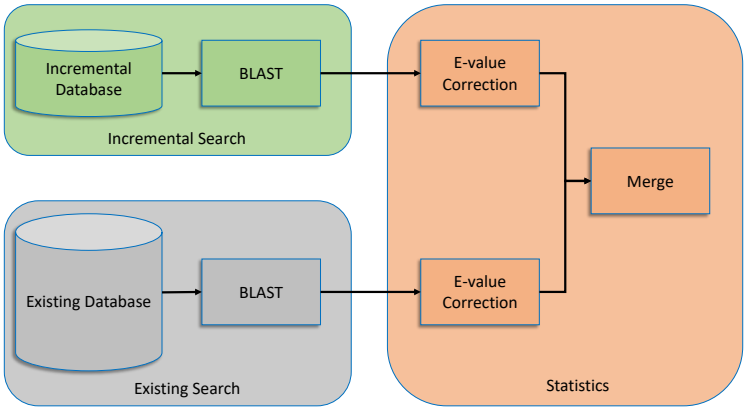


Figure 4.5: Sub-modules of the *Incremental Logic* module. Whenever the user initiates a BLAST job, the above "Incremental Logic" module first checks if an existing search result is available. If there is a search result against an outdated BLAST database, a delta database consisting of the newly added sequences is constructed. A BLAST search is then performed against the delta database (i.e., incremental database). In the final stage, the existing search result and the incremental search result are merged and the associated E-values corrected.

4.3.4 Case Studies

To demonstrate the efficiency and benefits of using the iBLAST program over standard NCBI BLAST, we analyze different actual nucleotide and protein sequence datasets as case studies.

Case study I: method verification

We explore the scenario where hits from a collection of 100 *query sequences are updated to account for the growth of NCBI sequence databases across the duration of the project. To demonstrate the application's use for BLAST programs that use Karlin-Altschul statistics, we ran blastn against a nucleotide database (growing subsets of NCBI nt) for* 100 *nucleotide sequences from Bombus impatiens available at* `ftp://ftp.ncbi.nlm.nih.gov/genomes/Bombus_impatiens/CHR_Un/bim_ref_BIMP_2.1_chrUn.fa.gz`. *To*

demonstrate its utility on BLAST programs that use Spouge statistics, we ran blastp against a non-redundant protein database (growing subset of NCBI nr) for 100 *protein sequences from Bombus impatiens assembly available at* `ftp://ftp.ncbi.nlm.nih.gov/genomes/Bombus_impatiens/protein/protein.fa.gz`.

We demonstrate iBLAST's fidelity and performance over three time periods (Figure 4.6(A)). The instances for nucleotide database changes through time as following:

- ***Time 0:*** *the nucleotide database comprises* 44.5% *of the full nt database. Both tool conduct search on the same database.*

- ***Time 1:*** *the nucleotide database comprises* 62.7% *of nt. While NCBI BLAST searches* 62.7% *of nt, iBLAST performs search only on* 18.2% *of nt. The database grew by* 40.8% *(=* $(62.7 - 44.5)/44.5$*) from time* 0.

- ***Time*** 2***:*** *the nucleotide database comprises* 84.1% *of nt. While NCBI BLAST searches* 84.1% *of nt, iBLAST performs search only on* 21.4% *of nt. The database grew by* 34.1% *(=* $(84.1 - 62.7)/62.7$*) from time* 1.

The instances for protein database changes through time as it comprises 35.4%, 47.5%, *and* 60.0% *of nr. The protein database grew by* 34.1% *(=* $(47.5 - 35.4)/35.4$*) and* 26.3% *(=* $(67.5 - 35.4)/48$*) in time* 1 *and* 2 *respectively from the earlier time periods. The performances of NCBI BLAST and iBLAST for each of these time periods were then compared.*

More details on incremental database creation is provided in B.3.

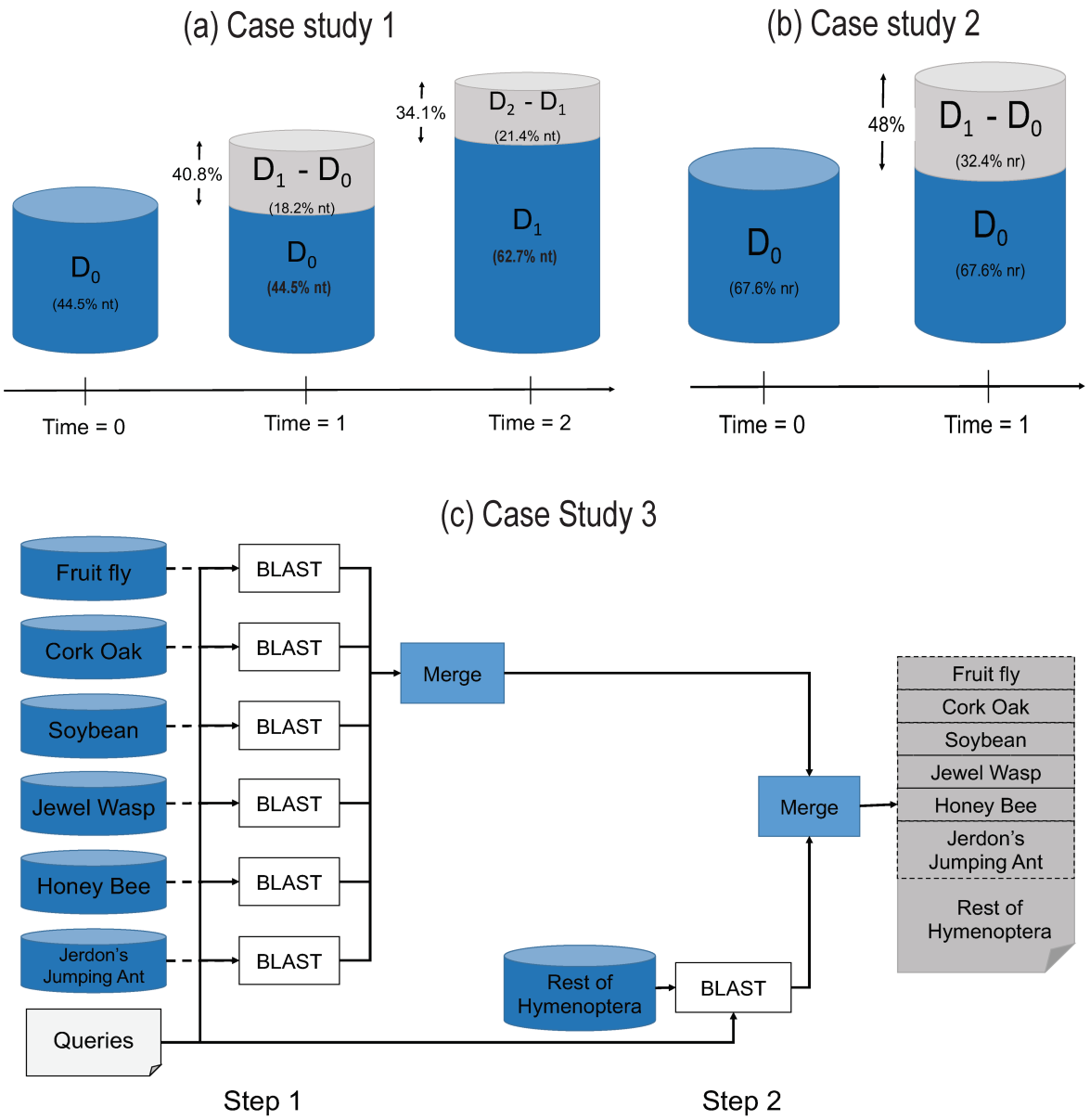


Figure 4.6: Experimental design of three case studies. (a) Case study I: Incremental addition of sequences in the nt database over three time periods. (b) Case study II: Incremental addition of sequences in the nr database over two time periods. (c) Case study III: Incremental search of taxon-specific databases.

Case study II: updating a query re-annotation of a novel transcriptomics dataset

Our second case study mimics a typical scenario in a transcriptome re-annotation project where a transcriptome is BLAST-ed after a certain period as a part of a re-annotation pipeline. This case study uses a novel dataset not yet available on NCBI BLAST database - a de novo assembled transcriptome of the venom gland of an Oak gall wasp (see below) - and thus the identity of the assembled sequence was unknown, and the sequence was not available to BLAST to itself.

We conduct a BLAST search for the same query set for database instances in two time periods (Figure 4.6(B)):

- ***Time 0:*** *the database comprises* 67.6% *of the non-redundant database nr (nr accessed on August 2018). Both tools perform the search on this same* 67.6% *of the database.*

- ***Time 1:*** *the database comprises* 100% *of the non-redundant database nr. While NCBI BLAST performs a search on* 100% *of nr, iBLAST performs a search on only* 32.4% *and reuses search result from time* 0*. iBLAST merges the result from time* 0 *with the incremental search result after E-value correction.*

We constructed these two database instances by combining database parts using blastdb_aliastool utility packaged with BLAST+.

Given that the number of queries from the de novo assembled transcriptome, it would take few months to complete the search on a single core. We ran this experiment with 640 *cores distributed across* 20 *compute nodes (each node has* 2 *Intel Xeon Processor E5-2683 v4), partitioning the* 17927 *queries into* 20 *query files and assigning each file per node. Given that each node will run a subset of queries against the same database,*

there is no need to recompute the statistics for these results before we merge them. The time investment and results for traditional and incremental approaches were compared.

It is essential that workload across all the nodes are balanced so that computation for each of the 20 *query files finishes roughly at the same time. To ensure such load balancing, we partitioned the queries using the following strategy. We randomize the order of the queries and partition them so that each partition has roughly the same number of residues. We compare this strategy against the straightforward approach of partitioning the queries in a linear order by putting roughly the same number of queries in each partition.*

Case study III: taxon-based incremental approach

*Our third case study presents a special case of using a taxon-based incremental approach to obtain a fast, cost-effective and biologically relevant sequence similarity results. To achieve this goal, we examine the genes contained within an assembled transcriptome of the venom gland of a gall wasp of oak trees, the hedgehog gall wasp (**Acraspis erinacei**), a taxon lacking a closely related species with a genome. Gall wasps are a group of parasitic wasps that inject their eggs into plant tissues and through processes yet unknown, induce changes in plant development at the site of injection. These changes result in the construction of a niche for the gall wasp by inducing predictable modifications of plant tissues that both protect the wasp from the environment and feed the developing wasp. Genes important for inducing changes in the plant's development are thought to be produced in the female venom gland during oviposition ([176]). We performed separate BLAST searches of the hedgehog gall wasp venom gland against transcriptomes of the closest relatives to gall wasps with curated genomes including three fairly equidistant taxa ([143]) - the parasitic wasp **Nasonia vitripennis**, the honey bee **Apis mellifera**, and the ant **Harpegnathos saltator**, - as well as the*

more distant model insect, ***Drosophila melanogaster****, upon which many insect gene annotations are based. We also perform BLAST searches the transcripts to an oak tree,* ***Quercus suber****, to determine if some genes belonged to the host, and a model plant the soybean,* ***Glycine max****.*

A blastp search was conducted individually against each of the databases and results were merged using the statistics module of the iBLAST. After this initial search, we then added to this analysis all remaining Hymenopteran species using iBLAST, to assess the impact of adding more taxa on the top BLAST hits and further demonstrate the potential of iBLAST to add taxa progressively (Figure 4.6(C)). We performed a blastp search against the merged database of those seven subsets to determine whether the same hits would have been found from our concatenated incremental analysis as from a combined single-instance run. These results were further compared with blastp results obtained by searching the complete nr database, thus allowing us to determine how well we captured of the full dataset with this taxon sub-sampling approach.

Data collection for case studies II and III

To obtain the venom gland transcriptome, 15 *venom glands were dissected from newly emerged adult females from wild collected oak (oak spp.) hedgehog galls of Acraspis erinacei and placed in RNAlater. Pooled tissues were homogenized in lysis buffer using a Bead Ruptor 12 (Omni International) with additional lysis with a 26-gauge syringe. RNA was extracted from the sample using the RNaqueous Micro kit followed by DNase I treatment as specified by the kit and confirmed to be of good quality using the Bioanalyzer 2100 (Agilent). The Illumina HiSeq library was prepared from* 200 *ng RNA using the TruSeq Stranded mRNA kit and sequenced in* 150 *bp single-end reads across two Rapid Run lanes on the Illumina HiSeq 2500 (Penn State Genomics Core Facility, University Park, USA) along with nine other barcoded wasp samples.*

*Raw sequence data (*30.6 *million reads, available at NCBI SRA achieve under Accession ID XXXXXXXXX) quality was assessed using FastQC v0.10.0 (available at* `http://www.bioinformatics.babraham.ac.uk/projects/fastqc/`*), and appropriate trimming was conducted using Trimmomatic v0.35 [26] with following parameters: ILLUMINACLIP:TruSeq3- PE.fa:2:30:10 LEADING:20 TRAILING:20 MINLEN:50 AVGQUAL:20. This procedure removed* 0.12% *of total reads. de novo transcriptome assembly from these QC-passed trimmed reads was performed on the Trinity RNA- Seq de novo Assembler (version: trinityrnaseq r20140717)([69]). The transcriptome assembly consists of* 44, 440 *transcripts with the contig N50 of* 865 *bases. Transdecoder v5.3.0 (available at* `https://github.com/TransDecoder/TransDecoder/wiki`*) was used to predict* 17927 *protein sequences which were used as queries for case Study II and III.*

4.4 Results

In this section, we present iBLAST software and its efficacy in sequence similarity search tasks through three case studies.

4.4.1 iBLAST Program

The iBLAST program v1.0 includes a collection of python scripts which can be downloaded at `https://bitbucket.org/sajal000/incremental-blast`*. The user needs to copy the source folder and run the following command from this directory to install iBLAST:*

```
./IncrementalBLAST-installer.sh
```

The python scripts are:

Main program: iBLAST.py *This program provides incremental BLAST search options. It takes in a regular BLAST search command and performs incremental search. An example usage of the script is:*

```
python iBLAST.py "blastp -db nr -query Trinity-tx.fasta -outfmt 5 -out result.x
```

Merge scripts *These scripts are used to merge two BLAST search results in xml format and produce an xml output with corrected E-values.*

1. ***BlastpMergerModule.py:*** *This is used to merge results obtained using Karlin-Altschul statistics (e.g., blastn results)*
2. ***BlastnMergerModule.py:*** *This is used to merge results obtained using Spouge statistics (e.g., blastp results)*
3. ***BlastpMegerModuleX.py*** *These scripts merge more than two BLAST results. They require number of results to merge, the input results and output.*

```
python BlastpMergerModule.py input1.xml input2.xml output.xml
python BlastnMergerModule.py input1.xml input2.xml output.xml
python BlastpMergerModuleX.py 3 input1.xml input2.xml input3.xml output.xml
```

4.4.2 Case Study I: Method Verification

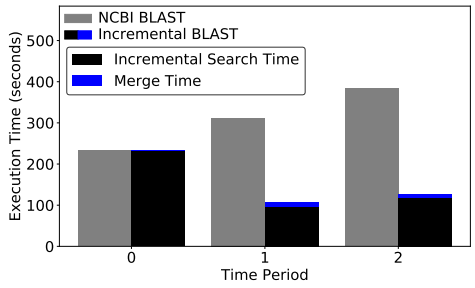
In case study I, we validate whether we can get the same results from a single NCBI BLAST search as from the iBLAST (Table 4.3). In all three time periods, we find all the hits reported by NCBI BLAST for blastn were recovered by iBLAST in the same order, including 3964 *hits at time* 0, 4150 *hits at time* 1, *and* 4924 *hits at time* 2, *thus validates iBLAST for Karlin-Altschul statistics. We observe similar results for blastp as well. For each of these three time periods, iBLAST reports the exact same hits in the*

exact same order as NCBI BLAST. The numbers of reported hits in these three time periods for blastp are 45154, 46,356, *and* 46869. *Reporting the same results for blastp validates iBLAST for Spouge statistics.*

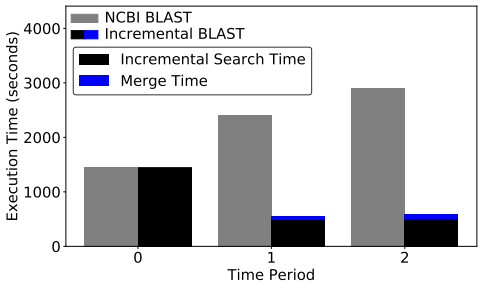
Period	Database	Database Size		E-value Match	Hit Match
		Current	Incremental		
0	nt	80,740,533,243	80,740,533,243	100%	100%
	nr	17,686,779,866	17,686,779,866	100%	100%
1	nt	113,749,495,340	33,008,962,097	100%	100%
	nr	23,752,080,639	6,065,300,773	100%	100%
2	nt	152,471,828,601	38,722,333,261	100%	100%
	nr	30,030,148,449	6,278,067,810	100%	100%

Table 4.3: Fidelity of iBLAST in three consecutive time periods.

For δ increase in database size iBLAST performs $(1+\delta)/(\delta)$ times faster than NCBI BLAST. Figure 4.7 shows the time saved for both blastp and blastn using iBLAST.



(a) Performance comparison between regular blastn and incremental blastn at three periods when nt database is growing over time, using 100 nucleotide queries. For 40.8% and 34.0% increase in the database size, iBLAST performs 2.93 and 3.03 times faster respectively.



(b) Performance comparison between regular blastp and incremental blastp at three periods when nr database is growing over time, using 100 protein queries. For 34.1% and 26.3% increase in the database size, iBLAST performs 4.33 and 4.98 times faster respectively.

Figure 4.7: Performance for case study I.

4.4.3 Case Study II: High Efficiency of iBLAST for Large Alignment Search Tasks on Novel Datasets

We performed searches using iBLAST and NCBI BLAST where a gall wasp transcriptome dataset were utilized as queries in two time periods, applying a 48% *increase in the nr database in between. In both time periods, iBLAST reports the same hits in the same order as NCBI BLAST.*

Period	Database Size		E-value Match	Hit Match
	Current	Incremental		
0	40,077,622,077	40,077,622,077	100%	100%
1	162,267,258	19,192,851,238	100%	100%

Table 4.4: Fidelity of iBLAST (blastp) in two consecutive time periods.

