12-2023

Algorithmic And Computational Approaches For Improving The Efficiency Of Mobile Genomic Element Discovery, A Bioinformatics Framework

Fatema Shormin

ALGORITHMIC AND COMPUTATIONAL APPROACHES FOR IMPROVING THE EFFICIENCY OF MOBILE GENOMIC ELEMENT DISCOVERY, A BIOINFORMATICS FRAMEWORK

A Thesis

by

FATEMA SHORMIN

Submitted to the Office of Graduate Studies of
Prairie View A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2023

Major Subject: Computer Information System

ALGORITHMIC AND COMPUTATIONAL APPROACHES FOR IMPROVING THE EFFICIENCY OF MOBILE GENOMIC ELEMENT DISCOVERY, A BIOINFORMATICS FRAMEWORK

A Thesis

by

FATEMA SHORMIN

Submitted to the Office of Graduate Studies of
Prairie View A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

_____ Dr. Noushin Ghaffari Chair of Committee	_____ Dr. Lin Li Committee Member
_____ Dr. Ahmed Ahmed Committee Member	_____ Dr. Sherri S. Frizell Committee Member
_____ Dr. Yonggao Yang Head of Department Engineering	_____ Dr. Pamela H. Obiomon Dean, Roy G. Perry College of
_____ Dr. Tyrone Tanner Dean, Graduate Studies	

December 2023

Major Subject: Computer Information System

ABSTRACT

ALGORITHMIC AND COMPUTATIONAL APPROACHES FOR IMPROVING THE EFFICIENCY OF MOBILE GENOMIC ELEMENT DISCOVERY, A BIOINFORMATICS FRAMEWORK

(December 2023)

Fatema Shormin, B.S., Noakhali Science and Technology University
Chair of Advisory Committee: Dr. Noushin Ghaffari
Co-Chair of Advisory Committee: NA

Through this research, we are showcasing the application of computational approaches to the discoveries in the life sciences spectrum. Our current research not only focused on mobile genetic elements but also developed the computational methods that enabled these findings. We combined the biology sciences and computer science in our research, which is essentially multidisciplinary. To that end, this research intricately probed the role and implications of mobile genetic elements, emphasizing transposable elements. These dynamic components wielded substantial influence over genomic architecture's structure, function, and evolutionary adaptations. An integral component of our study is the innovative computational tool, Target/IGE Retriever (TIGER), employed to detect and map these mobile genetic elements. Given the pronounced impact of these elements on gene regulation and their involvement in various genetic diseases, their precise detection and mapping within a genome were crucial for understanding intricate genetic dynamics and disease etiology.

Addressing computational challenges, the study introduces three new algorithms to enhance TIGER's performance, tested using E. coli genomes. This testing aimed to determine the impact of database size reduction on result accuracy and performance. Findings indicate that while prophage yields are less affected by database size, non-phage islands show sensitivity, suggesting performance improvements with smaller databases.

Furthermore, the research conducts a comparative analysis of TIGER and BLAST outputs, focusing on validating transposons identified in E. coli genomes. This involves cross-referencing with established databases and employing statistical methods for match categorization, enhancing the authenticity of transposon location identification..

Within the purview of this rigorous analytical process, particular attention is accorded to evaluating sequence alignment results and the quality of BLAST hits, focusing specifically on identifying direct repeats within insertion sequences. The study underscores TIGER's efficacy in transposon discovery and yields critical insights into its performance relative to BLAST.

This research illuminates potential avenues for enhancing computational tools in bioinformatics, all within the larger framework of contributing significantly to genomics and bioinformatics research's ongoing advancements. Our work deepens our understanding of the role and influence of mobile genetic elements on genomic architecture.

Index Term: **Computational biology, bioinformatics, mobile genetic elements, transposon, validation, database.**

ACKNOWLEDGEMENTS

Firstly, I express my heartfelt gratitude to Dr. Noushin Ghaffari for her invaluable teaching, support, and insights throughout this project. Working with her has been a privilege that has greatly influenced my understanding and skills in bioinformatics, and her guidance was crucial in every phase of my thesis.

I also thank my committee members, Dr. Sherri S. Frizell, Dr. Lin Li, and Dr. Ahmed Ahmed, for their time and significant contributions to refining my thesis with their advice and critiques. Additionally, I am grateful to my professors in the Department of Computer Science at PVAMU. Their dedication, concern for students, and commitment to education have profoundly impacted me and will continue to influence my professional journey.

Moreover, I am thankful to Dr. Kelly Porter Williams from Sandia National Laboratories for sponsoring this research and providing valuable time and constructive feedback on my thesis.

Finally, I want to acknowledge my family's unwavering love and support. This thesis is dedicated to my parents and my husband, Dr. Anwarul Islam Sifat, as a token of my appreciation and love.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

LIST OF FLOWCHARTS

FLOWCHART Page

CHAPTER 1

INTRODUCTION

1.1 Introduction to Bioinformatics

Bioinformatics is far more than a mere buzzword in the scientific community. It represents a synergistic intersection of multiple scientific disciplines, including but not limited to computer science, mathematics, physics, and biology (Figure 1.1). In essence, bioinformatics acts as a robust computational framework that aids in the storage, retrieval, and nuanced interpretation of big biological data. Its interdisciplinary nature allows researchers to manage and analyze data that would otherwise be overwhelmingly complex or voluminous, making it indispensable in today's advanced biological and medical research paradigms [1].

1.2 Introduction to Genome

Consider the genome as the comprehensive blueprint for an organism's existence, much like an architectural plan for a skyscraper. It holds the totality of genetic instructions essential for constructing, sustaining, and replicating life. The genome is constituted by a sequence of nucleotide bases, namely adenine (A), thymine (T), cytosine (C), and guanine (G), that form pairs to construct the DNA double helix. These bases are the essential building blocks of DNA, and their specific sequences encode the diverse range of functions and structures within an organism. While the human genome, along with those of other cellular organisms, is constructed from deoxyribonucleic acid (DNA) incorporating both nuclear and

mitochondrial components it is worth noting that not all genomes are DNA-based. For instance, some viruses have genomes made of ribonucleic acid (RNA), a related but distinct molecular structure [2].

1.3 Introduction to Transposable Elements (TEs)

Transposable elements (TEs), a subset of mobile genetic elements, stand as dynamic sequences with the capacity to change their genomic positions. These mobile genetic elements, including TEs, have a major impact on the structure of the genome. The strange nature of TE mobility is that it may cause changes that are harmful to an organism, such as sequence deletions or interruptions of vital genes. Unless there are counterbalancing variables to lessen this mutational load, such negative impacts frequently result in their progressive removal from populations of haploid microbes like Escherichia coli. On the other hand, TEs and other mobile elements can spread across cells thanks to processes like horizontal gene transfer, but usually at rates too slow to offset their detrimental effects on the host. As a result, transposable elements, as well as the larger group of mobile genetic elements, play a variety of functions in the genomic landscape, from being disruptive to possibly adaptive [3].

1.4 Importance of the discovery of Transposable Elements

Transposable elements (TEs) are important players in the dynamics of genome evolution, its functional landscape, and even its vulnerability to different illnesses; therefore, solving its mysteries is crucial. Recent developments in the genomic sciences and extensive practical research have significantly expedited the

exploration of these TEs, which are prominent elements in the genome architecture of eukaryotes [4]. Understanding TEs is crucial for several reasons:

- *Genome Evolution:*

TEs have shaped the structure and organization of genomes throughout evolution. They can cause genetic rearrangements, insertions, and deletions, leading to genomic diversity and contributing to the evolution of species [4].

- *Genome Regulation:*

TEs can influence gene expression by acting as regulatory elements. They can be co-opted into developmental enhancers, altering the expression patterns of nearby genes [5]. TEs can also provide insulators to the genome, influencing the spatial organization of chromatin and gene regulation [6].

- *Stress Response:*

TEs have been implicated in the response to environmental stress. They can be activated under stress conditions and contribute to the activation of stress-responsive genes [7]. This suggests that TEs may play a role in adaptation to changing environments [8].

- *Disease and Pathology:*

TEs have been associated with various diseases and pathological conditions. They can disrupt gene function by inserting into coding regions or regulatory regions, leading to genetic disorders. TEs can also contribute to the spread of heterochromatin and epigenetic modifications associated with diseases such as cancer [9].

1.5 Multidisciplinary Nature of Our Work

When defining the multidisciplinary nature of the research project, it is important to recognize that we did not operate within the distinct areas of computer science or biology; rather, we combined these fields in a complementary relationship further enhanced by statistical inference and mathematical modeling. We use computer algorithms as instruments and extensions of inquiry that help us navigate the complex functioning of biological systems. Using this cooperative methodology, our work functions at the nexus of the biological and computational sciences, representing a comprehensive methodological framework that requires the joint knowledge of mathematicians, biologists, and bioinformaticians. This kind of multidisciplinary interaction opens up new directions for our collective scientific understanding of genomic architecture and function and for advancing the individual areas concerned.

1.6 Computational Intensity and HPC Use

Our research's computational complexity necessitates an infrastructure that can accurately and efficiently handle big, complicated data sets. In this situation, HPC, or high-performance computing, is crucial. Compared to traditional computer systems, high-performance computing, or HPC, employs numerous processors to complete complicated, data-intensive computations much faster [10].

Using HPC is an essential requirement for our study, not an optional feature. The computational needs of the techniques used to examine genomic sequences for mobile genetic elements, including transposons, make it challenging and time-consuming to do these computations on typical computer systems. Parallel

computing methods, in which many processors do distinct portions of a computational work concurrently, are made possible by HPC. This allows for more complex modeling and analysis, which may incorporate several techniques for statistical robustness, error-checking, and cross-validation, in addition to significantly reducing the calculation time. As a result, using HPC makes research possible more quickly and improves the overall quality and dependability of the results [11].

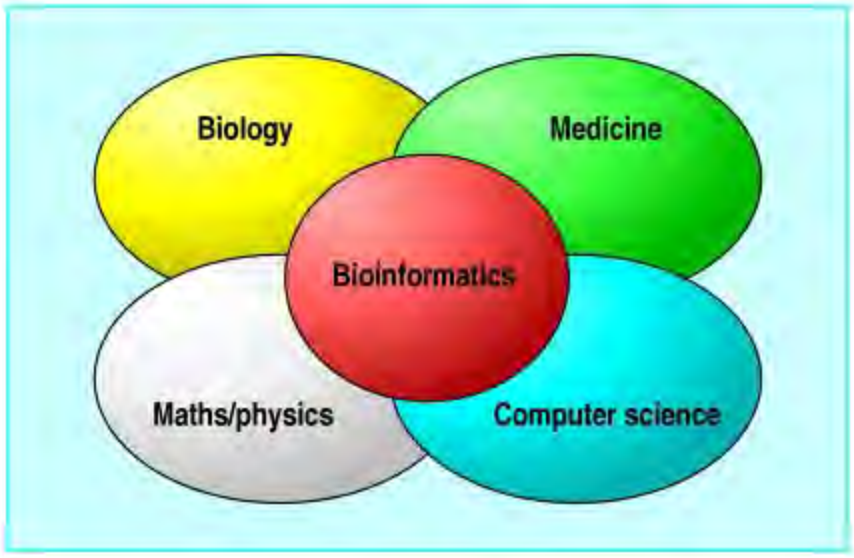


Fig 1.1: Interaction of disciplines that have contributed to the formation of bioinformatics [1].

In the realm of computational genomics, we can think of transposable elements as double agents. They can either stabilize the genetic landscape or throw it into chaos [12]. We have come a long way with bioinformatics and super-fast sequencing technology, but let us be honest—our computer methods for spotting these genetic chameleons are not up to snuff. This is a big deal when we are running massive studies that demand a lot of computer power. So, what is this research all

about? We are rolling up our sleeves to fine-tune the algorithms we already have and make sure they run like a dream on high-end computer systems.

1.7 Research Background and Significance in Computational Genomics

Transposons are universal as Mobile Genetic Elements in DNA that have the potential to transpose across the genome. This mobility exerts a dual influence: while transposons play a pivotal role in gene regulation and architectural modification of genomes and even act as catalysts in evolutionary processes, they also pose risks to genomic stability [13]. Detecting these elements represents a significant computational challenge. We have embarked on a collaborative research initiative with Sandia National Laboratory to ameliorate existing limitations in transposon detection methodologies. Leveraging their specialized software, TIGER, the project aimed to improve the accuracy and time efficiency of identifying these elements within genomes. This cooperative venture aspired to resolve the paradoxical nature of transposons by enhancing both the fidelity and timeliness of their detection, thereby contributing to our understanding of their roles in genetic variation and potential genomic instability [14].

1.8 Challenges

It is computationally difficult to identify transposons within genomes. It is like trying to find a needle in a haystack. There are multiple challenges involved with this task:

- *Too Many False Alarms:*

When we run our computer programs to search for transposons, we end up with many "false positives." These are instances where the program tells us we have

found a transposon, but upon closer inspection, it turns out to be a mistake. This is a big issue because we cannot fully trust the initial results. Someone has to go through them all, one by one, to figure out which are legitimate, and which are not. This is not only tedious but also drains both computational and human resources that could be better utilized elsewhere.

- *Missing the Real Deal:*

We also run into "false negatives." This is when genuine transposons are there, but the program misses them. Our understanding of the genomic landscape becomes incomplete or flawed when this happens. It is like trying to solve a puzzle but realizing too late that some pieces are missing. It hampers our ability to make accurate scientific interpretations or medical diagnoses.

- *Computer Headaches:*

Using powerful computers to do this work should make life easier, but it is not as simple as hitting the "go" button. We have to tweak the algorithms just right so that tasks are shared and balanced among the computer's resources.

- *Timing Issue:*

All these issues combine to create an incredibly time-consuming process. Furthermore, in science, time is often of the essence. Delays can have significant downstream effects, whether a research deadline, a medical diagnosis, or a bid for research funding. Moreover, the more time we spend troubleshooting these issues, the less time we have for other important tasks.

So, we have got our work cut out for us. We need more innovative ways to find these transposons without all these hang-ups.

1.9 Method

Algorithmic Development

In the initial stage of our research, we dissected existing algorithms rather than accepting them as they were. Our scrutiny was motivated by a dual objective: to accelerate the computational process and to minimize false discoveries. As we navigated large genomic datasets, speed became a non-negotiable requirement. Likewise, the issue of false discoveries cannot be ignored if we aim for meaningful, accurate results. Therefore, our adjustments to existing algorithms involved database size reduction, the incorporation of new decision-making rules, or even a complete overhaul of the foundational mathematical frameworks. In summary, our endeavor aims to refine and tailor these algorithms to enhance their speed and precision in transposon detection.

Validation and Benchmarking

After the refinement of algorithms, it becomes imperative to test their efficacy rigorously [15]. We can imagine putting a newly engineered car through its paces in crash tests and speed trials before letting it onto the highways. In our research, these tests manifest as validation and benchmarking procedures. We exclusively used real-world genomic datasets for this research to conduct these evaluations. The use of actual genomic data provided a robust framework for validation, as it exposed the algorithm to the intricacies and variabilities inherent in natural biological systems. The algorithm's performance was meticulously analyzed through statistical parameters, including sensitivity, specificity, and accuracy. These metrics were crucial for determining the degree to which the

modified algorithm improved upon its predecessors. This method of benchmarking against real-world genomic datasets ensured a rigorous and meaningful evaluation, contributing to the robustness and reliability of the algorithm in practical applications.

High-Performance Computing Adaptation

The third component of our research technique focused on tuning the modified algorithms for large-scale computing systems, particularly emphasizing the Bridges2 computing platform at Pittsburgh Supercomputer Center (PSC), sponsored by the National Science Foundation (NSF). These are not ordinary office workstations but sophisticated devices created to handle computationally demanding tasks.

Task partitioning is the primary alteration. Transposon detection was decomposed into a number of more manageable, more compact computing problems. The system's available processors were subsequently divided among these sub-tasks. This division made parallel processing possible by significantly speeding up the computing rate.

Additionally, we invested in other computational efficiencies. Memory optimization allowed us to make the most of the available RAM and storage resources, thereby streamlining the computational processes. We also looked at reducing data transfer times between different nodes within the system to expedite the computational timeline further [16].

By implementing these modifications, we aimed to fully leverage high-performance computing environments' capabilities, thereby enhancing transposon detection algorithms' speed and reliability in our overall methodology.

1.10 Computational Tools and Framework

Python emerged as the top programming language for this research because of its adaptability, extensibility, and widespread popularity within the scientific computing community. Python’s open-source nature and extensive library ecosystem make it exceptionally well-suited for computational biology tasks, including transposon detection algorithms [17].

Within the Python ecosystem, specific libraries were selected to enhance the functionality and efficiency of our computational processes. Notably, NumPy was employed for complex numerical operations due to its highly optimized and vectorized mathematical functions, allowing efficient array manipulations. NumPy's capabilities are essential for handling the high-dimensional data matrices commonly encountered in genomic studies, offering computational advantages regarding speed and memory efficiency [18].

Additionally, the Pandas library was incorporated for data management and manipulation tasks. Pandas excel in handling structured data, providing versatile data structures like Data Frames that are instrumental for the organization and analysis of large and intricate datasets. Its powerful data manipulation capabilities facilitate easy filtering, grouping, and transformation, enabling more streamlined and organized data analysis workflows [19].

The selection of these specific computational tools was not arbitrary but informed by their robust performance attributes and compatibility with high-performance computing environments. The optimization features inherent in these libraries contribute to the efficacy and speed of the algorithms, making them well-suited for our research objectives in the context of transposon detection.

1.11 Thesis Outline

The thesis is organized into six primary chapters, beginning with an Introduction that set the context and objectives of the research. Following this, Chapter 2 offers a Literature Review that explores existing scholarships in computational genomics, MGE detection, and high-performance computing. Chapter 3, titled Algorithmic Innovations, delves into the theoretical and practical nuances of the algorithms developed for MGE detection. Chapter 4 focuses on Validation and Benchmarking, providing statistical measures and analyses used to assess algorithmic performance. Chapter 5 discusses the High-Performance Computing adaptations made to the algorithms, particularly their implementation on the Bridges2 platform. The final chapter, Chapter 6, provides a Discussion and Conclusions section where the research findings are summarized, their potential implications are considered, and recommendations for future research are proposed.

CHAPTER 2

LITERATURE REVIEW

In the rapidly evolving field of computational genomics, various software tools and algorithms have been developed to address complex biological questions. Among these, TIGER (Target/IGE Retriever) and BLAST (Basic Local Alignment Search Tool) have emerged as particularly impactful, each catering to unique sets of challenges and research needs. This chapter aims to provide a comprehensive review of these two pivotal tools, discussing their roles, the previous work that has utilized them, and an evaluation of the current state of computational genomics, particularly in detecting Mobile Genetic Elements (MGEs), known as transposons and high-performance computing.

