ENHANCE THE UNDERSTANDING OF WHOLE-GENOME EVOLUTION BY DESIGNING, ACCELERATING AND PARALLELIZING PHYLOGENETIC ALGORITHMS

A Dissertation
Presented to
The Academic Faculty

by

Zhaoming Yin

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering

Georgia Institute of Technology
May 2014

ENHANCE THE UNDERSTANDING OF WHOLE-GENOME EVOLUTION BY DESIGNING, ACCELERATING AND PARALLELIZING PHYLOGENETIC ALGORITHMS

Approved by:

Professor David A. Bader
School of Computational Sience and Engineering
Georgia Institute of Technology

Professor Mark Borodovsky
School of Biology
Georgia Institute of Technology

Professor Srinvas Aluru
School of Computational Sience and Engineering
Georgia Institute of Technology

Professor Jijun Tang
Deptartment of Computer Science and Engineering
University of South Carolina

Professor Richard W. Vuduc
School of Computational Sience and Engineering
Georgia Institute of Technology

Date Approved: 25 March 2014

To my parents,

Xiaoping Yin and Lanying Xiao,

Who can hear the voice of my heart, even from thousands of miles away.

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. David A. Bader for his powerful support for my study, research, and life of living. I would like to thank Prof. Jijun Tang for his help guiding me in the direction of computational biology and genome rearrangement problems. I would like to thank Prof. Steve W. Schaeffer for his careful assist in criticizing on and revising my papers. I would like to thank Dr. Satish Nadathur and Dr. Narayanan Sundaram in Intel labs for their hand-in-hand instruction to provide me a profound understanding on graph analytics and code optimizations. I would like to thank Dr. Seunghwa Kang for his insightful instruction to help me to initiate my research work. I would like to thank the members of my committee for their extreme patience in the face of numerous obstacles. I would like to thank Dr. Wei Xu, Xing Liu, Dr. Oded Green, Piyush Sao, Mingfu Shao, and Jia Zhao for their feedback on various aspects of this project. I would like to thank Xue Wu, who has spent a whole year with me and help me conquer the odds in the extremely busy days of my life. I would like to thank Dr. Xuan Shi, Jian Zhao, Mizan Rahman and Cong Hou, as friends, they provided me as much help as they can to make the world I lived a happier place. Last, but certainly not the least, I would like to thank those who gave me a lot of brave to survive this hardship of PhD study.

SUMMARY

The advent of new technology enhance the speed and reduce the cost for sequencing biological data. Making biological sense of this genomic data is a big challenge to the algorithm design as well as to the high performance computing society. There are many problems in Bioinformatics, such as how new functional genes arise, why genes are organized into chromosomes, how species are connected through the evolutionary tree of life, or why arrangements are subject to change. Phylogenetic analysis has became essential to research on the evolutionary tree of life. It can assist us to track the history of species and the linkage between diverse genes or genomes through millions of years.

One of the fundamentals for phylogenetic analysis is the computation of distances between genomes. The distance computation can be generally divided into two categories based on the granularity of the input data. One with fine grained granularity is on the gene level, which is generally rest on aligning nucleotide or amino acid between two gene sequences to minimize the number of insertions/deletions among them. Another is more coarse grained, which takes the rearrangement events of genes into consisderation. Since there are much more complicated combinatorial patterns in rearrangement events, the distance computation is still a hot topic as much belongs to mathematics as to biology. For distance computation with input of two genomes containing unequal gene contents (with insertions/deletions *Indels* and duplications), the problem is especially difficult. In this thesis, we will discuss about our contributions to both of these two distance methods.

The problem of finding the median of three genomes is the key process in building

the most parsimonious (MP) phylogenetic trees from genome rearrangement data. When dealing with input genomes having the same gene content, the median problem employing Double-Cut-and-Join (DCJ) distance is *NP-hard* and the best exact algorithm is anchored in a branch-and-bound (BnB) search strategy to explore sub-graph patterns in Multiple Break Point Graph (MBG). As the search space is prohibitively large, it may take months if not years to finish when the genomes are distant. For genomes with unequal contents, to the best of our knowledge, there is no algorithm to be utilized for finding the median. In this thesis, we make our contributions to the median computation in two aspects: 1) Algorithm engineering aspect, we harness the power of streaming graph analytics methods to implement an exactDCJ median algorithm which run as fast as the heuristic algorithm and is able to construct a better phylogenetic tree. 2) Algorithmic aspect, we explore the way of leveraging genetic algorithm to tackle with the median problem, which achieved good results on synthetic data. In addition, we theoretically formulate the problem of finding median with input of genomes having unequal gene content, which leads to the design and implementation of an efficient median algorithm build around Lin-Kernighan heuristic.

Having the knowledge for the two fundamental issues of distance and median computation. Inferring phylogeny (evolutionary history) of a set of given species is the ultimate goal. For more than a decade, biologists and computer scientists have studied how to infer phylogeny by the measurement of genome rearrangement events using gene order data. While evolution is not an inherently parsimonious process, MP phylogenetic analysis has been supported by being widely applied to the phylogeny inference to study the evolutionary patterns of genome rearrangements events. There are generally two problems with the MP phylogenetic aroused by genome rearrangement: One is, given a set of modern genomes, how to compute the topologies of the according phylogenetic tree; Another is, given the topology of a model tree,

how to infer the gene orders of the ancestor species. To assemble a *MP* phylogenetic tree algorithm, there are multiple *NP-hard* problems involved. Unfortunately, these problems are organized as one on top of others. Which means, to solve a *NP-hard* problem, we need to solve multiple *NP-hard* sub-problems first. For phylogenetic tree construction with the input of unequal content genomes, there are three layers of *NP-hard* problems. In this thesis, we will mainly discuss about our contributions to the design and implementation of the software package *DCJUC* (Phylogeny Inference using **DCJ** model to cope with **U**nequal **C**ontent Genomes), which is able to assist achieving both of these two goals.

Aside from the biological problems, another issue to be concerned is on how to utilize the power of parallel computing to assist accelerating algorithms handling huge data sets. For example, the high resolution gene order data. For one thing, most of the algorithms we used in phylogenetic problems are grounded on branch-and-bound method, which is quite irregular and unfriendly for parallel computing. To parallelize these algorithms, we need to properly select way to boost localized memory access and load balance, in hope that each thread can put their potentials into full play. For another, there is a revolution taking place in computing with the availability of commodity graphical processors such as *Nvidia GPU* and many-core *CPUs* such as *Cray-XMT*, and *Intel Xeon Phi*^{TM} co-processor. These architectures pave a new way to uplift performance at much lower cost. However, code running on these machines are not so easily programmed, and scientific computing is hard to tune well on them. We try to explore the potentials of these architectures to help us accelerate *BnB* like phylogenetic algorithms. As a result, a software package *OPT-Kit* (**OPT**imization Tool-**Kit** for Prallellizing Discrete Combinatoric Problems in Emerging Platforms) is designed and implemented on state of the art high performance architectures.

Contents

List of Tables

List of Figures

Chapter I

INTRODUCTION

1.1 Biological Background

A phylogenetic tree captures speciation events among multiple organisms. Constructing a phylogenetic tree requires inferring ancestral relationship among multiple organisms based on currently available data. Traditionally, scientists had studied this problem by inspecting fossils or comparing the morphology and the physiology of living creatures, but these approaches revealed limitations due to an incomplete set of fossils or the complex nature of evolutionary mechanisms affecting the morphology and the physiology of organisms. The increasing availability of genetic data opens a new opportunity to solve the problem. Constructing a phylogenetic tree by comparing nucleotide sequences of a single gene or a few genes has been intensively studied. With the advancement of the multiple sequence alignment technology, phylogenetic trees can be constructed simultaneously with multiple sequence alignment (*MSA*). An example of multiple sequence alignment is shown in Figure 1

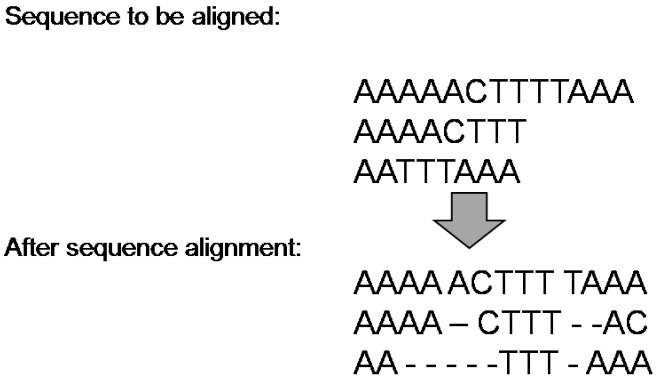
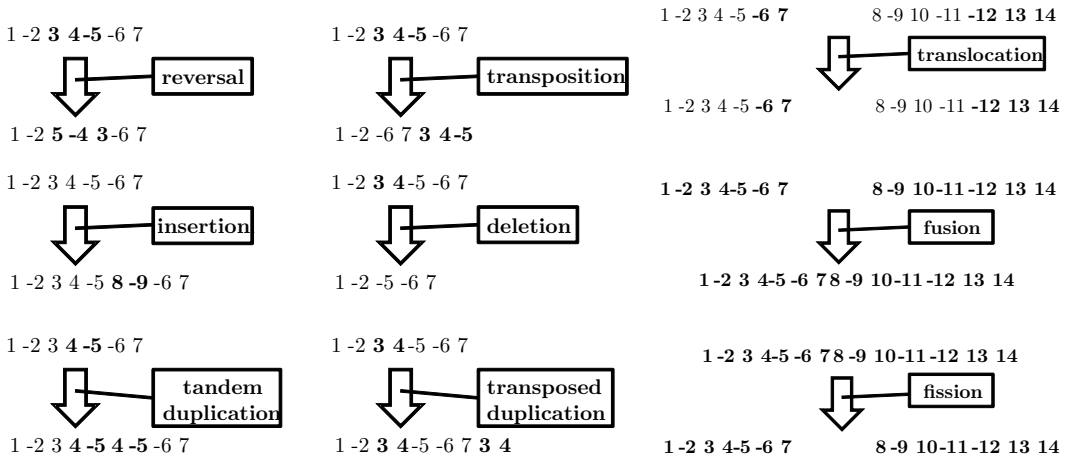


Figure 1: Example of multiple sequence alignment.