Figure 4.8 shows the time comparison between NCBI BLAST and iBLAST. We see that for 48% *increase in the database size, iBLAST is* 3.1 *times faster than the NCBI BLAST in achieving the new result. The time needed for E-value correction and merging the results is minimal (less than a minute using only* 20 *cores).*

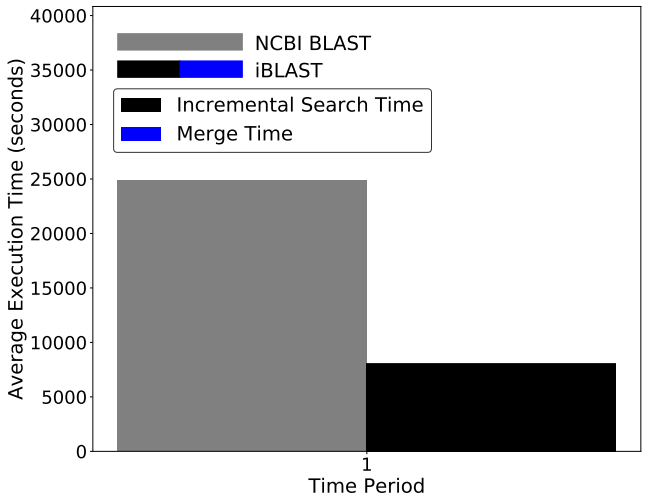


Figure 4.8: Performance for case study II. Time required by NCBI BLAST and iBLAST. The average time taken is 24862 seconds (6 hours, 54 minutes) for NCBI BLAST and 8009 seconds (2 hours, 14 minutes) for iBLAST. Merge time for each of these tasks is 40 seconds on average. Maximum time for these three are 25835 seconds (7 hours, 11 minutes), 8334 seconds (2 hours, 19 minutes), and 49 seconds. By both accounts, iBLAST is 3.1 times faster than NCBI BLAST. This speedup complies with our projected speedup $(1 + 0.48)/0.48 = 3.08$.

We observe the effect of load balancing. When we partition the queries based on number of queries without disrupting the original order, incremental search time ranges from 3777 *seconds to* 12351 *seconds with a standard deviation of* 2787 *seconds. With the same strategy, NCBI BLAST search time ranges from* 11626 *seconds to* 39247 *seconds with a standard deviation of* 8727 *seconds.*

In contrast, when we partition the queries after reordering the queries and partition them based on number of residues the standard deviation becomes 150 *seconds and* 487 *seconds respectively. It clearly demonstrates the superior load balancing of our adopted strategy over the straight forward partitioning. We demonstrate this further in Section B.4.*

4.4.4 Case Study III: Expedited Informatics via Taxon-Specific Searches

To examine the fidelity of iBLAST when merging multiple (taxon-specific) databases, first, we compared the iBLAST merged results from multiple individual BLAST(blastp) searches on seven biologically relevant taxa to results obtained when a BLAST search was performed against a database combining all the sequences belonging to these taxa. The result exhibits 100% *fidelity. Then, we compared (presented in Table 4.5) the merged BLAST results of individual taxon-level database search with the BLAST results obtained in case-study II (time period 1) where the same queries were searched against the entire nr database to better understand relative time savings vs. accuracy of taxon-guided approaches. The taxon-specific approach is much more time-efficient and computationally inexpensive as it searched much smaller databases. With our initial set of 6 taxa we sampled only* 0.35% *of the nr database and retrieved* 8.124% *of the top hits obtained when searching nr. Although this number is low, the identity of top hits is likely to be similar even if the best taxonomic hit to that gene was not retrieved, as gall wasps do not have any close relatives sequenced but rather many equidistant relatives. Given this, we then added in the rest of Hymenoptera to see if this improves the number of shared top hits. With this analysis we BLAST-ed only* 1.17% *of the total nr yet obtained* 87.75% *similarity in top hits to a full nr BLAST. This demonstrates the potential of performing more taxon-guided approaches to save on the costs of large-scale BLAST searching jobs. Performing the analysis in this way has also enabled improved curation of hits by taxon which facilitates better biological interpretation of these results.*

NCBI taxon id	Species	% nr sequences covered	Number of nr top hits covered	% nr top hits covered
7425	*Nasonia vitripennis* (jewel wasp)	0.02%	853	4.84%
7460	*Apis mellifera* (honey bee)	0.02%	207	1.17%
10380	*Harpegnathos saltator* (Jerdon's jumping ant)	0.03%	347	1.96%
7227	*Drosophila melanogaster* (fruit fly)	0.08%	6	0.034%
58331	*Quercus suber* (cork oak)	0.09%	0	0.00%
3847	*Glycine max* (soybean)	0.11%	22	0.12%
7399	Rest of Hymenoptera	0.83%	14281	80.98%
Total	**multiple**	**1.17%**	**15476**	**87.75%**

Table 4.5: Potential for taxon-guided searches. Comparison of merged BLAST results from multiple individual BLAST searches with a separate BLAST search conducted against a completed nr database, showing that biologically relevant taxa can be added incrementally to obtain similar results to nr by searching against much smaller database size.

4.4.5 Identification of Better Scoring Hits by iBLAST Than NCBI BLAST

While iBLAST finds all the hits reported by NCBI BLAST in the same relative order, iBLAST reports some better scoring hits that NCBI BLAST misses for all the case studies. Since case study II covers the most number of hits, we quantified these missed hits for this case study. NCBI BLAST misses 1.57% *(13171 out of 837942 top hits) of the better scoring hits. Command line NCBI BLAST uses a search pa-*

rameter max_target_seqs *in an unintended way where instead of reporting all the* best max_target_seqs *hits, it has a bias toward* first max_target_seqs *hits. A comprehensive discussion about this issue was carried out by Sujai Kumar (*`https://gist.github.com/sujaikumar/504b3b7024eaf3a04ef5/`*) and two other teams of researchers [67, 159]. In this process, it misses some of the better scoring hits that is discovered in a later phase of the search (Details can be found in Section B.5). This is an extra advantage of iBLAST over NCBI BLAST. Since the former works on smaller databases and then combines the results instead of searching a single large database, it has more candidate hits to choose from for reporting final hits.*

4.5 Conclusion and Discussion

In this paper, we have introduced iBLAST, an incremental local alignment tool that delivers identical results to NCBI BLAST (along with additional better hits that are missed by NCBI BLAST) and does so with much better performance than NCBI BLAST.

Our approach is enabled by the development of novel statistical methods of E-value correction. This approach can be used to combine multiple search results performed at different times or with different groups (e.g., taxa), thus facilitating novel ways of performing sequence alignment tasks and incorporating domain knowledge. For a δ *fraction increase in the database size, iBLAST can perform* $(1+\delta)/\delta$ *times faster than NCBI BLAST (i.e.,* 10% *growth in database size will yield 11 times speedup for iBLAST over NCBI BLAST). It should be noted that for a small increase in the database size (which is the most likely scenario between two searches), iBLAST provides a large speedup factor.*

iBLAST is more successful than NCBI BLAST in discovering better hits. While iBLAST finds 100% *of the hits that NCBI BLAST reports in the same order, it reports many*

additional high scoring hits that NCBI BLAST misses due to an early approximation used by the heuristic search algorithm in NCBI BLAST.

With the expansion of genetic data available in NCBI, computational time is becoming ever more burdensome, resulting in analyses that take months to complete with substantial financial cost. This problem is aggravated by cheaper sequencing technology leading to ever-larger genome assembly/transcriptomics projects with substantially more samples to analyze. Our program can help relieve the cost burden. It enables iterative updates for re-annotation of genome and transcriptome assemblies, useful given rapid changes in the nr databases across the duration of a project. Specific datasets of interest can be added to previous searches, such as scenarios involving availability of new genome releases or conducting large phylogenetic studies. As demonstrated in Case Study III, the program can be applied to transcriptomic or metagenomics projects by merging the results of knowledge-guided BLAST searches only on groups that are biologically relevant. This enables iterative exploration by taxon and facilitates curation of BLAST results.

Chapter 5

Mitigating Catastrophic Forgetting Using Historical Summary

5.1 Introduction

In many real-life and scientific applications, observational data arrives in a stream with a need to discover knowledge gradually with the arrival of the data. For example, we need to make instantaneous routing decisions from real-time traffic surveillance images from traffic cameras. In the case of a developing storm, weather scientists need to make a live predictive map of the storm's path from streaming meteorological data. In scientific simulations where outcomes of previous steps determine the next steps and their parameters, a decision has to be made online from the simulation's streaming data.

Training Machine Learning and Deep Learning models in streaming settings can be useful in scenarios like the ones mentioned above. Besides naturally occurring streaming data sources, analytics of exceptionally large data can also benefit from learning in a streaming setting. If the data is too large so that learning from it at once overwhelms system memory and compute capability, we can break down the data into manageable chunks and emulate a streaming setting. Incremental learning from a streaming data source can be beneficial when we have access to the streaming data only for a limited amount of time, and we cannot store the entire dataset due to memory constraints, or we cannot analyze the whole dataset due to memory and compute constraint.

Like other machine learning models, incrementally trained deep neural networks suffer from the changing nature of the data in a stream. From related literature [64], we see few challenges that arise in training machine learning models in streaming settings.

1. *Online model parameter adaptation. How to update the model parameters with the new data?*
2. *Concept drift. How to deal with data when the data distribution changes over time?*
3. *The stability-plasticity dilemma. If the model is too stable, it can't learn from the new data. When the model is too plastic, it forgets knowledge learned from the old data. How to balance between these two?*
4. *Adaptive model complexity and meta-parameters. In the presence of concept drift or the introduction of a new class of data might necessitate updating and increasing model complexity against limited system resources.*

The primary focus of our work is identifying the challenges associated with catastrophic forgetting. Catastrophic forgetting is a special case of stability-plasticity problem, where the model becomes too plastic and forgets about the tasks learned earlier in the stream. In Section 2, we will present our survey on various state of the art approaches to tackle these challenges in training deep learning models in a streaming setting.

5.2 Background and Related Work

Different body of research addresses various aspects of these challenges. The following two frameworks/algorithms address various challenges.

ADAIN *framework [77] describes an adaptive learning method in a streaming setting that learns a distribution of the current data based on its classification performance*

on the past model and then adjusts the past model using the current distribution. The distribution assigns higher weights to the data points that are heard to learn and lower weights to the data points, which are easier to learn. At any given time t*, this framework assumes there is a known distribution* P_{t-1} *of the data from the earlier arrival at time* $t-1$ *and a hypothesis (model)* h_{t-1}*. It learns current distribution by first mapping the past distribution* P_{t-1} *to current data and then by assessing the error when this mapped distribution is tested against the past distribution,* h_{t-1}*.*

ADAIN is tested on simple datasets comprising up to 6 *classes and several models, including Multi-layer Perceptron (MLP). While ADAIN shows comparable performance, its usability for more complex data such as ImageNet (1000 classes) and Deep learning models is unknown. ADAIN addresses concept drift.*

SCIL *algorithm [89] deals with streaming data by maintaining a set of neurons per class and by occasionally merging the neurons to keep the network size under control. Versatile Elliptic Basis Function (VEBF) is used to combine the neurons. The set of neurons form a set of hyper-ellipsoids to capture clusters of data in the same class. VEBF is used to define the boundary of the discarded data from earlier arrivals, and with the arrival of new data and new class, these boundaries are updated.*

SCIL was tested on the dataset with the number of classes up to 26*, and data points up to* $581K$*, and it outperformed other reported streaming algorithms in the classification task. One obvious limitation of this work is it was not tested on a dataset with a large number of classes. SCIL addresses concept drift and adaptive model complexity.*

Sahoo et al. [150] proposed an online training model to address adaptive model complexity. It uses Hedge Backpropagation, which dynamically decides on the depth of the training model. Since too shallow model converges too fast and performs poorly on new data, and too deep model converges very slowly to achieve a reasonable amount of learning, this training model maintains the range of layers. With time, the number of

layers required converges.

Besedin et al. [25] designed an evolutive deep model to allow training in an online streaming setting. This model addresses concept drift and catastrophic forgetting. Their model learns on the available data, and to forward the current data, they construct a Generative Adversarial Network (GAN) on the current data and forward this GAN to the next time step. Every time new data arrives, this approach trains the model on that data and the data generated by the GAN from the previous step.

5.2.1 State-of-the-Art Approaches for Mitigating Catastrophic Forgetting

Kemker et al. [91] categorized approaches to mitigate catastrophic forgetting in the following classes.

1. *Regularization methods. Elastic Weight Consolidation (EWC) [95] adds a regularization term to the loss function to preserve weights contributing to learning past tasks.*

2. *Ensemble methods. Learn++ [145] learns a new model per episode with the help of the distribution of the data and ensembles these models.*

3. *Rehearsal methods. There are a few rehearsal methods that reuse past data; one such method GeppNet preserves past data and incorporates these to current episode's training through a Self Organizing Map (SOM).*

4. *Dual-memory methods. GeppNet+STM [65] maintains a short term memory for new uncertain predictions and consolidates these with more long-term memory.*

5.3 Mitigating Catastrophic Forgetting Using Historical Summary

We propose to train the model on historical data as well as current data without stressing the storage requirement. In every episode, we train our model on the currently arrived data and the summary of historical data. Figure 5.1 illustrates the proposed pipeline. First, the arriving data unit will go through a data-reduction tool (clustering, dimension reduction using PCA/auto-encoder, core-set approximation, sampling, etc.). We will train the model with the summary data and feed the summary data to a summary-over-time module, which will be used to train the model.

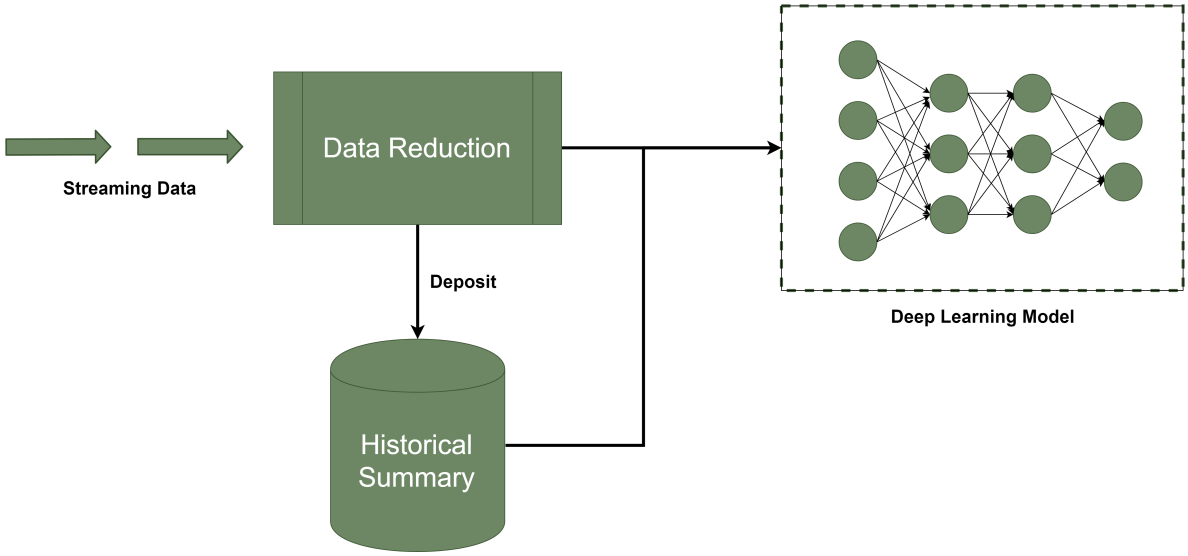


Figure 5.1: A streaming model of training Deep Learning (DL) models using less data. As the data arrives in a stream, we perform a fast summarizing process on this. We train the model on this summarized data. We combine this summary with the data in "summary over time" buffer. This way, we train the model incrementally on newly arrived data along with a representative set from the past.

Summarizing data *We can summarize data using coreset computation methods, clustering, random sampling, or dimension reduction techniques.*

Saving summarized history *A summary of data from the previous episodes would be saved into a buffer, and we will train the model using the current data along with the saved summary.*

Effectiveness of historical summary in mitigating catastrophic forgetting *Catastrophic forgetting prevents effective learning while training deep learning models in a streaming setting. There are various efforts to mitigate this phenomenon with varying degrees of success. We propose historical summary construction as a means to counteract this. Several summarization techniques such as k-means clustering, random sampling, autoencoder, and PCA can be useful for constructing historical summary.*

We investigated the conditions under which catastrophic forgetting occurs. When data arrives uniformly in all episodes, catastrophic forgetting is less likely. When a subset of classes arrives in each episode, we observe catastrophic forgetting to take place.