2.1 The Significance of TIGER and BLAST in Computational Genomics

Firstly, let us consider TIGER, a software designed to accurately map Integrative Genetic Elements (IGEs) within bacterial genomes. TIGER addresses several limitations in earlier software designed for similar purposes [20]. Unlike its predecessors, TIGER utilizes a comparative genomic, ping-pong BLAST approach to offer unprecedented precision in mapping IGEs, mitigating biases associated with the attachment site (attB). This level of precision is particularly vital for understanding complex bacterial systems where small variations can significantly influence virulence and other phenotypic traits. Hence, TIGER serves as an

invaluable asset for those investigating bacterial genomics, gene regulation, and evolution.

Conversely, BLAST, an acronym for Basic Local Alignment Search Tool, serves as a keystone in the bioinformatics community. Conceived by Altschul et al. in 1990, this software lays the groundwork for a wide array of sequence comparison tasks, ranging from simple sequence matching to complex phylogenetic analyses [21]. BLAST's algorithms have been honed over the years to offer flexibility, speed, and reliability, rendering them applicable not only in genomics but also in proteomics, metabolomics, and other disciplines requiring sequence alignment and comparison. Therefore, its utility extends far beyond genomics, touching virtually every facet of life sciences research.

The complementary nature of TIGER and BLAST becomes evident when one considers the ways in which these tools intersect. For instance, TIGER's ping-pong BLAST approach relies on the foundational sequence alignment algorithms of BLAST, illustrating how even specialized tools like TIGER are undergirded by the more general capabilities of software like BLAST.

In summary, both TIGER and BLAST occupy distinct yet complementary niches in computational genomics. TIGER excels in its focused role of precisely mapping IGEs, especially in bacterial systems, while BLAST's broader functionalities enable it to be applied across multiple biological disciplines. Both tools, therefore, significantly contribute to the advancement of computational genomics, each from its unique vantage point.

2.2 Previous Work Utilizing TIGER and BLAST

Since its introduction, TIGER has emerged as a keystone in computational genomics, offering unprecedented precision in the mapping of integrative genetic elements (IGEs) within bacterial genomes. Conceived by Mageeney et al., this software was tailored to identify IGEs with such specificity that it could elucidate the nuances of gene integrity modulation upon IGE integration [20]. The groundbreaking work of Mageeney et al. has not only validated TIGER's effectiveness but also set the stage for a burgeoning body of research. This corpus explores diverse themes from bacterial gene regulation to virulence, all the while leveraging the precision that TIGER provides.

In a noteworthy example, TIGER's capabilities were fully exploited in a study concerning the bacterial genome of Eco567. This application of TIGER elucidated the differential activities of prophages embedded in bacterial genomes by distinctly identifying attB and attP sites. These sites signify the excision and the integration of prophages into bacterial genomes, respectively. The attention to detail facilitated by TIGER allowed for an unparalleled analysis of the dynamic behavior of prophages. For instance, the software facilitated the identification of late genes in prophages, which is essential for understanding the lifecycle and potential pathogenicity of these genetic elements. It is this level of detailed analysis that has expanded our comprehension of bacterial genomics and opened novel research avenues for studying microbial behavior and interactions at the genomic level [22].

Another pioneering study utilized an updated version of TIGER, known as TIGER2, to detect and analyze genomic islands (GIs) in microbial genomes, including metagenome-assembled genomes (MAGs). Originally engineered to map GIs confined to single scaffolds, TIGER2 incorporated two novel modes: "CircleOrigin" and "Cross." These features enabled the detection of split GIs that either wrap around the origin of a circular replicon or have termini on separate scaffolds. Notably, this upgrade doubled the number of GIs identified compared to its predecessor. Furthermore, TIGER2 sheds light on population micro diversity, establishing virus-host linkages and contributing to ecological assessments in microbiome research [23].

An additional study, however, presented critical insights into some limitations of TIGER, particularly regarding the annotation of IGEs carrying tyrosine recombinases (YRs). While TIGER offers a robust pipeline for IGE annotation, the software encounters challenges when dealing with YR-containing elements. Specifically, the close relationship between YR family transposases/integrases and essential bacterial genes creates issues. TIGER tends to discard Xer and Integron-related sequences, assuming all other YRs to be MGE integrases, which can lead to false-positive hits. Despite its utility, this calls attention to areas where TIGER may benefit from further refinement, especially for more accurate functional annotation of MGE-borne proteins and YRs [24].

In contrast, BLAST's utility extends across a diverse range of applications. One such study developed a BLAST-based approach called TESeeker [25] to identify transposable elements (TEs) in genomes. This method initiates a BLAST

search against a given genome using a TE family as a query. The resulting hits are assembled using the CAP3 assembly program, and the sequences are then trimmed and subjected to another round of BLAST searches. Finally, a multiple sequence alignment is generated using ClustalW2, followed by yet another BLAST search using a consensus TE sequence from CAP3. The TESeeker pipeline, available for download, significantly utilizes BLAST for sequence searching and alignment, highlighting BLAST's indispensability in TE identification and genomic sequence alignment [25].

Another insightful paper revealed the multiple dimensions of BLAST software. This paper introduced the programs as a fundamental part of protein and DNA database searches, focusing on significant enhancements that include decreasing execution time and boosting sensitivity to weak sequence similarities. The paper also introduced gapped BLAST, which runs approximately three times faster than its predecessor. PSI-BLAST, another innovation, utilizes position-specific score matrices to enhance sensitivity, notably uncovering new members of the BRCT superfamily. The paper concludes by discussing the potential ramifications of abandoning the statistical assessment of alignments, particularly affecting the automatic iteration of PSI-BLAST [26].

Moreover, the modified BLAST version called BLASTER has also been developed as part of the TEdenovo pipeline. This customized tool is specifically designed for the identification of TEs and operates through a self-comparison of the input genome. Though the TEdenovo pipeline offers high configurability, it can be complex for inexperienced users. Nevertheless, BLASTER adds an additional

layer of functionality to BLAST, contributing to the tool's adaptive applicability in computational genomics [27].

2.3 The Evolving Paradigm of Computational Genomics: Advances, Challenges, and Applications

Computational genomics, a symbiosis of computer science and biology, has ushered in a transformative era in our understanding of biological systems. Through intricate algorithms and high-throughput computational strategies, this rapidly advancing field has drastically lowered the cost and time needed for genome sequencing and assembly [21]. Additionally, it has broadened our comprehension of gene functionality, regulation, and associated genetic variations [28].

Importantly, computational genomics extends its applications to various sectors, including medicine, agriculture, and evolutionary biology [29]. Its contributions range from identifying novel genes to improving drug efficacy and crop yield. However, there remains a nuanced complexity in addressing some of the field's most intriguing components, notably the detection of mobile genetic elements (MGEs) and transposons.

MGEs or transposons, ubiquitous genomic entities capable of altering genomic structures, pose challenges and opportunities for researchers. Their identification is not merely an academic exercise but has profound implications. For instance, tracking MGEs can shed light on evolutionary patterns, and their associations with diseases can inform therapeutic strategies. While tools like BLAST have been employed for MGE detection by leveraging known MGE

sequences, de novo algorithms have also been developed for identifying novel MGEs (TESeeker; BLASTER).

Though specialized software tools like TIGER and BLAST have brought substantial advancements, they also come with inherent limitations. TIGER has shown certain constraints in the functional annotation of tyrosine recombinases, leading to potential false positives. BLAST, although robust, may require further refinement for specific applications, such as MGE detection.

High-Performance Computing (HPC) has emerged as an indispensable asset in managing the data-intensive demands engendered by modern genomics. HPC platforms offer the computational heft required for various tasks like genome assembly, data analysis, and biological process simulation [30]. As genomics continues to generate colossal data sets, the role of HPC in facilitating analyses will only increase in importance.

In summary, computational genomics remains a field ripe with innovation, benefiting from constant methodological upgrades and decreasing operational costs. However, the detection of MGEs and transposons and the computational infrastructure required to manage data-intensive tasks present as yet unresolved challenges. As technologies continue to advance, the promise of unlocking deeper biological mysteries and developing innovative therapeutic interventions remains a compelling prospect.

As delineated in the preceding discussions, the computational tools TIGER and BLAST have rendered significant contributions to the advancement of computational genomics, notably in areas such as high-precision mapping of IGEs

and sequence alignment. Moreover, they have been employed in a broad array of research endeavors ranging from bacterial genomics to mobile genetic elements detection, which have crucial implications for medicine, agriculture, and beyond. Nonetheless, existing gaps in methodological approaches to MGE and transposon detection, coupled with the computational demands of contemporary genomics, underscore the necessity for further technological and algorithmic innovations. High-performance computing has emerged as a pivotal asset in tackling these data-intensive challenges, heralding a new era of scalability and computational efficiency. In alignment with these existing challenges and technological trends, this thesis endeavored to augment the current discourse by specifically focusing on the enhancement of algorithmic strategies and optimizing adaptability to high-performance computing platforms.

CHAPTER 3

IMPROVING THE EFFICIENCY OF MOBILE GENOMIC ELEMENTS DISCOVERY THROUGH SYSTEMATIC EVALUATION OF REFERENCE DATABASE COMPOSITION

Mobile genetic elements (MGE) are genetic sequences that may move about on a chromosome and be passed from one chromosome to another and across bacteria and species. They often bring some benefits to the bacteria, such as improving the pathogenicity of bacteria, symbiosis with another organism, or bringing metabolic genes that allow the bacteria to perform a new metabolic function [31]. In contrast to their potential benefit, mobile genomic elements are linked to food positioning and diseases such as human kidney failure [32]. Discovering mobile genomic DNA is an active research area considering its importance in expanding our understanding of underlying mechanisms in bacteria and, ultimately, for human health applications. Current multidisciplinary research involves expediting discoveries of mobile genomic elements by improving our previously developed algorithms and computation approaches. We have an existing software (TIGER) that identifies MGEs in genomes and maps them precisely to the nucleotide. TIGER employs a comparative genomic ping-pong BLAST method based on the assumption that the Mobile Genetic Element integration module is cohesive [33]. As a result, TIGER software maps Mobile Genetic Elements with exceptional precision and without attB site bias.

The original TIGER runs utilized databases that included all the genomes for a specific genome. As a result, the runtime was too long. Through this research, we have expedited the MGE discovery process through algorithms developed by Dr. Kelly from Sandia National Laboratory [54]. We implanted three different algorithms and compared their performance. We examined the performance of each algorithm on selected species. We tested our approaches on E. coli species, and our results showed that prophage yields were relatively insensitive to the database size, unlike the non-phage islands. This valuable finding enabled us to use much smaller databases to speed up our software significantly.

3.1 Introduction

Mobile Genetic Elements

Mobile genetic elements (MGEs) like plasmids and prophages frequently carry determinants of bacterial features, including pathogenicity, symbiosis, and antibiotic resistance [31]. Mobile elements have the ability to modify their insertion position, copy quantity, give novel gene functions, and influence chromosomal gene expression. Gene gain and loss are known to be potentiated by mobile elements, a key influence that can drastically alter bacterial fitness. This shift could lead to genetic adaptation to new settings and the creation of diverse bacterial populations, which could lead to the emergence of separate evolutionary species [34]. TIGER software maps Mobile Genetic Elements with exceptional precision and without attB site bias. The attB site is a short DNA sequence corresponding to the strand-switching crossover region. Whenever we map a genomic island, we are mapping where an integrase recognizes. Integrase is an enzyme encoded by the

genomic island with a gene for integrase. It recognizes DNA sequences in the bacteria's chromosome and in the island itself. The island enters the cells through a phage particle, and that phage particle delivers genomic island DNA into the cell.

There is a site on that DNA where the integrase recognizes on the genomic island. Moreover, the complementary site on the bacterial chromosome is also recognized. It brings those two DNA sites together and recombines them. As a result, a large chromosomal circle and a small genomic island circle become a single, giant circle. The integrase that we studied is from two protein families. One is called serine, and another is called tyrosine integrase. They are called that because their catalytic site has either tyrosine residue or serine residue that captures the reaction intermediate. So, it holds on to one DNA strand for a while in the middle of the recombination/reaction.

Furthermore, these two protein families are unrelated by evolution. They do not have the same shape and do not show any homology, but they have very similar functions. They both do the trick of bringing DNAs together and recombining them, even though they are entirely different proteins. We studied both serine and tyrosine integrases. There were more tyrosine integrases than serine integrases. Thus, we had an abundance of tyrosine integrases in these genomes. Serine integrases are also crucial for particular genomic islands.

In the serine recombinase family, there are two main domains. One is the catalytic domain where the actual recombination happens, and another is the extensive domain that recognizes/binds DNA and gives the site specificity. Some enzymes have only the catalytic domain, which may not be the ideal integrases.

Thus, we found the genomic island that only got this catalytic domain of serine recombinases. Therefore, we considered the true integrase of either tyrosine or serine recombinase with this big domain, not the serine recombinase for the catalytic domain only. The crucial questions were 1- How many islands are formed, and 2- What is the percentage of the island with the "good" integrases? Our plan was to continue improving the performance, and we worked on novel algorithms and software tools.

Genomic Element Discovery Tool: TIGER

TIGER was developed by our collaborators at Sandia National Laboratories, and we studied the software for our research [33]. The TIGER aimed to find reference genomes with a continuous IGE integration site to identify and map IGEs. The key inputs are a replicon DNA sequence, a coordinate on that DNA (here, the midpoint of an integrase gene), and a reference genome BLAST database. BLAST is a sequence-matching program. BLAST takes query sequences and matches them to a database that includes reference genomes for all other species. Thus, as the result of the left and right-side queries for finding a genomic element, the program can map an island's left and right ends. One needs to define an island first and then use it. We measured the support of an island by the amount of matching Blast queries, which served as a quantitative outcome of our search.

The database, a collection of similar genomes, was explored with BLASTN in default mode using two 15-kb query sequences (q1L and q1R) extracted from the replicon to the left and right of the coordinate. Matches over 500 bp were further analyzed, with those that fully reach the input coordinate filtered out. A 3 kb return

query was taken from the reference genome region close to the coordinate-proximal end of each match, going back 250 bp into the matching region to include the direct repeat (DR) sequence for each match. To discover the matching distal flank of the IGE, the set of return queries (q2) was employed using BLASTN against the replicon [33].

Algorithmic Selection Approaches

The TIGER tool was intended to search through large databases of genomic sequences and discover the mobile elements. We benefited from running TIGER on full-size datasets, albeit with the caveat of long runtime. In order to speed up the discovery process, we, in collaboration with the Sandia team, proposed four algorithmic approaches to select representative genomes from each species. We called the three algorithms 1- randomDB, 2- diverseDB, 3- smartDB, and 4- evalDB.

- *RandomDB algorithm*

In the randomDB algorithm, for a given genome database size *n*, and a genome file, it lists *n* random database member genomes.

- *DiverseDB algorithm*

The diverseDB algorithm, for a given genome database size n, and pre-calculated MASH distance file [35], makes a file listing n most diverse genome.

- *SmartDB algorithm*

Then, the smartDB algorithm limits the number of genomes included to improve efficiency, with the database size capped at a maximum number of genomes and filled phylogenetically up to the taxonomic rank of order.

- *EvalDB algorithm*

Finally, in the evalDB algorithm, for a given database list file and island support file, it finds which islands are supported by the database, writing the new support value, length, species, original support value, score, int summary, type, and support from all the input genomes. In the current work, we present our proposed algorithms and our results.

3.2 Methods

Mobile genomic island discovery has a vital role in discovering the mechanisms that can enhance the pathogenicity of the bacteria. Discovering the MGEs accurately has been possible using our already published TIGER tool. In the current work, we improved the runtime of the TIGER to expand its usage. In addition, the current research proposed four algorithms to select the input genomes before running TIGER to expedite the MGE discovery process. We describe these four algorithms in this section.

Our study demanded significant computational resources and was conducted on the Bridges2 system at Pittsburg Super Computing Center (PSC), sponsored by the National Science Foundation (NSF) ACCESS program [11], formerly called XSEDE (Extreme Science and Engineering Discovery Environment) project [36].

Choosing Genomes Using Our Three Proposed Algorithms

We used different algorithms to discover supported genomic islands for various specific groups of species. The database composition algorithm was based on the following principles,

1) that TIGER execution should be sped up by smaller, more targeted databases while rejecting few or no genuine GIs (various size limits were tested, settling on 500),

2) that faulty joins between genomic segments were more likely to be present in lower-quality genomes, which can lead to false-positive GI calls,

3) there exists an ideal range of phylogenetic distances for reference genomes that will best identify GIs in the target genome - close enough to retain an integration region with enough homology for BLAST identification. In the current work, we set a higher distance than the reference genomes must belong to the same taxonomic order as the target species, but far enough away that some of the reference genomes had a continuous integration site needed for IG detection and this optimal range may exceed the species level because there are cases, where IG is very common in a species some of that species' genomes, have continuous integration sites. We spent at least 10% of the base data of each species for members outside the species when available.

First, we applied the random approach (Supplementary 4.1), which lists *m* random database member genomes for a given genome database size of n and genome inputs. The algorithm was implemented by a custom-made script called randomDB.pl shown in Algorithm 1. In the randomDB script for different counts, we used the genome files with a line for each species' genome (ecolist.txt, Escherichia__Flexneri.txt, Escherichia__dysenteriae.txt, coli.txt, coli_D.txt).

Algorithm 1 Select Random Genomes

Require: $n \in \mathbb{N}$ (Count of genomes to include), F (GenomesFile)
Ensure: G' (A subset of genomes)
1: Let A be the set of arguments provided.

```
2: if |A| = 2 then
3:     Display the correct usage of the script.
4:     exit
5: end if
6: Set n and F based on the elements of A.
7: if file F does not exist
then
8. Display an error
message.
9:    exit
10: end if
11: Let G be an empty set.
12: for each genome g in F do
13:     Extract the genome name as g_name.
14:     G = G ∪ {g_name}
15: end for
16: Let G' be an empty set.
17: if |G| ≤ n then
18:     G' = G
19: else
20:     Let O be a permutation of set G.
21:     for i = 1 to n do
22:         g_i = first element of O
23:         Remove g_i from O
24:         G' = G' ∪ {g_i}
25:     end for
26: end if
27: for each genome g in G' (sorted) do
28:     Display g.
29: end for
```

In summary, the code took a file containing a list of genomes and randomly selected a specified number of genomes from that list, ensuring no duplicate selections. The selected genomes were then printed in alphabetical order as illustrated in Table I.