Nonetheless, the content of the *DNA* molecules is often similar, but their organization usually differs dramatically. As a consequence, the multiple sequence alignment method is limited by its appliance to only one or few genes but not to the whole picture of the genome. Mutations that affect the organization of genes are called genome rearrangements, the phylogenetic tree construction employing genome rearrangement events is associated with numerous combinatorial optimization problems.



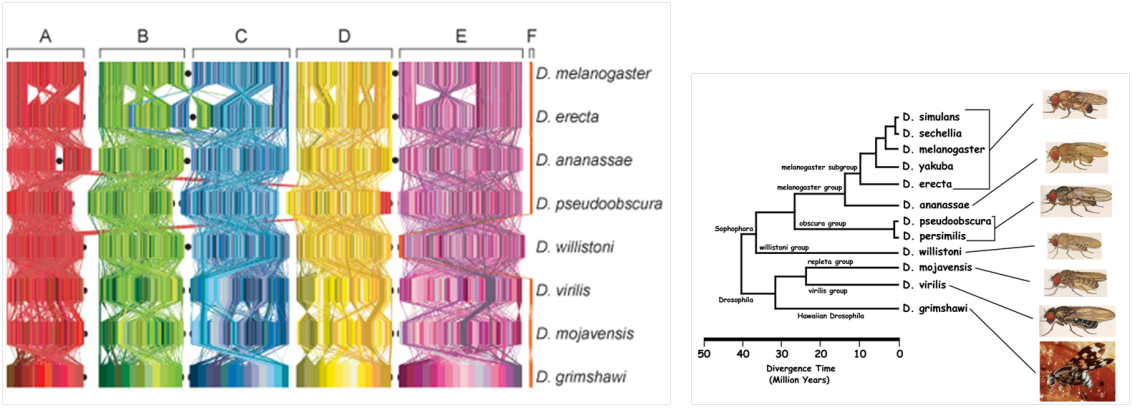
(a) Intra genome rearrangement events

(b) Inter genome rearrangement events

Figure 2: Different genome rearrangement events.

The genome of a taxa, is consisted of multiple chromosomes. For example, the human being's genome is consisted of 23 chromosomes, and each chromosome is consisted of multiple genes. If two genes (possibly located in chromosomes of different species) are originated from a common ancestral gene, then the two genes are homologous. If a unique number is assigned to a set of homologous genes, a chromosome can be represented as a sequence of numbers; numbers appear in the order the genes corresponding to the numbers appear in a chromosome. If a genome has multiple chromosomes, we can represent the genome with multiple sequences of numbers-one sequence per chromosome. Such sequences of numbers are gene order data. There are various kinds of operations that is included in the genome rearrangement events which can roughly be divided into two classes: Intra-chromosomal genome rearrangement

events, and Inter-chromosomal genome rearrangement events. Figure 2 shows the examples for rearrangement events in multiform.



(a) Example of drosophilla gene content

(b) Example of drosophilla phylogenetic tree

Figure 3: Example of sequence data of 12 species drosophila [19, 127]

Though genome rearrangement events are comparatively rare in evolutionary history, through millions of years aggregation, it is an ineligible factor for distinguishing species. For example, in Fig 3(a), the genome content for 12 species drosophilla is displayed. We can see that there are very obvious changes in their gene contents. Build from their gene order data, the phylogenetic tree for the 12 species drosophilla is shown in Fig 3(b). Another point of view is, gene duplication events are also very widely occurred. Fig 4 shows an example of 55 species yeast genome. In the figure, triangles with different color represents the orientation and type of different genes. It's easy to notice that there are a lot of duplications happened (triangles with the same color in a single genome).

1.2 Distance Computation between Species

Biologist has spent over decades to develop methods on gene level sequence comparison, which is mainly focused on the research of multiple sequence alignment. Notable softwares include *Clustal-W*, *T-Coffee*, *Muscle*, *DiAlign*, *Satchmo*, *ProbCons*,

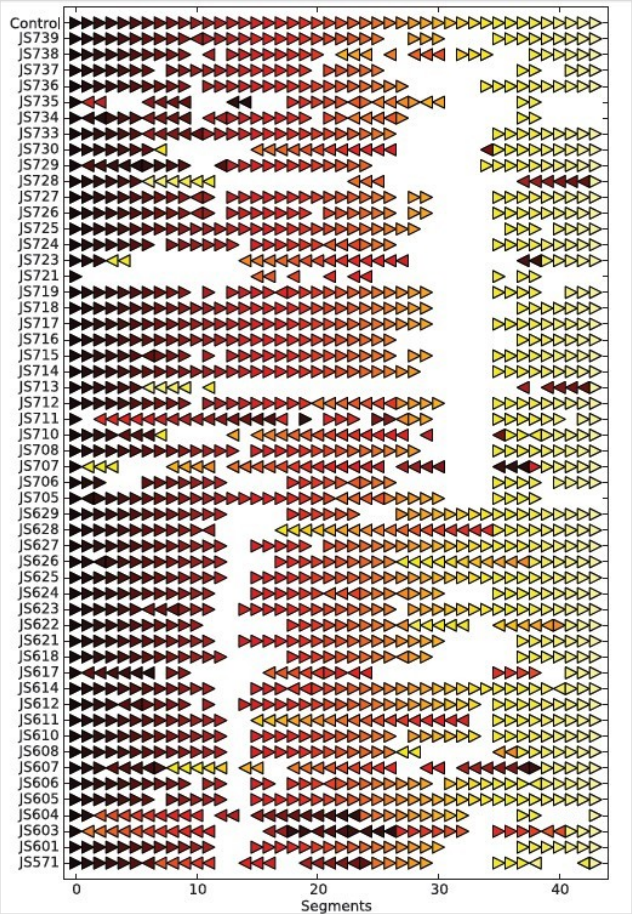


Figure 4: Example of yeast gene order data with duplicated genes marked with the same color [50].

Mafft, and *Sam* [44, 52, 138, 103, 158, 53, 54]. Although *DNA* sequence data is still the dominant source, building phylogeny from higher-level changes such as genome rearrangements has gained increasing attentions from the field. The distance calculation between two genes is the canonical longest common sequence problem (LCS). There are multiple approaches to solve this problem. Such as the dot-mat method [63], dynamic programming method [72, 41], probability based method [47, 46] and various of heuristics [78, 137]. These algorithms usually has regular computational patterns. However, with the advancement of high throughput sequencing technology, more and more sequence data becomes available to be processed. How to handle this huge amount of data with sequence alignment technologies becomes a challenging

problem.

There are many distance metrics to calculate the dissimilarity between two genomes [21, 14]. Among them, Double-cut-and-join (DCJ) distance [150] is currently the most widely used in the research of rearrangement distance modeling. It can abstract the rearrangement operations of reversal, translocation, fusion, and fission, with the additional merits that the model can be easily applied to multi-chromosomes. All these distance metrics have the same assumption that two input genome contains the same number of genes and every gene has exact one copy in each genome. This assumption limits the true modeling of genome evolution, because gene insertions/deletions (*Indel*) and duplications are widely spreaded events in different species. Computational biologists tried multiple ways to surpass this limit, such as the effort of using exemplar distance [123, 104] to measure the difference of two genomes with duplications happened. Very useful attempt is tried to compute the approximate and exact DCJ distance with duplications [129, 98]. And for genomes with *Indels*, there is exact algorithm to compute their DCJ distance [37]. However, there is no way to measure distance of genomes with gene orders that contain both *Indels* and duplications.

1.3 Median Computation

The question of finding median of three genomes rooted on genome rearrangement, is the building block for constructing a phylogenetic tree under maximum parsimony criteria [100, 22]. Given three genomes, it inquires a "median" genome which has the minimum accumulate genome rearrangement operations (distance) between these three genomes. There are different methods to solve the median problem using different measurement of distance metrics, such as break-point [112], reversal [30], translocation [16] and double-cut-and-join(DCJ) [150]. The DCJ median problem is proved *NP-hard* and *APX-hard* [136] and several exact algorithms have been implemented to

solve the DCJ median problems on both circular [146, 143] and linear chromosomes [142, 145]. Some heuristics are introduced to improve the speed of median computation, such as the linear programming (LP) [30]; local search [49, 87]; evolutionary programming [62, 49]; or simply searching on one promising direction [118]. All these algorithms are intended for solving the median problems with equal content, which is highly unrealistic in practice.

1.4 Phylogenetic Inference and Ancestor Genome Reconstruction

There are several representative approaches for constructing phylogenetic trees. The distance based method is a polynomial time approach, typical algorithms includes Neighbor joining (NJ) and $UPGMA$ [122, 132], they are essentially hierarchical clustering employing different method to guide the merging process. Maximum parsimony (MP) methods find a topology with the minimum number of mutations (or the lowest parsimony score), and maximum likelihood (ML) methods attempt to find a tree with the highest likelihood value under a certain evolutionary model. MP and ML methods are generally more accurate than the distance oriented method but also more computationally expensive. These methods, using nucleotide sequence data, find a reasonably accurate tree topology for close genomes. For distant genomes, these methods become significantly less accurate due to the high rate of nucleotide sequence level mutations.