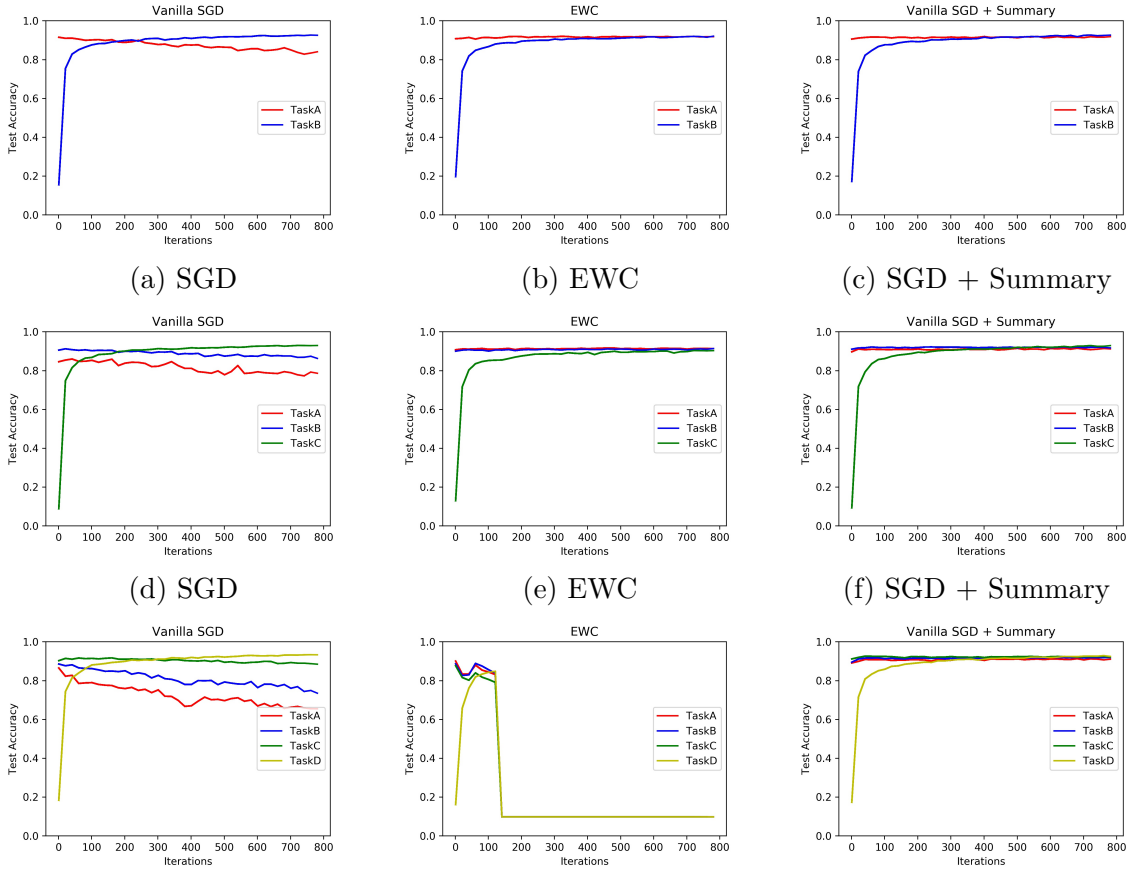
To investigate the potential of historical summary in mitigating catastrophic forgetting, we compare a simple summary method (i.e., random sampling) against the EWC method on two different types of tasks.

1. ***Task type I:*** *In the first type of task, we use the MNIST dataset, but in each episode, the images of the MNIST are permuted in a certain way. So, the classifier model is incrementally trained to learn a new task (classify newly permuted images into their correct classes) while retaining knowledge on previously learned tasks.*

2. ***Task type II:*** *In this setting, MNIST images are not permuted, but at any episode, only a subset of classes of the images are available. We simulated the arrival of images from six randomly chosen classes from a total of ten classes at any episode.*

On the first type of task, the historical summary is as good as the EWC when recovering

from catastrophic forgetting for up to three episodes. During the fourth episode, the EWC becomes unstable, but historical summary performs consistently through all four episodes (Figure 5.2).



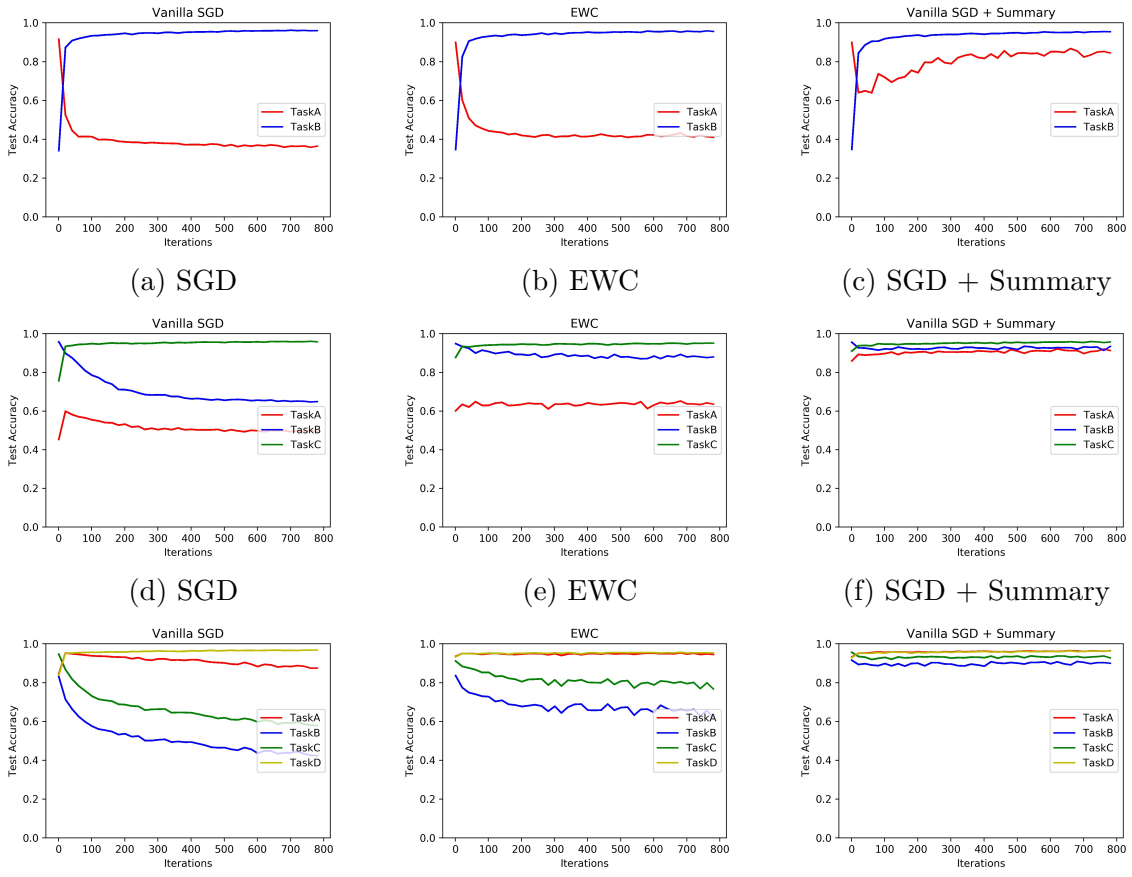
(a) SGD (b) EWC (c) SGD + Summary

(d) SGD (e) EWC (f) SGD + Summary

(g) SGD (h) EWC (i) SGD + Summary

Figure 5.2: Test accuracy while training the models using three paradigms. The left column shows the performance when the model is incrementally trained only with SGD. The middle columns shows the performance when EWC is used. The rightmost columns shows the performance when a historical summary constructed through random sampling is used in training the model with SGD.

On the second type of task, EWC suffers in recovering from catastrophic forgetting even though it performs better than vanilla stochastic gradient descent (SGD). The historical summary outperforms both of these methods (Figure 5.3) by a margin.



(a) SGD (b) EWC (c) SGD + Summary

(d) SGD (e) EWC (f) SGD + Summary

(g) SGD (h) EWC (i) SGD + Summary

Figure 5.3: Test accuracy while training the models using three methods. The left column shows the performance when the model is incrementally trained only with SGD. The middle column shows the EWC's performance. The rightmost column shows the performance when a historical summary constructed through random sampling is used in training the model with SGD.

5.4 Conclusion and Discussion

In this chapter, we have proposed a training paradigm which uses historical summary to mitigate catastrophic forgetting. There are many choices for summary construction, either through dimension reduction, or through choosing a smaller number of samples. To illustrate a simple summarization technique such as random sampling can be effective in mitigating catastrophic forgetting, we designed experiments with two types of tasks and compared our method against a state of the art method, EWC. Our experiments on both types of tasks show that historical summary performs better than EWC and vanilla

stochastic gradient descent (SGD).

The findings of this work show promise in solving a long-standing problem in streaming learning. It might be useful to experiment with more summarization methods and more complex task learning to develop effective training methodologies to discover knowledge from high velocity big data.

Chapter 6

Conclusion and Future Work

Standard practices in exploring the landscape of big data analytics involve choosing or designing an appropriate statistical or machine learning method, distributing the parallel workload of the selected method across parallel processing units through efficient scheduling and software design methodologies, developing high performance computing (HPC) platforms through software or hardware innovation. Using HPC platforms for big data analytics can be expensive, so this thesis proposes frugality in resource usage by investigating three properties of the big data: volume, variety, and velocity. This thesis proposes three guidelines, each targeting one property of the big data.

1. ***Guideline targeting volume:*** *Explore geometric properties of high dimensional data for succinct representation*
2. ***Guideline targeting variety:*** *Design domain-aware algorithms through mapping of domain problems to computation problems*
3. ***Guideline targeting velocity:*** *Leverage incremental arrival of data through incremental analysis and invention of problem-specific merging methodologies*

Throughout this thesis, we demonstrated applications of these guidelines through the solution approaches of representative problems. To demonstrate Guideline 1, we chose the problem of developing a portable parallel dimension reduction tool (WMDS). To demonstrate the application of Guideline 2, we chose the problem of identifying multi-hit combinations of carcinogenic gene mutations. Guideline 3 was demonstrated through two

problems: (1) incremental sequence similarity search, and (2) mitigating catastrophic forgetting while training deep learning models in a streaming setting. The solution approaches show that multiple guidelines can be useful in solving a big data problem since challenges might arise from any of its three characteristics.

6.1 Applications and Artifacts

1. *The Claret [45] offers a portable and parallel dimension reduction tool (WMDS) that leverages geometric properties of high-dimensional data (preservation of pairwise distances after random projection) and portable parallel programming paradigm offered by OpenCL that can run on available single accelerator systems. (Guideline 1)*

2. *Claret enabled Web Andromeda shows through the use of geometric properties of high dimensional data, parallelization on consumer accelerator, and incremental gradient computation, visual analytics on domain problems relying on WMDS can be run in real-time without significant delay. (Guideline 1, Guideline 3)*

3. *Identification of multi-hit combinations of carcinogenic genes [48] by mapping a domain problem to a well-known NP-Complete problem weighted set cover problem demonstrates the effectiveness of incorporating domain knowledge in designing approximate algorithms to solve domain problems. (Guideline 2)*

4. *Scaling up weighted set cover algorithm to identify 3-hit combinations of carcinogenic genes [46] by representing cancer genomics data using a compressed binary matrix demonstrates that the application of domain guided data representation and domain-aware algorithm design can help us discover scientific knowledge. (Guideline 1, Guideline 2)*

5. *Scaling out the combinatorial workload of a weighted set cover algorithm on Summit supercomputer to identify 4-hit combinations of carcinogenic genes through various memory optimizations and equitable scheduling demonstrates how the guidelines can be used to co-design effective solution for domain problems. (Guideline 1, Guideline 2)*

6. *iBLAST [47] performs incremental sequence similarity search against gradually growing sequence databases through the design of novel statistical formulas to correct e-values. iBLAST merges two partial results by re-computing or re-scaling e-values through these novel formulas. (Guideline 3)*

7. *A comparative study into various approaches to mitigate catastrophic forgetting in training deep learning models in a streaming setting demonstrates the efficacy of using data summary. This work also sheds light on the challenges of incremental training of deep learning models. (Guideline 1, Guideline 3)*

6.2 Future Work

There are several future research directions concerning the applications solved in this thesis. Like weighted multi-dimensional scaling (WMDS) solved in Claret [45], parallel and portable tools to perform other dimension reduction techniques that use metric distances can be designed and developed.

In identifying carcinogenic gene combinations, more precise domain knowledge needs to be incorporated to make the identified combinations more biologically meaningful. Many of the identified combinations consist of passenger mutations, removing the passenger mutations and their combinations from the search space can be beneficial in two ways: (1) the identified combinations will more likely consist cancer-specific driver mutations, (2) fewer candidate genes means fewer candidate combinations, this will result in less

computation. After refining the combinations through passenger mutation elimination, validation of these combinations as bio-markers of cancers needs to be done through bio-chemical experiments. After refinement and validation, these combinations can be used to design precision medicine and diagnostic tools.

The incremental analysis of iBLAST can be extended to other sequence similarity search tools such as DIAMOND by developing appropriate statistics. There are a number of workflows in bioinformatic research that use BLAST tool as one part of the pipeline. Replacing traditional BLAST with iBLAST can expedite those workflows significantly.

We have identified a few challenges and approaches to mitigate catastrophic forgetting while training a deep learning model in a streaming setting. Similar studies can be conducted to address other problems in streaming learning (e.g., concept drift).

Additional Information *Additional information is available in Appendix B.*

Appendices

Appendix A

Identifying Multi-Hit Combinations

A.1 Identifying Somatic Variants Using MuTect2 and VEP: Command Parameters

Preliminary step of our approach is to identify meaningful somatic variations. We identify somatic variations by comparing tumor and normal tissue samples with blood-derived normal samples using MuTect2.

We use the following command for germline variant calling.

```
java -jar GenomeAnalysisTK.jar
     -T MuTect2 -R GRCh38.d1.vd1.fa
     -I:tumor tissue.bam -I:normal blood-normal.bam -o output.vcf
     --disable_auto_index_creation_and_locking_when_reading_rods
     --max_alt_alleles_in_normal_count 2
     --contamination_fraction_to_filter 0.02 --dbsnp dbsnp.vcf
     --cosmic cosmic.vcf 279
```

Once the variants are identified, we compute the effect of these variants using Variant Effect Predictor(VEP). We used the following VEP command for effect prediction:

```
perl variant_effect_predictor.pl
     -i input.vcf -o output.vep --fasta GRCh38-directory --nostats
```

A.2 Algorithm and Data Structure

Input and Data Structure *We first prepare the following data structures from the tumor tissue samples and normal tissue samples.*

Data	*Description*
$\mathsf{G}_{selected}$	*a list of selected genes.* $List < Gene >$.
$TumorCoverage$	*a dictionary object mapping any gene to a set of tumor samples it covers.* $Dict < Gene, Set < TumorSample >>$.
$NormalCoverage$	*a dictionary object mapping any gene to a set of normal samples it covers.* $Dict < Gene, Set < NormalSample >>$.
$TumorSamples$	*a set of all tumor tissue samples.* $Set < TumorSamples >$
$NormalSamples$	*a set of all normal tissue samples.* $Set < NormalSamples >$
$CandidateCombination$	$G_{selected} \times G_{selected}$, *a set of all candidate Combinations.*

Algorithm 14 Identifying two-hit combinations.

Require: $\mathsf{G}_{selected}$, $TumorCoverage$, $NormalCoverage$, $TumorSamples$, $NormalSamples$, $CandidateCombinations$

Initialize $C \leftarrow \phi$
$CoveredTumorSamples \leftarrow \phi$
$CoveredNormalSamples \leftarrow \phi$
$\mathbb{C}$
while $|coveredTumorSamples| \neq |TumorSamples|$ **do**
    $weights \leftarrow \phi$
    for $combination \in CandidateCombinations$ **do**
        $g_1, g_2 \leftarrow extract(combination)$
        $Cov_T(g_1) \leftarrow TumorCoverage.getSamples(g_1)$
        $Cov_T(g_2) \leftarrow TumorCoverage.getSamples(g_2)$
        $Cov_T(combination) \leftarrow Cov_T(g_1) \cap Cov_T(g_2) - CoveredTumorSamples$

        $Cov_N(g_1) \leftarrow NormalCoverage.getSamples(g_1)$
        $Cov_N(g_2) \leftarrow NormalCoverage.getSamples(g_2)$
        $Cov_N(combination) \leftarrow Cov_N(g_1) \cap Cov_N(g_2) - CoveredNormalSamples$
        $weight(combination) \leftarrow f(Cov_N(combination), Cov_T(combination))$
        $weights[combination] \leftarrow weight(combination)$

    $bestCombination \leftarrow argmin\{weights\}$
    Update $CoveredTumorSamples$ and $CoveredNormalSamples$ using $bestCombination$
    $\mathbb{C} \leftarrow \mathbb{C} \cup \{bestCombination\}$
    $CandidateCombinations \leftarrow CandidateCombinations - bestCombination$

return $\mathbb{C}$

A.3 Robustness of Our Algorithm Across Sets of Partitions

We partitioned the data in three different ways, and the average classification performance in each case is comparable. Table A1 shows the result when we run our algorithm on the second partition.

			Discovery Set								Validation Set								
			Tumor Samples				Normal Samples				Tumor Samples					Normal Samples				
Cancer Type	#Hits	#Combinations	True Positives	False Negatives	Total	Sensitivity	True Negatives	False Positives	Total	Specificity	True Positives	False Negatives	Total	Sensitivity	95% Confidence Interval	True Negatives	False Positives	Total	Specificity	95% Confidence Interval
BLCA	2	17	283	0	283	1.00	239	1	240	1.00	78	7	85	0.92	83-96%	78	15	93	0.84	74-90%
BRCA	2	7	674	0	674	1.00	234	6	240	0.97	235	2	237	0.99	96-99%	82	11	93	0.88	79-93%
CESC	2	9	209	0	209	1.00	240	0	240	1.00	57	8	65	0.88	77-94%	82	11	93	0.88	79-93%
COAD	2	10	287	0	287	1.00	239	1	240	1.00	93	5	98	0.95	88-98%	89	4	93	0.96	89-98%
GBM	2	10	252	0	252	1.00	240	0	240	1.00	71	8	79	0.90	81-95%	88	5	93	0.95	87-98%
HNSC	2	12	360	0	360	1.00	238	2	240	0.99	101	9	110	0.92	85-96%	87	6	93	0.94	86-97%
KIRP	2	10	181	0	181	1.00	239	1	240	1.00	44	3	47	0.94	82-98%	84	9	93	0.90	82-95%
LGG	2	11	360	0	360	1.00	236	4	240	0.98	112	7	119	0.94	88-97%	85	8	93	0.91	83-96%
LIHC	2	8	223	0	223	1.00	239	1	240	1.00	83	6	89	0.93	85-97%	85	8	93	0.91	83-96%
LUAD	2	12	310	0	310	1.00	239	1	240	1.00	93	6	99	0.94	87-97%	85	8	93	0.91	83-96%
LUSC	2	11	228	0	228	1.00	239	1	240	1.00	65	12	77	0.84	74-91%	89	4	93	0.96	89-98%
OV	2	9	231	0	231	1.00	240	0	240	1.00	83	3	86	0.97	90-99%	90	3	93	0.97	90-99%
PRAD	2	21	321	0	321	1.00	239	1	240	1.00	87	13	100	0.87	78-92%	72	21	93	0.77	67-85%
SARC	2	5	151	0	151	1.00	240	0	240	1.00	66	2	68	0.97	89-99%	93	0	93	1.00	96-100%
STAD	2	17	297	0	297	1.00	239	1	240	1.00	84	7	91	0.92	84-96%	77	16	93	0.83	73-89%
THCA	2	16	323	0	323	1.00	239	1	240	1.00	92	6	98	0.94	87-97%	83	10	93	0.89	81-94%
UCEC	2	6	360	0	360	1.00	234	6	240	0.97	135	0	135	1.00	97-100%	87	6	93	0.94	86-97%
Total	2	191	5050	0	5050	1.00	4053	27	4080	0.99	1579	104	1683	0.94	92-94%	1436	145	1581	0.91	89-92%

Table A1: Result for two-hit combinations.