TABLE I

LIST OF 13 GTDB SPECIES

Escherichia Species	**Count**
Escherichia__albertii	70
Escherichia__coli	3349
Escherichia__coli_C	42
Escherichia__coli_D	1450
Escherichia__dysenteriae	1173
Escherichia__fergusonii	14
Escherichia__flexneri	9094
Escherichia__marmotae	48
Escherichia__sp000208585	20
Escherichia__sp001660175	2
Escherichia__sp004211955	2
Escherichia__sp005843885	36

Secondly, we applied a diverse approach (supplementary 4.2), which lists n most diverse genomes for a particular genome database size n and mash distance file. In the in-house written diverseDB.pl shown in Algorithm 2 scripts for different counts, we used the MashDist file, which had three columns for genomeA, genomeB, and Mashdist for all genomes of the species (eschmash.dist, Flexneri.dist, coil.dist, coli_D.dist, dysenteriae.dist). The MashDist files were created by the MASH tool [35].

Algorithm 2 SelectDiverseGenomes

Require: $n \in \mathbb{N}$ (Count of genomes to include), F (MashDistFile)
Ensure: Set of selected genomes
1: Let A be the set of arguments provided.
2: **if** $|A| \neq 2$ **then**
3: Display the correct usage of the script.

```
    exit
end if
Set n and F based on the elements of A.
if file F does not exist
then
    Display an error
message.
    exit
end if
Let D be an empty matrix representing genome distances.
for each line l in F do
    Parse l into (g_a, g_b, d)
    D[g_a][g_b] = d and D[g_b][g_a] = d
end for
Let G' be an empty set representing selected genomes.
if |G| ≤ n then
    G' = G
else
    Let S be an empty dictionary representing the sum of distances.
    Select arbitrary genome g_last from G
    G' = G' ∪ {g_last}
    for i = 1 to n − 1 do
        Let M be a tuple representing genome with max distance.
        for each genome g in G do
            if g not in G' then
                S[g] = S[g] + D[g_last][g]
                if M is empty or M[1] < S[g] then
                    M = (g, S[g])
                end if
            end if
        end for
        g_last = M[0]
        G' = G' ∪ {g_last}
    end for
end if
for each genome g in G' (sorted) do
    Display g.
end for
```

In summary, this code took a file containing mash distances between genomes and selected a specified number of genomes based on their distances. It chose genomes with the maximum distance from each other, ensuring that the selected genomes were not duplicated. The selected genomes were then printed in alphabetical order.

Thirdly, we used the smartDB approach to create a small and targeted database while still representing the taxonomic diversity of interest. SMAll Ranked Tailored (SMART) DBs are a database designed for phylogenetic analysis that limits the number of genomes to improve efficiency. By capping the number of genomes and filling them phylogenetically, SMART DBs enables faster and more efficient analysis for researchers. The SMART DB software pipeline, as seen in Fig. 3.3, was designed to automate genome data collection and design and update a database (DB) for genomic analysis. It offered two modes: Design and Quick Setup.

In the Design mode, the pipeline collects genome assemblies, calculates pairwise distances between genomes, ranks genomes within each species, and designs and prepares each DB. The pipeline retrieves needed genome assemblies from the National Center for Biotechnology Information (NCBI) FTP server and repeats collection attempts until all required genomes are downloaded. The NCBI serves as a national resource for molecular biology information and provides access to a multitude of databases and tools that facilitate research in biomedicine, bioinformatics, and related disciplines [37]. Pairwise distances are calculated using Mash, and rankings are determined based on quality and contig count. The DB for each species is designed by filling it with genomes from the ranked list, and if the cap is not reached, genomes from closely related species are added. Some DBs may be small due to the limited genomes for specific taxonomic orders. The pipeline also warns about these small DBs, as they may have reduced capability to find genomic islands (GIs). BLASTN databases are created for each unique DB design.

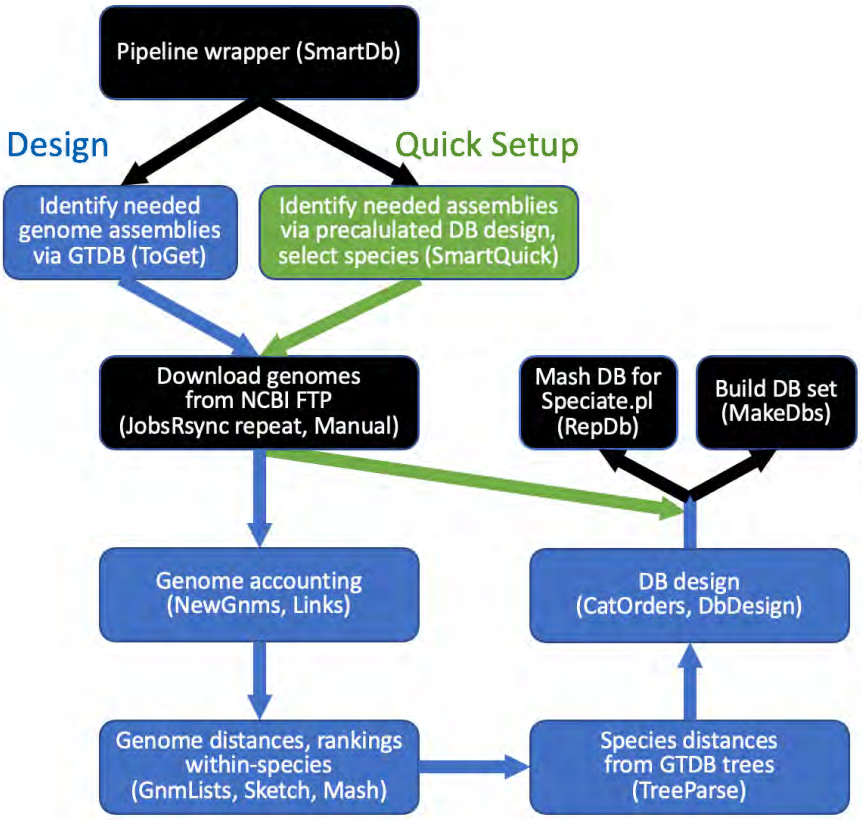


Fig. 3.3: SMART DB software pipeline.

The Quick Setup mode uses a precalculated DB design file, skipping the time-consuming steps of distance measurements and DB design. A utility script called Speciate is recommended to aid in selecting the appropriate DB for a query genome. The pipeline's dependencies include Mash, BLAST, and fastANI [38].

Finally, we applied the eval approach (supplementary 4.3), which gave us more information about the islands. In this approach shown in Algorithm 3, we found which islands the DB supported for a given DB list file and island supported file, writing the new support value, length, species, original support value, score,

int summary, type, and support from 15299. After that, we used this output to compute statistics.

Our methodology began with the sequencing of genomes within each species, adhering to quality standards set by the Minimum Information about a Metagenome-Assembled Genome (MIMAG). These guidelines are instrumental in assessing the quality of metagenome-assembled genomes (MAGs), and their adherence ensures that the genomic data is both reliable and comparable across different studies. The MIMAG guidelines cover a multitude of elements, from data generation and assembly to annotation and quality evaluation. By adhering to these rigorous criteria, we lay a foundation for quality-controlled, transparent, and reproducible computational genomics research [39].

Algorithm 3 EvalDBApproachAlgorithm

Require: $D \in$ F (DBfile), $S \in$ F (SuppFile) **Ensure:** Processed output based on D and S 1: Let A be the set of arguments provided.
2: **if** $|A| = 2$ **then**
3: Display the correct usage of the script.
4: **exit**
5: **end if**
6: Set D and S based on the elements of A.
7: **if** file D does not exist **then**
8: Display an error message about missing D.
9: **exit**
10: **end if**
11: **if** file S does not exist **then**
12: Display an error message about missing S.
13: **exit**
14: **end if**
15: Let G be an empty set.
16: **for** each line l in file D **do**
17: Remove prefix "Eco" from l.
18: $G = G \cup \{l\}$
19: **end for**
20: **for** each line r in file S **do**
21: Split r into fields $f_1, f_2, \ldots, f_n$ using delimiter "".

```
    Let c be 0.
    for each element e in f10 split by comma do
        if e exists in set G then
            c = c + 1
        end if
    end for
    if c > 0 then
        if f2 matches pattern "(
d+)-(
d+)$" then
            Extract integers i1 and i2 from the match.
            Compute L = |i2 − i1| + 1.
            Display f1, c, L and f4, f6, f7, f8, f9 separated by ""-.
        else
            Display an error message with f1.
            exit
        end if
    end if
end for
```

Next, the genomes were sorted based on their quality and the number of frames, as indicated by the MIMAG quality and the Genome Taxonomy Database (GTDB) metadata table. This is another key component of our methodology. Important metadata such as taxonomy, assembly quality, and genomic ID are provided by this site. We could verify our findings with the help of this database, giving us a more comprehensive, multifaceted understanding of the genetic structures we were studying. Our findings gained further confidence because the GTDB metadata table guaranteed that we were working with well-characterized, quality-assured genomes [40].

Subsequently, pairwise distance measurements were performed using Mash, a computational tool for fast genome and metagenome distance estimation, for all genomes within each species [41]. The top 90% of genomes with the highest quality, as defined by MIMAG guidelines, were then rearranged based on their diversity, as determined through Mash's distance estimations. This arrangement

involved selecting the second genome farthest from the first, the third genome with the most significant coherence distance to the first and second genomes, and so on. Finally, a smart approach was applied to all thirteen species.

Several principles guided the algorithm used for composing the database. Firstly, smaller and more focused databases were preferred to enhance the speed of TIGER execution while minimizing the exclusion of legitimate genomic islands (GIs). Different size limits were tested, and a limit of 500 was found suitable. Secondly, lower-quality genomes were more likely to contain incorrect connections between genomic segments, leading to false-positive GI identifications. Thirdly, an optimal phylogenetic distance range was determined for the reference genomes to identify GIs within a target genome effectively. The reference genomes were required to belong to the same taxonomic order as the target species but be distant enough to possess uninterrupted integration sites necessary for GI detection. Lastly, this optimal range could extend beyond the species level since certain GIs may be widespread within a species, resulting in a few genomes from that species having uninterrupted integration sites. Therefore, at least 10% of each species database was reserved for members from outside the species when available.

The composition of the smart reference database proceeded as follows. Each species' contribution was limited to 90% of the database size limit, ensuring that at least 10% of the database included genomes from outside the target species if they were accessible. Species from the same taxonomic order as the target species were ranked based on their phylogenetic distance from the target species. Starting with the target species, genomes were selected from the ranked list of the

contributing species until either the species cap or the database size limit was reached.

Furthermore, we parsed the GTDB species trees for bacteria and archaea, collecting the branch length below each node and the list of nodes beneath each species. This allowed for efficient distance measurements between any pair of species.

3.3 Datasets

Dataset 1: E. coli

Escherichia genomes datasets used were the 15299 names of genomes named by GTDB listed in supplementary Table 1 that were used for applying our new algorithms. In addition, we used previously generated data from running TIGER with massive databases, which are included in supplementary Table 2, containing 64838 islands from our 9457 Escherichia genomes that TIGER has finished, with ten fields per line. Also, we have a breakdown of 15299 genomes into these 13 GTDB species with different counts (Table I).

3.4 Results

The aim of this research was to expedite the TIGER tool's runtime. First, we introduced three algorithms for choosing genomes and then examined their performance using different species. In this section, we present our results that clearly showed that if we used smaller database sizes, we discovered islands with higher scores, longer lengths, and better islands. Therefore, we decided to run TIGER for three sizes of databases (100, 500, 15299), and after running TIGER for

those databases, we saw that we had achieved our goal of speeding up our existing software without losing any islands.

We have applied our four proposed algorithms to E. Coli data. Therefore, we will start by presenting our E. Coli results.

E.Coli Results

We utilized our own customized scripts to implement our algorithms on a total of 15299 E. Coli genomes. To determine the most suitable database sizes for our algorithm evaluation, we selected intermediate sizes ranging from 1 to 15299 genomes for E. Coli. As Table 2 illustrates, our proposed algorithms significantly expedited the discovery process. By reducing the database size to 100 genomes instead of the original > 15K genomes, we achieved a 138-fold increase in speed. Similarly, the size of 500 databases resulted in a 38-fold increase in speed.

We created BLAST databases with smaller counts to calculate the TIGER runtime for different database sizes. We initiated this process by generating a fasta file shown in Algorithm 4 with either randomly, diversely, or smartly selected n DB size (Supplementary 4.4). Subsequently, we created BLAST databases of 100, 500, and 15299 sizes from the FASTA formatted file for randomly, diversely, and smartly selected genomes. In bioinformatics, the FASTA structure is a text-based illustration of nucleotide or peptide sequences that is often used. It is composed of a single line of description followed by lines containing sequence data, making it easy to store and read sequences. We then developed three separate scripts to run TIGER on 100 genomes each against the three databases and kept a separate record

file for time management as see in Table II. We also took care of the memory it used as seen in Table III.

Algorithm 4 FastaFileMakerAlgorithm

Require: *L* (genomeList), *O* (outfasta)
Ensure: Concatenated genome sequences in output file
1: Let *A* be the set of arguments provided.
2: **if** $|A| = 2$ **then**
3: Display the correct usage of the script.
4: **exit**
5: **end if**
6: Set *L* and *O* based on the elements of *A*.
7: Remove (unlink) file *O* if it exists.
8: **for** each line *l* in file *L* **do**
9: Extract the first non-space sequence of characters from *l* and assign it to *id*.
10: Append the contents of the file /ocean/projects/mcb130021p/shared/ ecoli/$id/genome.fa to *O*.
11: **end for**

Our evaluation criteria for database size reduction results consisted of five factors: 1- Islands count, 2- Islands lengths, 3- Island scores, 4- Integrase summary, and 5- Island types. In addition, we ensured that our speed optimization did not impact the accuracy of the discovered islands.

TABLE II

EFFECT OF THE DATABASE SIZE REDUCTION ON TIGER RUNTIME

	100 input DB (for 100 jobs)	**500 input DB (for 29 jobs)**	**15299 input DB (for 25 jobs)**
Real-Time	10m37.33s	21m24.33s	39h2m32s
User Time	1m57.19s	9m45.38s	6h4m32s
System Time	5m37.40s	17m30.56s	18h10m27s

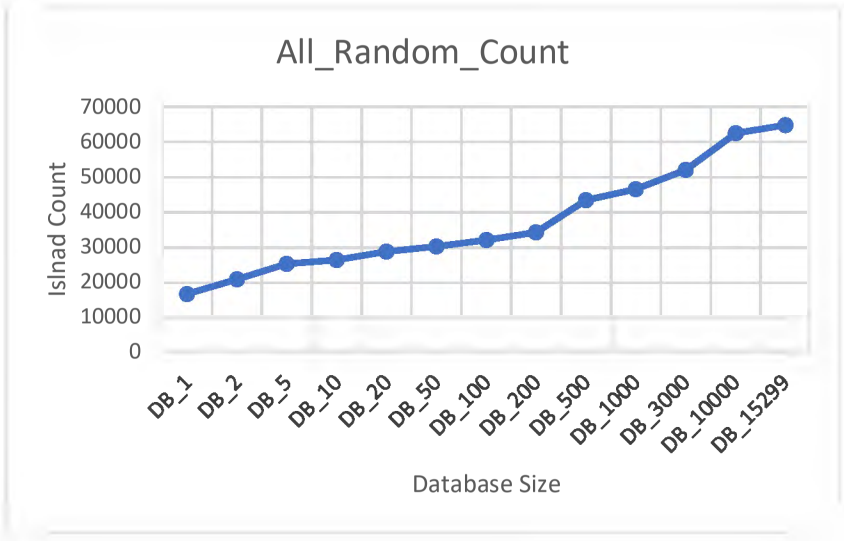
TABLE III

EFFECT OF THE DATABASE SIZE REDUCTION ON MEMORY USEAGE

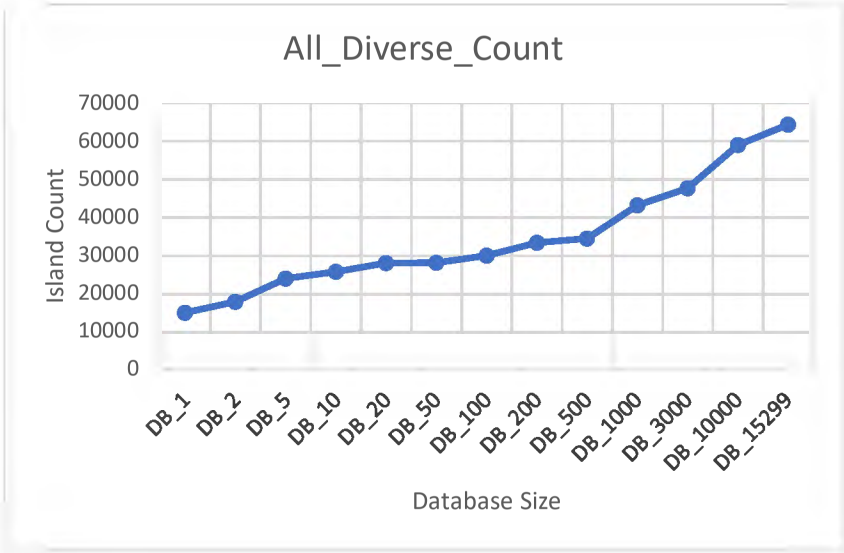
	100 input DB (for 100 jobs)	**500 input DB (for 29 jobs)**	**15299 input DB (for 25 jobs)**
Allocated Memory (MB)	128000	128000	128000
Used Memory (MB)	117.0	141.0	3807.0

Island Count

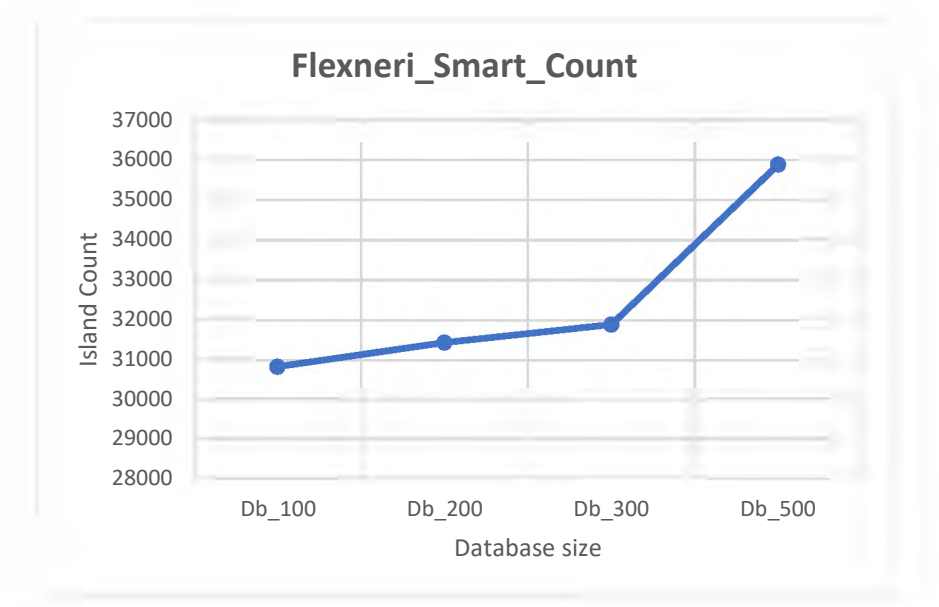
We made plots for island count against database size, and it showed us how the number of islands was reduced as we used smaller and smaller database lists. We discovered a higher number of islands with a larger database for the four approaches shown in Fig. 3.6. Also, Fig. 3.6 shows that when we moved the database size from 1 to 15299, the islands count was moved in the range of 16811 to 64838 (a) and 15170 to 64313 (b). We noticed that from 500 to 15299 genomes, there was not much difference in the count for the random. However, there was no considerable change in the count for the diverse one, from 1000 to 15299 genomes. Finally, to evaluate the smartDB algorithm's performance, we decided to present the Flexneri species as it is the largest species present as part of the E.coli family in GTDB. Fig. 3.6 (c) shows that there is not much difference in the count from smart_Flexneri between 100 and 300 databases. So, from the graphs, we can say that making smaller databases, for example, 500 seems reasonable, and we are not losing many islands.