NJ method is one of the most widely applied phylogenetic tree tool. It takes as input a distance matrix specifying the distance between each pair of taxa. The algorithm starts with a completely unresolved tree, whose topology corresponds to that of a star network, and iterates over the steps that trying to join the sub-tree and recompute the distance matrix until the tree is completely resolved and all branch lengths are

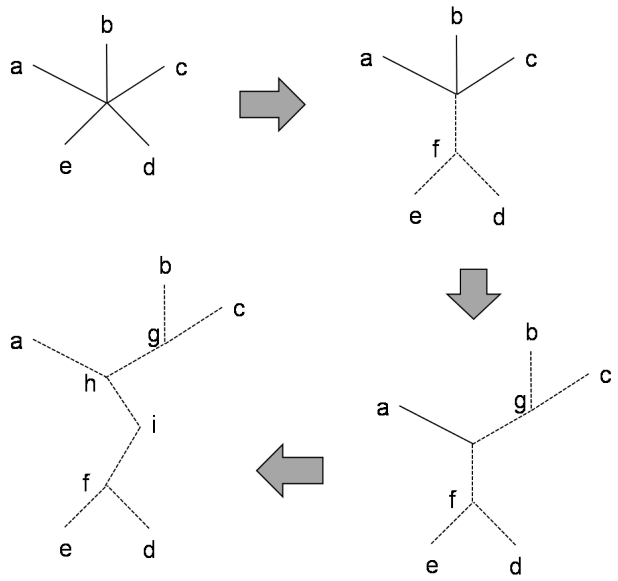


Figure 5: Example of using neighbor joining method to build a phylogenetic tree.

known. In addition, neighbor-joining method also serves as the indicator to estimate how easy a data set can be infered for phylogenetic purposes. Because in general, when neighbor-joining method performs well, other methods can perform good as well, and vise versa. Figure 5 shows an example of using neighbor joining method to construct a phylogenetic tree.

The principle of *MP* is to choose one tree which entails the smallest amount of evolutionary change. The *MP* criterion favors hypotheses that maximize congruence and minimize homoplasy. In the sequence based phylogeny, in which taxa are represented by aligned sequences,parsimony criteria are classified into Fitch Parsimony, Wagner Parsimony, Dollo Parsimony and Generalized Parsimony (Sankoff Parsimony). Several algorithms of the heuristic search for *MP* trees are available, however, to get the exact optimal, branch-and-bound (*BnB*) algorithm is used. See Figure 6 for an example of using *BnB* method to infer a phylogenetic tree. The expansion of active nodes in the *BnB* search of phylogeny is stepwise addition, trying to insert the next taxon into arbitrary branches of an incomplete tree. Many researchers have been

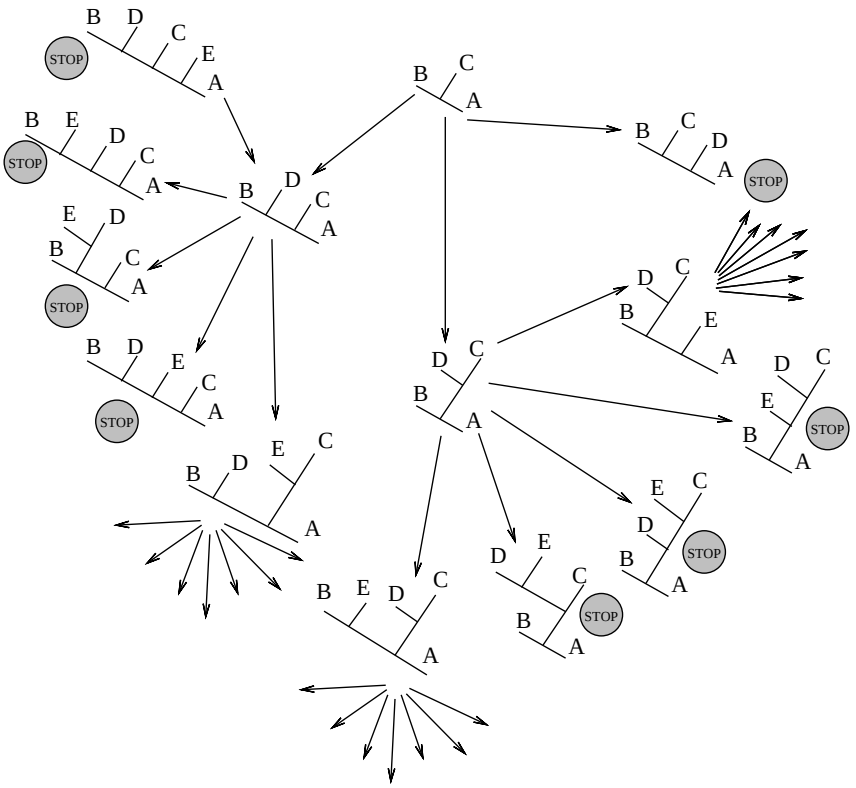


Figure 6: Example of using BnB to search a maximum parsimonious tree.

working on different tree search strategies to escape local optima or prune branches more quickly. On the other hand, because parsimony problems require evaluating enormous number of trees, rapid evaluation of a candidate tree generated by stepwise addition or branch swapping is a crucial factor to the performance of a parsimony program as well.

The maximum likelihood method to construct a phylogenetic tree is a probability driven method. It can be formulated as the problem of finding θ which maximize the likelihood of $L(I|\theta)$. Of which I represents the input of different species with some specific encoding, such as sequence data of nucleic acid, protein, or gene sequences. Recent research on *ML* method targeting at genome rearrangement events can be found at [134, 88] , they can cope with genome sequences with very high resolution and deal with very large data sets.

1.5 Supertree and Consensus Tree Problems

There are in general two approaches to tackle with the phylogeny topology inference problem. One is based on exhaustive search that select the best tree by enumerating every possible tree, another is a heuristic method using sequential addition that builds the tree by adding species one by one. The exhaustive approach can usually find better tree than heuristic approach at the cost of much more computational time. And the Disk-Covering Method (*DCM*) [74, 80], provide a divide-and-conquer way to assemble the sub-trees which are inferred by exhaustive approach into one tree, with little loss of accuracy. There are three versions of *DCMs* [75, 77, 121]. The first *DCM* (*DCM1*) was designed for use with distance-based methods and has provable theoretical guarantees about the sequence length required to reconstruct the true tree with high probability under Markov models. It produces good sub-problems (small enough in size), but the structure induced by the decomposition is often poor. The second *DCM* (*DCM2*) was designed to speed up heuristic searches for *MP* trees. It computes a fixed structure (a graph separator) to overcome *DCM1*'s drawback, but the resulting sub-problems tend to be too large. An both of *DCM1* and *DCM2* used only one distance matrix for parition. *DCM3* overcome their problems by using a dynamically updated guide tree and do the computation with iterative and recursive manner. *DCM3* tend to produce good tree with one order of magnitude more number of species. Since all these *DCM* methods are designed for sequence alignment based phylogenetic tree construction. They do not adapt well to the genome rearrangement based phylogenetic tree construction. A spectral partition based method is introduced to improve the existing *DCM* method [80]. In synthetic analysis, it out performs all of the *DCM* methods.

1.6 Ancestor Genome Reconstruction

As for the ancestor genome reconstruction problem, once the tree topology is fixed, there are multiple ways to reconstruct the ancestry gene orders which can be generally divided into two categories. One is based on heuristics that tries to recover all common child adjacencies, and for the uncommon ones, recover the one that is more likely to be present in the parent genome. [91] introduced the method for reconstructing contiguous regions of an ancestral genome. We can see this process as a way to infer the "median" genome of two given genomes. Given a phylogenetic tree, it constructs the genome sequences of the ancestral species from modern species. They also designed an algorithm named *DUPCAR* [93] to help reconstruct contiguous ancestral regions (*CAR*) with duplications which contains a sub-routine of gene tree reconciliation [85, 135]. However, Ma's method requires the input of phylogenetic tree and gene tree, which is not feasible if we want to "build" a phylogenetic tree from the scratch. And usually, this method is not capable of finding the mathematically optimized median genome. Another category is more mathematical sound, it tries to recover the ancestor genome by finding the median genome of three given genomes. The method is a two step process to iteratively improve the median results for each ancestor genome. The first step is the initialization step that initiate the internal nodes in the model tree. The second step is the refinement step that tries to improve the results, if one of the three neighbors' genome is changed, the algorithm will update the genome to see if it improves or not. The initialization step is critical for the quality of the final ancestor genome reconstruction.

1.7 Parallel Branch-and-Bound Methods

Branch-and-bound method is one of the most well know method to get the exact solution for the *NP-hard* optimization problems. It's basically originated from a tree search approach. There are different search strategies for *BnB*, such as depth-first-search (*DFS*) and best-first-search ($A*$), $A*$ has the advantage over *DFS* for it can ensure to have the minimum search space. In [154], the number of search nodes expanded by *DFS* is no more than $O(dN)$, where d is the tree depth and N is the expected number of nodes expanded by $A*$. However, *DFS* only use $O(d)$ of space (d is the depth of the search tree), while the space $A*$ used is usually way more than that, even though there are some memory saving technologies such as frontier search [84] or Delayed Duplicate Detection [83]. Though there are methods that combine the idea of $A*$ and *DFS*, such as iterative deepening [82] or recursive best-first-search [155], they usually do not perform better than *DFS* or $A*$ in practice.