A.4 Identified Combinations for 17 Cancer Types

Table A2-A18 show identified 2*-hit combinations for* 17 *cancer types using the first partition.*

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000205277	ENSG00000184956	781	0.857299670692
2	ENSG00000149531	ENSG00000211896	347	0.380900109769
3	ENSG00000219481	ENSG00000173213	305	0.334796926454
4	ENSG00000185567	ENSG00000090512	81	0.0889132821076
5	ENSG00000170471	ENSG00000205869	79	0.0867178924259
6	ENSG00000178104	ENSG00000275113	47	0.0515916575192
7	ENSG00000149531	ENSG00000084731	29	0.0318331503842
8	ENSG00000137210	ENSG00000198888	8	0.00878155872667

Table A2: Sample coverage by combinations for BRCA.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000172199	ENSG00000173213	214	0.646525679758
2	ENSG00000149531	ENSG00000211896	116	0.350453172205
3	ENSG00000149531	ENSG00000161905	109	0.329305135952
4	ENSG00000127481	ENSG00000158445	103	0.311178247734
5	ENSG00000237541	ENSG00000171862	91	0.274924471299
6	ENSG00000177731	ENSG00000124092	88	0.26586102719
7	ENSG00000198601	ENSG00000100151	41	0.123867069486
8	ENSG00000149531	ENSG00000050438	41	0.123867069486
9	ENSG00000125498	ENSG00000166272	18	0.0543806646526
10	ENSG00000241322	ENSG00000176302	6	0.0181268882175

Table A3: Sample coverage by combinations for GBM.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000205277	ENSG00000149531	334	0.674747474747
2	ENSG00000184956	ENSG00000173213	282	0.569696969697
3	ENSG00000196126	ENSG00000171862	224	0.452525252525
4	ENSG00000205277	ENSG00000278662	188	0.379797979798
5	ENSG00000171862	ENSG00000211896	172	0.347474747475
6	ENSG00000141510	ENSG00000174501	159	0.321212121212
7	ENSG00000079841	ENSG00000205277	79	0.159595959596
8	ENSG00000198128	ENSG00000102890	22	0.0444444444444
9	ENSG00000197887	ENSG00000161031	19	0.0383838383838
10	ENSG00000243073	ENSG00000196460	10	0.020202020202

Table A4: Sample coverage by combinations for UCEC.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	192	0.521739130435
2	ENSG00000169862	ENSG00000149531	175	0.475543478261
3	ENSG00000186409	ENSG00000139687	35	0.0951086956522
4	ENSG00000213928	ENSG00000163435	24	0.0652173913043
5	ENSG00000204479	ENSG00000124762	24	0.0652173913043
6	ENSG00000169862	ENSG00000119720	24	0.0652173913043
7	ENSG00000141510	ENSG00000171502	23	0.0625
8	ENSG00000109758	ENSG00000171936	21	0.0570652173913
9	ENSG00000099917	ENSG00000116044	16	0.0434782608696
10	ENSG00000147050	ENSG00000108840	15	0.0407608695652
11	ENSG00000240864	ENSG00000196498	13	0.0353260869565
12	ENSG00000171680	ENSG00000071626	11	0.0298913043478
13	ENSG00000153933	ENSG00000244482	11	0.0298913043478
14	ENSG00000158290	ENSG00000089041	9	0.0244565217391
15	ENSG00000156650	ENSG00000139910	9	0.0244565217391
16	ENSG00000163959	ENSG00000153815	6	0.0163043478261
17	ENSG00000205356	ENSG00000125810	5	0.0135869565217
18	ENSG00000126262	ENSG00000166736	5	0.0135869565217

Table A5: Sample coverage by combinations for BLCA.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	129	0.565789473684
2	ENSG00000134775	ENSG00000149531	112	0.491228070175
3	ENSG00000197915	ENSG00000227152	68	0.298245614035
4	ENSG00000149531	ENSG00000204525	66	0.289473684211
5	ENSG00000169174	ENSG00000145920	37	0.162280701754
6	ENSG00000197915	ENSG00000204661	26	0.114035087719
7	ENSG00000213516	ENSG00000175193	5	0.0219298245614
8	ENSG00000159409	ENSG00000137337	4	0.0175438596491
9	ENSG00000142798	ENSG00000180767	4	0.0175438596491
10	ENSG00000070413	ENSG00000140795	4	0.0175438596491
11	ENSG00000004866	ENSG00000178188	3	0.0131578947368

Table A6: Sample coverage by combinations for KIRP.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	176	0.577049180328
2	ENSG00000141510	ENSG00000149531	145	0.475409836066
3	ENSG00000141510	ENSG00000130226	37	0.12131147541
4	ENSG00000141510	ENSG00000221900	32	0.104918032787
5	ENSG00000141510	ENSG00000170959	31	0.101639344262
6	ENSG00000155657	ENSG00000205246	29	0.0950819672131
7	ENSG00000133863	ENSG00000141510	29	0.0950819672131
8	ENSG00000187537	ENSG00000179603	11	0.0360655737705
9	ENSG00000127507	ENSG00000121898	8	0.0262295081967
10	ENSG00000170382	ENSG00000173531	7	0.0229508196721
11	ENSG00000143882	ENSG00000167822	5	0.016393442623
12	ENSG00000007923	ENSG00000197841	3	0.00983606557377

Table A7: Sample coverage by combinations for LUSC.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000205277	ENSG00000149531	153	0.698630136986
2	ENSG00000184956	ENSG00000173213	137	0.625570776256
3	ENSG00000205277	ENSG00000197978	87	0.397260273973
4	ENSG00000145506	ENSG00000205277	68	0.310502283105
5	ENSG00000169047	ENSG00000273976	26	0.118721461187
6	ENSG00000173662	ENSG00000006377	11	0.0502283105023

Table A8: Sample coverage by combinations for SARC.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	188	0.686131386861
2	ENSG00000205277	ENSG00000149531	168	0.613138686131
3	ENSG00000205277	ENSG00000278662	128	0.467153284672
4	ENSG00000204525	ENSG00000149531	77	0.28102189781
5	ENSG00000205277	ENSG00000157423	62	0.226277372263
6	ENSG00000197915	ENSG00000227152	46	0.167883211679
7	ENSG00000279804	ENSG00000131951	12	0.043795620438
8	ENSG00000004455	ENSG00000178199	7	0.0255474452555
9	ENSG00000237541	ENSG00000153391	6	0.021897810219

Table A9: Sample coverage by combinations for CESC.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000149531	ENSG00000134775	197	0.63141025641
2	ENSG00000227152	ENSG00000184956	137	0.439102564103
3	ENSG00000186844	ENSG00000134775	125	0.400641025641
4	ENSG00000177212	ENSG00000173213	120	0.384615384615
5	ENSG00000149531	ENSG00000188162	92	0.294871794872
6	ENSG00000147234	ENSG00000145920	77	0.246794871795
7	ENSG00000141510	ENSG00000130558	34	0.108974358974
8	ENSG00000198128	ENSG00000171368	13	0.0416666666667
9	ENSG00000171680	ENSG00000204310	5	0.0160256410256

Table A10: Sample coverage by combinations for LIHC.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	252	0.616136919315
2	ENSG00000169862	ENSG00000149531	205	0.501222493888
3	ENSG00000198216	ENSG00000133703	81	0.19804400978
4	ENSG00000227152	ENSG00000010438	70	0.171149144254
5	ENSG00000116147	ENSG00000141510	63	0.154034229829
6	ENSG00000172765	ENSG00000118046	36	0.0880195599022
7	ENSG00000187741	ENSG00000081842	21	0.0513447432763
8	ENSG00000146648	ENSG00000140323	17	0.041564792176
9	ENSG00000204479	ENSG00000179593	16	0.039119804401
10	ENSG00000181396	ENSG00000156650	15	0.0366748166259
11	ENSG00000171680	ENSG00000185640	10	0.0244498777506
12	ENSG00000184677	ENSG00000165370	9	0.0220048899756
13	ENSG00000146830	ENSG00000272514	7	0.0171149144254

Table A11: Sample coverage by combinations for LUAD.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000138413	ENSG00000184956	375	0.782881002088
2	ENSG00000184956	ENSG00000173213	294	0.613778705637
3	ENSG00000149531	ENSG00000158865	115	0.240083507307
4	ENSG00000179912	ENSG00000149531	101	0.210855949896
5	ENSG00000173826	ENSG00000177468	7	0.0146137787056
6	ENSG00000167395	ENSG00000144791	6	0.0125260960334
7	ENSG00000130244	ENSG00000112984	6	0.0125260960334
8	ENSG00000144381	ENSG00000204516	5	0.0104384133612
9	ENSG00000134184	ENSG00000122257	5	0.0104384133612

Table A12: Sample coverage by combinations for LGG.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	308	0.655319148936
2	ENSG00000141510	ENSG00000211896	195	0.414893617021
3	ENSG00000149531	ENSG00000204525	92	0.195744680851
4	ENSG00000214324	ENSG00000055609	83	0.176595744681
5	ENSG00000146555	ENSG00000149531	64	0.136170212766
6	ENSG00000276644	ENSG00000141510	21	0.0446808510638
7	ENSG00000154222	ENSG00000099957	18	0.0382978723404
8	ENSG00000204442	ENSG00000063169	16	0.0340425531915
9	ENSG00000065526	ENSG00000021645	15	0.031914893617
10	ENSG00000146112	ENSG00000166343	12	0.0255319148936
11	ENSG00000197429	ENSG00000154175	11	0.0234042553191
12	ENSG00000198793	ENSG00000179588	4	0.00851063829787
13	ENSG00000117148	ENSG00000211967	4	0.00851063829787

Table A13: Sample coverage by combinations for HNSC.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	221	0.569587628866
2	ENSG00000149531	ENSG00000204525	89	0.229381443299
3	ENSG00000163283	ENSG00000149531	84	0.216494845361
4	ENSG00000141510	ENSG00000177548	39	0.100515463918
5	ENSG00000110046	ENSG00000171936	39	0.100515463918
6	ENSG00000162927	ENSG00000141510	36	0.0927835051546
7	ENSG00000116251	ENSG00000198216	36	0.0927835051546
8	ENSG00000163629	ENSG00000141510	32	0.0824742268041
9	ENSG00000234745	ENSG00000039068	29	0.0747422680412
10	ENSG00000116251	ENSG00000198929	24	0.0618556701031
11	ENSG00000168702	ENSG00000120963	20	0.0515463917526
12	ENSG00000159650	ENSG00000211896	15	0.0386597938144
13	ENSG00000153201	ENSG00000100151	14	0.0360824742268
14	ENSG00000196126	ENSG00000158488	12	0.0309278350515
15	ENSG00000124466	ENSG00000185177	11	0.0283505154639
16	ENSG00000167548	ENSG00000131203	8	0.020618556701
17	ENSG00000211721	ENSG00000203933	7	0.0180412371134
18	ENSG00000184677	ENSG00000184814	3	0.00773195876289
19	ENSG00000116350	ENSG00000197245	3	0.00773195876289

Table A14: Sample coverage by combinations for STAD.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000133056	ENSG00000158445	209	0.659305993691
2	ENSG00000141510	ENSG00000149531	173	0.545741324921
3	ENSG00000204501	ENSG00000173213	128	0.403785488959
4	ENSG00000141510	ENSG00000065534	90	0.283911671924
5	ENSG00000153820	ENSG00000141510	72	0.227129337539
6	ENSG00000155657	ENSG00000133193	71	0.223974763407
7	ENSG00000141298	ENSG00000133112	45	0.141955835962
8	ENSG00000080031	ENSG00000077782	32	0.10094637224

Table A15: Sample coverage by combinations for OV.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	256	0.608076009501
2	ENSG00000169862	ENSG00000149531	185	0.439429928741
3	ENSG00000163283	ENSG00000149531	102	0.242280285036
4	ENSG00000038358	ENSG00000157764	92	0.218527315914
5	ENSG00000157764	ENSG00000100290	70	0.166270783848
6	ENSG00000157764	ENSG00000170369	68	0.161520190024
7	ENSG00000157764	ENSG00000186818	33	0.0783847980998
8	ENSG00000104974	ENSG00000213281	27	0.0641330166271
9	ENSG00000175216	ENSG00000211896	23	0.0546318289786
10	ENSG00000204479	ENSG00000205246	16	0.0380047505938
11	ENSG00000118777	ENSG00000100626	11	0.0261282660333
12	ENSG00000113649	ENSG00000186395	8	0.0190023752969
13	ENSG00000137492	ENSG00000155034	3	0.00712589073634

Table A16: Sample coverage by combinations for THCA.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000184956	ENSG00000173213	247	0.586698337292
2	ENSG00000169862	ENSG00000149531	203	0.482185273159
3	ENSG00000163283	ENSG00000149531	90	0.21377672209
4	ENSG00000211896	ENSG00000168096	37	0.0878859857482
5	ENSG00000213516	ENSG00000121067	31	0.0736342042755
6	ENSG00000211896	ENSG00000135341	26	0.061757719715
7	ENSG00000100401	ENSG00000008988	25	0.0593824228029
8	ENSG00000154358	ENSG00000112559	19	0.0451306413302
9	ENSG00000187545	ENSG00000227152	17	0.0403800475059
10	ENSG00000196498	ENSG00000159625	14	0.0332541567696
11	ENSG00000204442	ENSG00000221923	12	0.0285035629454
12	ENSG00000196539	ENSG00000205445	12	0.0285035629454
13	ENSG00000198502	ENSG00000197595	11	0.0261282660333
14	ENSG00000177548	ENSG00000180104	8	0.0190023752969
15	ENSG00000152661	ENSG00000043355	8	0.0190023752969
16	ENSG00000196187	ENSG00000152086	7	0.0166270783848
17	ENSG00000198128	ENSG00000162009	6	0.0142517814727
18	ENSG00000134545	ENSG00000185519	6	0.0142517814727
19	ENSG00000116721	ENSG00000142546	4	0.00950118764846
20	ENSG00000143226	ENSG00000099889	2	0.00475059382423

Table A17: Sample coverage by combinations for PRAD.

Combination	Gene1	Gene2	#Samples Covered	Fraction Coverage
1	ENSG00000134982	ENSG00000149531	281	0.72987012987
2	ENSG00000184956	ENSG00000173213	214	0.555844155844
3	ENSG00000149531	ENSG00000180329	158	0.41038961039
4	ENSG00000120314	ENSG00000157423	60	0.155844155844
5	ENSG00000184634	ENSG00000198786	42	0.109090909091
6	ENSG00000176542	ENSG00000100151	39	0.101298701299
7	ENSG00000204130	ENSG00000188766	24	0.0623376623377
8	ENSG00000154330	ENSG00000000971	20	0.0519480519481
9	ENSG00000162620	ENSG00000110074	10	0.025974025974

Table A18: Sample coverage by combinations for COAD.

A.5 Correlation Between Genes Within Combinations

We investigate whether the sample coverages of genes within a combination are correlated in normal samples to determine if we may be identifying passenger mutations as part of the combinations. Figure A1 shows $-\log_{10} p$ *against Pearson's correlation coefficients, where* p *is the p-value. We find no evidence of the genes within combinations being significantly correlated.*

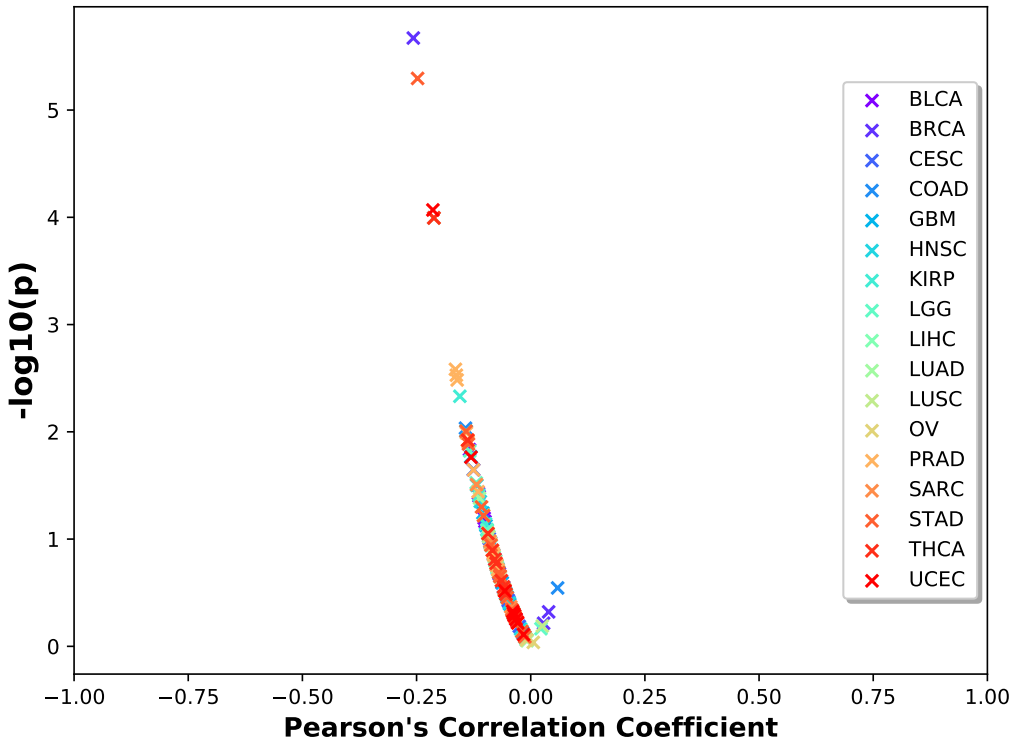


Figure A1: $-\log_{10} p$ vs Pearson correlation coefficient plot for all pairs of genes in all combinations identified by our method using the normal samples, where p is the p-value. Since no pair has a p-value < 0.005 (or $-\log_{10} p \geq 2.301$) and absolute Pearson correlation coefficient greater than 0.50, none of the combinations appear to have correlated genes.