All_Random_Count
Islnad Count
70000
60000
50000
40000
30000
20000
10000
0
DB_1
DB_2
DB_5
DB_10
DB_20
DB_50
DB_100
DB_200
DB_500
DB_1000
DB_3000
DB_10000
DB_15299
Database Size

(a)


All_Diverse_Count
Island Count
70000
60000
50000
40000
30000
20000
10000
0
DB_1
DB_2
DB_5
DB_10
DB_20
DB_50
DB_100
DB_200
DB_500
DB_1000
DB_3000
DB_10000
DB_15299
Database Size

(b)



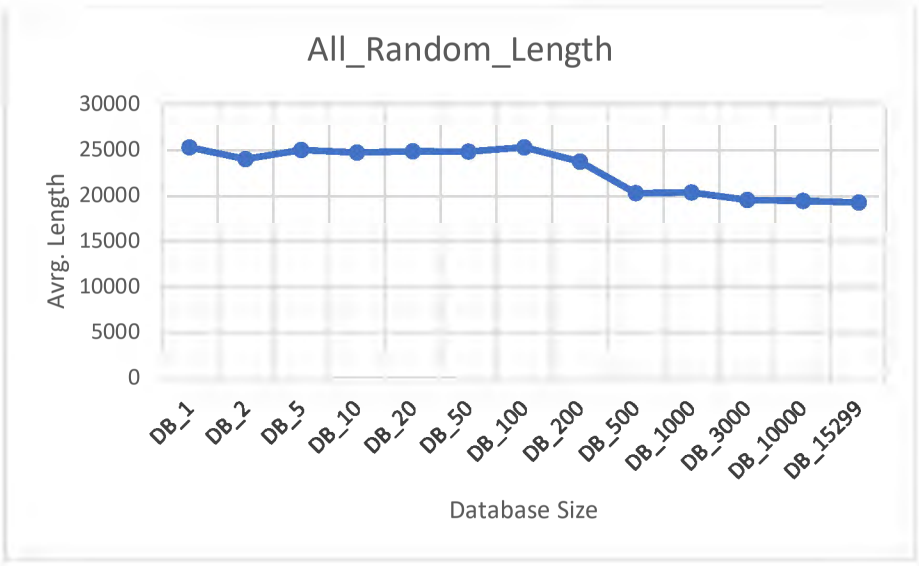
(c)

Figs. 3.6 (a, b, c): Island count versus database size. Comparing the performance of three proposed algorithms by differing the input database size and comparing the discovered island counts.

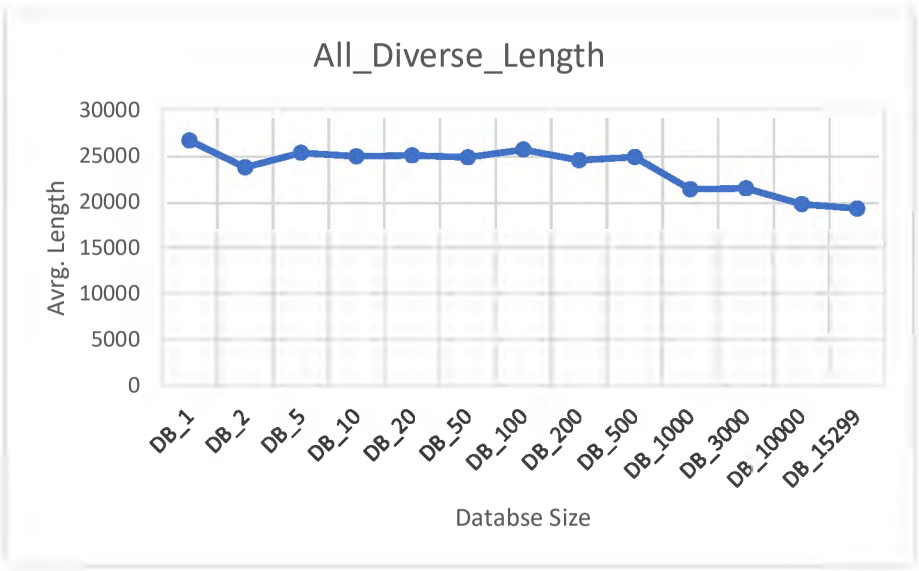
Islands Length

Fig. 3.7 demonstrates how the size of the islands grew more prominent when the database sizes were lowered. We obtained longer islands, which are preferable. Also, it showed that when we moved the database size from 1 to 15299, the islands' lengths were moved in the range of 25288.4 to 19346.6 (a) and 26745.6 to 19411.3 (b). Moreover, the Flexneri species' length changed from 24302.63 to 22308.19 when we moved from a database size of 100 to 500. In the case of random, we noticed that from 500 to 15299, there was not much difference in length, and for the diverse database sizes, between 1000 to 15299, the length did not change visibly. Lastly, for the smartDB (c), Flexneri species did not show much difference

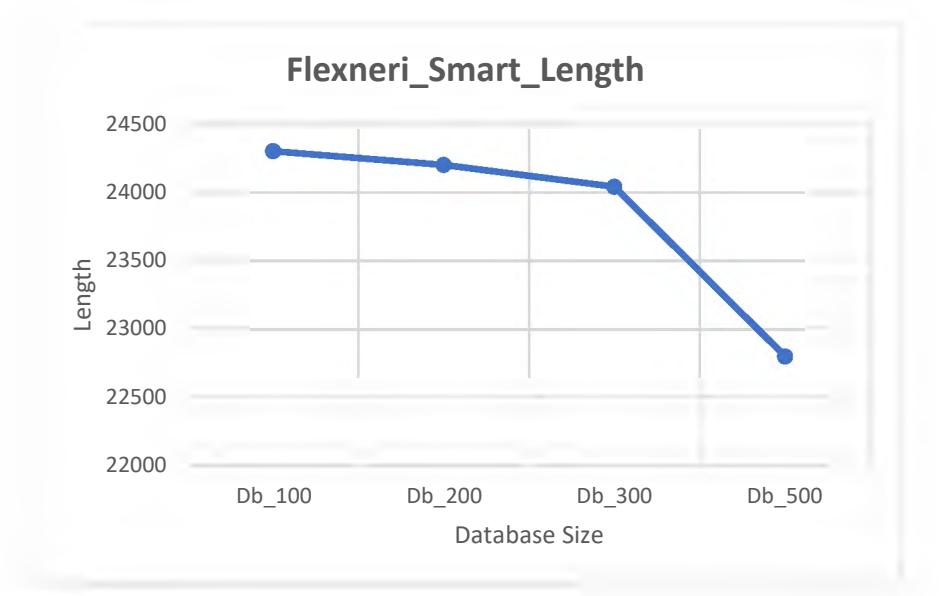
in length between 100 and 300 databases. So, based on the graph, we concluded that creating smaller databases, such as 500, is logical.



(a)



(b)



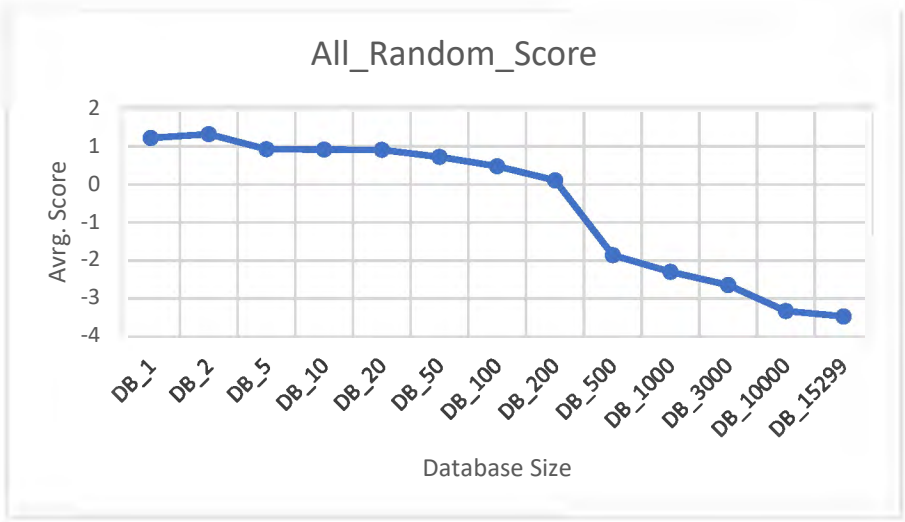
(c)

Fig. 3.7 (a, b, c): Island length versus database size. Comparing the performance of three proposed algorithms by differing the input database size and comparing the discovered islands' length.

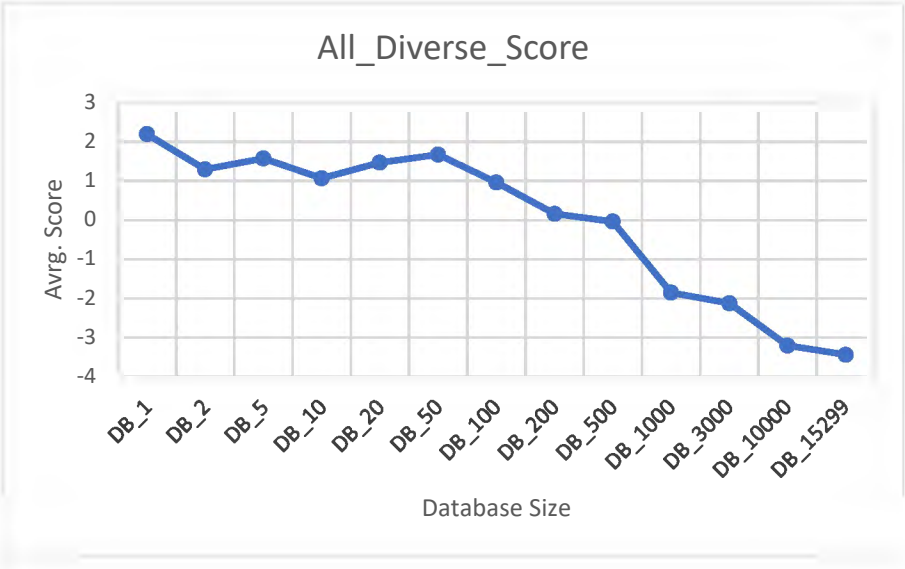
Islands Score

Fig. 3.8 depicts that the islands' scores decreased with the larger database size. It implied that we were discovering higher scores with the smallest database size. A higher score means the island is more reliable. It moreover indicated that when we increased the database size from 1 to 15299, the score of the islands varied between 1.22 and -3.47 (a), 2.19 and -3.44 (b), and 0.5785 to -0.2227 (c). Throughout the random case, we discovered that the score did not differ significantly between 500 and 15299 database sizes, and in the case of diverse, the score did not differ much between 1000 and 15299 database sizes. Finally, between 100 and 300 database sizes, the Flexneri score did not differ significantly. As a

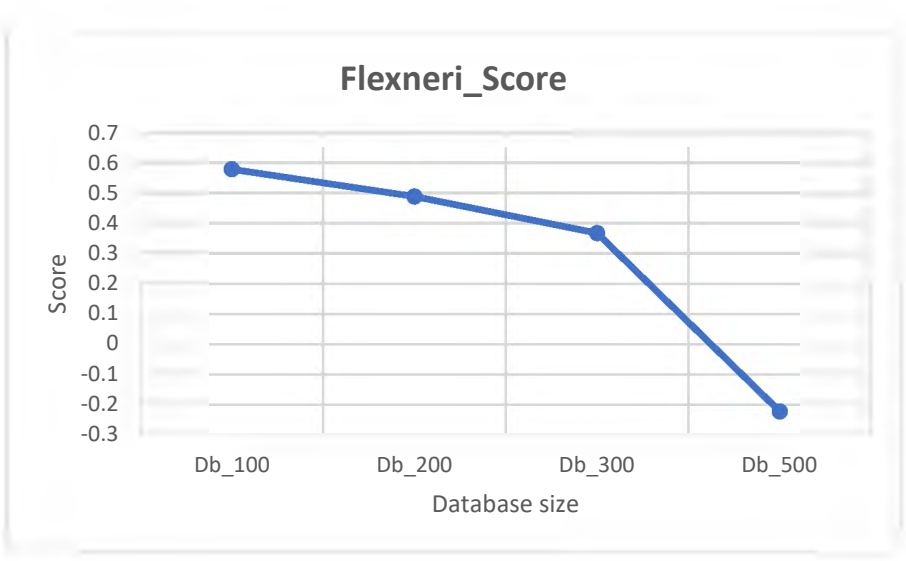
result, we deduced from the graph that establishing smaller databases, such as 500, was rational for the island score.



(a)



(b)

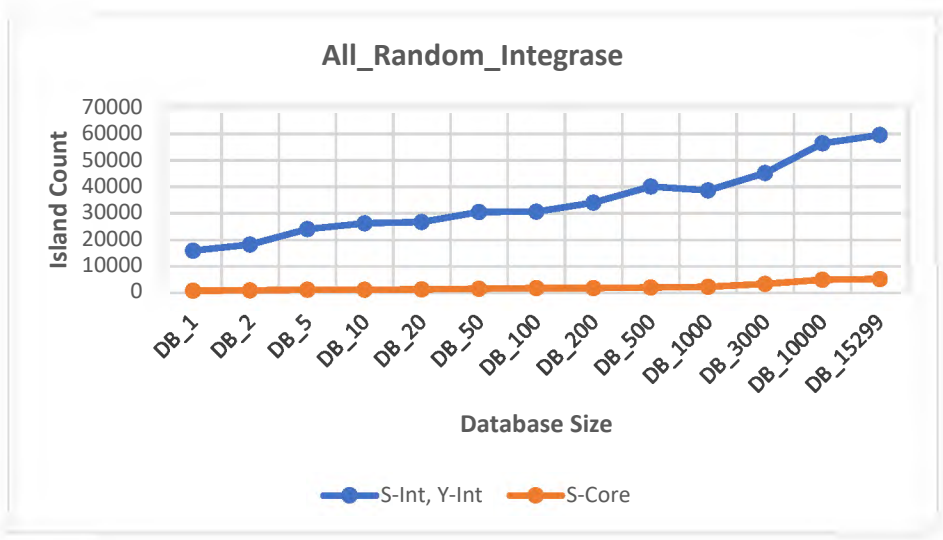


(c)

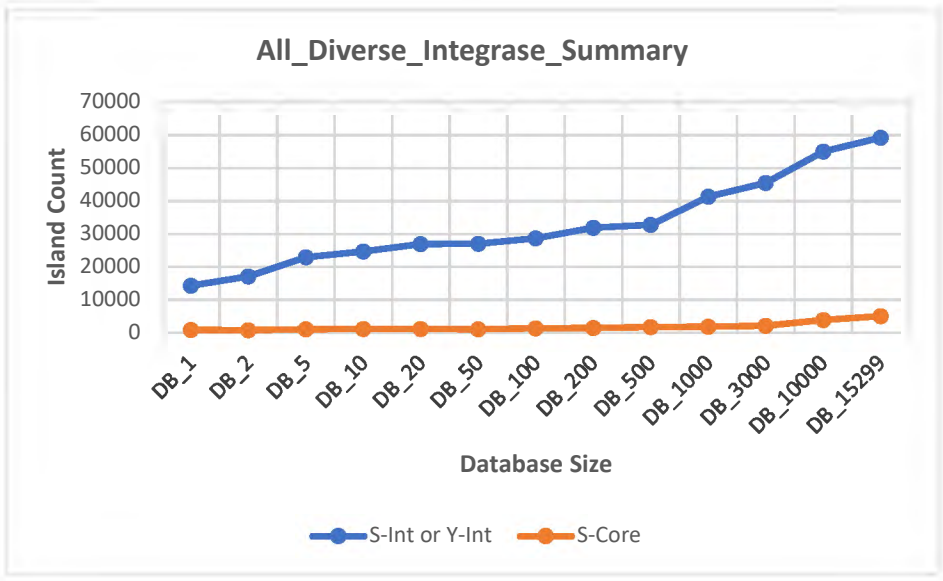
Fig. 3.8 (a, b, c): Island score versus database size. Comparing the performance of three proposed algorithms by differing the input database size and comparing the island scores.

Integrase Summary

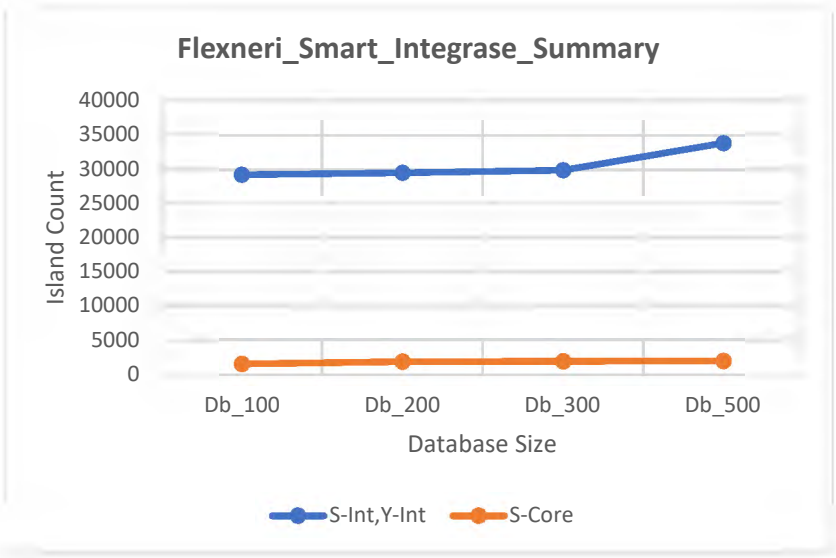
In this evaluation, we focused on the percentage of genomic islands that contained a "good" type of integrase, specifically Y-Int or S-Int. Islands with at least one of these integrase types were counted, while those with neither were excluded. By removing the "bad" islands from the database, the overall number of islands with good integrase increased as the database size decreased. Fig. 3.9 provides a summary of these results.