There are many frameworks that enable people who has background in optimization area to write parallel *BnB* programs to solve optimization problems such as: *PEBBLE* [115] and *PICO* [51], they are written in *C++* language and distributed memory system, they can help developers to design different kind of *BnB* algorithms with relatively less effort. *ALPS* [147] , which is designed for distributed memory system, and adopted many traditional strategies for parallel *BnB* algorithms. It can support $A*$ search, and use a knowledge pool to share intermediate search nodes. However, this method does not scale well, as the central node pool inevitably becomes a computational and communications bottleneck. *DryadOpt* [29] is also a distributed memory framework for designing parallel branch and bound algorithms, it is based on the big data processing engine. The limit of this method is, it does not fully support $A*$ algorithm, therefore the knowledge sharing is just at the ramp up stage for every iteration. And there is no dynamic load balancing in this method. *BoB++* [43] can be used very

flexibly, which support many architectures like *SMP* and distributed memory systems, it can also support different load balancing strategies including master-worker and master-hub-worker. Nevertheless, it does not have a public available source for $A*$. There are also old parallel packages for branch-and-bound algorithm such as *PUBB* [130] and *PPBB-lib* [1], they are targeting at old architectures, which might not suit today's high performance parallel architecture.

BnB algorithm has in its nature parallelism, however, Performance of the parallel *BnB* algorithm is highly dependent on different factors, such as storage of open sub trees, choice of data structures, communication protocols, and choice of granularity [12]. Since there are different parallel algorithms implemented in various hardware architectures such as distributed memory system, grid architecture, and shared memory system [13]. Different techniques are used for these architectures, such as ramp up and hub-worker strategies for the load balancing in distributed memory systems [51],the master-worker paradigm which is well suited for the grid computing [64] and the use of priority queue in the share memory system [12]. With the upcoming of "Big Data" era, there are parallel *BnB* methods based on the distributed data-parallel execution engine such as *DryadOpt* [29].

There is one type of the optimization problem that to solve these problems, it's necessary to solve a set of embedded *NP-complete* problems. For example, the verification process is *NP-complete*. The time used to solve these problems is a constant raised to the power of an exponential function, which has a general formula of $f(x) = a^{bx}$ and grows much more quickly than an exponential function. These problems are called *EXPTIME* or *NEXPTIME* problems [69]. *EXPTIME* or *NEXPTIME* problems problems are widely spreaded across different domains. For example in the game theory [61, 120, 119], computational biology [34, 100, 22]. However, there is no software packages focused on these kind of problems. We think these problems will be a

big challenge in the next generation petascale or exascale computation. Because they are involved with multiple memory management, layered scheduling, load balancing, *I/O* problems.

1.8 Emerging Parallel Computing Architectures

Recent graphics architectures provide tremendous memory bandwidth and computational horsepower at a very inexpensive price compared with the same amount of computing capability of traditional *CPUs*. Meanwhile, because of its design principle for arithmetic intensity, the speed of graphics hardware is increasing quickly, nearly doubling every six months. The stream programming model provides a compelling solution for applications that require high arithmetic rates and data bandwidths. Compute Unified Device Architecture (*CUDA*) introduced by *NVIDIA* is a programming environment for writing and running general-purpose applications on the *NVIDIA GPUs*, such as *GeForce*, *Quadro*, and *Tesla* hardware architectures. *CUDA* allows programmers to develop *GPGPU* applications using the extended *C* programming language instead of graphics *APIs*. In *CUDA*, threads are organized in a hierarchy of grids, blocks, and threads, which are executed in a *SIMT* (single-instruction, multiple-thread) manner; threads are virtually mapped to an arbitrary number of streaming multiprocessors (*SMs*). In *CUDA*, the wider memory is shared, the slower it is accessed. Therefore, how to organize the memory access hierarchy in the application is the golden rule for *CUDA* programmers.

Recently, *Intel* announced the *Intel*® *Xeon Phi*™ Coprocessor, codenamed Knights Corner (*KNC*), which is the first commercial release of the *Intel*® Many Integrated Core (*Intel*® *MIC*) architecture. Unlike previous microprocessors from *Intel*, *KNC* works on a *PCIe* card with *GDDR5* memory and offers extremely high memory bandwidth. The first model of *KNC* has 60 cores, featuring a new 512-bit *SIMD*

instruction set. With a clock speed in excess of 1 GHz, *KNC* has over 1 Tflops double precision peak performance from a single chip.

Chapter II

DISTANCE COMPUTATION

2.1 Using HMM *based Method to Compute Distance using Gene Sequence Data*

Biological Sequence alignment is very important in homology modeling, phylogenetic tree reconstruction, sub-family classification, and identification of critical residues. When aligning multiple sequences, the cost of computation and storage of traditional dynamic programming algorithms becomes unacceptable on current computers. Many heuristic algorithms for the multiple sequence alignment problem run in a reasonable time with some loss of accuracy. However, with the growth of the volume of sequence data, the heuristic algorithms are still very costly in performance.

The Hidden Markov Model (*HMM*) includes three canonical problems: evaluation, decoding, and learning [9]. Typical sequence alignment approaches often involve the last two problems. The learning problem [131, 26] often needs less time compared with the decoding problem – Biological Sequence Alignment Oriented Viterbi Algorithm [48]. Based on our experimental results on *Satchmo* [54] (a multiple sequence alignment tool), the *Viterbi* algorithm occupies more than approximately 80% of the total execution time. Therefore, it is critical to design an efficient method to speed up the *Viterbi* algorithm.

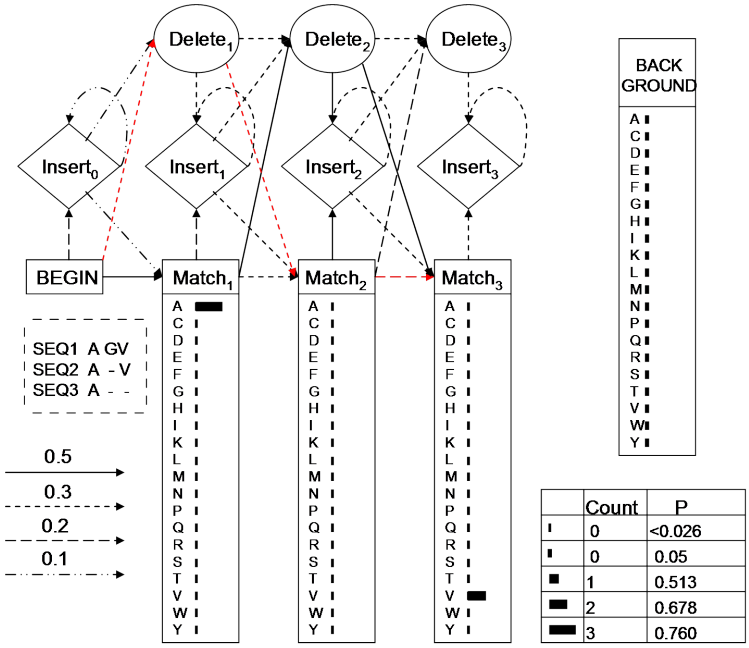


Figure 7: A typical *HMM* model built from a sequence profile, Match stands for the match state of the i^{th} profile column, as do Deletet$_i$ and Insert$_i$. There is a probability assigned to each state-transition of every column, which is represented by different type of line. For every match state, there are emit probabilities for each of 20 amino acids in accordance with that state. These probabilities are represented by different lengths of bars. For delete and insert states, emit probabilities are the background probabilities trained from massive amounts of data in the real world.

2.1.1 Hidden Markov Model

An *HMM* is a quintuple; it can be described as follows:

$$HMM = \{O, S, P^{(e)}, P^{(t)}, \pi\} \tag{1}$$

O stands for the set of observations; S stands for the set of states, $P(e)$ stands for the probability of emitting an observation for the given state, $P(t)$ stands for the probability of transferring from one state to another, and π stands for the probability of an initial state. Figure 7 shows a typical hidden Markov model build on a sequence profile of length 3 consisting 3 aligned sequences. In this Figure, 1) the observation stands for the residue(s) in a specific column of a profile, which includes one or the combination of 20 amino acid residues. 2) There are three possible states for each

Algorithm 1: VITERBI

```
1 for observationO_t ∈ Observations do
2     for stateS_i ∈ states of observation O_t do
3         for stateS_j ∈ states of previous observation O_{t−1} do
4             calculate δ(t, i) ;
```

columns of a profile: Match, Delete and Insert. States could transfer to other states of which the arrow point to, there are probabilities assigned to each state transition. In addition, once a state of a column is decided, there will be an emitting probability. For example, in Figure 7, the match state of column 1 is 50%, and the emit probability of A under match state is 76%.

2.1.2 *Viterbi* Algorithm

Viterbi algorithm is used to solve the so-called "Decoding" problem, which could be described as: Given a set of observations, find a route of their according states, which maximize the probability of observations. As for an event that is consisting of n observations, if one observation could be mapped to m states, there will be mn possible routes. *Viterbi* algorithm applies the dynamic programming method, suppose $\delta(t, i)$ is the probability to be calculated of state i for observation t, and the *Viterbi* algorithm could be described as Algorithm 7:

This algorithm gives a general description of *Viterbi* algorithm, the time complexity for this algorithm is $O(nm^2)$. n stands for the number of observations; m stands for the number of states of an observation.

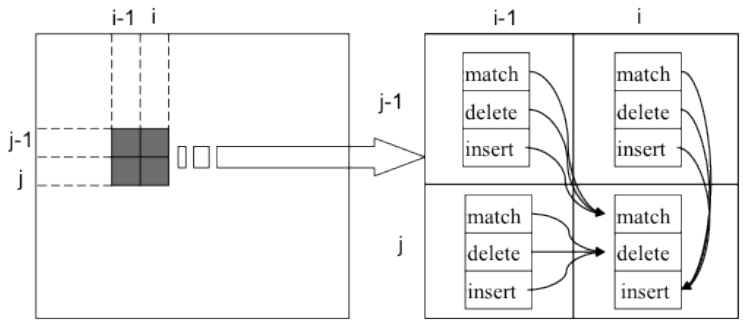


Figure 8: Foundation of the biological sequence alignment oriented *Viterbi* algorithm. The rectangle on the left represents the whole matrix to be computed by the *Viterbi* algorithm, and the right side of the figure shows the process of updating a single block of the matrix.