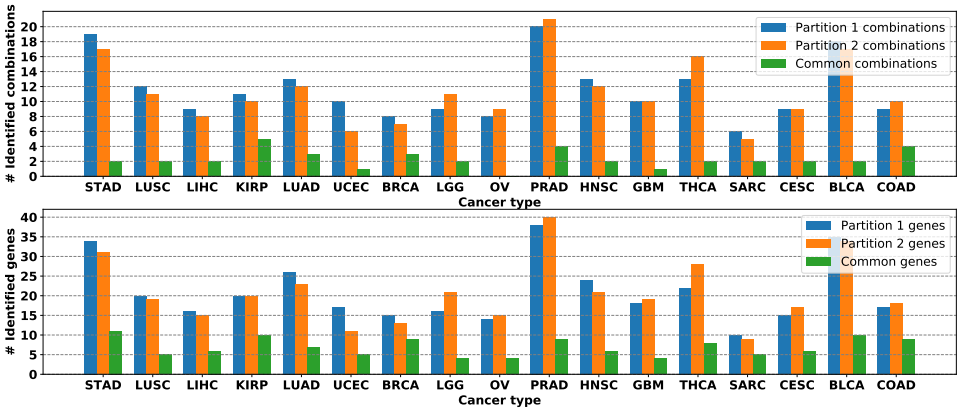


Figure A2: Identified genes and combinations shared between two sets of partitions. $(1-5)$ combinations are shared between two sets and $(4-10)$ genes are shared between two sets.

A.6 Coverage of Samples by Identified Combinations

Figure A3 shows fractions of tumor samples covered by identified combinations. Most samples are covered by the top combinations, while a very small number of samples require a large number of combinations. Figure A4 shows that this distribution is similar for different ways of partitioning data.

Figure A5 shows that many samples can be covered by more than one combinations. These overlapping combinations might constitute more than two hits.

A.7 Distinguishing Between Driver and Passenger Mutations

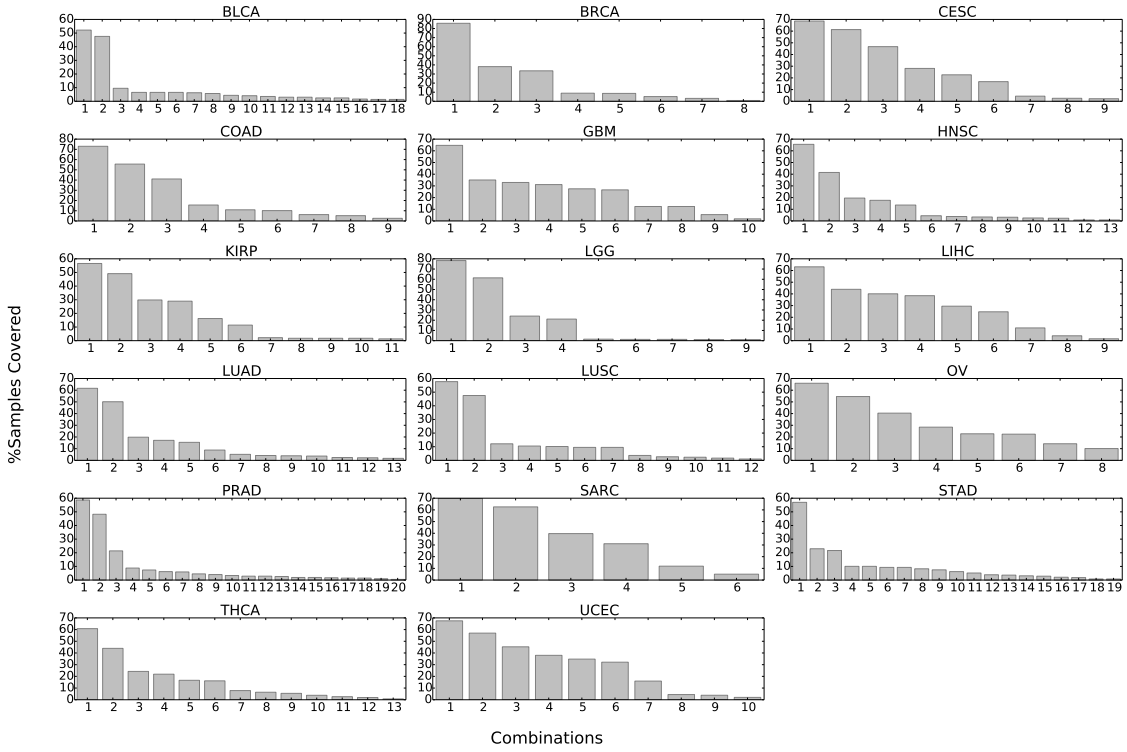


Figure A3: Occurrence of two-hit combinations identified in tumor samples, for the seventeen cancer types considered. The top combination occurs in 65% of tumor samples, on average, while 42% of the combinations occur in less than 5% of the samples. Total percentage exceeds 100% because samples can contain multiple combinations.

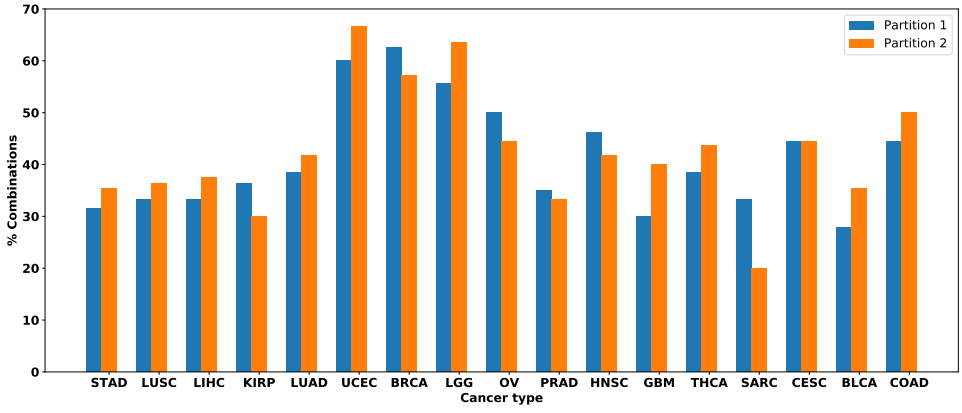


Figure A4: Percentage of identified combinations covering the last 5% of samples for the 17 cancer types. On average, the last 42% of the combinations occur in 5% of the tumor samples.

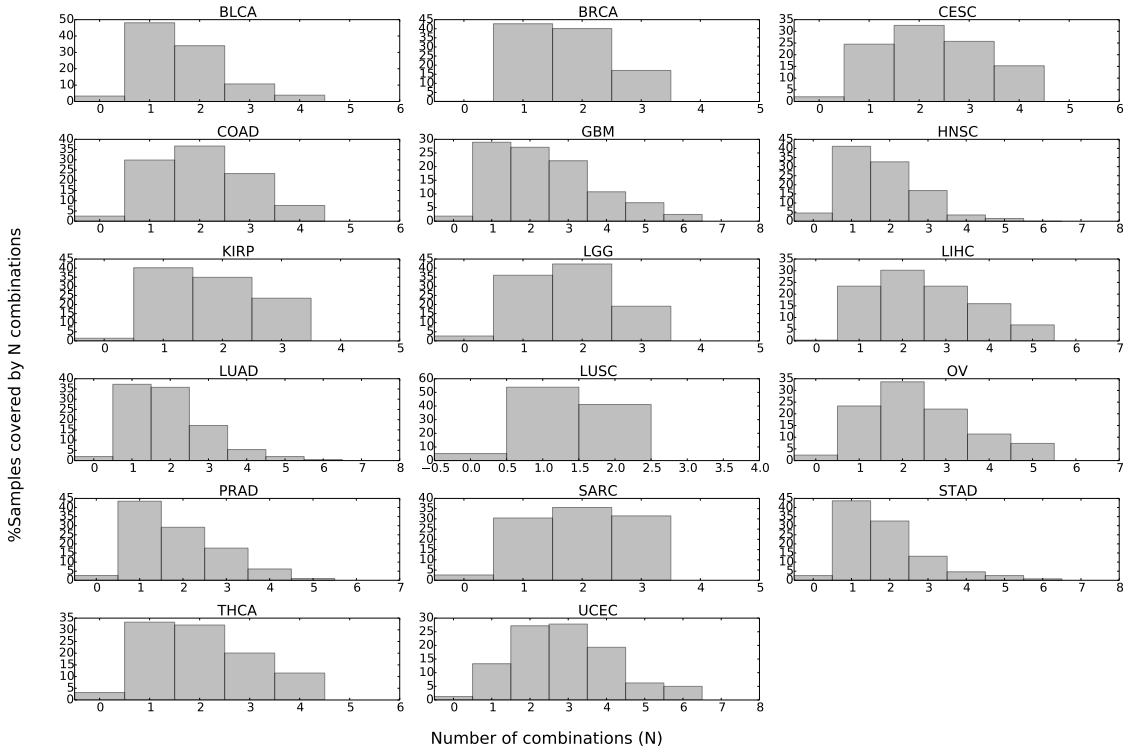


Figure A5: Distribution of overlapping combinations for 17 cancer types. 64.5% of the tumor samples contain multiple combinations, suggesting that the two-hit combinations represent subsets of three or more hits.

Table A19: Reactome superpathways associated with genes in the top three two-hit combinations identified [58].

		Reactome Superpathways																				
Cancer	***Gene***	*Signal Transduction*	*Metabolism*	*Immune System*	*Gene Expression*	*Metabolism of proteins*	*Developmental Biology*	*Disease*	*Vesicle-mediated transport*	*Cell Cycle*	*Homeostasis*	*Cellular response to stress*	*Organelle biogenesys and maintenance*	*Neuronal Systems*	*DNA Repair*	*Extracellular matrix organization*	*Chromatin organization*	*Muscle contraction*	*Programmed Cell Death*	*Cell-Cell communication*	*DNA Replication*	*Circadian Clock*	*Reproduction*
	MUC6					***X***		***X***															
	TUBB8P1																						
BLCA	*CTNND2*																						
	FRG1BP																						

	CCDC30																						
	RB1									X		X									X		
BRCA	*MUC12*					X		X															
	MUC6					X		X															
	FRG1BP																						
	IGHG1			X																			
	AHNAK2																						
	FETUB																						
CESC	*MUC6*					X		X															
	TUBB8P1																						
	MUC12					X		X															
	GOLGA6L10																						
	MUC12					X		X															
	FRG1BP																						

COAD	*APC*	X				X		X											X						
	FRG1BP																								
	MUC6					X		X																	
	TUBB8P1																								
	FRG1BP																								
	CCDC43																								
GBM	*OR8U1*	X																							
	TUBB8P1																								
	FRG1BP																								
	ALOX15		X																						
	UBR4			X																					
	KCNB1		X											X											
HNSC	*MUC6*					X		X																	
	TUBB8P1																								
	TP53	X		X	X	X				X	X	X			X				X						

	IGHG1			X																			
	FRG1BP																						
	HLA-C			X																			
	MUC6					X		X															
	TUBB8P12																						
KIRP	*FHOD3*																						
	FRG1BP																						
	HRNR			X																			
	OR2T7																						
	IDH1		X	X				X															
	MUC6					X		X															
LGG	*MUC6*					X		X															
	TUBB8P12																						
	R3HDM2																						

	FRG1BP																						
LIHC	*FRG1BP*																						
	FHOD3																						
	OR2T7																						
	MUC6					X		X															
	OR2T33	X																					
	TUBB8P12																						
LUAD	*MUC6*					X		X															
	TUBB8P1																						
	CTNND2																						
	FRG1BP																						
	OR2T7																						
	PRSS3		X	X																			
LUSC	*MUC6*					X		X															
	TUBB8P12																						

	TP53	X		X	X	X				X	X	X			X				X				
	FRG1BP																						
	TP53	X		X	X	X				X	X	X			X				X				
	DPP6																						
	PIK3C2B		X																				
	KCNB1		X											X									
OV	*TP53*	X		X	X	X				X	X	X			X				X				
	FRG1BP																						
	TP53	X		X	X	X				X	X	X			X				X				
	MYLK	X																X					
	MUC6					X		X															
	TUBB8P12																						
PRAD	*CTNND2*																						
	FRG1BP																						

	RBMXL1																							
	SPOP	X																						
	MUC12					X		X																
	FRG1BP																							
SARC	*MUC6*					X		X																
SARC	*TUBB8P1*																							
	MUC12					X		X																
	GOLGA6L9																							
	MUC6					X		X																
	TUBB8P12																							
STAD	*ALPP*								X															
STAD	*FRG1BP*																							
	ATG2A																							
	OR10H3	X																						
	MUC6					X		X																
THCA																								

	TUBB8P1																					
	CTNND2																					
	FRG1BP																					
	EDC4				**X**																	
	BRAF	**X**		**X**			**X**	**X**						**X**								
	MUC12					**X**		**X**														
	FRG1BP																					
UCEC	*MUC6*					**X**		**X**														
	TUBB8P1																					
	HLA-DRB1			**X**																		
	PTEN	**X**	**X**	**X**	**X**	**X**		**X**														

Appendix B

iBLAST

B.1 Existing E-Value Correction Software and Their Features

Several sequence similarity tools such as mpiBLAST and NOBLAST require E-value correction. A close inspection of the approaches used by these tools will help understand the challenges in E-value correction.

mpiBLAST *mpiBLAST [44] is a parallel implementation of NCBI BLAST on the cluster. It segments the database, ports the segments into different nodes of a cluster, and runs parallel BLAST search jobs against database segments on different nodes. Once the parallel search jobs return, it aggregates the search result. It has two important contributions. First, it achieves super-linear speedup by reducing IO overhead (time spent in reading and writing hard-disk storage). Second, it is the first parallel BLAST tool to provide exact E-value statistics in contrast to approximate E-value statistics of other contemporary parallel implementations of NCBI BLAST.*

mpiBLAST's exact E-value statistics requires two steps. First, it collects the necessary statistical parameters for the entire database by performing a pseudo-run of the BLAST engine against the global database. Once it has the global parameters, it passes the global parameters (such as whole database length n, the total number of sequences N) to the parallel search jobs against segmented databases. mpiBLAST modifies some

functionalities of NCBI BLAST (blast.c, blastdef.h, blastkar.c, and blastutl.c) so that global parameters can be fed externally and that information can be used to calculate exact E-values.

For accurate E-value correction, mpiBLAST requires prior knowledge of the entire database.

NOBLAST *NOBLAST [101] provides new options for NCBI BLAST. It offers a way to correct E-values when split databases are used and the results need to be aggregated. E-value computation requires knowledge about the entire database size, the number of sequences in the whole database N and the total length of the database n. Using the values N, n and Karlin-Altschul statistical parameters which are independent of database size, the E-value can be computed using Karlin-Altschul statistics. First, NOBLAST computes the length adjustment using the knowledge about the complete original database, then, it computes effective search space using length adjustment, and finally, it computes the E-value using effective search space.*

In principle, NOBLAST takes a similar approach to mpiBLAST, as both provide global statistical parameters to the search jobs against a segmented database so that that exact E-value can be computed. While mpiBLAST's main contribution is a parallel implementation and E-value correction comes from the need of producing the same output as the sequential counterpart, NOBLAST's main contribution is an E-value correction. Both tools require prior knowledge about the entire database. Both tools were developed before Spouge's E-value statistics were introduced, so they didn't address E-value corrections for the BLAST programs that use Spouge's statistics.

B.2 E-Value Correction

We use algorithm 15 to recompute E-values for BLAST programs using Karlin-Altschul statistics. We first aggregate database sizes from two input results and use the aggregated sizeN to compute length adjustment l. Using N and l, we recompute E-values for both results.

Algorithm 15 Recomputing E-values for Karlin-Altschul statistics.

1: Input: $result1, result2$

2: $n \leftarrow result1.n + result2.n$

3: $m \leftarrow result1.m$

4: $N \leftarrow result1.N + result2.N$

5: $l \leftarrow recompute\_length\_adjustment(n, m, N)$

6: $recompute\_evalues(result1, l, N)$

7: $recompute\_evalues(result2, l, N)$

We use algorithm 16 to re-scale E-values for BLAST programs using Spouge statistics. First we aggregate the database sizes for two input results, and scale the E-values by a factor of the ratio between aggregated database size and the individual database size.

Algorithm 16 Re-scale E-values for Spouge statistics.

1: Input: $result1, result2$

2: $db\_length1 \leftarrow result1.db\_length$

3: $db\_length2 \leftarrow result2.db\_length$

4: $db\_length \leftarrow db\_length1 + db\_length2$

5: $re\text{-}scale\_evalues(result1, \frac{db\_length}{db\_length1})$

6: $re\text{-}scale\_evalues(result2, \frac{db\_length}{db\_length2})$

B.3 Creating Experimental Databases

Pre-formatted BLAST databases such nt *and* nr *come in incremental parts. With progression of time, new sequences are packaged in parts and added to the databases.*

B.3.1 Databases for Case Study I

For case study I, we consider three time steps when the nt and nr databases had 30*,* 40*, and* 50 *parts. For these three time periods, we construct three databases as instances of nt and nr by combining* 30*,* 40*, and* 50 *parts using BLAST tool* blastdb_aliastool*. The incremental databases between two periods are also constructed. Table B1 shows different instances of nt databases in three different periods.*

Period	Database parts	Number of sequences	Number of bases	Longest sequence length
0	0-29	25,117,275	80,740,533,243	774,434,471 bases
0-1	30-39	8,389,596	33,008,962,097	275,920,749 bases
1	0-39	33,506,871	113,749,495,340	774,434,471 bases
1-2	40-59	8,891,258	38,722,333,261	129,927,919 bases
2	0-49	42,398,129	152,471,828,601	774,434,471 bases

Table B1: Incremental nt databases for case study I.