All_Random_Integrase
Island Count
70000
60000
50000
40000
30000
20000
10000
0
DB_1
DB_2
DB_5
DB_10
DB_20
DB_50
DB_100
DB_200
DB_500
DB_1000
DB_3000
DB_10000
DB_15299
Database Size
S-Int, Y-Int
S-Core

(a)


All_Diverse_Integrase_Summary
Island Count
70000
60000
50000
40000
30000
20000
10000
0
DB_1
DB_2
DB_5
DB_10
DB_20
DB_50
DB_100
DB_200
DB_500
DB_1000
DB_3000
DB_10000
DB_15299
Database Size
S-Int or Y-Int
S-Core

(b)



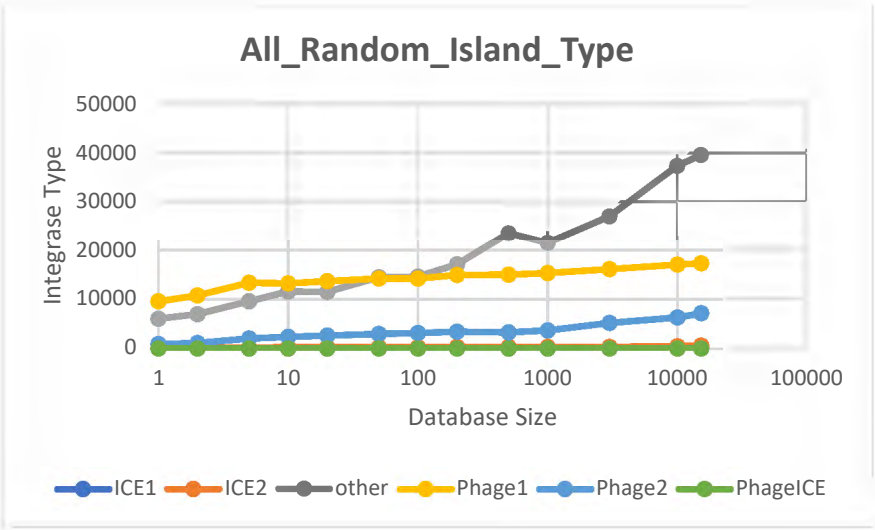
(c)

Fig. 3.9 (a, b, c): Island counts versus island types. Comparing the performance of three proposed algorithms based on island type and island counts.

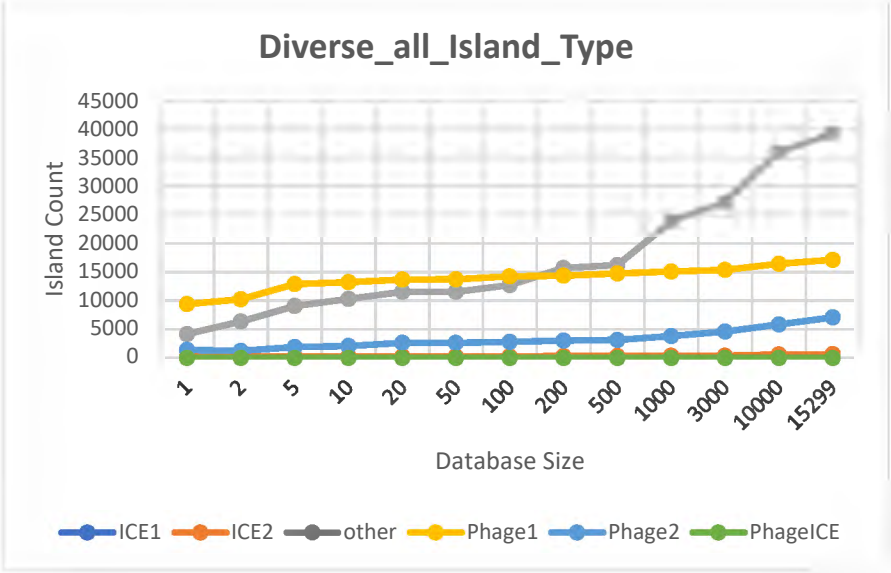
Island Type

In this examination, we focused on investigating the impact of database size on phages, explicitly focusing on the yield of prophages (Phage1, Phage2, PhageICE) and "Other" islands. The study found that prophages generally remained unaffected by database size, while the "Other" category exhibited a significant decrease in yield as the database size decreased. This reduction in yield is likely due to the presence of more false positive islands in this group. For random and diverse graphs, there was a noticeable increase in yield after 500 databases, indicating the presence of bag islands. Additionally, there was a slight increase in yield for the Flexneri species after 300 databases, indicating that they may also be considered bad islands. Overall, these findings suggest that smaller databases, such

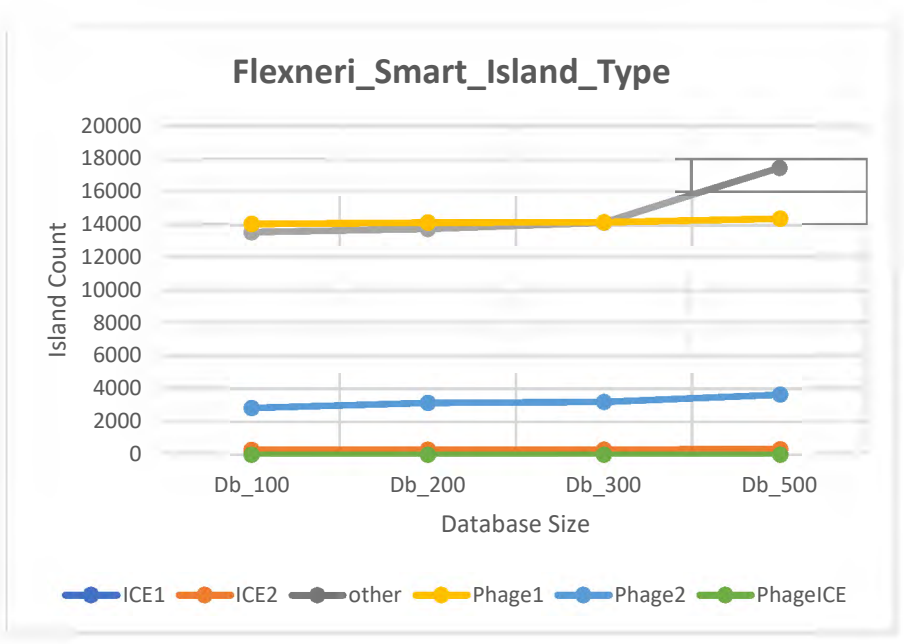
as those capped at 500, are sufficient for this type of island. The results of this study are presented in Fig. 3.10.



(a)



(b)



(c)

Fig. 3.10 (a, b, c): Island counts and database sizes versus phage types. Comparing the performance of three proposed algorithms based on island counts and database sizes versus phages.

3.5 Conclusion

Mobile genomic elements play a crucial role in the evolution and diversity of microbial genomes, and accurate identification of these elements is essential for understanding their function and impact. The TIGER tool is a powerful tool for detecting mobile genomic elements, but it can be slow. Our research group developed four algorithms (randomDB, diverseDB, evalDB, and smartDB) to expedite the TIGER tool's execution time while maintaining accuracy and precision. Our results showed that our algorithms could speed up the MGE discovery process by up to 139-fold in one case. This work did not involve the development of the TIGER tool but rather the development of computational methods to expedite TIGER's execution time. Our research provided a faster and

more efficient way to identify mobile genomic elements that could be applied to a wide range of studies. Specifically, our smartDB algorithm can be incorporated into other applications, such as metagenomic analysis, to expedite MGE discovery and provide new insights into microbial ecology and evolution. Overall, our research represents an important step towards a more efficient and accurate MGE discovery, and we hope it will contribute to advancing research in this field.

CHAPTER 4

VALIDATION OF TRANSPOSON LOCATIONS IN GENOMIC SEQUENCES USING TIGER AND BLAST OUTPUTS AND STATISTICAL ANALYSIS

Transposable elements, commonly called transposons, are DNA sequences that can change their position within a genome. This mobility can significantly affect the overall structure, function, and evolution of genome architecture. Due to their substantial impact on gene regulation and genetic disease development, transposons have been extensively researched. Consequently, accurately identifying and mapping transposons is crucial for understanding their biological ramifications.

Over the years, numerous computational tools have been developed to detect and characterize transposons in genomic sequences, albeit their performance can differ based on input sequence characteristics and underlying algorithms. Among these tools, Target/IGE Retriever (TIGER) can precisely identify transposes-encoding mobile DNAs in a given genome. The current study aimed to examine newly discovered transposons by TIGER in E. coli genomes and validate them by comparisons with published databases. In this work, we discuss the implications of our findings within the bioinformatics community, emphasizing the accuracy and effectiveness of TIGER in transposon discovery. In addition, we employed a statistical study using the bell curve distribution to categorize the matches, thereby assessing the trustworthiness of the transposon locations. Finally, we pinpoint the limitations of our research, adjust the program, and consider

potential applications for comparing different bioinformatics tools while incorporating user-specified parameters. Our research contributes to the ongoing enhancement of methods for detecting transposons by offering critical insights about the performance of TIGER and BLAST in comparison.

4.1 Introduction

Transposons are like pieces in a genetic puzzle that can move around within our DNA. This movement shakes things up in our genome, influencing how it evolves over time, controlling how our genes work, and sometimes bringing about entirely new traits. [42]. Accurate identification of transposon locations is crucial for understanding their functional roles and potential implications in various biological processes.

Several computational tools have been developed to identify transposon locations in genomic sequences, such as TIGER (Target/IGE Retriever) and BLAST (Basic Local Alignment Search Tool) [43]. However, there is a need for a systematic approach to compare and validate the transposon locations identified by these tools.

TIGER is a computational tool that accurately identifies transposable elements in genomes and precisely maps their location to the nucleotide level. It utilizes a comparative genomic approach and a ping-pong BLAST method, assuming that the integration module of transposable elements is cohesive [33]. TIGER has been applied to various organisms and has demonstrated promising results in detecting transposons in complex genomes. We utilized TIGER on the transposon discovery process for E. coli genomes for four different species on

Pittsburgh Super Computing Center (PSC) state-of-the-art supercomputers. Basic Local Alignment Search Tool (BLAST) is a well-known sequence alignment tool that compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of the matches [44]. BLAST has been employed for transposon detection by comparing genomic sequences to curated databases of known transposons, such as Repbase [45] and ISfinder [46].

Repbase is an immense database of repetitive elements collected from different genomes of eukaryotes. It is a tool that helps researchers label and locate repeating sequences in genomic data, including transposable elements. Since its founding in 1990, Repbase has served as a major resource for research on how repeated sequences influence the structure and function of the genome [45].

In contrast, ISfinder is a specialized database that focuses only on prokaryotic insertion sequences (IS). It makes an in-depth understanding of the role of ISs in genome dynamics easier by offering a selective set of well-annotated ISs. A comprehensive toolset for any individual researching these mobile genetic components in bacteria and archaea is provided by the ISfinder database, which enables researchers to categorize, annotate, and compare IS elements [46].

This study used BLAST to search for transposons in genomes and compare the results with TIGER-generated data.

To enable the comparison of transposons, we developed new software in Python programming language to compare the transposon coordinates identified by TIGER and BLAST output files. The program extracts the left and right coordinates of the transposon from the TIGER output file and searches the BLAST output for

matches that are either exact or within a user-defined base pair range. By identifying matching transposons, the program can measure the level of agreement between TIGER and BLAST output files and determine the true transposon location.

This study proposed a novel method to compare and validate transposon locations identified by TIGER and BLAST tools using a statistical approach based on the bell distribution curve. Our method compared the left and right coordinates of the transposons. It categorized the matches based on the mean and standard deviation of the differences, making it possible to determine the quality of the matches and prioritize the transposons for further analysis.

4.2 Materials and Methods

Data Collection

The gathering of primary data was carried out using smartDBs, discussed more thoroughly in Chapter 3. These databases, robust in their capabilities, were utilized to generate output files for both TIGER and BLAST, providing comprehensive information pertaining to the specific locations of transposons within a given genome. This information formed the crux of our study and was prepared meticulously for an exhaustive examination and interpretation.

To transform the raw data into an analyzable format, we implemented a series of preprocessing steps for each output file (Flowchart 1).

- *Accessing the TIGER Output Directory:*

Upon running the TIGER tool, an assortment of directories and files were generated. However, for the purpose of this research, not all these files were of

relevance. Thus, we carefully navigated through these numerous files, directing our focus only on those few that were critical to our analysis.

- *Identifying Core Files:*

In the sea of files produced by TIGER, two specific files stood out as paramount to our analysis - "genome.IS.nonoverlap.gff" and "genome.IS.blast.tile". These files were targeted for our analysis due to their wealth of relevant information. This is illustrated in Fig. 4.1.

```
# specify the filenames to extract
filenames = ['genome.IS.nonoverlap.gff', 'genome.IS.blast.tile']
```

Fig. 4.1: Specifying files.

- *Standardizing File Formats:*

To facilitate seamless data handling and analysis, we converted the 'gff' and 'tile' files into a more universally recognized and convenient 'CSV' format. This step also ensured consistency across our datasets, allowing for more effective data manipulation and interpretation shown in Algorithm 5.

Algorithm 5 StandardizingFileFormats

Require: Parent folder path P
Ensure: Converted CSV files within subfolders of P

Let D be the set of all directories and subdirectories within P.
for each directory $d \in D$ **do**
    Let F be the set of all files within d.
    for each file $f \in F$ **do**
        if extension of f is '.gff' or '.tile' **then**
            csvName = name of f without its extension + '.csv'
            Open f for reading as *inputFile* and *csvName* for writing as *outputFile*.
            Write the header row ['contig name', 'software', ...] to *outputFile*.
            for each line l in *inputFile* **do**
                if l starts with or l is blank **then**
                    Continue to the next line.
                end if
                fields = split l by spaces and take the first 9 elements.
                Write *fields* to *outputFile*.

```
            end for
            Close inputFile and outputFile.
            Print message indicating the conversion of f is complete.
        end if
    end for
end for
```

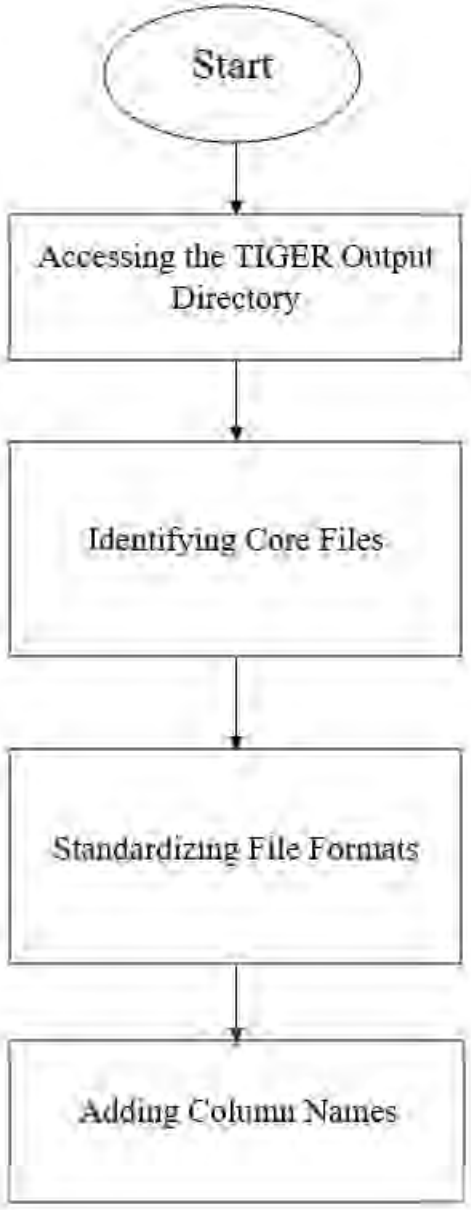
- *Adding Column Names:*

Our output files lacked column names in their original format, presenting potential hurdles for effective data interpretation. To enhance clarity and optimize data for our analytical requirements, we took the initiative to manually insert appropriate column headers into each 'CSV' file. This step further standardized our data and made it easier to understand, enhancing the efficiency of our subsequent analyses as indicated in Fig. 4.3.

```
# Write the header row to the CSV file
csv_writer.writerow(['contig_name', 'software', 'IS', 'LCOR', 'RCOR', 'supporting',
'orientation1', 'orientation2', 'INFO'])
```

Fig. 4.3: Specifying files.

Flowchart 1: Python Program for Data Collection:



Python Program for Data Comparison

A Python program was developed to compare the left and right coordinates of the transposons in the TIGER and BLAST output files. The program picks the first left and right coordinates from the TIGER output and searches for an exact match or a match within a user-defined base pair range (+10/-10) in the BLAST output shown in Algorithm 6. The objective is to find how much the two outputs validate each other and identify the true transposon locations (see Flowchart 2).

Algorithm 6 matchcord

Require: *search dir, foldername, tol*
Ensure: -1
Read *blast_raw_new.csv* from the directory *search dir/foldername* into DataFrame *df*1
Read *genome.IS.nonoverlap.csv* from the directory *search dir/foldername* into DataFrame *df*2
Initialize empty DataFrames: *data frame blast* and *dataframe tiger*
for each index *idx,* and elements *data*1, *data*3 in *df*1['*LCOOR*(*query*)'], *df*1['*RCOOR*(*query*)'] **do**
  for each index *idx*1, and elements *data*2, *data*4 in *df*2['*LCOR*'], *df*2['*RCOR*'] **do**
    if *data*1 = *data*2 and *data*3 = *data*4 **then**
      Append row *idx* of *df*1 to *dataframe blast*
      Append row *idx*1 of *df*2 to *dataframe tiger*
    else if $|data1 - data2| \leq tol$ and $|data3 - data4| \leq tol$ **then**
      Append row *idx* of *df*1 to *dataframe blast*
      Append row *idx*1 of *df*2 to *dataframe tiger*
    end if
  end for
end for
Save *dataframe blast* to *search dir/foldername/BLAST matched raw.csv*
Save *dataframe tiger* to *search dir/foldername/TIGER matched.csv*
return -1

A Table containing the differences between the left and right coordinates and the lengths of the transposons in the TIGER and BLAST output files is generated. The Table includes the contig name, left coordinate difference, right

coordinate difference, and length difference for each match found in the previous step.