2.1.3 Biological Sequence Alignment Oriented *Viterbi* Algorithm

The task of using *HMM* to align sequence can be described as: Given a template profile (*HMM* is build on template profile), and a target profile (which is to be aligned to template profile), find an optimal route of states assigned to each columns of template. For example, in Figure 7, if the target is GV, and the red line marks the route of aligning template to target, so the aligned result is: $-GV$ The 'insert' state is permitted to loop over itself, so there would be many repeat 'insert' states to be calculated for one observation. When target is long, it is very time consuming to calculating so many insert states. The *Viterbi* Algorithm for Biological Sequence Alignment is used to solve the redundant computation of 'insert self-loop' problem of general *Viterbi* algorithm. It reduce the time complexity from $O(m^2n)$ to $O(len\_t * len\_a)$, (len_t stands for the length of template, len_a stands for the length of target), for the detail of this algorithm please refer to [48]. As Figure 8 shows, the fundamental task of *Viterbi* Algorithm for Biological Sequence Alignment is to calculate a matrix consists of $len\_t * len\_a$ blocks, each block is consisting of three states: Match, Delete and Insert, and the value for each state of a block is dependent on the value of previous block. The Match state is dependent on the upper-left block; the Delete state depends

Algorithm 2: ViterbiAlign

```
for Block_mn ∈ Matrix do
    for stateS_i ∈ states of Block_mn do
        for stateS_j ∈ three states of dependent Block do
            //when S_i is match, the dependent block is Block_(m−1)(n−1)
            //Delete Blcok_(m−1)n, Insert Blcok_m(n−1)
            calculate δ(mn, i) ;
```

on the left block and the Insert state depends on the up block. Suppose $\delta(mn, i)$ is the maximal probability of state i for block in matrix of column m and row n, The *Viterbi* Algorithm for Biological Sequence Alignment could be described as:

Because there are $len\_t * len\_a$ blocks, and each block has three states where each state refer to three states of the dependent block, therefore, the time complexity for this algorithm is $O(9 * len\_t * len\_a)$. It is an order of magnitude faster than general *Viterbi* algorithm used in biological sequence alignment.

2.2 Using Graph based Method to Compute Distance Using Gene Order Data

2.2.1 Preliminaries

Given a gene set which contains g non-duplicate genes, each gene is marked with a signed number $1 \leq |i| \leq g$. We define a breakpoint graph $BPG = (V, E)$, such that V is consists of $2g$ vertices, Each gene i is represented by a pair of vertices *head* (marked $|i| * 2 - 2$) and *tail*(marked $|i| * 2 - 1$). These vertices are ordered following the gene sequence: $head(i)$ is ordered in front of $tail(i)$ if i is positive, Otherwise, $tail(i)$ is put in front of $head(i)$. If two genes i and j are adjacent to each other in the gene order$(..., i, j....)$, an edge will connect their adjacent vertices. When dealing with end points, if the chromosome is circular, two end points of the breakpoint graph will be connected by an single edge, if the chromosome is linear, each of the two end points

will connect by an edge to a separate vertex called a *CAP* vertex. Fig 9 shows the *BPG* for gene order Γ (1,-2,3,-6,5) which is a genome with one circular chromosome and Π (1,2,3,7,4) which is a genome with one linear chromosome. Double-cut and join (*DCJ*) operation is able to simulate all aforementioned rearrangement events using *BPG*. The operation cut two edges (one for each genome) and rejoin them using two possible combinations of end vertices (shown in Fig 10). *DCJ* distance of genomes with the same content can be easily calculated by enumerating the number of cycles in the *BPG*. comparing with the complex model based on reversal operations, *DCJ* operation is simple and powerful.

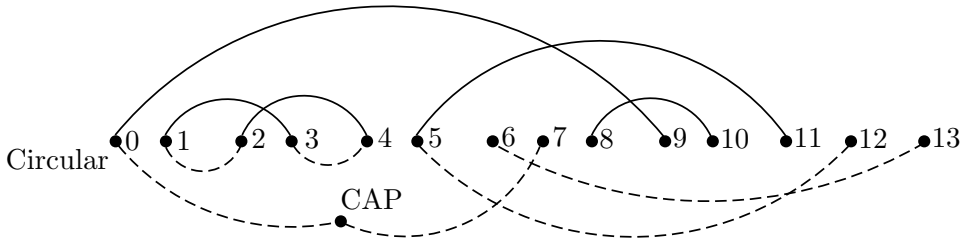


Figure 9: Example of breakpoint graph for gene order (1,-2,3,-6,5) which is a genome Γ formed by one circular chromosome, the genome is represented by solid edges. And gene order (1,2,3,7,4) is a genome Π that has one linear chromosome, the genome is represented by dashed edges.

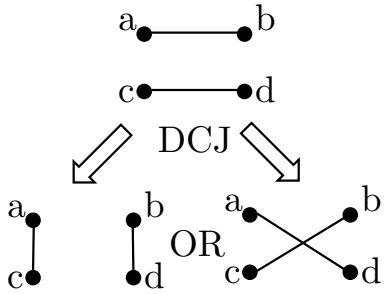


Figure 10: *DCJ* operation on *BPG*, which selected two edges (one for each genome); cut these two edges and rejoin them using two possible combinations of end vertices.

In the *BPG* that has two genomes Γ and Π, the vertices and the edges of a closed walk form a cycle. In Fig 9, the walk $(1,(1;3),3,(3;4),4,(4;2),2,(2;1),1)$ is a cycle. A vertex v is π-*open* (γ-*open*) if $v \notin \Gamma$ $v \notin \Pi$. An unclosed walk in *BPG* is a path.

Based on different kinds of end points of the paths, we can classify paths into different types. If the two ends of a path are *CAP* vertices, we simply denote this path as p^0, if a path is ended by one open vertex and one *CAP*, we denote it as p^π (p^γ). If a path is ended by two open vertices, it is denoted by the type of its two open vertices, for example, $p^{\pi,\gamma}$ represent a path that ends with a π-*open* vertex and a γ-*open* vertex. In Fig 9, the walk $(9,(9;0),0,(0;CAP),CAP)$ is a p^γ path, and the walk (11,(11;5),5,(5;12)12) is a $p^{\gamma,\pi}$ path. A path is even (odd), if it contains even (odd) number of edges. In [37], the *DCJ* distance between two genomes with *Indels* but without duplications is calculated by equation (2). From this equation, we can easily get the distance between Γ and Π in Fig 9 as 4.

$$\begin{aligned} distance_{indel}(\Gamma,\Pi) = N - [c + p^{\pi,\pi} + p^{\gamma,\gamma} + \lfloor p^{\pi,\gamma} \rfloor] \\ + \frac{1}{2}(p^0_{even} + min(p^\pi_{odd}, p^\pi_{even}) + min(p^\gamma_{odd}, p^\gamma_{even}) + \delta) \end{aligned} \tag{2}$$

Where $\delta = 1$ only if $p^{\pi,\gamma}$ is odd and either $p^\pi_{odd} > p^\gamma_{even}, p^\gamma_{odd} > p^\gamma_{even}$ or $p^\pi_{odd} < p^\gamma_{even}, p^\gamma_{odd} < p^\gamma_{even}$; Otherwise, $\delta = 0$.

An imortant thing to notice is, mathematically optimized distance might not reflect the true number of biological events, distance estimation methods such as *EDE* or *IEBP* are used to rescale these computed distances [101].

2.2.2 Using *DCJ-Indel-Exemplar* Distance to Evaluate Dissimilarity

For input of two genomes, an *exemplar* string is constructed by deleting all but one occurrence of each gene family. The exemplar distance is the distance that among all possible exemplar strings, the minimum distance that one exemplar string returns. Figure 11 shows an example of transforming genomes with duplicated genes into exemplar genome strings. In [123] the author present a branch and bound algorithm to compute the exemplar distance.

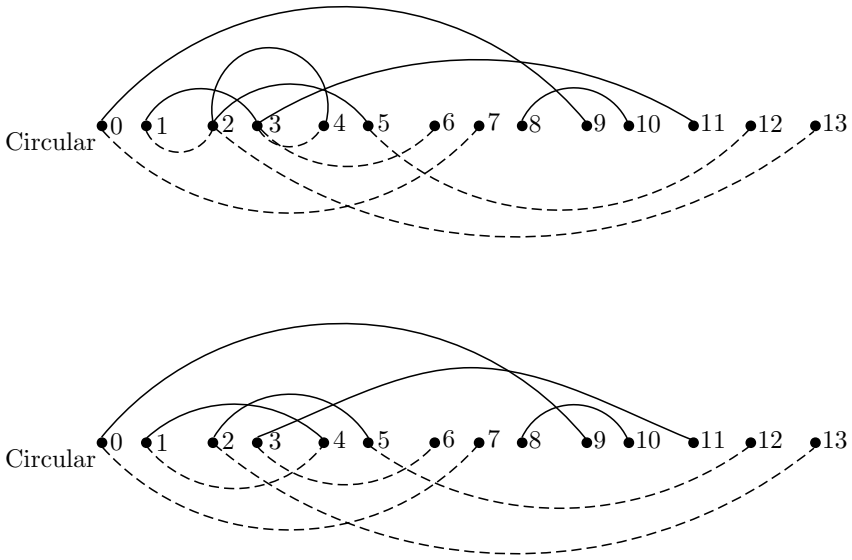


Figure 11: An example of exemplar mapping from genome Γ (1, -2, 3, 2, -6, 5) and genome Π (1, 2, 3, 7, 2, 4) to Γ (1, 3, 2, -6, 5) and genome Π (1, 3, 7, 2, 4).