Table B2 shows different instances of nr databases in three different periods.

Period	Database parts	Number of sequences	Number of residues	Longest sequence length
0	0-29	49,468,463	17,686,779,866	36,507 residues
0-1	30-39	15,878,318	6,065,300,773	35,523 residues
1	0-39	65,346,781	23,752,080,639	36,507 residues
1-2	40-49	16,448,075	6,278,067,810	38,105 residues
2	0-49	81,794,856	30,030,148,449	38,105 residues

Table B2: Incremental nr databases for case study I.

B.3.2 Databases for Case Study II

We construct nr database instances for time 0 and 1 by combining 64 and 90 parts respectively. We combine these parts using blastdb_aliastool.

Period	Database	Number of sequences	Total residues	Longest sequence length
0	0-63	109,407,071	40,077,622,077	38,105 residues
0-1	64-90	52,860,187	19,192,851,238	74,488 residues
2	0-90	162,267,258 sequences	59,270,473,315	74,488 residues

Table B3: Incremental nr databases for case study II.

B.4 Load-Balancing via Query Partitioning

For case study II and III, we have partitioned 17927 *queries into* 20 *query files based on number of residues after randomizing the order instead of a more straightforward partitioning based on number of queries while keeping the original order.*

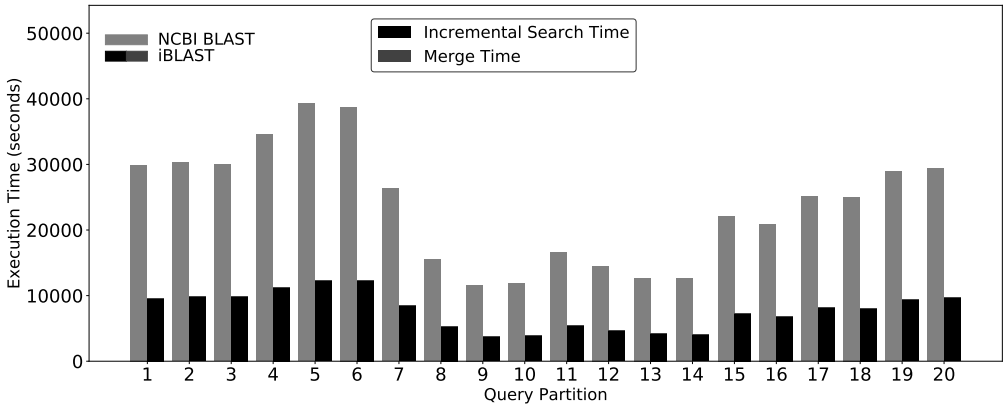


Figure B1: Execution time when a straightforward query partitioning scheme is adopted, which results in significant lack of load balancing. The standard deviation for the execution times for both iBLAST and NCBI BLAST searches are large (2748 and 8727 seconds, respectively).

If we partition the queries by making sure each partition has roughly same number of queries without disrupting their order, we get a range of execution times demonstrating lack of proper load balancing. The standard deviation in iBLAST search times is 2748 *seconds and standard deviation in NCBI BLAST search times is* 8727 *seconds. This means the compute nodes have to wait idly for* $1 - 3$ *hours on average. Figure B1 demonstrates the lack of load balancing.*

In contrast, when we first randomize the order of the queries and then partition the queries by making sure that all partitions have the roughly same number of residues, the standard deviations fall to 150 *and* 487 *seconds respectively (Figure B2).*

B.5 Explanation for NCBI BLAST Missing Many Top Hits

Due to the early cutoff of max target sequence used by its heuristic algorithm. NCBI BLAST performs search in two phases. In earlier phase (ungapped extension), it starts

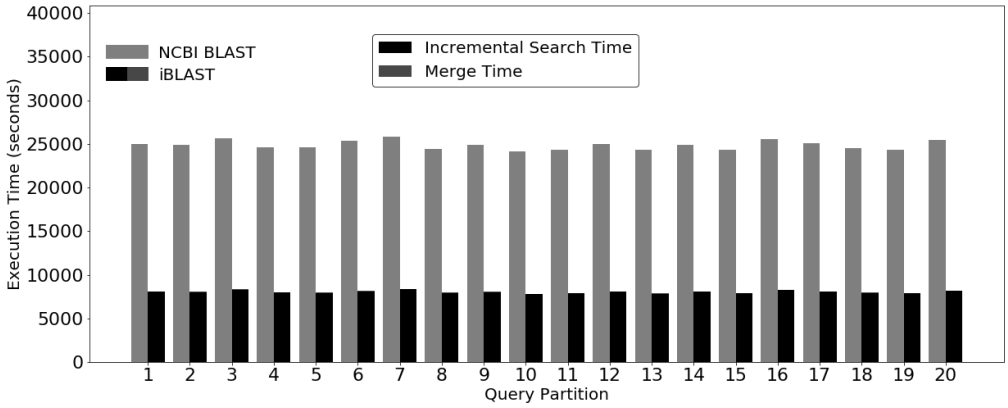


Figure B2: Execution time when our improvised query partitioning scheme is adopted, which results in better load balancing. The standard deviation for the execution times for both incremental and NCBI BLAST searches are minimal compared to the naive strategy (150 and 487 seconds, respectively).

with matching a seed sub-string between target and query sequence and then extends the matching pair in both direction without allowing any gap. In this phase, BLAST algorithm assigns some scores to these matching pairs and keeps only the very high scoring pairs using a cutoff determined by E-value cutoff or number of maximum hits. In the gapped phase, these selected high scoring pairs are further extended in both directions while allowing gaps and these evolved pairs get changed scores. Some of the pairs that did not make the cut during the ungapped extension, can become high scoring pairs. For a larger database, these missed opportunities are higher in number because there are more potential pairs in the ungapped phase. Since iBLAST is combining results from smaller databases, it misses relatively smaller number of those high scoring hits compared to NCBI BLAST.

Bibliography

[1] *Manifold learning.* `http://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html`.

[2] *Volume of data/information created worldwide from 2010 to 2024 (in zettabytes).* `https://www.statista.com/statistics/871513/worldwide-data-created/`.

[3] *Summit: America's newest and smartest supercomputer.* `https://www.olcf.ornl.gov/summit/`.

[4] *NVIDIA Tesla V100 GPU Architecture: The world's most advanced datacenter GPU. Technical report, NVIDIA, 08 2017. Also available at https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.*

[5] *Alex Abramovici, William E Althouse, Ronald WP Drever, Yekta Gürsel, Seiji Kawamura, Frederick J Raab, David Shoemaker, Lisa Sievers, Robert E Spero, Kip S Thorne, et al. Ligo: The laser interferometer gravitational-wave observatory.* science*, 256(5055):325–333, 1992.*

[6] *Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins.* Journal of computer and System Sciences*, 66 (4):671–687, 2003.*

[7] *Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets.* Combinatorial and computational geometry*, 52:1–30, 2005.*

[8] *Ahmed Ashour Ahmed, Dariush Etemadmoghadam, Jillian Temple, Andy G Lynch, Mohamed Riad, Raghwa Sharma, Colin Stewart, Sian Fereday, Carlos Caldas, Anna DeFazio, et al. Driver mutations in TP53 are ubiquitous in high grade serous carcinoma of the ovary.* The Journal of Pathology*, 221(1):49–56, 2010.*

[9] *Qais Al Hajri, Sajal Dash, Wu-chun Feng, Harold R Garner, and Ramu Anandakrishnan. Identifying multi-hit carcinogenic gene combinations: Scaling up a weighted set cover algorithm using compressed binary matrix representation on a gpu.* Scientific Reports*, 10(1):1–18, 2020.*

[10] *Bissan Al-Lazikani, Udai Banerji, and Paul Workman. Combinatorial drug therapy for cancer in the post-genomic era.* Nature biotechnology*, 30(7):679, 2012.*

[11] *Luay Almassalha, Greta Bauer, John Chandler, Scott Gladstein, Igal Szleifer, Hemant Roy, and Vadim Backman. The greater genomic landscape: The heterogeneous rvolution of cancer.* Cancer Res*, 76(19):5605–9, 2016.*

[12] *Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool.* Journal of molecular biology*, 215(3): 403–410, 1990.*

[13] *Amina Amadou, Maria I Waddington Achatz, and Pierre Hainaut. Revisiting tumor patterns and penetrance in germline tp53 mutation carriers: temporal phases of li–fraumeni syndrome.* Current opinion in oncology*, 30(1):23–29, 2018.*

[14] *Qian An, Sarah L Wright, Anthony V Moorman, Helen Parker, Mike Griffiths, Fiona M Ross, Teresa Davies, Christine J Harrison, and Jon C Strefford. Heterogeneous breakpoints in patients with acute lymphoblastic leukemia and the dic (9; 20)(p11~ 13; q11) show recurrent involvement of genes at 20q11. 21.* Haematologica*, 94(8):1164–1169, 2009.*

[15] Ramu Anandakrishnan. Estimating the number of genetic mutations (hits) required for carcinogenesis based on the distribution of somatic mutations. PLOS Comp Bio*, In Review, 2018.*

[16] Ramu Anandakrishnan, Tom RW Scogland, Andrew T Fenley, John C Gordon, Wu-chun Feng, and Alexey V Onufriev. Accelerating electrostatic surface potential calculation with multi-scale approximation on graphics processing units. Journal of Molecular Graphics and Modelling*, 28(8):904–910, 2010.*

[17] Ramu Anandakrishnan, Robin T Varghese, Nicholas A Kinney, and Harold R Garner. Estimating the number of genetic mutations (hits) required for carcinogenesis based on the distribution of somatic mutations. PLoS computational biology*, 15(3):e1006881, 2019.*

[18] Yoshimi Arima, Yasumichi Inoue, Tatsuhiro Shibata, Hidemi Hayashi, Osamu Nagano, Hideyuki Saya, and Yoichi Taya. Rb depletion results in deregulation of e-cadherin and induction of cellular phenotypic changes that are characteristic of the epithelial-to-mesenchymal transition. Cancer research*, 68(13):5104–5112, 2008.*

[19] Peter Armitage and Richard Doll. The age distribution of cancer and a multi-stage theory of carcinogenesis. Br J Cancer*, 8(1):1, 1954.*

[20] DJ Ashley. The two" hit" and multiple" hit" theories of carcinogenesis. British journal of cancer*, 23(2):313, 1969.*

[21] Alon Ben-Arie, Zion Hagay, Herzel Ben-Hur, and Ram Dgani. Elevated serum alkaline phosphatase may enable early diagnosis of ovarian cancer. European Journal of Obstetrics & Gynecology and Reproductive Biology*, 86(1):69–71, 1999.*

[22] Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, James

Ostell, Kim D Pruitt, and Eric W Sayers. GenBank. Nucleic Acids Research*, 46 (D1):D41–D47, 11 2017. ISSN 0305-1048. doi: 10.1093/nar/gkx1094.*

[23] Andrew Berchuck, Kerrie-Ann Heron, Michael E Carney, Johnathan M Lancaster, Elisa G Fraser, Vickie L Vinson, Amie M Deffenbaugh, Alexander Miron, Jeffrey R Marks, P Andrew Futreal, et al. Frequency of germline and somatic brca1 mutations in ovarian cancer. Clinical Cancer Research*, 4(10):2433–2437, 1998.*

[24] Alice H Berger, Angela N Brooks, Xiaoyun Wu, Yashaswi Shrestha, Candace Chouinard, Federica Piccioni, Mukta Bagul, Atanas Kamburov, Marcin Imielinski, Larson Hogstrom, et al. High-throughput phenotyping of lung cancer somatic mutations. Cancer cell*, 30(2):214–228, 2016.*

[25] Andrey Besedin, Pierre Blanchart, Michel Crucianu, and Marin Ferecatu. Evolutive deep models for online learning on data streams with no storage. In Workshop on Large-scale Learning from Data Streams in Evolving Environments*, 2017.*

[26] Anthony M Bolger, Marc Lohse, and Bjoern Usadel. Trimmomatic: a flexible trimmer for illumina sequence data. Bioinformatics*, 30(15):2114–2120, 2014.*

[27] Benjamin Buchfink, Chao Xie, and Daniel H Huson. Fast and sensitive protein alignment using diamond. Nature methods*, 12(1):59, 2014.*

[28] Sergiy Butenko and Wilbert E Wilhelm. Clique-detection models in computational biochemistry and genomics. European Journal of Operational Research*, 173(1): 1–17, 2006.*

[29] Jeffrey D Calhoun, Carlos G Vanoye, Fernando Kok, Alfred L George, and Jennifer A Kearney. Characterization of a KCNB1 variant associated with autism, intellectual disability, and epilepsy. Neurology Genetics*, 3(6):e198, 2017.*

[30] *Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L Madden. Blast+: architecture and applications.* BMC bioinformatics*, 10(1):421, 2009.*

[31] *Brandi L Cantarel, Ian Korf, Sofia MC Robb, Genis Parra, Eric Ross, Barry Moore, Carson Holt, Alejandro Sánchez Alvarado, and Mark Yandell. Maker: an easy-to-use annotation pipeline designed for emerging model organism genomes.* Genome research*, 18(1):188–196, 2008.*

[32] *Ethan Cerami, Jianjiong Gao, Ugur Dogrusoz, Benjamin E. Gross, Selcuk Onur Sumer, Bülent Arman Aksoy, Anders Jacobsen, Caitlin J. Byrne, Michael L. Heuer, Erik Larsson, Yevgeniy Antipin, Boris Reva, Arthur P. Goldberg, Chris Sander, and Nikolaus Schultz. The cbio cancer genomics portal: An open platform for exploring multidimensional cancer genomics data.* Cancer Discovery*, 2(5): 401–404, 2012.*

[33] *Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In* Visualization'96. Proceedings.*, pages 127–131. IEEE, 1996.*

[34] *Sisi Chen, Hao Yu, Michihiro Kobayashi, Rui Gao, H Scott Boswell, and Yan Liu. Gain-of-function mutant p53 enhances hematopoietic stem cell self-renewal.* Blood*, 124:260, 2014.*

[35] *Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In* Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.*

[36] *Feixiong Cheng, Junfei Zhao, and Zhongming Zhao. Advances in computational approaches for prioritizing driver mutations and significantly mutated genes in cancer genomes.* Briefings in Bioinformatics*, 17(4):642–656, 2015.*

[37] Vasek Chvatal. A greedy heuristic for the set-covering problem. Mathematics of Operations Research*, 4(3):233–235, 1979.*

[38] Charles J Clopper and Egon S Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika*, pages 404–413, 1934.*

[39] Francis S Collins, Michael Morgan, and Aristides Patrinos. The human genome project: lessons from large-scale biology. Science*, 300(5617):286–290, 2003.*

[40] Mireia Coma, Rubén Vicente, Silvia Busquets, Neus Carbó, Michael M Tamkun, Francisco J López-Soriano, Josep M Argilés, and Antonio Felipe. Impaired voltage-gated k+ channel expression in brain during experimental cancer cachexia. FEBS letters*, 536(1-3):45–50, 2003.*

[41] ENCODE Project Consortium et al. Identification and analysis of functional elements in 1% of the human genome by the encode pilot project. Nature*, 447 (7146):799, 2007.*

[42] Ellen R Copson, Tom C Maishman, Will J Tapper, Ramsey I Cutress, Stephanie Greville-Heygate, Douglas G Altman, Bryony Eccles, Sue Gerty, Lorraine T Durcan, Louise Jones, et al. Germline brca mutation and outcome in young-onset breast cancer (posh): a prospective cohort study. The lancet oncology*, 19 (2):169–180, 2018.*

[43] Ben Corden, Julian Jarman, Nicola Whiffin, Upasana Tayal, Rachel Buchan, Joban Sehmi, Andrew Harper, William Midwinter, Karen Lascelles, Vias Markides, et al. Association of Titin-truncating genetic variants with life-threatening cardiac arrhythmias in patients with dilated cardiomyopathy and implanted defibrillators. JAMA Network Open*, 2(6):e196520–e196520, 2019.*

[44] Aaron E Darling, Lucas Carey, and Wu-chun Feng. The design, implementation,

and evaluation of mpiblast. Technical report, Los Alamos National Laboratory, 2003.