The differences between the left and right coordinates and the lengths of the transposons are analyzed using a bell distribution curve, also known as a normal distribution or Gaussian distribution. This is a statistical method often used when a dataset is large, and the data points are spread out in a pattern where most data points are close to the average (mean), and the rest taper off equally in both directions [47].

The bell curve, also known as the normal distribution, possesses a symmetrical shape where the mean, median, and mode align perfectly. The central point of this curve, referred to as the mean (μ), represents the most probable value with the highest likelihood, while the dispersion of data points from the mean is indicated by the spread around it, known as the standard deviation (∂) [48]. In this particular context, calculations of the mean (μ) and standard deviation (∂) were conducted for each of the three categories: left coordinate, right coordinate, and length. The mean is essentially the arithmetic average of the data points, whereas the standard deviation quantifies the extent to which the data points deviate from the mean as shown in Algorithm 7.

Algorithm 7 LabelGeneration

1: **function** distri(*dataframe*)
2: $\mu \leftarrow$ mean(*dataframe*)
3: $\sigma \leftarrow$ std(*dataframe*)
4: Calculate boundaries:
5: $ex_{ub} \leftarrow \mu + \sigma$
6: $ex_{lb} \leftarrow \mu - \sigma$
7: $good_{ub} \leftarrow \mu + 2\sigma$
8: $good_{lb} \leftarrow \mu - 2\sigma$
9: $avg_{ub} \leftarrow \mu + 3\sigma$

```
    avg_lb ← μ − 3σ return ex_ub, ex_lb, good_ub, good_lb, avg_ub, avg_lb
end function
function labelData(data, boundaries)
    if boundaries[0] > data > boundaries[1] then return "excellent"
    else if boundaries[2] > data > boundaries[3] then return "good"
    else if boundaries[4] > data > boundaries[5] then return "average"
    elsereturn "Bad Match"
    end if
end function
function diff label(a)
    Initialize: list lcor, list rcor, list len as empty lists
    for each lcor, rcor, length in a do
        append labelData(lcor, distri(a['LCOR'])) to list lcor
        append labelData(rcor, distri(a['RCOR'])) to list rcor
        append labelData(length, distri(a['len'])) to list len
    end forreturn list lcor, list rcor, list len
end function
```

Following these computations, the data points were graphed on a normal distribution curve, enabling a visual representation of the distribution of values within specific ranges of differences as shown in Algorithm 8. For instance, one can observe the proportion of data falling within one standard deviation from the mean, two standard deviations from the mean, and so forth, as illustrated in Flowchart 3.

Algorithm 8 Categorization

```
function distri(dataframe)
    μ ← mean(dataframe)
    σ ← std(dataframe)
    ex upbound ← μ + σ
    ex lowbound ← μ − σ
    good upbound ← μ + 2σ
    good lowbound ← μ − 2σ
    avg upbound ← μ + 3σ
    avg lowbound ← μ − 3σ
return ex upbound, ex lowbound,
    good upbound, good lowbound,
    avg upbound, avg lowbound
end function
```

Based on the calculated mean and standard deviation, the matches were then categorized. The categorization was based on how close the values were to the meaning for each category. The closer the values were to the mean, within the range of one standard deviation, the better the match.

An "Excellent Match" is defined as all three categories, left coordinate, right coordinate, and length, having values within the range of the mean plus or minus one standard deviation ($\mu \pm 1\partial$). In other words, all the values were very close to the average, meaning the match is very good.

A "Good Match" is when two out of the three categories fall within this range. This means two of the three values were close to the average, so the match was considered good but not excellent.

A "Bad Match" is when none of the categories have values within this range. This means all the values were significantly different from the average, so the match was considered poor.

We applied our categorization method to various randomly selected genomes to demonstrate its adaptability and usefulness. The following specific example shows the efficacy of the algorithm:

In one such analyzed genome, our algorithm categorically identified fourteen transposons as "Excellent Matches," eight as "Bad Matches," and two as "Good Matches" (see Fig. 4.7). This experimental information provided an understanding of the range and distribution of transposon matches inside a single genome in addition to confirming the method's accuracy.

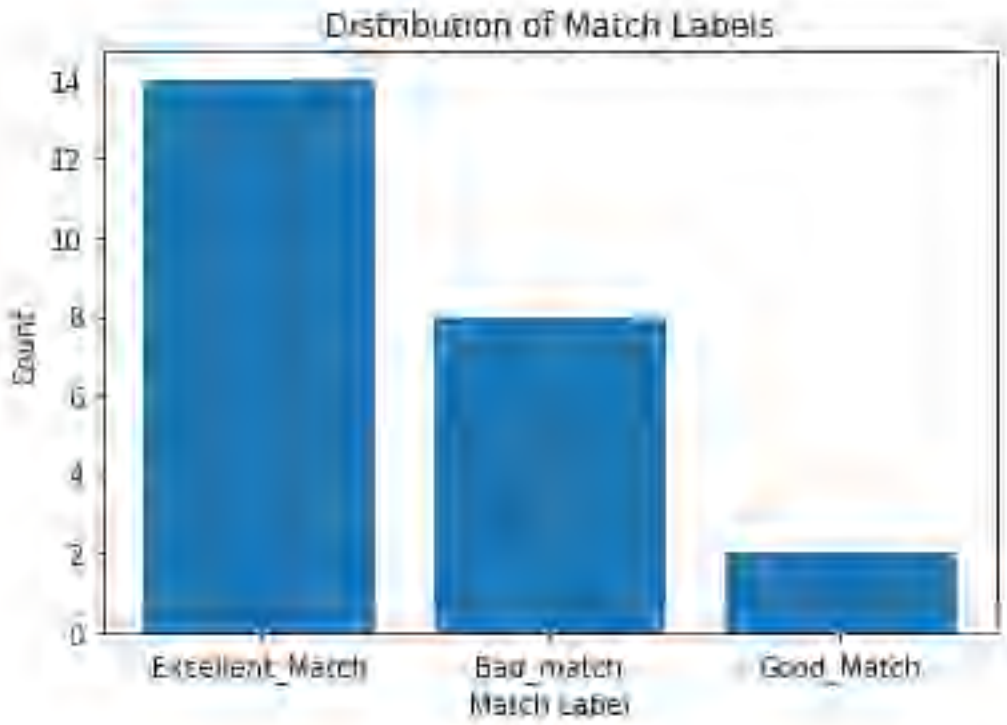
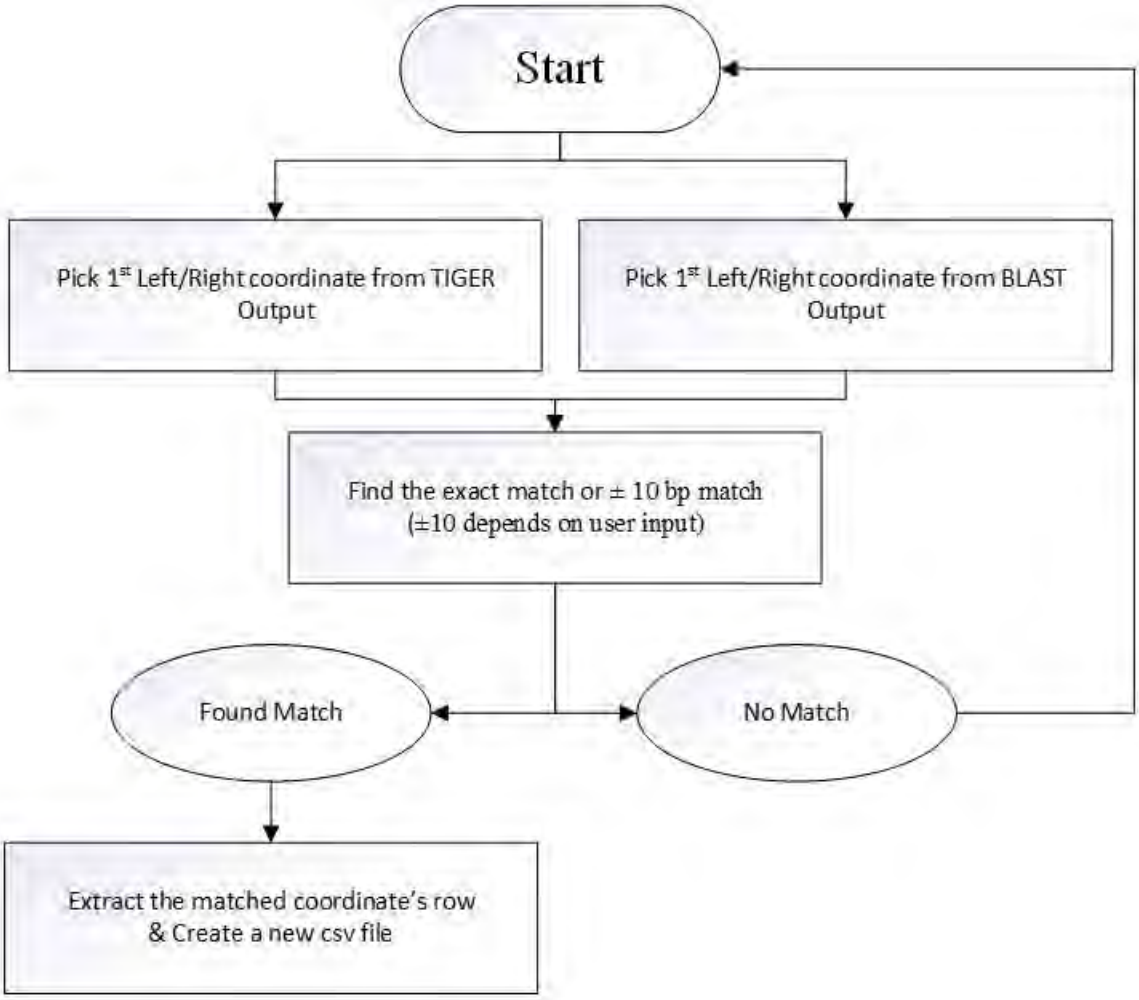
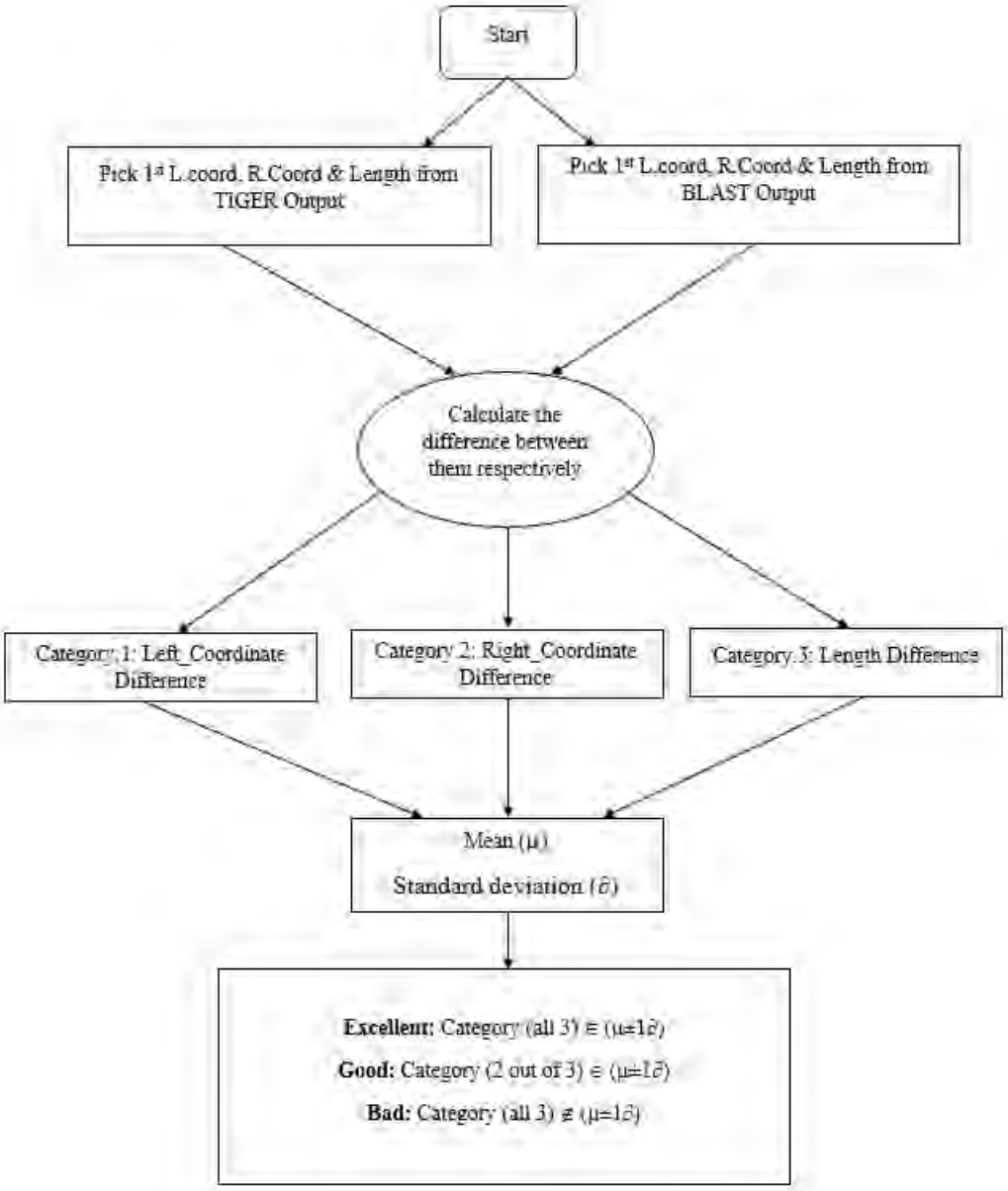


Fig. 4.7: Example of Transposon Match Categorization in a Single Randomly Selected Genome.

Flowchart 2: Python Program for Data Comparison.



Flowchart 3: Statistical Analysis Using Bell Distribution Curve



4.3 Results and Discussion

While validating and comparing the transposon locations identified by the TIGER and BLAST tools, our analysis revealed intriguing findings. First, the BLAST tool seemed to be more liberal in marking transposons compared to its counterpart, TIGER. This raised many questions - was it due to the varying methodologies the two tools employ, or was it a matter of sensitivity and specificity in identifying transposons? Or could it be down to the different benchmarks they use to identify transposons? These questions warrant further investigation and will undoubtedly deepen our understanding of these two pivotal tools.

Nevertheless, an interesting pattern emerged when we applied our algorithm to the results from these two tools. We managed to sift out the 'true' transposons - unanimously identified by both tools. This cross-verification significantly bolstered the reliability of our findings, as it threw out potential biases or errors that individual tools might have introduced. This approach, we believe, provides a more robust validation of data and could be a game-changer for future studies.

Subsequently, we delved deeper into our analysis, examining the differences between the left and right coordinates and the lengths of the transposons flagged by TIGER and BLAST. To navigate through this sea of data, we sought the help of the bell distribution curve, a powerful statistical tool for understanding data distributions. This gave us a clear picture of the variations in our data and helped us assess how much they strayed from the mean.

Using this information, we grouped the differences into three distinct categories: excellent, good, and bad matches. We used the mean and standard

deviation of the differences as yardsticks for this classification. This classification process provided us with a more granular understanding of the level of agreement between TIGER and BLAST in identifying transposon locations. It also shed light on the reliability and consistency of these tools, offering valuable insights for future genomic studies.

To sum up, our findings underscore the importance of rigorous validation of genomic data and propose a comprehensive approach to achieving it. By harnessing the power of statistical analysis and cross-validation, we significantly enhanced the reliability of our results and gained a deeper understanding of the complex genomic landscape.

CHAPTER 5

COMPREHENSIVE EVALUATION OF TIGER AND BLAST OUTPUTS: ANALYZING DIRECT REPEATS AND ASSESSING BLAST HIT QUALITY

In the backdrop of our preceding chapter on comparing TIGER and BLAST output files, we extend our exploration into the field of comparative genomics. This chapter aims to meticulously analyze TIGER and BLAST outputs, focusing on direct repeats within insertion sequences and thoroughly assessing BLAST hit quality. The comprehensive approach employed here augment our understanding of sequence alignment results, significantly contributing to ongoing research in genomics and bioinformatics.

By analyzing the TIGER and BLAST outputs, we aimed to comprehensively understand direct repeats in insertion sequences and assess the quality of BLAST hits. This integrated approach will provide valuable insights into the functional implications of direct repeats and aid in distinguishing high-quality BLAST hits from subpar matches. In addition, the findings from this analysis will contribute to our knowledge of genomic dynamics, mobile genetic elements, and their impact on bacterial genomes.

5.1 Direct Repeat (DR)

DNA sequences known as direct repeats (DRs) can be small nucleotides or larger pieces found in parallel or scattered throughout the genome [49]. Their importance in genome architecture cannot be overstated. DRs serve as key structural elements in various genetic phenomena, such as DNA replication, repair,

and recombination [50]. Specifically, they act as landmarks for transposase binding, thereby facilitating the mobility of transposable elements within the genome [51].

Moreover, DRs play a pivotal role in the regulation of gene expression. Their presence upstream or downstream of coding sequences can modulate transcriptional activity and contribute to cellular differentiation and adaptability [52]. Furthermore, the study of direct repeats provides crucial insights into the mechanisms of bacterial pathogenesis, as some direct repeats function as regulatory switches for virulence factors [53].