The *DCJ-Indel* distance can handle genomes which only have *Indels*, while the exemplar distance can only handle duplications. In this section, we discuss the method that combine these two distances together to handle genomes with both *Indels* and duplications. We call this distance *DCJ-Indel-Exemplar* distance. The idea is, after finding a one-to-one mapping of duplicated genes between two genomes, we get an exemplar genome pair that have *Indels*. Of all exemplar pairs, find the one with minimum *DCJ-Indel* distance, and return this distance as the *DCJ-Indel-Exemplar* distance. The *DCJ-Indel-Exemplar* distance does not reflect the true number of evolutionary events. For one thing, the number of duplications are not counted, for another, when there are large number of mutations, *DCJ* distance will underestimate the distance. Therefore, two steps are followed to adjust the *DCJ-Indel-Exemplar* distance. The first step is use *EDE* [101] to rescale the distance. The second step is to add the count of duplicated genes. The *DCJ-Indel-Exemplar* distance after the adjustment of *EDE* distance and the addition of number of duplications is set as the final distance. The *DCJ-Indel-Exemplar* distance after the adjustment of *EDE* distance and the addition of duplication counts is the final distance. The algorithm

Algorithm 3: DCJIndelExem

Input: G_1 and G_2

Output: Minimum distance d

$G_1^{'}, G_2^{'} \leftarrow$randomly init mapping ;
    of all duplicated genes of G_1, G_2;
$G_1^{*}, G_2^{*} \leftarrow$remove all duplicated genes of G_1, G_2;
$min\_ub \leftarrow DCJIndel(G_1^{'}, G_2^{'})$;
$min\_lb \leftarrow DCJIndel(G_1^{*}, G_2^{*})$;
Init search list L of size $min\_ub - min\_lb$ and insert G_1, G_2;
while $min\_ub > min\_lb$ **do**
    $G_1^{+}, G_2^{+} \leftarrow$ pop from $L[min\_lb]$;
    for *pair* $\in$ *all mappings of next available duplicated gene* **do**
        $G_1^{+}, G_2^{+} \leftarrow G_1^{+}, G_2^{+}$ fix the mapping of *pair* $G_1^{+'}, G_2^{+'} \leftarrow$randomly init mapping ;
            of rest duplicated genes of G_1^{+}, G_2^{+};
        $G_1^{+*}, G_2^{+*} \leftarrow$remove rest duplicated genes of G_1^{+}, G_2^{+};
        $ub \leftarrow DCJIndel(G_1^{+'}, G_2^{+'})$;
        $lb \leftarrow DCJIndel(G_1^{+*}, G_2^{+*})$;
        if $lb > min\_ub$ **then**
            discard G_1^{+}, G_2^{+}
        if $ub < min\_ub$ **then**
            $min\_ub = ub$;
        else if $ub = max\_lb$ **then**
            return $d = ub$;
        else
            insert G_1^{+}, G_2^{+} into $L[lb]$
return $d = min\_lb$;

is described in Algorithm 3.

Because the computation of *DCJ-Indel-Exemplar* distance is *NP-hard*, we need to find a low cost strategy to compute it efficiently. The computation of an exemplar distance is to set one specific edge for each vertex in *BPG*. If a gene family has multiple copies of the gene, its according two vertices (*head* and *tail*) in *BPG* will have degree of more than 1, and there will be multiple choices picking one edge. Once we set a "exemplar", only the edges of the vertices that have degree of more than one have been changed, Those vertices that has degree of only one are always

stable. We can actually see the computation of the *DCJ-Indel-Exemplar* distance as the process of computing number of cycles/paths on a streaming breakpoint graph [152], We can bridge out all the vertices that are stable and just leave those vertices that are changing. And the result of computation on the changed graph will stay the same. Suppose there are V vertices in the *BPG*, and V_s stable vertices and V_d dynamic vertices, the speedup will be $\frac{V}{V_d}$.

2.2.3 Using *DCJ-Indel-CD* Distance to Evaluate Dissimilarity

Suppose there are two genomes with the same gene content: each gene family has the same number of genes in both genomes. Assuming that two genomes *G1* and *G2* have the same gene content. A bijection between *G1* and *G2* is defined as it specifies n homologous gene pairs, where n is the number of genes in each genome. If *G1* and *G2* contain only singleton gene families (exactly one gene in each family in each genome), then there is a unique valid bijection between *G1* and *G2*, and the *DCJ* distance between *G1* and *G2* can be computed in linear time [150]. If *G1* and *G2* contain gene families with multiple genes in each genome, then there are many valid bijections between *G1* and *G2*. Different valid bijections define different one-to-one correspondences between homologous genes, yielding possibly different *DCJ* distances between *G1* and *G2*. We study the following generalized *DCJ* distance problem: given two genomes *G1* and *G2* with the same gene content, find a valid bijection between *G1* and *G2* that minimizes the *DCJ* distance. We denote the generalized *DCJ* distance between *G1* and *G2* as $d(G1, G2)$. Because finding bijections of gene families between two genomes can be view as the *maximum cycle decomposition problem*. The problem can be view as: given two genomes and their breakpoint graph representation, find a cycle decomposition that minimize the number of resulting cycles in the *BPG*. Figure 12 shows an example of two genomes with duplication on gene family 2. A

cycle decomposition is given by mapping the first gene 2 of genome Γ to the second gene 2 of genome Π. There is Integer Linear Programming method compute the exact solutions for this problem [98]. Which is based on constraints that all singleton gene families will be present in the final solution and the cycle selection can correctly reflect the gene bijection.

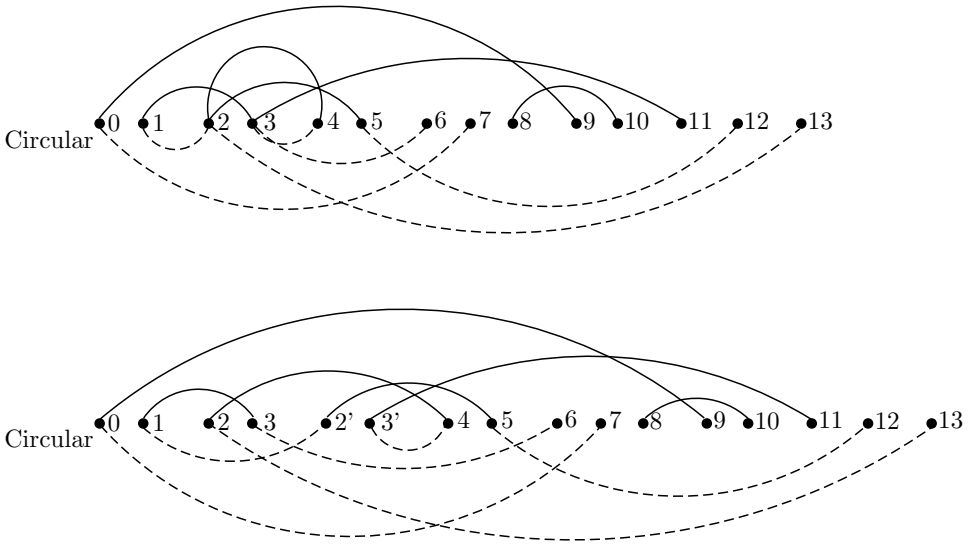


Figure 12: An example of cycle decomposition results for a bijection from genome Γ (1, -2, 3, 2, -6, 5) and genome Π (1, 2, 3, 7, 2, 4) to Γ (1, -2, 3, 2', -6, 5) and genome Π (1, 2', 3, 7, 2, 4).

There are some restrictions with the previous maximum cycle decomposition work. For one thing, it restrict that all duplicated gene families should have exactly the same number of genes in each genome; for another, it does not allow genomes to have *Indels*. Here, we introduce the distance we called *DCJ-Idel-CD* distance. Which is formally described as, given a *MBG* representations of two genomes which contains both *Indels* and duplications (there might be different number of duplication copies in each genome), find a *maximum cycle decomposition* in this *MBG* that minimize the *DCJ-Indel* distance of the resulting *MBG*.

We can use the similar methods in *DCJ-Indel-Exemplar* distance algorithm to design a branch-and-bound algorithm to solve the *DCJ-Idel-CD* distance problem. To begin

with, we need to specify the way to compute upper and lower bound. The computation of upper bound is easy, once we fixed some duplicated gene families' bijection, we can randomly initialize other duplicated gene families' bijection, and compute a *DCJ-Indel* distance as the upper bound. For the lower bound, we can prove that by removing all the occurrences of unfixed duplicated gene families, the resulting *DCJ-Indel* distance is monotony decreasing, therefore we can use this distance as the lower bound. The pseudo code for the branch and bound algorithm is as Algorithm 4 shows. In [98], they mentioned that the cycles of length two can be directly fixed without being considered different possible combinations of bijections. It's also applicable to our algorithm.

2.2.4 Experimental Results

2.2.4.1 Results for DCJ-Indel-Exemplar *Distance*

We simulated the data sets using genomes with 200 genes. To show how *Indels* and duplications affect the estimation of the distance, we divide the data set into multiple groups with varied *Indels* rate (γ, which varied from 0% to 10%), and duplication rate (ϕ, which varied from 0% to 10% as well). For each *Indels* or duplication event, only one gene is inserted/deleted or duplicated. We compare the change of distance estimation with the change of mutation rate (θ, which varied from 10% to 100%), with one specific setting of γ and ϕ. We used reversal operation to simulate the mutation mainly because *DCJ* distance and reversal distance is quite similar when using genome data of same contents.