[45] Sajal Dash, Anshuman Verma, Chris North, and Wu-chun Feng. Portable parallel design of weighted multi-dimensional scaling for real-time data analysis. In 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 10–17. IEEE, 2017.*

[46] Sajal Dash, Nick Kinney, Robin Varghese, Harold Garner, Wu-chun Feng, and Ramu Anandakrishnan. Identifying carcinogenic multi-hit combinations using weighted set cover algorithm. ICPP PhD Forum*, 2018.*

[47] Sajal Dash, Sarthok Rahman, Heather M Hines, and Wu-chun Feng. Incremental blast: incremental addition of new sequence databases through e-value correction. bioRxiv*, page 476218, 2018.*

[48] Sajal Dash, Nicholas A Kinney, Robin T Varghese, Harold R Garner, Wu-chun Feng, and Ramu Anandakrishnan. Differentiating between cancer and normal tissue samples using multi-hit combinations of genetic mutations. Scientific reports*, 9(1):1005, 2019.*

[49] Mark L Davison. Introduction to multidimensional scaling and its applications. Applied Psychological Measurement*, 7(4):373–379, 1983.*

[50] Marcelo Rodrigo de Castro, Catherine dos Santos Tostes, Alberto MR Dávila, Hermes Senger, and Fabricio AB da Silva. Sparkblast: scalable blast processing using in-memory operations. BMC bioinformatics*, 18(1):318, 2017.*

[51] Jan De Leeuw and Patrick Mair. Multidimensional scaling using majorization: Smacof in r. Department of Statistics, UCLA*, 2011.*

[52] *Nathan D. Dees, Qunyuan Zhang, Cyriac Kandoth, Michael C. Wendl, William Schierding, Daniel C. Koboldt, Thomas B. Mooney, Matthew B. Callaway, David Dooling, Elaine R. Mardis, Richard K. Wilson, and Li Ding. Music: identifying mutational significance in cancer genomes.* Genome Res*, 22(8):1589–1598, 2012.*

[53] *Yajun Deng, Qiqi Xie, Guangzhi Zhang, Shaoping Li, Zuolong Wu, Zhanjun Ma, Xuegang He, Yicheng Gao, Yonggang Wang, Xuewen Kang, et al. Slow skeletal muscle troponin t, titin and myosin light chain 3 are candidate prognostic biomarkers for Ewing's sarcoma.* Oncology Letters*, 18(6):6431–6442, 2019.*

[54] *Sohita Dhillon. Ivosidenib: first global approval.* Drugs*, 78(14):1509–1516, 2018.*

[55] *Sean R. Eddy. Profile hidden markov models.* Bioinformatics (Oxford, England)*, 14(9):755–763, 1998.*

[56] *Alaa A Elbendary, Frank D Cirisano, AC Evans, Penelope L Davis, JD Iglehart, Jeffrey R Marks, and Andrew Berchuck. Relationship between p21 expression and mutation of the p53 tumor suppressor gene in normal and malignant ovarian epithelial cells.* Clinical Cancer Research*, 2(9):1571–1575, 1996.*

[57] *Aristides G Eliopoulos, David J Kerr, Jonathan Herod, Liz Hodgkins, Stanislaw Krajewski, John C Reed, and Lawrence S Young. The control of apoptosis and drug resistance in ovarian cancer: influence of p53 and Bcl-2.* Oncogene*, 11(7): 1217–1228, 1995.*

[58] *Antonio Fabregat, Konstantinos Sidiropoulos, Phani Garapati, Marc Gillespie, Kerstin Hausmann, Robin Haw, Bijay Jassal, Steven Jupe, Florian Korninger, Sheldon McKay, Lisa Matthews, Bruce May, Marija Milacic, Karen Rothfels, Veronica Shamovsky, Marissa Webber, Joel Weiser, Mark Williams, Guanming Wu, Lincoln Stein, Henning Hermjakob, and Peter D'Eustachio. The reactome pathway knowledgebase.* Nucleic Acids Research*, 44(D1):D481–D487, 2016.*

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

[60] *Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In* Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1434–1453. Society for Industrial and Applied Mathematics, 2013.*

[61] *Thilo Fester, Falk Schreiber, Marc Strickert, and IPK Gatersleben. Cuda-based multi-core implementation of mds-based bioinformatics algorithms. In* GCB*, pages 67–79. Citeseer, 2009.*

[62] *Jodie M Fleming, Erika Ginsburg, Shannon D Oliver, Paul Goldsmith, and Barbara K Vonderhaar. Hornerin, an s100 family protein, is functional in breast cells and aberrantly expressed in breast cancer.* BMC cancer*, 12(1):266, 2012.*

[63] *Jianjiong Gao, Bülent Arman Aksoy, Ugur Dogrusoz, Gideon Dresdner, Benjamin Gross, S Onur Sumer, Yichao Sun, Anders Jacobsen, Rileen Sinha, Erik Larsson, et al. Integrative analysis of complex cancer genomics and clinical profiles using the cbioportal.* Sci. Signal.*, 6(269):pl1–pl1, 2013.*

[64] *Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In* ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2016.*

[65] *Alexander Gepperth and Cem Karaoguz. A bio-inspired incremental learning architecture for applied perceptual problems.* Cognitive Computation*, 8(5):924–934, 2016.*

[66] *BA Goff, K Shy, BE Greer, HG Muntz, M Skelly, and AM Gown. Overexpression and relationships of her-2/neu, epidermal growth factor receptor, p53, ki-67, and*

tumor necrosis factor alpha in epithelial ovarian cancer. European journal of gynaecological oncology, *17(6):487, 1996.*

[67] Raúl A González-Pech, Timothy G Stephens, and Cheong Xin Chan. Commonly misunderstood parameters of ncbi blast and important considerations for users. Bioinformatics, *35(15):2697–2698, 2019.*

[68] Sara Goodwin, John D McPherson, and W Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. Nature Reviews Genetics, *17(6):333, 2016.*

[69] Manfred G Grabherr, Brian J Haas, Moran Yassour, Joshua Z Levin, Dawn A Thompson, Ido Amit, Xian Adiconis, Lin Fan, Raktima Raychowdhury, Qiandong Zeng, et al. Full-length transcriptome assembly from rna-seq data without a reference genome. Nature biotechnology, *29(7):644, 2011.*

[70] Robert C Grant, Iris Selander, Ashton A Connor, Shamini Selvarajah, Ayelet Borgida, Laurent Briollais, Gloria M Petersen, Jordan Lerner-Ellis, Spring Holter, and Steven Gallinger. Prevalence of germline mutations in cancer predisposition genes in patients with pancreatic cancer. Gastroenterology, *148(3): 556–564, 2015.*

[71] Tanya Guha and David Malkin. Inherited tp53 mutations and the li–fraumeni syndrome. Cold Spring Harbor Perspectives in Medicine, *7(4):a026187, 2017.*

[72] Huadong Guo, Lizhe Wang, Fang Chen, and Dong Liang. Scientific big data and digital earth. Chinese science bulletin, *59(35):5066–5073, 2014.*

[73] Xin Guo. g3viz: Interactively Visualize Genetic Mutation Data using a Lollipop-Diagram. `https://github.com/G3viz/g3viz`, *2019. Accessed 2019-12-27.*

[74] Daniel H Haft, Michael DiCuccio, Azat Badretdin, Vyacheslav Brover, Vyacheslav Chetvernin, Kathleen O'Neill, Wenjun Li, Farideh Chitsaz, Myra K Derbyshire,

Noreen R Gonzales, et al. Refseq: an update on prokaryotic genome annotation and curation. Nucleic acids research*, 46(D1):D851–D860, 2017.*

[75] Antonina Harłiozińska and Julia K Bar. Relationship between p53 and c-erbB-2 overexpression in tissue sections and cyst fluid cells of patients with ovarian cancer. Tumor Biology*, 15(4):223–229, 1994.*

[76] Mark Harris. Optimizing parallel reduction in CUDA. `https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf`*, 2019. Accessed 2019-12-27.*

[77] Haibo He, Sheng Chen, Kang Li, and Xin Xu. Incremental learning from stream data. IEEE Transactions on Neural Networks*, 22(12):1901–1914, 2011.*

[78] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.*

[79] Vannini I, Zoli W, Tesei A, Rosetti M, Sansone P, Storci G, Passardi A, Massa I, Ricci M, Gusolfino D, Fabbri F, Ulivi P, Brigliadori G, Amadori D, and Bonafe M. Role of p53 codon 72 arginine allele in cell survival in vitro and in the clinical outcome of patients with advanced breast cancer. Tumour Biol*, 29(3):145–51, 2008.*

[80] i5K Consortium. The i5k initiative: advancing arthropod genomics for knowledge, human health, agriculture, and the environment. Journal of Heredity*, 104(5): 595–600, 2013.*

[81] Stephen Ingram, Tamara Munzner, and Marc Olano. Glimmer: Multilevel mds on the gpu. IEEE Transactions on Visualization and Computer Graphics*, 15(2): 249–261, 2009.*

[82] *Intel. Product specifications: Intel Xeon Processor E5-2630 v4.* `https://ark.intel.com/content/www/us/en/ark/products/92981/intel-xeon-processor-e5-2630-v4-25m-cache-2-20-ghz.html`*, 2017. Accessed 2019-12-30.*

[83] *Alan Julian Izenman. Linear discriminant analysis. In* Modern multivariate statistical techniques*, pages 237–280. Springer, 2013.*

[84] *Tarush Jain and Tanmay Agrawal. The haswell microarchitecture-4th generation processor.* International Journal of Computer Science and Information Technologies*, 4(3):477–480, 2013.*

[85] *Joon Young Jang, Yulhyun Park, Dae-Hyun Jang, Ja-Hyun Jang, and Ju Seok Ryu. Two novel mutations in TTN of a patient with congenital myopathy: A case report.* Molecular Genetics & Genomic Medicine*, 2019.*

[86] *Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele P Scarpazza. Dissecting the nvidia volta gpu architecture via microbenchmarking.* arXiv preprint arXiv:1804.06826*, 2018.*

[87] *Mark Johnson, Irena Zaretskaya, Yan Raytselis, Yuri Merezhuk, Scott McGinnis, and Thomas L Madden. Ncbi blast: a better web interface.* Nucleic acids research*, 36(suppl_2):W5–W9, 2008.*

[88] *William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space.* Contemporary mathematics*, 26(189-206):1, 1984.*

[89] *Prem Junsawang, Suphakant Phimoltares, and Chidchanok Lursinsap. Streaming chunk incremental learning for class-wise data stream classification with fast learning speed and low structural complexity.* PloS one*, 14(9):e0220624, 2019.*

[90] *Dalma Kellermayer, John E Smith, and Henk Granzier. Titin mutations and muscle disease.* Pflügers Archiv-European Journal of Physiology*, 471(5):673–682, 2019.*

[91] *Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In* Thirty-second AAAI conference on artificial intelligence*, 2018.*

[92] *Brian Kernighan and Dennis M Ritchie.* The C programming language*. Prentice hall, 2017.*

[93] *Amjad Khan, Rongrong Wang, Shirui Han, Muhammad Umair, Safdar Abbas, Muhammad Ismail Khan, Mohammad A Alshabeeb, Majid Alfadhel, and Xue Zhang. Homozygous missense variant in the TTN gene causing autosomal recessive limb-girdle muscular dystrophy type 10.* BMC Medical Genetics*, 20(1):166, 2019.*

[94] *Kenneth W Kinzler and Bert Vogelstein. Lessons from hereditary colorectal cancer.* Cell*, 87(2):159–170, 1996.*

[95] *James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks.* Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.*

[96] *Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, and Ahmed Fasih. Pycuda and pyopencl: A scripting-based approach to gpu run-time code generation.* Parallel Computing*, 38(3):157–174, 2012.*

[97] *Klaus-Peter Koepfli, Benedict Paten, Genome 10K Community of Scientists, and Stephen J O'Brien. The genome 10k project: a way forward.* Annu. Rev. Anim. Biosci.*, 3(1):57–111, 2015.*

[98] *Eugene F Krause. Taxicab geometry.* The Mathematics Teacher*, 66(8):695–706, 1973.*

[99] *Karoline B Kuchenbaecker, John L Hopper, Daniel R Barnes, Kelly-Anne Phillips, Thea M Mooij, Marie-José Roos-Blom, Sarah Jervis, Flora E Van Leeuwen, Roger L Milne, Nadine Andrieu, et al. Risks of breast, ovarian, and contralateral breast cancer for BRCA1 and BRCA2 mutation carriers.* JAMA*, 317(23):2402–2416, 2017.*

[100] *Runjun D Kumar, S Joshua Swamidass, and Ron Bose. Unsupervised detection of cancer driver mutations with parsimony-guided learning.* Nature genetics*, 48 (10):1288–1294, 2016.*

[101] *Jacques Lagnel, Costas S Tsigenopoulos, and Ioannis Iliopoulos. Noblast and jamblast: New options for blast and a java application manager for blast results.* Bioinformatics*, 25(6):824–826, 2009.*

[102] *Doug Laney. 3d data management: Controlling data volume, velocity and variety.* META group research note*, 6(70):1, 2001.*

[103] *Xénia Latypova, Naomichi Matsumoto, Cécile Vinceslas-Muller, Stéphane Bézieau, Bertrand Isidor, and Noriko Miyake. Novel kcnb1 mutation associated with non-syndromic intellectual disability.* Journal of Human Genetics*, 62 (5):569, 2017.*

[104] *Michael S Lawrence, Petar Stojanov, Paz Polak, Gregory V Kryukov, Kristian Cibulskis, Andrey Sivachenko, Scott L Carter, Chip Stewart, Craig H Mermel, Steven A Roberts, et al. Mutational heterogeneity in cancer and the search for new cancer-associated genes.* Nature*, 499(7457):214, 2013.*

[105] *Heidi Ledford. Cocktails for cancer with a measure of immunotherapy.* Nature*, 532(7598):162–164, 2016.*

[106] *Mark DM Leiserson, Matthew A Reyna, and Benjamin J Raphael. A weighted exact test for mutually exclusive mutations in cancer.* Bioinformatics, *32(17): i736–i745, 2016.*

[107] *Scotland C Leman, Leanna House, Dipayan Maiti, Alex Endert, and Chris North. Visual to parametric interaction (v2pi).* PloS one, *8(3):e50474, 2013.*

[108] *Harris A Lewin, Gene E Robinson, W John Kress, William J Baker, Jonathan Coddington, Keith A Crandall, Richard Durbin, Scott V Edwards, Félix Forest, M Thomas P Gilbert, et al. Earth biogenome project: Sequencing life for the future of life.* Proceedings of the National Academy of Sciences, *115(17):4325–4333, 2018.*

[109] *MP Little and EG Wright. A stochastic carcinogenesis model incorporating genomic instability fitted to colon cancer data.* Mathematical biosciences, *183(2): 111–134, 2003.*

[110] *DP Liu, Hoseok Song, and Yang Xu. A common gain of function of p53 cancer mutants in inducing genetic instability.* Oncogene, *29(7):949, 2010.*

[111] *Xiang Liu and Zhi-Qiang Ling. Role of isocitrate dehydrogenase 1/2 (IDH 1/2) gene mutations in human tumors.* Histology and Histopathology, *30(10):1155–1160, 2015.*

[112] *Xiuli Liu, Maureen Jakubowski, and Jennifer L Hunt. Kras gene mutation in colorectal cancer is correlated with increased proliferation and spontaneous apoptosis.* American journal of clinical pathology, *135(2):245–252, 2011.*

[113] *Ru-Sen Lu, Avery E Broderick, Fabien Baron, John D Monnier, Vincent L Fish, Sheperd S Doeleman, and Victor Pankratius. Imaging the supermassive black hole shadow and jet base of m87 with the event horizon telescope.* The Astrophysical Journal, *788(2):120, 2014.*

[114] *E Georg Luebeck and Suresh H Moolgavkar. Multistage carcinogenesis and the incidence of colorectal cancer.* Proc Natl Acad Sci USA, *99(23):15095–15100, 2002.*

[115] *Ruibang Luo, Binghang Liu, Yinlong Xie, Zhenyu Li, Weihua Huang, Jianying Yuan, Guangzhu He, Yanxiang Chen, Qi Pan, Yunjie Liu, et al. Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler.* Gigascience, *1(1):18, 2012.*

[116] *Jun Ma, Lei Zhang, Jianguo Zhang, Mengmeng Liu, Liuping Wei, Tingting Shen, Cui Ma, Yanyan Wang, Yingli Chen, and Daling Zhu. 15-lipoxygenase-1/15-hydroxyeicosatetraenoic acid promotes hepatocellular cancer cells growth through protein kinase b and heat shock protein 90 complex activation.* The international journal of biochemistry & cell biology, *45(6):1031–1041, 2013.*

[117] *Phuong L Mai, David Malkin, Judy E Garber, Joshua D Schiffman, Jeffrey N Weitzel, Louise C Strong, Oliver Wyss, Luana Locke, Von Means, Maria Isabel Achatz, et al. Li-fraumeni syndrome: report of a clinical research workshop and creation of a research consortium.* Cancer genetics, *205(10):479–487, 2012.*

[118] *Carla Marini, Michele Romoli, Elena Parrini, Cinzia Costa, Davide Mei, Francesco Mari, Lucio Parmeggiani, Elena Procopio, Tiziana Metitieri, Elena Cellini, et al. Clinical features and outcome of 6 new patients carrying de novo KCNB1 gene mutations.* Neurology Genetics, *3(6):e206, 2017.*

[119] *Iñigo Martincorena, Amit Roshan, Moritz Gerstung, Peter Ellis, Peter Van Loo, Stuart McLaren, David C Wedge, Anthony Fullam, Ludmil B Alexandrov, Jose M Tubio, et al. High burden and pervasive positive selection of somatic mutations in normal human skin.* Science, *348(6237):880–886, 2015.*

[120] *Iñigo Martincorena, Joanna C Fowler, Agnieszka Wabik, Andrew RJ Lawson,*

Federico Abascal, Michael WJ Hall, Alex Cagan, Kasumi Murai, Krishnaa Mahbubani, Michael R Stratton, et al. Somatic mutant clones colonize the human esophagus with age. Science*, 362(6417):911–917, 2018.*

[121] Takatoshi Matsuyama, Toshiaki Ishikawa, Kaoru Mogushi, Tsuyoshi Yoshida, Satoru Iida, Hiroyuki Uetake, Hiroshi Mizushima, Hiroshi Tanaka, and Kenichi Sugihara. Muc12 mrna expression is an independent marker of prognosis in stage ii and stage iii colorectal cancer. International journal of cancer*, 127(10): 2292–2299, 2010.*

[122] Simon Kebede Merid, Daria Goranskaya, and Andrey Alexeyenko. Distinguishing between driver and passenger mutations in individual cancer genomes by network enrichment analysis. BMC Bioinformatics*, 14:308, 2014.*

[123] Pu Miao, Jianhua Feng, Yufan Guo, Jianda Wang, Xiaoxiao Xu, Ye Wang, Yanfang Li, Liuyan Gao, Chaoguang Zheng, and Haiying Cheng. Genotype and phenotype analysis using an epilepsy-associated gene panel in Chinese pediatric epilepsy patients. Clinical Genetics*, 94(6):512–520, 2018.*