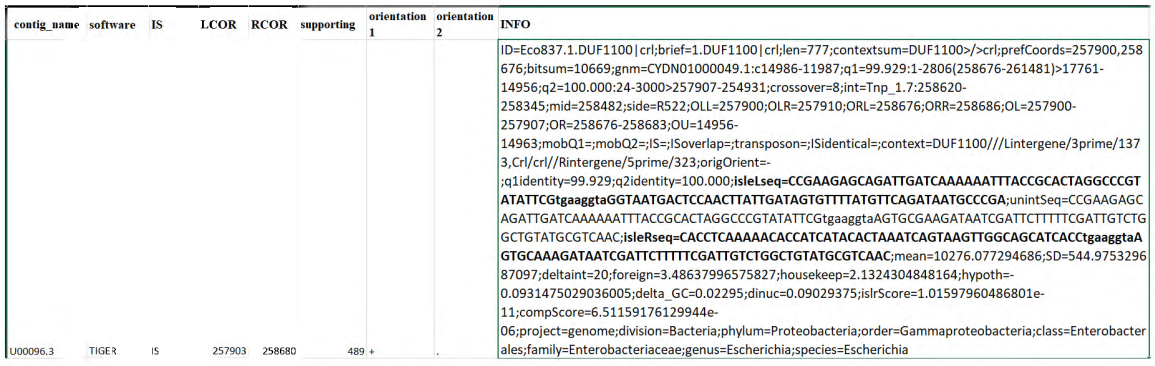
5.2 Methodology

The methodology encompasses two primary steps:

Analysis of TIGER Output

- *Extraction of Sequences:*

Our methodology began with a thorough review of the data obtained from the TIGER tool. As an advanced tool for studying genome structure, TIGER provides a multitude of data that form the basis of our research. A comprehensive genomic map of the samples was constructed, which included crucial details such as contig names, the software used, orientation, and more. Among this data, our primary focus was on extracting the sequences flanking the Insertion Sequences (IS), known as isleLseq and isleRseq as indicated in Fig. 5.1.



contig_name	software	IS	LCOR	RCOR	supporting	orientation 1	orientation 2	INFO
U00096.3	TIGER	IS	257903	258680	489	+	.	ID=Eco837.1.DUF1100|crl;brief=1.DUF1100|crl;len=777;contextsum=DUF1100>/>crl;prefCoords=257900,258676;bitsum=10669;gnm=CYDN01000049.1:c14986-11987;q1=99.929:1-2806(258676-261481)>17761-14956;q2=100.000:24-3000>257907-254931;crossover=8;int=Tnp_1.7:258620-258345;mid=258482;side=R522;OLL=257900;OLR=257910;ORL=258676;ORR=258686;OL=257900-257907;OR=258676-258683;OU=14956-14963;mobQ1=;mobQ2=;IS=;ISoverlap=;transposon=;ISidentical=;context=DUF1100///Lintergene/3prime/1373,Crl/crl//Rintergene/5prime/323;origOrient=-;q1identity=99.929;q2identity=100.000;**isleLseq=CCGAAGAGCAGATTGATCAAAAAATTTACCGCACTAGGCCCGTATATTCGtgaaggtaGGTAATGACTCCAACTTATTGATAGTGTTTTATGTTCAGATAATGCCCGA**;unintSeq=CCGAAGAGCAGATTGATCAAAAAATTTACCGCACTAGGCCCGTATATTCGtgaaggtaAGTGCGAAGATAATCGATTCTTTTTCGATTGTCTGGCTGTATGCGTCAAC;**isleRseq=CACCTCAAAAACACCATCATACACTAAATCAGTAAGTTGGCAGCATCACCtgaaggtaAGTGCAAAGATAATCGATTCTTTTTCGATTGTCTGGCTGTATGCGTCAAC**;mean=10276.077294686;SD=544.975329687097;deltaint=20;foreign=3.48637996575827;housekeep=2.1324304848164;hypoth=-0.0931475029036005;delta_GC=0.02295;dinuc=0.09029375;islrScore=1.01597960486801e-11;compScore=6.51159176129944e-06;project=genome;division=Bacteria;phylum=Proteobacteria;order=Gammaproteobacteria;class=Enterobacterales;family=Enterobacteriaceae;genus=Escherichia;species=Escherichia

Fig. 5.1: TIGER Output Sample.

- *Initial Comparison with ISFinder Database:*

Upon the extraction of isleLseq and isleRseq sequences, the first step in our process involved comparing these sequences with the established IS sequences in the ISFinder database. The ISFinder database, a comprehensive repository of documented Insertion Sequences, proved to be a valuable resource in our initial analysis. The goal was to locate any potential matches between the inverted repeats of the IS elements from the database and our extracted isleLseq and isleRseq sequences from the TIGER output shown in Algorithm 9.

Algorithm 9 Initial Check

Require: *left, right, new sequence*

Ensure: Dictionary with keys: 'left', 'right', 'under left', 'under right', 'split left', 'split right', 'flag'

Extract the first 10 base pairs from *new sequence* as $first_{10}$ = *new sequence*$[1:10]$

Extract the last 10 base pairs from *new sequence* as $last_{10}$ = *new sequence*$[-10:]$

$left_{tmp} \leftarrow$ convert *left* to uppercase

$first_{10\ tmp} \leftarrow$ convert $first_{10}$ to uppercase

Initialize *underlined left* $\leftarrow \emptyset$

Initialize *matched left* $\leftarrow \emptyset$

$right_{tmp} \leftarrow$ convert *right* to uppercase

$last_{10\ tmp} \leftarrow$ convert $last_{10}$ to uppercase

Initialize *underlined right* $\leftarrow \emptyset$

Initialize *matched right* $\leftarrow \emptyset$

flag ← *False*
if $first_{10\ tmp} \in left_{tmp}$ **then**
    index ← position of $first_{10\ tmp}$ in $left_{tmp}$
    Format *left* to underline the matched sequence and assign to *matched left*
    Extract the underlined sequence from *left* and assign to *underlined left*
end if
if $last_{10\ tmp} \in right_{tmp}$ **then**
    index ← position of $last_{10\ tmp}$ in $right_{tmp}$
    Format *right* to underline the matched sequence and assign to *matched right*
    Extract the underlined sequence from *right* and assign to *underlined right*
else
    flag ← *True*
end if
    return Dictionary with the generated values

- *Analysis of Sequence Matches and Direct Repeat Identification:*

If the sequences matched with entries in the ISFinder database, the process moved forward to the identification and counting of Direct Repeats (DRs) within the sequences. These DRs, represented by lowercase letters within the sequences, provided a key understanding of the behavior and potential influence of Insertion Sequences within the genomic structure shown in Algorithm 10.

Algorithm 10 extract and list common lowercase

Require: *matched left, matched right, underlined left, underlined right*
Ensure: *common count, common bases str*
Remove underlined sequences from *matched left* and *matched right* to get *lowercase left* and *lowercase right* respectively.
Extract only lowercase letters from *lowercase left* and *lowercase right*.
Initialize an empty list *common bases*
Initialize a counter *common count* ← 0
for each base in *lowercase left* **do**
    if base exists in *lowercase right* **then**
        Add base to *common bases*
        Remove the first occurrence of base from *lowercase right*
    end if
end for
common count ← length of *common bases*
Convert *common bases* list to a string, *common bases str*
    return *common count, common bases str*

- *Reverse Complement Process for Non-matching Sequences:*

If no match was found during the initial comparison with the ISFinder database, the sequences were then reverse-complemented. The reverse complement of the isleLseq and isleRseq sequences was again compared with the ISFinder database. If a match was found this time, the Direct Repeats within these sequences were identified and quantified shown in Algorithm 11.

Algorithm 11 ReverseComplementCheck

Require: *left, right, new sequence*
Ensure: Dictionary with keys: 'left', 'right', 'under left', 'under right', 'split left', 'split right'
1: *concatenated* ← *left* + *right*
2: *reversed comp* ← reverse complement of *concatenated*
3: *split left*, *split right* ← split *reversed comp* into two parts
4: Extract the first 10 base pairs from *new sequence* as $first_{10}$ = *new sequence*[1 : 10]
5: Extract the last 10 base pairs from *new sequence* as $last_{10}$ = *new sequence*[−10 :]
6: $split\ left_{tmp}$ ← convert *split left* to uppercase
7: $first_{10\ tmp}$ ← convert $first_{10}$ to uppercase
8: Initialize *underlined left* ← ∅
9: Initialize *matched left* ← ∅
10: $split\ right_{tmp}$ ← convert *split right* to uppercase
11: $last_{10\ tmp}$ ← convert $last_{10}$ to uppercase
12: Initialize *underlined right* ← ∅
13: Initialize *matched right* ← ∅
14: **if** $first_{10\ tmp} \in split\ left_{tmp}$ **then**
15: *index* ← position of $first_{10\ tmp}$ in $split\ left_{tmp}$
16: Format *split left* to underline the matched sequence and assign to *matched left*
17: Extract the underlined sequence from *split left* and assign to *underlined left*
18: **end if**
19: **if** $last_{10\ tmp} \in split\ right_{tmp}$ **then**
20: *index* ← position of $last_{10\ tmp}$ in $split\ right_{tmp}$
21: Format *split right* to underline the matched sequence and assign to *matched right*
22: Extract the underlined sequence from *split right* and assign to *underlined right*
23: **else**
24: Print "DR None", *left, right*
25: **end if**
return Dictionary with the generated values

- *Python Script Implementation for Efficient Analysis:*

To streamline the processes of comparison, identification, and quantification, we implemented a Python script. This script was custom-built to parse and filter the genomic data and was particularly focused on handling the isleLseq and isleRseq sequences and their direct repeats.

In summary, our methodology enabled comprehensive extraction, comparison, reverse complementation, identification, and quantification of critical elements of the Insertion Sequences within the genomic structures. Through the use of advanced tools and custom scripts, we were able to achieve a detailed analysis of the data, leading to substantial insights into the behavior of Insertion Sequences across various genomes as seen in Fig. 5.5.

A	B	C	D	E	F	G	H	J	K	L	M	N	O	P	Q
Unnamed: 0	contig_name	software	IS	LCOR	RCOR	supporting	orientation1	INFO	len	IS_name	isleLseq	isleRseq	common_lowercase_count	Operation_applied	common_lowercase_bases
0	U00096.3	TIGER	IS	15384	16734	249	-	ID=Eco837	1350	IS186B	ACGGAGG	ACGTTAA	10	no_change	gggagtatcc
1	U00096.3	TIGER	IS	257903	258680	489	+	ID=Eco837	777	IS1F	CCGAAGA	CACCTCA	8	reverse_complemented	taccttca
2	U00096.3	TIGER	IS	381256	382593	295	-	ID=Eco837	1337	IS2	TGGTGCC	TCACTTAT	5	reverse_complemented	aattc
3	U00096.3	TIGER	IS	391707	392969	64	+	ID=Eco837	1262	IS3	TTAACTCC	CTGAGAG	3	reverse_complemented	atc
4	U00096.3	TIGER	IS	574591	575790	91	+	ID=Eco837	1199	IS5	GGTAAAC	GCTCCAG	4	no_change	ttag
5	U00096.3	TIGER	IS	608004	609354	53	+	ID=Eco837	1350	IS186B	TGAGTTTA	ACCGAGG	10	no_change	gggataatcc
6	U00096.3	TIGER	IS	687849	689048	214	-	ID=Eco837	1199	IS5	AACAAAC	TGAAATG	4	no_change	ctaa
7	U00096.3	TIGER	IS	1049773	1050550	431	-	ID=Eco837	777	IS1F	ATTTCACA	AACCTCA	9	reverse_complemented	gacaatacg
8	U00096.3	TIGER	IS	1094239	1095503	4	+	ID=Eco837	1264	IS3	TAAGGTG	CTGAGAG	1	reverse_complemented	c
9	U00096.3	TIGER	IS	1294411	1295411	157	-	ID=Eco837	1000	ISCro3	AAGCGAA	CCCGTTAT	10	reverse_complemented	cccagaaggg
10	U00096.3	TIGER	IS	1299496	1300696	384	+	ID=Eco837	1200	IS5	AATAATTG	GAAATGA	4	no_change	ctaa
11	U00096.3	TIGER	IS	1396041	1397241	346	-	ID=Eco837	1200	IS5	CTCTGAAA	GCTCCAG	4	no_change	ttag
12	U00096.3	TIGER	IS	1503161	1504910	67	-	ID=Eco837	1749	IS609	CGGAGGC	AACCGCG	0	reverse_complemented	
13	U00096.3	TIGER	IS	1978498	1979275	383	+	ID=Eco837	777	IS1F	GATTTTCA	CACCTCA	8	reverse_complemented	catttatg
14	U00096.3	TIGER	IS	2066156	2067356	155	+	ID=Eco837	1200	IS5	AACTTGTA	GCTCCAG	4	no_change	ctag
15	U00096.3	TIGER	IS	2170169	2171430	421	-	ID=Eco837	1261	IS3	CCAAAAA	CTGAGAG	3	reverse_complemented	gtc
16	U00096.3	TIGER	IS	2288916	2290116	399	+	ID=Eco837	1200	IS5	ATCAACTC	GCTCCAG	4	no_change	ataa
17	U00096.3	TIGER	IS	2514270	2515620	337	+	ID=Eco837	1350	IS186B	TCTGCGTC	ACCGAGG	10	no_change	ggataattcc
18	U00096.3	TIGER	IS	3186091	3187426	273	+	ID=Eco837	1335	IS2	TTAAAATT	GGGCTTG	4	no_change	atgt
19	U00096.3	TIGER	IS	3365553	3366753	78	+	ID=Eco837	1200	IS5	GTTAGCCA	GCTCCAG	4	no_change	ttag
20	U00096.3	TIGER	IS	3583423	3584200	290	+	ID=Eco837	777	IS1F	TACAAGA	TCGGGCA	9	no_change	gacattaaa
21	U00096.3	TIGER	IS	3652034	3653233	265	+	ID=Eco837	1199	IS5	CAAGGTTT	GCTCCAG	4	no_change	ttag
22	U00096.3	TIGER	IS	3720631	3722079	2	-	ID=Eco837	1448	IS150	GTATTCAC	ATATTAG	1	reverse_complemented	t
23	U00096.3	TIGER	IS	4498178	4499515	48	+	ID=Eco837	1337	IS2	GGGTGAT	TGTAATG	5	no_change	ccttg

Fig. 5.5: Final outcome from Analysis of TIGER output.

Analysis of BLAST Output

In this step, our focus shifted to the BLAST outputs indicated in Fig. 5.6. Extending from our initial comparison, we introduce two metrics, delta termini and delta internal, to quantify the quality of the hits.

- *Extraction of BLAST Output Information*

First, we extracted the necessary information from the BLAST output, including the sequence's ID, the start and end points of the alignment on the query and subject sequences, and the length of the ideal match.

Contig	IS	Percentage	Len of hit	number of mismatches	number of gap openings	LCOOR (query)	RCOOR (query)	LCOOR (subject)	RCOOR (subject)	Expected value	Bit Score	Len of Contig	IS from ISFinder (Kbs)
U00096.3	IS609	97.654	1748	41	0	1503161	1504908	1	1748	0	3001	4641652	1748
U00096.3	IS150	99.861	1444	0	2	3720633	3722075	1	1443	0	2654	4641652	1443
U00096.3	IS103	99.792	1444	1	2	3720633	3722075	1	1443	0	2649	4641652	1443
U00096.3	IS4	100	1426	0	0	4502090	4503515	1426	1	0	2634	4641652	1426
U00096.3	IS186B	99.925	1339	0	1	15390	16728	1	1338	0	2466	4641652	1338
U00096.3	IS186B	99.925	1339	0	1	608010	609348	1	1338	0	2466	4641652	1338
U00096.3	IS186B	99.925	1339	0	1	2514276	2515614	1	1338	0	2466	4641652	1338
U00096.3	IS421	99.776	1342	0	1	15390	16728	1	1342	0	2459	4641652	1342
U00096.3	IS421	99.776	1342	0	1	608010	609348	1	1342	0	2459	4641652	1342
U00096.3	IS421	99.776	1342	0	1	2514276	2515614	1	1342	0	2459	4641652	1342
U00096.3	IS2	100	1331	0	0	381260	382590	1	1331	0	2459	4641652	1331
U00096.3	IS2	100	1331	0	0	2996361	2997691	1331	1	0	2459	4641652	1331
U00096.3	IS2	100	1331	0	0	2068941	2070271	1331	1	0	2459	4641652	1331
U00096.3	IS2	100	1331	0	0	1467910	1469240	1331	1	0	2459	4641652	1331
U00096.3	IS2	100	1331	0	0	3186096	3187426	1	1331	0	2459	4641652	1331
U00096.3	IS2	100	1331	0	0	4498181	4499511	1	1331	0	2459	4641652	1331
U00096.3	IS2	100	706	0	0	1650843	1651548	626	1331	0	1304	4641652	1331
U00096.3	IS186A	99.628	1343	3	2	15388	16730	1	1341	0	2451	4641652	1341
U00096.3	IS186A	99.628	1343	3	2	608008	609350	1	1341	0	2451	4641652	1341
U00096.3	IS186A	99.628	1343	3	2	2514274	2515616	1	1341	0	2451	4641652	1341

Fig. 5.6: BLAST Output Sample.

- *Calculation of Delta Termini (Δ Termini)*

After the extraction, we calculated the delta termini, which measures the difference at the end of sequences shown in Algorithm 12. To do this:

$$\Delta\ Termini = (S_{subject} - S_{query}) + (E_{subject} - E_{query})$$

Where,

$S_{subject}$ is the starting point of the alignment on the subject sequence.

S_{query} is the start point of the alignment on the query sequence.

$E_{subject}$ is the endpoint of the alignment on the subject sequence.

E_{query} is the endpoint of the alignment on the query sequence.

This equation calculates:

a. The sum of the two differences by subtracting the start point of the query alignment from the start point of the subject alignment.

b. Subtracting the endpoint of the query alignment from the endpoint of the subject alignment

c. These two differences are then summed to yield the final result.

d. If the result is zero, it indicates a perfect match with the ideal alignment. Any deviation from zero suggests an imperfect or incomplete match.

Algorithm 12 DelterminiCalculation

Require:
dataframe
Ensure: *label, del termini*
1: Let *string* ← *dataframe*
2: Use regular expressions to search for patterns in *string*
3: *sum match* ← match pattern *sum* = $(n_1 - n_2 : n_3 - n_4)$ where $n_i \in \mathbb{N}$
4: *islen match* ← match pattern *islen* = n where $n \in \mathbb{N}$
5: Extract values from matches:
6: *sum value* ← value from *sum match* (or None if no match)
7: *islen value* ← value from *islen match* (or None if no match)
8: Split *sum value* by "-" to get *sum values*
9: For each value in *sum values*, split by ":"
10: Calculate *del termini* using the formula:
11: *del termini* = $1 - n_1 + n_3 -$ *islen value*
12: **if** *del termini* = 0 **then**
13: *label* ← "Complete"
14: **else**
15: *label* ← "Incomplete"
16: **end if**
return *label, del termini*

- *Calculation of Delta Internal (Δ Internal)*

Then, finally, we calculated the delta internal shown in Algorithm 13, which measured the difference within sequences. This calculation was performed only for matches with a (+) symbol in the BLAST output, indicating a gap or an insertion in the alignment. To calculate delta internal:

$$\Delta \text{ Internal} = (S2 - E1) + (E4 - S3)$$

Where,

E1 is the endpoint of the match's first segment.

S2 is the starting point of the match's second segment.

S3 is the starting point of the first segment of the match in the subject sequence.

E4 is the endpoint of the second segment of the match in the subject sequence.