The result for *DCJ-indel-exemplar* distance and *DCJ-indel-exemplar* distance corrected by *EDE* are shown in Fig 13. As for the impact of different evolution operation rates, the main factor that affect the accuracy of distance estimation is the change of rate γ. This is mainly because an *Indel* after a duplication can offset the count

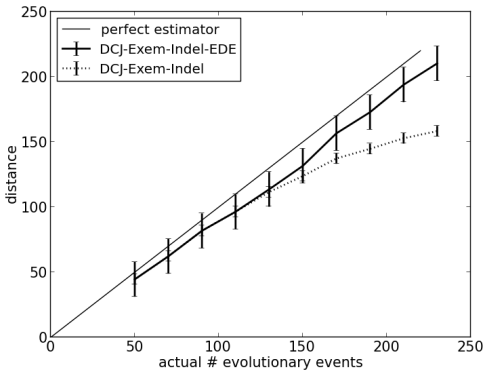
Algorithm 4: DCJIndelCD

Input: G_1 and G_2

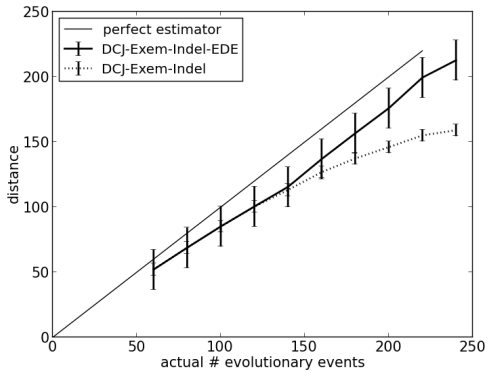
Output: Minimum distance d

G_1, G_2 fix the cycle of length two ;
G_1', G_2' ←randomly init bijection ;
    of all duplicated genes of G_1, G_2;
G_1^*, G_2^* ←remove all duplicated genes of G_1, G_2;
$min\_ub \leftarrow DCJIndel(G_1', G_2')$;
$min\_lb \leftarrow DCJIndel(G_1^*, G_2^*)$;
Init search list L of size $min\_ub - min\_lb$ and insert G_1, G_2;
while $min\_ub > min\_lb$ **do**
    $G_1^+, G_2^+ \leftarrow$ pop from $L[min\_lb]$;
    for *pair* $\in$ *all bijection of next available duplicated gene* **do**
        $G_1^+, G_2^+ \leftarrow G_1^+, G_2^+$ fix the bijection of *pair* $G_1^{+'}, G_2^{+'}$ ←randomly init bijection ;
            of rest duplicated genes of G_1^+, G_2^+;
        G_1^{+*}, G_2^{+*} ←remove rest duplicated genes of G_1^+, G_2^+;
        $ub \leftarrow DCJIndel(G_1^{+'}, G_2^{+'})$;
        $lb \leftarrow DCJIndel(G_1^{+*}, G_2^{+*})$;
        if $lb > min\_ub$ **then**
            discard G_1^+, G_2^+
        if $ub < min\_ub$ **then**
            $min\_ub = ub$;
        else if $ub = max\_lb$ **then**
            return $d = ub$;
        else
            insert G_1^+, G_2^+ into $L[lb]$
return $d = min\_lb$;

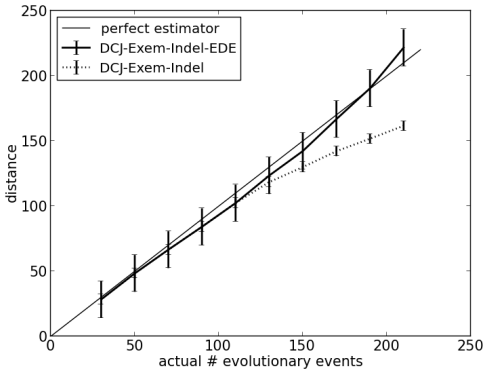
of both *Indel* and duplication and makes the distance underestimated. Both θ and ϕ seem to have little effect on the distance estimation. As for the effect of different distance estimation models, the result indicates that when the *DCJ* rate is moderate, both distances are close to the true result. While with the *DCJ* rate increases, the *EDE* corrected distance is more accurate than the *DCJ-indel-exemplar* distance with the growth of θ. *EDE* tends to underestimate distances when γ is 0%, but tends to overestimate them when $\gamma > 0\%$



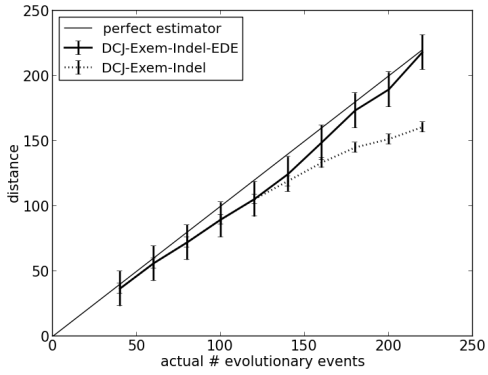
(a) $\gamma = 0.0, \phi = 0.05$



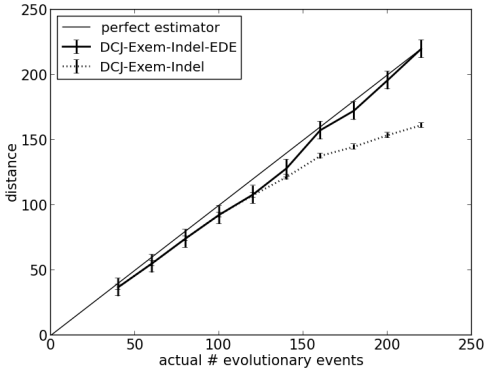
(b) $\gamma = 0.0, \phi = 0.1$



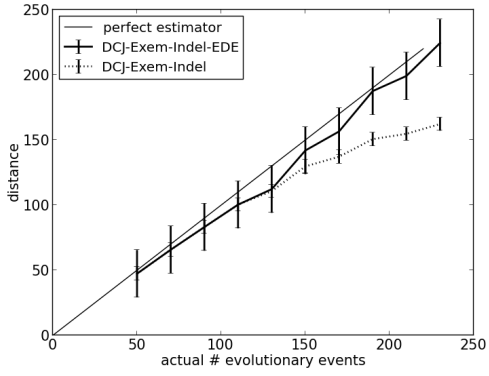
(c) $\gamma = 0.05, \phi = 0.05$



(d) $\gamma = 0.05, \phi = 0.1$

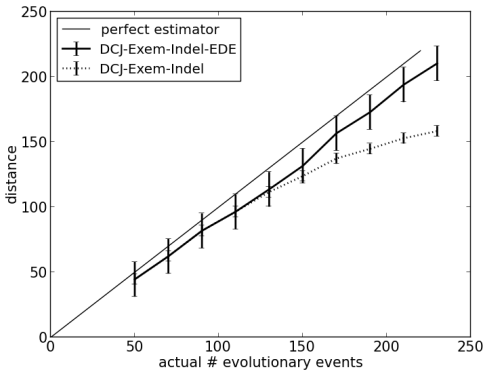


(e) $\gamma = 0.1, \phi = 0.05$

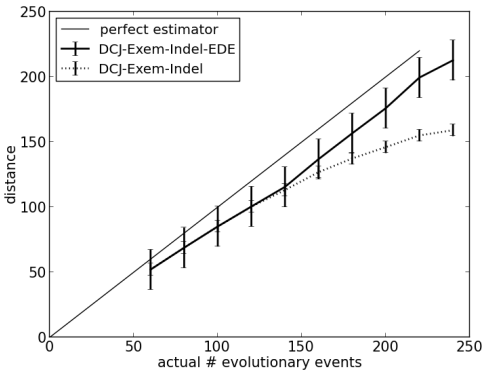


(f) $\gamma = 0.1, \phi = 0.1$

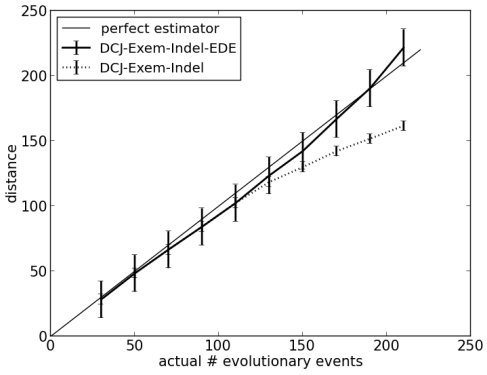
Figure 13: Distance computation results, the x-axis represents the actual number of *DCJ* operations and the y-axis represent the computed distance for the methods using *DCJ-indel-exemplar* distance, *DCJ-Indel-Exemplar* distance corrected by *EDE*, and the true estimator.