[124] Bruce Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. IEEE transactions on automatic control*, 26 (1):17–32, 1981.*

[125] Shailendra Mundhada, Rajyalakshmi Luthra, and Pedro Cano. Association of hla class i and class ii genes with bcr-abl transcripts in leukemia patients with t (9; 22)(q34; q11). BMC cancer*, 4(1):25, 2004.*

[126] Aaftab Munshi. The opencl specification. In 2009 IEEE Hot Chips 21 Symposium (HCS)*, pages 1–314. IEEE, 2009.*

[127] Rachael Natrajan, Suzanne E Little, Jorge S Reis-Filho, Lara Hing, Boo Messahel, Paul E Grundy, Jeffrey S Dome, Toni Schneider, Gordan M Vujanic, Kathy

Pritchard-Jones, et al. Amplification and overexpression of cacna1e correlates with relapse in favorable histology wilms' tumors. Clinical cancer research*, 12 (24):7284–7293, 2006.*

[128] Cristobal A Navarro and Nancy Hitschfeld. Gpu maps for the space of computation in triangular domain problems. In 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*, pages 375–382. IEEE, 2014.*

[129] Cristóbal A Navarro, Benjamín Bustos, and Nancy Hitschfeld. Potential benefits of a block-space gpu approach for discrete tetrahedral domains. In 2016 XLII Latin American Computing Conference (CLEI)*, pages 1–5. IEEE, 2016.*

[130] CO Nordling. A new theory on the cancer-inducing mechanism. Br J Cancer*, 7 (1):68, 1953.*

[131] NVIDIA. Cuda C++ Best Practices Guide. `https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#instruction-optimization`*, 2019. Accessed 2019-12-30.*

[132] Nuala A O'Leary, Mathew W Wright, J Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, et al. Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation. Nucleic acids research*, 44(D1):D733–D745, 2015.*

[133] Hasmik Osipyan, Martin Kruliš, and Stéphane Marchand-Maillet. A survey of cuda-based multidimensional scaling on gpu architecture. In OASIcs-OpenAccess Series in Informatics*, volume 49. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.*

[134] *Aisling O'Driscoll, Jurate Daugelaite, and Roy D Sleator. 'big data', hadoop and cloud computing in genomics.* Journal of biomedical informatics*, 46(5):774–781, 2013.*

[135] *Bin Pan, Shaobo Zheng, Chunxiao Liu, and Yawen Xu. Suppression of IGHG1 gene expression by siRNA leads to growth inhibition and apoptosis induction in human prostate cancer cell.* Mol Biol Rep*, 40(1):27–33, 2013.*

[136] *Pan Pantziarka. Primed for cancer: Li fraumeni syndrome and the pre-cancerous niche.* ecancermedicalscience*, 9, 2015.*

[137] *Sungin Park, Soo-Yong Shin, and Kyu-Baek Hwang. Cfmds: Cuda-based fast multidimensional scaling for genome-scale data.* BMC bioinformatics*, 13(17):1, 2012.*

[138] *Yonil Park, Sergey Sheetlin, Ning Ma, Thomas L Madden, and John L Spouge. New finite-size correction for local alignment score distributions.* BMC research notes*, 5(1):286, 2012.*

[139] *Alejandro Parrales and Tomoo Iwakuma. Targeting oncogenic mutant p53 for cancer therapy.* Frontiers in oncology*, 5:288, 2015.*

[140] *Nikki R Paul, Jennifer L Allen, Anna Chapman, Maria Morlan-Mairal, Egor Zindy, Guillaume Jacquemet, Laura Fernandez del Ama, Nermina Ferizovic, David M Green, Jonathan D Howe, et al. $\alpha 5\beta 1$ integrin recycling promotes arp2/3-independent cancer cell invasion via the formin fhod3.* Journal of Cell Biology*, 210(6):1013–1031, 2015.*

[141] *Karl Pearson. Mathematical contributions to the theory of evolution. iii. regression, heredity, and panmixia.* Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 187:253–318, 1896.*

[142] Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. Stringtie enables improved reconstruction of a transcriptome from rna-seq reads. Nature biotechnology*, 33(3): 290, 2015.*

[143] Ralph S Peters, Lars Krogmann, Christoph Mayer, Alexander Donath, Simon Gunkel, Karen Meusemann, Alexey Kozlov, Lars Podsiadlowski, Malte Petersen, Robert Lanfear, et al. Evolutionary history of the hymenoptera. Current Biology*, 27(7):1013–1018, 2017.*

[144] Erin Pleasance, Keira Cheetham, Philip Stephens, David McBride, Sean Humphray, Chris Greenman, Ignacio Varela, Meng-Lay Lin, Gonzalo Ordóñez, Graham Bignell, Kai Ye, Julie Alipaz, Markus Bauer, David Beare, Adam Butler, Richard Carter, Lina Chen, Anthony Cox, Sarah Edkins, Paula Kokko-Gonzales, Niall Gormley, Russell Grocock, Christian Haudenschild, Matthew Hims, Terena James, Mingming Jia, Zoya Kingsbury, Catherine Leroy, John Marshall, Andrew Menzies, Laura Mudie, Zemin Ning, Tom Royce, Ole Schulz-Trieglaff, Anastassia Spiridou, Lucy Stebbings, Lukasz Szajkowski, Jon Teague, David Williamson, Lynda Chin, Mark Ross, Peter Campbell, David Bentley, Andrew Futreal, and Michael Stratton. A comprehensive catalogue of somatic mutations from a human cancer genome. Nature*, 463(14):191–196, 2010.*

[145] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.*

[146] Julia R Pon and Marco A Marra. Driver and passenger mutations in cancer. Annual Review of Pathology: Mechanisms of Disease*, 10:25–50, 2015.*

[147] Yun Qin, Xicai Tang, and Mingxing Liu. Tumor-suppressor gene nbpf1 inhibits invasion and pi3k/mtor signaling in cervical cancer cells. Oncology Research Featuring Preclinical and Clinical Cancer Therapeutics*, 23(1-2):13–20, 2016.*

[148] Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. Systematic biology*, 45(3):380–385, 1996.*

[149] Lauren L. Ritterhouse, Lori J. Wirth, Gregory W. Randolph, Peter M. Sadow, Douglas S. Ross, Whitney Liddy, and Jochen K. Lennerz. Ros1 rearrangement in thyroid cancer. Thyroid*, 26(6):1, 2016.*

[150] Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. Online deep learning: Learning deep neural networks on the fly. arXiv preprint arXiv:1711.03705*, 2017.*

[151] Hirotomo Saitsu, Tenpei Akita, Jun Tohyama, Hadassa Goldberg-Stern, Yu Kobayashi, Roni Cohen, Mitsuhiro Kato, Chihiro Ohba, Satoko Miyatake, Yoshinori Tsurusaki, et al. De novo KCNB1 mutations in infantile epilepsy inhibit repetitive neuronal firing. Scientific Reports*, 5:15199, 2015.*

[152] Sarah Sandmann, Aniek O De Graaf, Mohsen Karimi, Bert A Van Der Reijden, Eva Hellström-Lindberg, Joop H Jansen, and Martin Dugas. Evaluating variant calling tools for non-matched next-generation sequencing data. Scientific reports*, 7:43169, 2017.*

[153] Hamid Sarbazi-Azad. Advances in GPU Research and Practice: A volume in Emerging Trends in Computer Science and Applied Computing*, chapter 23, pages 649–705. Morgan Kaufmann, 2017.*

[154] Hamid Sarbazi-Azad. Advances in GPU Research and Practice: A volume in Emerging Trends in Computer Science and Applied Computing*, chapter 9, pages 543–580. Morgan Kaufmann, 2017.*

[155] *Joellen M Schildkraut, Ellen L Goode, Merlise A Clyde, Edwin S Iversen, Patricia G Moorman, Andrew Berchuck, Jeffrey R Marks, Jolanta Lissowska, Louise Brinton, Beata Peplonska, et al. Single nucleotide polymorphisms in the TP53 region and susceptibility to invasive epithelial ovarian cancer.* Cancer Research*, 69(6):2349–2357, 2009.*

[156] *Bertil Schmidt, Jorge Gonzalez-Dominguez, Christian Hundt, and Moritz Schlarb.* Parallel programming: concepts and practice*. Morgan Kaufmann, 2017.*

[157] *Günter Schneider, Marc Schmidt-Supprian, Roland Rad, and Dieter Saur. Tissue-specific tumorigenesis: context matters.* Nature Reviews Cancer*, 17(4):239, 2017.*

[158] *Jessica Zeitz Self, Leanna House, and Chris North. Andromeda: Observation-level and parametric interaction for exploratory data analysis, 2015.*

[159] *Nidhi Shah, Michael G Nute, Tandy Warnow, and Mihai Pop. Misunderstood parameter of ncbi blast impacts the correctness of bioinformatics workflows.* Bioinformatics*, page bty833, 2018. doi: 10.1093/bioinformatics/bty833.*

[160] *Jared T Simpson, Kim Wong, Shaun D Jackman, Jacqueline E Schein, Steven JM Jones, and Inanç Birol. Abyss: a parallel assembler for short read sequence data.* Genome research*, 19(6):1117–1123, 2009.*

[161] *Inderpreet Singh, Arrvindh Shriraman, Wilson WL Fung, Mike O'Connor, and Tor M Aamodt. Cache coherence for gpu architectures. In* 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 578–590. IEEE, 2013.*

[162] *Amit Singhal et al. Modern information retrieval: A brief overview.* IEEE Data Eng. Bull.*, 24(4):35–43, 2001.*

[163] *Ramin Skibba. Geneticists hope to unlock secrets of bats' complex sounds.* Nature News*, 539(7630):481, 2016.*

[164] *Hoseok Song, Monica Hollstein, and Yang Xu. p53 gain-of-function cancer mutants induce genetic instability by inactivating atm.* Nature Cell Biology*, 9(5): 573, 2007.*

[165] *YH Song and CJ Zhang. Effect of hydralazine on demethylation status and expression of apc gene, proliferation and apoptosis of human cervical cancer cell lines.* Zhonghua bing li xue za zhi= Chinese journal of pathology*, 36(9):614, 2007.*

[166] *Tyagi M. Vallania F. Bredemeyer A.J. Pfeifer J.D. Mitra R.D. Duncavage E.J. Spencer, D.H. Performance of common analysis methods for detecting low-frequency single nucleotide variants in targeted next-generation sequence data.* J Mol Diag*, 16(1):75–88, 2014.*

[167] *Maximilian Stahl, Nathan Kohrman, Steven D Gore, Tae Kon Kim, Amer M Zeidan, and Thomas Prebet. Epigenetics in cancer: a hematological perspective.* PLoS Genetics*, 12(10):e1006193, 2016.*

[168] *David Tamborero, Abel Gonzalez-Perez, and Nuria Lopez-Bigas. Oncodriveclust: exploiting the positional clustering of somatic mutations to identify cancer genes.* Bioinformatics*, 29(18):2238–2244, 2013.*

[169] *Ernest Y Tan, Cynthia L Richard, Hong Zhang, David W Hoskin, and Jonathan Blay. Adenosine downregulates dppiv on ht-29 colon cancer cells by stimulating protein tyrosine phosphatase (s) and reducing erk1/2 activity via a novel pathway.* American Journal of Physiology-Cell Physiology*, 291(3):C433–C444, 2006.*

[170] *Tatiana Tatusova, Michael DiCuccio, Azat Badretdin, Vyacheslav Chetvernin, Eric P Nawrocki, Leonid Zaslavsky, Alexandre Lomsadze, Kim D Pruitt, Mark Borodovsky, and James Ostell. Ncbi prokaryotic genome annotation pipeline.* Nucleic acids research*, 44(14):6614–6624, 2016.*

[171] Isabelle Thiffault, David J Speca, Daniel C Austin, Melanie M Cobb, Kenneth S Eum, Nicole P Safina, Lauren Grote, Emily G Farrow, Neil Miller, Sarah Soden, et al. A novel epileptic encephalopathy mutation in KCNB1 disrupts Kv2.1 ion selectivity, expression, and localization. Journal of General Physiology*, 146(5): 399–410, 2015.*

[172] Rui Tian, Malay K Basu, and Emidio Capriotti. Contrastrank: a new method for ranking putative cancer driver genes and classification of tumor samples. Bioinformatics*, 30(17):i572–i578, 2014.*

[173] Cristian Tomasetti, Luigi Marchionni, Martin A Nowak, Giovanni Parmigiani, and Bert Vogelstein. Only three driver gene mutations are required for the development of lung and colorectal cancers. Proc Natl Acad Sci USA*, 112(1):118–123, 2015.*

[174] Warren S Torgerson. Multidimensional scaling: I. theory and method. Psychometrika*, 17(4):401–419, 1952.*

[175] L-J Tsai, S-H Hsiao, L-M Tsai, C-Y Lin, J-J Tsai, D-M Liou, and J-L Lan. The sodium-dependent glucose cotransporter slc5a11 as an autoimmune modifier gene in sle. Tissue antigens*, 71(2):114–126, 2008.*

[176] Hege Vårdal. Venom gland and reservoir morphology in cynipoid wasps. Arthropod structure & development*, 35(2):127–136, 2006.*

[177] Sonja Verheyden, Soldano Ferrone, Arend Mulder, Frans H Claas, Rik Schots, Barbara De Moerloose, Yves Benoit, and Christian Demanet. Role of the inhibitory kir ligand hla-bw4 and hla-c expression levels in the recognition of leukemic cells by natural killer cells. Cancer immunology, immunotherapy*, 58 (6):855, 2009.*

[178] A Vincent, M Perrais, JL Desseyn, JP Aubert, P Pigny, and I Van Seuningen. Epigenetic regulation (dna methylation, histone modifications) of the 11p15 mucin genes (muc2, muc5ac, muc5b, muc6) in epithelial cancer cells. Oncogene*, 26(45): 6566–6576, 2007.*

[179] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. arXiv e-prints*, art. arXiv:1907.10121, Jul 2019.*

[180] Bert Vogelstein, Nickolas Papadopoulos, Victor E. Velculescu, Shibin Zhou, Jr. Diaz, Luis A., and Kenneth W. Kinzler. Cancer genome landscapes. Science*, 339 (6127):1546–58, 2013.*

[181] Hao-Yuan Wang, Wen Wang, Yan-Wei Liu, Ming-Yang Li, Ting-Yu Liang, Ji-Ye Li, Hui-Min Hu, Yang Lu, Chen Yao, Yong-Yi Ye, et al. Role of KCNB1 in the prognosis of gliomas and autophagy modulation. Scientific Reports*, 7(1):14, 2017.*

[182] Tao Wang, Yan-Hua Chen, Heng Hong, Yan Zeng, Jiao Zhang, Jian-Ping Lu, Beverly Jeansonne, and Qun Lu. Increased nucleotide polymorphic changes in the 5 -untranslated region of δ-catenin (ctnnd2) gene in prostate cancer. Oncogene*, 28(4):555–564, 2009.*

[183] *Donate Weghorn and Shamil Sunyaev. Bayesian inference of negative and positive selection in human cancers.* Nature genetics*, 49(12):1785–1788, 2017.*

[184] *John Weinstein, Eric Collisson, Gordon Mills, Kenna Shaw, Brad Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, and Joshua Stuart. The cancer genome atlas pan-cancer analysis project.* Nat Genet*, 48(10):1288–1294, 2016.*

[185] *John N Weinstein, Eric A Collisson, Gordon B Mills, Kenna R Mills Shaw, Brad A Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, Joshua M Stuart, Cancer Genome Atlas Research Network, et al. The cancer genome atlas pan-cancer analysis project.* Nature genetics*, 45(10):1113, 2013.*

[186] *Jianing Xi, Minghui Wang, and Ao Li. Discovering mutated driver genes through a robust and sparse co-regularized matrix factorization framework with prior information from mrna expression patterns and interaction network.* BMC bioinformatics*, 19(1):214, 2018.*

[187] *Shucai Xiao and Wu-chun Feng. Inter-block gpu communication via fast barrier synchronization. In* Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12. IEEE, 2010.*

[188] *Yawen Xu, Binshen Chen, Shaobo Zheng, Yong Wen, Abai Xu, Kai Xu, Bingkun Li, and Chunxiao Liu. IgG silencing induces apoptosis and suppresses proliferation, migration and invasion in LNCaP prostate cancer cells.* Cell Mol Biol Lett*, 21:27, 2016.*

[189] *Eun-Kyoung Yim, Guang Peng, Hui Dai, Ruozhen Hu, Kaiyi Li, Yiling Lu, Gordon B Mills, Funda Meric-Bernstam, Bryan T Hennessy, Rolf J Craven, et al. Rak functions as a tumor suppressor by regulating pten protein stability and function.* Cancer cell*, 15(4):304–314, 2009.*

[190] Junqi Yin, Shubhankar Gahlot, Jack Morrison, Ketan Maheshwari, Sajal Dash, and Mallikarjun Shankar. Deployment and evaluation of extreme-scale machine learning and deep learning on the summit supercomputer. SC 19*, 2019.*

[191] Meng Yu, Ying Zhu, Zhiying Xie, Yiming Zheng, Jiangxi Xiao, Wei Zhang, Ichizo Nishino, Yun Yuan, and Zhaoxia Wang. Novel TTN mutations and muscle imaging characteristics in congenital titinopathy. Annals of Clinical and Translational Neurology*, 2019.*

[192] Hongen Zhang, Paul Meltzer, and Sean Davis. Rcircos: an R package for Circos 2D track plots. BMC Bioinformatics*, 14(1):244, 2013.*

[193] Jing Zhang, Hao Wang, Heshan Lin, and Wu-chun Feng. cublastp: Fine-grained parallelization of protein sequence search on a gpu. In Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 251–260. IEEE, 2014.*

[194] Jing Zhang, Sanchit Misra, Hao Wang, and Wu-chun Feng. mublastp: database-indexed protein sequence search on multicore cpus. BMC bioinformatics*, 17(1): 443, 2016.*

[195] Xinan Zhang and Richard Simon. Estimating the number of rate limiting genomic changes for human breast cancer. Breast Cancer Res Treat*, 91(2):121–124, 2005.*