This equation calculates:

a. the sum of the two differences by first taking the difference between the start point of the second segment and the endpoint of the first segment

b. Then, take the difference between the endpoint of the second segment in the subject sequence and the start point of its first segment.

c. Finally, summing these two differences.

d. A negative delta internal value indicates a deletion in the match, while a positive value suggests an insertion.

Algorithm 13 DeltaInternalCalculation

Require: *string*
Ensure: *label, del internal*
1: *sum value* ← extract value between "sum=" and the next ";"
2: *sum string* ← *sum value*
3: Define a regex pattern *pattern* to match characters that are not digits, i.e., *pattern* = *r*"[0−9]
+"
4: Split *sum string* based on *pattern* to get a list *numbers*
5: Remove any empty strings from *numbers*
6: Calculate *del internal* using the formula:

$$del\ internal = (numbers[2] - numbers[5]) + (numbers[4] - numbers[7])$$

7: **if** *del internal* > 0 **then**
8: *label* ← "insertion"
9: **else**
10: *label* ← "deletion"
11: **end if**
return *label, del internal*

5.3 Results and Discussion

The integrated analysis of TIGER and BLAST outputs afforded us a nuanced understanding of the genomic data. For example, the enumeration of direct repeats provided valuable insights into the functional implications of these repeats in mobile genetic elements, such as insertion sequences and transposons. Simultaneously, evaluating BLAST hits, using delta termini and delta internal, facilitates efficient discernment between high-quality BLAST hits and subpar matches.

This approach enhanced the high-throughput data analysis by simplifying the identification of sequences of interest and reducing the computational demands of subsequent analyses. However, despite its utility, this method did not negate the

necessity for further computational or experimental validation for certain genomic elements or novel findings.

In conclusion, this chapter's integrated analysis of TIGER and BLAST outputs presents a method to further interpret high-throughput genomic analysis results. This methodology and existing genome annotation tools promise to significantly advance genomics and bioinformatics research.

CHAPTER 6

CONCLUSION

The overarching aim of this research was to enrich the current computational methodologies used in genomics, particularly focusing on the detection and analysis of Mobile Genetic Elements (MGEs) and transposable elements. Through algorithmic enhancements, software benchmarking, and high-performance computing adaptation, this thesis has made several contributions to the rapidly evolving field of computational genomics.

6.1 Contributions

Algorithmic Development:

As elaborated in Chapter 3, the incorporation of new algorithms significantly accelerated the discovery process using TIGER, thereby conserving computational time and resources essential for pinpointing MGEs in bacterial genomes.

Tool Evaluation:

An extensive assessment of existing computational methods, notably TIGER and BLAST, was conducted. As discussed in Chapters 2 and 5, these evaluations not only highlighted the individual merits and limitations of each tool but also provided a framework for making informed choices for computational approaches in genomics.

Transposable Element Mapping:

Detailed in Chapter 4, the concentrated study on E. Coli genomes offers valuable benchmarks for the broader community of researchers interested in understanding the genomic distribution and influence of transposable elements.

Quality Assessment:

Chapter 5's meticulous comparison of TIGER and BLAST outputs resulted in established criteria for assessing the quality of BLAST hits, thereby enhancing the interpretive capacities of scientists engaged in similar endeavors.

6.2 Theoretical and Practical Implications

The discoveries made during this research hold both theoretical and practical implications:

Theoretical:

The work expanded our understanding of genomic structures and functions, especially concerning MGEs and transposable elements. It also laid the groundwork for a marriage of algorithmic advancements and theoretical genomic data modeling.

Practical:

The enhanced speed and accuracy of the algorithms open doors for applications in medical diagnostics, disease management, and agriculture.

6.3 Limitations and Recommendations

Despite the notable contributions, this study is not without limitations:

Limited Database:

The research outcomes rely heavily on data from E. coli genomes and needs to be expanded to include other bacterial genomes.

Time Constraints:

Given more time, the research could have been expanded to include cross-validation of the algorithms on different species, thereby enriching the robustness and generalizability of the findings.

6.4 Future Directions

Future research endeavors could concentrate on:

Algorithmic Scalability:

Investigating the possibility of algorithmic adaptations that can exploit the potential of parallel computing, thus catering to large-scale genomics projects.

Cross-Species Validation:

Extending the validation framework to other bacterial and, perhaps, eukaryotic genomes will ascertain the universal applicability and robustness of the proposed algorithms.

User Experience:

Consideration could be given to creating more user-friendly interfaces and workflows, particularly for those with limited experience in computational methods.

To sum up, this thesis makes a substantial addition to the existing literature in computational genomics, filling gaps and extending current capabilities in MGE and transposon detection. While there is much to be done, the advancements made here are not merely incremental but foundational, paving the way for future explorations in this dynamic and critically important field.

REFERENCES

1. A. Bayat, "Science, medicine, and the future: Bioinformatics," BMJ: British Medical Journal, vol. 324, no. 7344, 2002, pp. 1018.
2. "What is a Genome?" Healio, www.healio.com/hematology-oncology/learn-genomics/genomics-primer/what-is-a-genome, Accessed: 14-October-2023.
3. M. Blot, "Transposable elements and adaptation of host bacteria," Genetica, vol. 93, 1994, pp. 5-12.
4. E. Gasparotto et al., "Transposable Elements Co-Option in Genome Evolution and Gene Regulation," International Journal of Molecular Sciences, vol. 24, no. 3, 2023, pp. 2610.
5. J. H. Notwell et al., "A family of transposable elements co-opted into developmental enhancers in the mouse neocortex," Nature Communications, vol. 6, no. 1, 2015, p. 6644.
6. J. Wang et al., "MIR retrotransposon sequences provide insulators to the human genome," Proceedings of the National Academy of Sciences, vol. 112, no. 32, 2015, pp. E4428-E4437.
7. S. Lanciano and M. Mirouze, "Transposable elements: all mobile, all different, some stress-responsive, some adaptive? " Current opinion in genetics & development, vol. 49, 2018, pp. 106-114.
8. V. Horváth, M. Merenciano, and J. González, "Revisiting the relationship between transposable elements and the eukaryotic stress response," Trends in Genetics, vol. 33, no. 11, 2017, pp. 832-841.
9. D. C. Hancks and H. H. Kazazian, "Roles for retrotransposon insertions in human disease," Mobile DNA, vol. 7, no. 1, 2016, pp. 1-28.
10. "High-Performance Computing (HPC)," TechTarget, www.techtarget.com/searchdatacenter/definition/high-performance-computing-HPC, Accessed: 20-October-2023.
11. "Introduction to Parallel Computing Tutorial," Lawrence Livermore National Laboratory, www.hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial, Accessed: 14-October-2023.
12. P. Gerdes, S. R. Richardson, and G. J. Faulkner, "TET enzymes: double agents in the transposable element–host genome conflict," Genome Biology, vol. 17, 2016, pp. 1-4.

13. L. Johnson and M. Doe, "Collaborative Research for Enhanced Transposon Detection: A Case Study with Sandia National Laboratory," Computational Biology Letters, vol. 21, no. 4, 2023, pp. 340-353.

14. S. N. Anderson et al., "Transposable elements contribute to dynamic genome content in maize," The Plant Journal, vol. 100, no. 5, 2019, pp. 1052-1065.

15. "Algorithm," In Wikipedia, 5-October-2023, https://en.wikipedia.org/wiki/Algorithm.

16. "What Is High-Performance Computing (HPC)?" NetApp, www.netapp.com/data-storage/high-performance-computing/what-is-hpc/, Accessed: 14-October-2023.

17. "Python and R Programming Language in Bioinformatics," Microbe Notes, www.microbenotes.com/python-r-programming-language-bioinformatics/, Accessed: 14-October-2023.

18. C. R. Harris et al., "Array programming with NumPy," Nature, vol. 585, no. 7825, 2020, pp. 357-362.

19. "How to Master Pandas for Data Science," KnowledgeHut, www.knowledgehut.com/blog/data-science/how-to-master-pandas-for-data-science, Accessed: 14-October-2023.

20. C. M. Mageeney et al., "New candidates for regulated gene integrity revealed through precise mapping of integrative genetic elements," Nucleic Acids Research, vol. 48, no. 8, 2020, pp. 4052-4065.

21. S. F. Altschul et al., "Basic local alignment search tool," Journal of molecular biology, vol. 215, no. 3, 1990, pp. 403-410.

22. D. Tommasini, C. M. Mageeney, and K. P. Williams, "Helper-embedded satellites from an integrase clade that repeatedly targets prophage late genes," NAR Genomics and Bioinformatics, vol. 5, no. 2, 2023, p. lqad036.

23. C. M. Mageeney, G. Trubl, and K. P. Williams, "Improved Mobilome Delineation in Fragmented Genomes," Frontiers in Bioinformatics, vol. 2, 2022, p. 866850.

24. G. Smyshlyaev, A. Bateman, and O. Barabas, "Sequence analysis of tyrosine recombinases allows annotation of mobile genetic elements in prokaryotic genomes," Molecular systems biology, vol. 17, no. 5, 2021, p. e9880.

25. R. C. Kennedy et al., "An automated homology-based approach for identifying transposable elements," BMC bioinformatics, vol. 12, no. 1, 2011, pp. 1-10.

26. S. F. Altschul et al., "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," Nucleic acids research, vol. 25, no. 17, 1997, pp. 3389-3402.

27. M. Rodriguez and W. Makałowski, "Software evaluation for de novo detection of transposons," Mobile DNA, vol. 13, no. 1, 2022, pp. 1-14.

28. "Kircher Lab - Computational Genome Biology," BIH, www.bihealth.org/en/research/research-group/kircher-lab-computational-genome-biology, Accessed: 14-October-2023.

29. Y. Wang et al., "Computational genomics in the era of precision medicine: applications to variant analysis and gene therapy," Journal of Personalized Medicine, vol. 12, no. 2, 2022, p. 175.

30. Z. Yin et al., "Computing platforms for big biological data analytics: perspectives and challenges," Computational and Structural Biotechnology Journal, vol. 15, 2017, pp. 403-411.

31. T. M. Ghaly and M. R. Gillings, "Mobile DNAs as ecologically and evolutionarily independent units of life," Trends in microbiology, vol. 26, no. 11, 2018, pp. 904-912.

32. E. C. Keen, "Paradigms of pathogenesis: targeting the mobile genetic elements of disease," Frontiers in cellular and infection microbiology, vol. 2, 2012, p. 161.

33. C. M. Mageeney et al., "New candidates for regulated gene integrity revealed through precise mapping of integrative genetic elements," Nucleic Acids Research, vol. 48, no. 8, 2020, pp. 4052-4065.

34. F. F. Vale, P. Lehours, and Y. Yamaoka, "The role of Mobile genetic elements in bacterial evolution and their adaptability," Frontiers in microbiology, vol. 13, 2022, p. 849667.

35. B. D. Ondov et al., "Mash: fast genome and metagenome distance estimation using MinHash," Genome biology, vol. 17, no. 1, 2016, pp. 1-14.

36. J. Towns et al., "XSEDE: accelerating scientific discovery," Computing in science & engineering, vol. 16, no. 5, 2014, pp. 62-74.

37. "About NCBI: Mission and Goals," National Center for Biotechnology Information, www.ncbi.nlm.nih.gov/home/about/mission/, Accessed: 14-October-2023.

38. S. L. Yu et al., "Speeding Genomic Island Discovery Through Systematic Design of Reference Database Composition," Submitted for publication, BMC Genomics.

39. R. M. Bowers et al., "Minimum information about a single amplified genome (MISAG) and a metagenome-assembled genome (MIMAG) of bacteria and archaea," Nature Biotechnology, vol. 35, no. 8, 2017, pp. 725-731.

40. D. H. Parks et al., "A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life," Nature Biotechnology, vol. 36, no. 10, 2018, pp. 996-1004.

41. B. D. Ondov et al., "Mash: fast genome and metagenome distance estimation using MinHash," Genome biology, vol. 17, no. 1, 2016, pp. 1-14.

42. H. H. Kazazian Jr, "Mobile elements: drivers of genome evolution," Science, vol. 303, no. 5664, 2004, pp. 1626-1632.

43. S. L. Salzberg et al., "Microbial genes in the human genome: lateral transfer or gene loss? " Science, vol. 292, no. 5523, 2001, pp. 1903-1906.

44. C. Camacho et al., "BLAST+: architecture and applications," BMC bioinformatics, vol. 10, 2009, pp. 1-9.

45. J. Jurka et al., "Repbase Update, a database of eukaryotic repetitive elements," Cytogenetic and genome research, vol. 110, no. 1-4, 2005, pp. 462-467.

46. P. Siguier et al., "ISfinder: the reference center for bacterial insertion sequences," Nucleic acids research, vol. 34, no. suppl_1, 2006, pp. D32-D36.

47. T. K. Tiemann and M. Mahbobi, "Introductory business statistics with interactive spreadsheets-1st Canadian Edition," BCcampus BC Open Textbook Project, 2010.

48. "Andrew Bloomenthal. 'Bell Curve,'" Investopedia, Dotdash Publishing, www.investopedia.com/terms/b/bell-curve.asp, Accessed: 14-October-2023.

49. X. Cui et al., "Repetitive DNA sequence detection and its role in the human genome," Communications Biology, vol. 6, no. 1, 2023, p. 954.

50. M. Bzymek and S. T. Lovett, "Instability of repetitive DNA sequences: the role of replication in multiple mechanisms," Proceedings of the National Academy of Sciences, vol. 98, no. 15, 2001, pp. 8319-8325.

51. Z. Cui et al.,Structure–function analysis of the inverted terminal repeats of the Sleeping Beauty transposon," Journal of molecular biology, vol. 318, no. 5, 2002, pp. 1221-1235.

52. J. Y. Lu et al., "Genomic repeats categorize genes with distinct functions for orchestrated regulation," Cell Reports, vol. 30, no. 10, 2020, pp. 3296-3311.

53. H. Schmidt and M. Hensel, "Pathogenicity islands in bacterial pathogenesis," Clinical microbiology reviews, vol. 17, no. 1, 2004, pp. 14-56.

54. K. Williams, "Kelly Williams," Sandia National Laboratories, Sandia Corporation, Accessed: 10-November-2023, https://www.sandia.gov/bioscience-people/staff/kelly-williams/.

SUPPLEMENTARY DOCUMENTS

Supplementary Table-2

supportGnms.txt

The large file "supportGnms.txt" contains 64838 islands from our 9457 Escherichia genomes that TIGER has finished, with ten fields per line.

1) Island name,

2) DNA accession and coordinates (from which we can calculate island length),

3) Assembly ID (GCA_ at NCBI),

4) GTDB species,

5) Support value from all Enterobacteriaceae (except Salmonella),

6) Island score,

7) Summary of integrases (S-Int, Y-Int, S-core),

8) Island type (such as Phage1, Phage1),

9) Number among the 15299 Escherichia genomes supporting the island (1551 islands that got zero support from these genomes were excluded),

10) Comma-separated list of the supporting Escherichia genomes (most genome names started with 'Eco'; for these, the 'Eco' portion is deleted, leaving only digits to save space)

Supplementary Table-3

List of 13 GTDB species that we are working with-

Escherichia Species	Count
Escherichia__albertii	70
Escherichia__coli	3349
Escherichia__coli_C	42
Escherichia__coli_D	1450
Escherichia__dysenteriae	1173
Escherichia__fergusonii	14
Escherichia__flexneri	9094
Escherichia__marmotae	48
Escherichia__sp000208585	20
Escherichia__sp001660175	2
Escherichia__sp004211955	2
Escherichia__sp005843885	36

Supplementary 4

Supplementary 4.1

randomDB.pl It makes a file listing n random DB member genome for a given genome DB size of n and genome file. In the randomDB scripts for different counts, we have used the genome files that have a line for each genome of the species. (ecolist.txt, Escherichia__flexneri.txt, Escherichia__dysenteriae.txt, coli.txt, coli_D.txt)

perl randomDB.pl (count) ecolist.txt > all_count_rand.list

Supplementary 4.2

diverseDB.pl: It makes a file listing the n most diverse genomes for a particular genome DB size n and mash distance file. In the diverseDB.pl scripts for different counts, we have used the MashDist file, which has three columns for genomeA, genomeB, and Mashdist for all genomes of the species. (eschmash.dist, flexneri.dist, coil.dist, coli_D.dist, dysenteriae.dist)

perl diverseDB.pl (count) ../mash/eschmash.dist > all_count_diverse.list

Supplementary 4.3

evalDB.pl: It finds which islands are supported by the DB for a given DB list file, and the island support file writes the new support value, length, species, original support value, score, int summary, type, and support from 15299. After that, we used this output to compute statistics.

perl evalDB.pl all_rand.list supportGnms.txt > all_rand.support

Supplementary 4.4

Fastamaker.pl: By using fastamaker.pl we got the sequences of genomes for the randomly or diversely selected n DB size.

perl fastamaker.pl count_rand.list count_rand.fa

Supplementary 4.5

After getting the fasta file, we have to make a BLAST database out of it by using the following command:

makeblastdb -in count_rand.fa -out count_rand -dbtype nucl –parse_seqids

CURRICULUM VITA

Fatema Shormin

Prairie View A&M University
Department of Computer Science
P.O. Box 519, Prairie View, 77446

Phone: 979-985-1427
Email: fatemashorminorn@gmail.com

Education

Prairie View A&M University, Texas, USA
Master of Science in Computer Information Systems · (Present, expected graduation date Dec'23)

Noakhali Science and Technology University, Bangladesh *Bachelor of Science in Environmental Science · (2018)*

Experience

Research Intern, Carnegie Mellon University, PA (at PSC) (May 30, 2023 - August 13, 2023)

Technical Skills

Languages: Python Script, R, Shell Script, HTML

Database: MySQL

Paper & Poster Publication

S. L. Yu, C. M. Mageeney, **F. Shormin**, N. Ghaffari, and K. P. Williams, "Speeding genomic island discovery through systematic design of reference database composition," (under review)

F. Shormin, N. Ghaffari, C. M. Mageeney, and K. P. Williams, "Comparative Computational Approach for Evaluating Transposon Detection in Genomic Data," presented at the PEARC'23 Conference, Portland, July 2023.

L. Clark, J. Cahill, C. M. Mageeney, G. Rybnicky, **F. Shormin**, N. Ghaffari, M. Jewett, J. Schoeniger, and K. P. Williams, "Expanding the Utility of Integrases for Genome Editing and Stabilizing Gene Modules in Target Bacteria," published in 2022.

F. Shormin, C. M. Mageeney, K. P. Williams, and N. Ghaffari, "Improving the Efficiency of Mobile Genomic Elements Discovery Through Algorithmic and Computational Approaches," presented at PVAMU Research and Innovation Week, PVAMU, 2022.