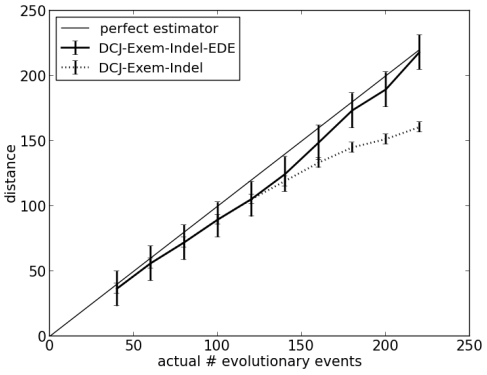
(a) $\gamma = 0.0, \phi = 0.05$



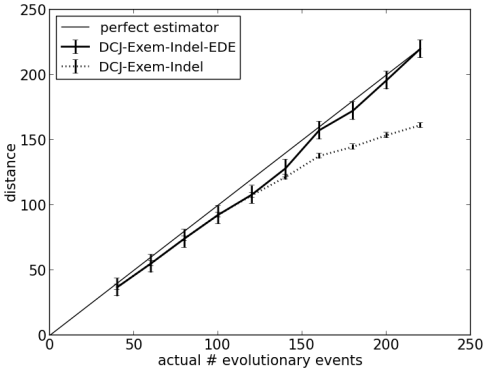
(b) $\gamma = 0.0, \phi = 0.1$



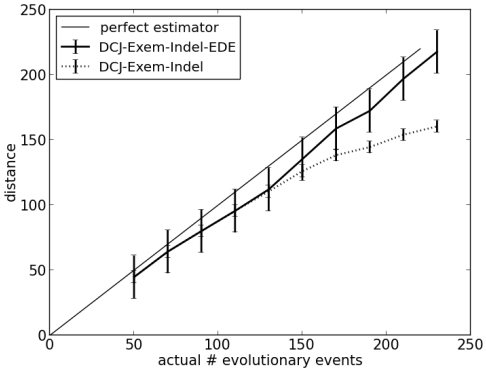
(c) $\gamma = 0.05, \phi = 0.05$



(d) $\gamma = 0.05, \phi = 0.1$



(e) $\gamma = 0.1, \phi = 0.05$



(f) $\gamma = 0.1, \phi = 0.1$

Figure 14: Distance computation results, the x-axis represents the actual number of *DCJ* operations and the y-axis represent the computed distance for the methods using *DCJ-indel-CD* distance, *DCJ-indel-CD* distance corrected by *EDE*, and the true estimator.

2.2.4.2 Results for DCJ-Indel-CD *Distance*

We run the experiment using the same data sets as in the *DCJ-indel-exemplar* distance experiment. The result for *DCJ-indel-CD* distance and *DCJ-indel-CD* distance corrected by *EDE* are shown in Fig 14. We can see that the result for *DCJ-indel-CD* distance and *DCJ-indel-exemplar* distance are quite similar, with the *DCJ-indel-CD* distance a little bit of more under estimate the distance. This is mainly due to the reason that for the *DCJ-Indel* distance, when computing the number of duplication events, if the occurrences of a duplicated gene family are ordered consecutively, the algorithm will output distance as one single *Indel* consequently. While the actual number of duplication events might be more than one. Another reason is the same as the under estimate of the *DCJ-indel-exemplar* distance, which is rooted from the process in the simulation, the *Indels* events will offset the duplication events. For example, when a duplication happened, the following *Indel* event might delete the duplicated gene. These two events will be counted in the simulation but not be detected in the distance computation algorithms. *DCJ-Indel-Exemplar* distance has the advantage of less search space reuirement, because it only need one to one mapping intead of multiple to multiple mapping for *DCJ-Indel-CD* distance. But considering that *DCJ-Indel-Exemplar* method is exact algorithm, and its accuracy is not dependent on the estimation on duplication events based on counting of number of different genes in a single gene family. It has its merit of being applicable to more complex scenarios.

Chapter III

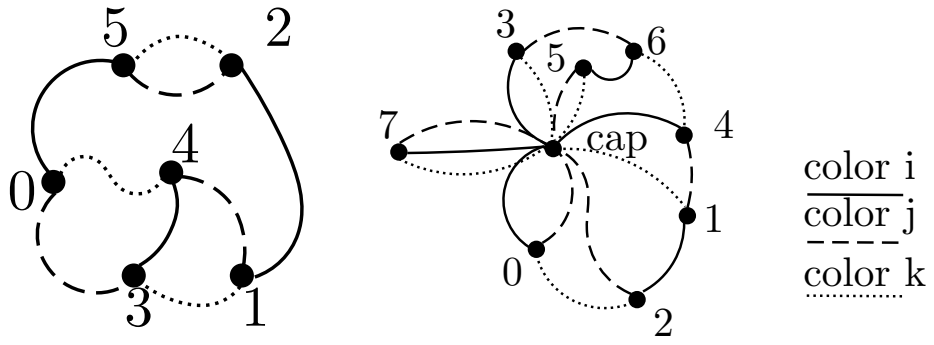
MEDIAN COMPUTATION

3.1 Introduction to Genome Median Problem under DCJ *Criteria*

We firstly introduce the following concepts: **Multiple Breakpoint Graph (*MBG*)**: Given multiple genomes which have the same set of non-duplicated genes, we can define a *MBG* by using different type of edges to represent different genomes. If genomes consist only of circular chromosomes, the constructed graph is a MBG. Figure 15(a) shows the *MBG* for three gene orders ((1,2,3);(1,3,2);(1,-2,-3)). It's easy to notice that MBG is 3-regular graph.

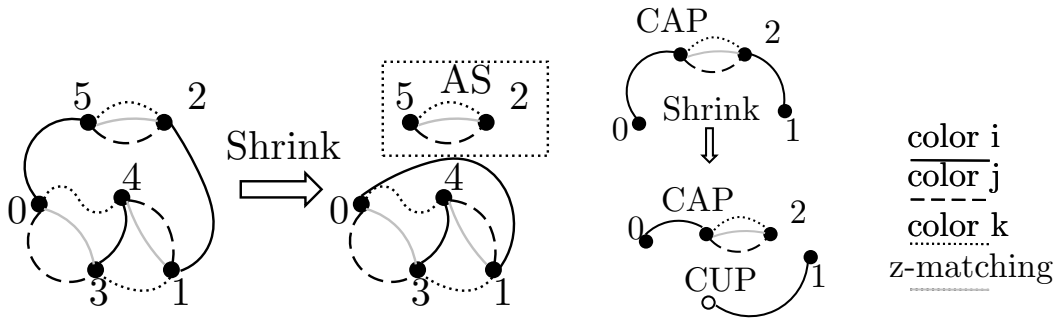
Capped Multiple Breakpoint Graph (*CMBG*): If genomes consist of single or multiple linear chromosomes, the constructed graph is $CMBG$. Figure 15(b) shows the *CMBG* for three gene orders that each are consisted of two chromosomes ((1,2;3,4);(1,3;2,4);(-1,2;-4,3)) (';' indicates the end of a chromosome). Other than CAP vertex, every vertex in $CMBG$ has degree of 3. If not specified, we use *BPG* to represent all these three classes of graphs.

Adequate Subgraphs (*ASs*): *ASs* defined in [146, 142] are such graphs that if a *BPG* is partitioned into two parts, one part is an *AS*, the other part is the rest of the graph, there will be no *0-matching* edges connecting these two parts. *ASs* can be used to partition the *BPG* and help to find the *DCJ median* without losing accuracy.



(a) MBG (b) CMBG

Figure 15: Examples of Breakpoint Graphs.



(a) 0-matching, AS, shrinking on edge without CAP vertex. (b) shrinking on edge with CAP vertex.

Figure 16: Examples of operations on BPG.

Edge Shrinking: *ASs* can be bridged out from the *BPG* by edge shrinking operations. When an *AS* is selected to partition the graph, its possible *0-matching* edges are selected at first. Then, if one *0-matching* edge does not contain *CAP* vertex, edges of the same type that incident to the same *0-matching* edge will be joined into a single edge; If the *0-matching* edge contains a *CAP* vertex, the edge incident to the non-*CAP* vertex will be connected to a *CUP* vertex. Figure 16(a) shows examples of *0-matching*, *AS*, and edge shrinking on the edge that does not contain *CAP* vertex. Figure 16(b) shows an example of edge shrinking on edge that contains a *CAP* vertex.

Edge Expansion: Edge expansion is the reverse operation of edge shrinking.

Best-first-search based Branch and Bound Algorithm (A* BnB): The best algorithm to solve the *DCJ* median problem is an *A* BnB* algorithm. When searching on a search node represented by a *BPG* with v vertices, **first** it traverses the *BPG* to detect *ASs*, which is a classic sub-graph isomorphism problem [139]. Since this step only detects small *ASs* of limited patterns, special algorithms have been developed which all have time/space complexity of $O(v)$; **second**, if there are *ASs* found, one or two child search nodes will be expanded by shrinking the *0-matching* edges in *ASs*, otherwise there will be $v-1$ child search nodes expanded. At the very beginning of this search process, there usually are a lot of *ASs* in *BPG*, but after a few steps, *BPG* will be altered to a state that for the first time there is no *ASs* exists in it, we call it *BPG kernel*, and the number of vertices in *BPG kernel* is marked as κ. When the searching process reaches *BPG kernel*, it's hard to generate new *ASs* by shrinking "assumed" *0-matching* edges, and the following search processes have a branching factor decided by κ. This step has time/space complexity of $O(v)$; **third**, each *BPG* of the expanded child search node will be traversed to get cycle/path number for updating the upper and lower bounds. In this step, if *ASs* are detected, the time/space complexity is $O(v)$. In contrast, if there is no *ASs* detected, the time/space complexity become $O(v^2)$.

The *DCJ* median algorithm can be briefly described by a branch and bound (*BnB*) process [146, 143, 142] on *MBG*, which we show an example in Fig 17. To begin with, the purpose of the *DCJ* median algorithm is to find a maximum matching in *MBG*, which we called 0-*matching*, and it is represented by grey edges in the example. If we use m to represent a possible median solution and M to represent the best median, the 0-*matching* for the final solution should satisfy the property $M = \underset{m}{argmin}\ distance(m, i)\ i \in (1, 2, 3)$.

The first step in the *BnB* process is to decompose the graph by using Adequate

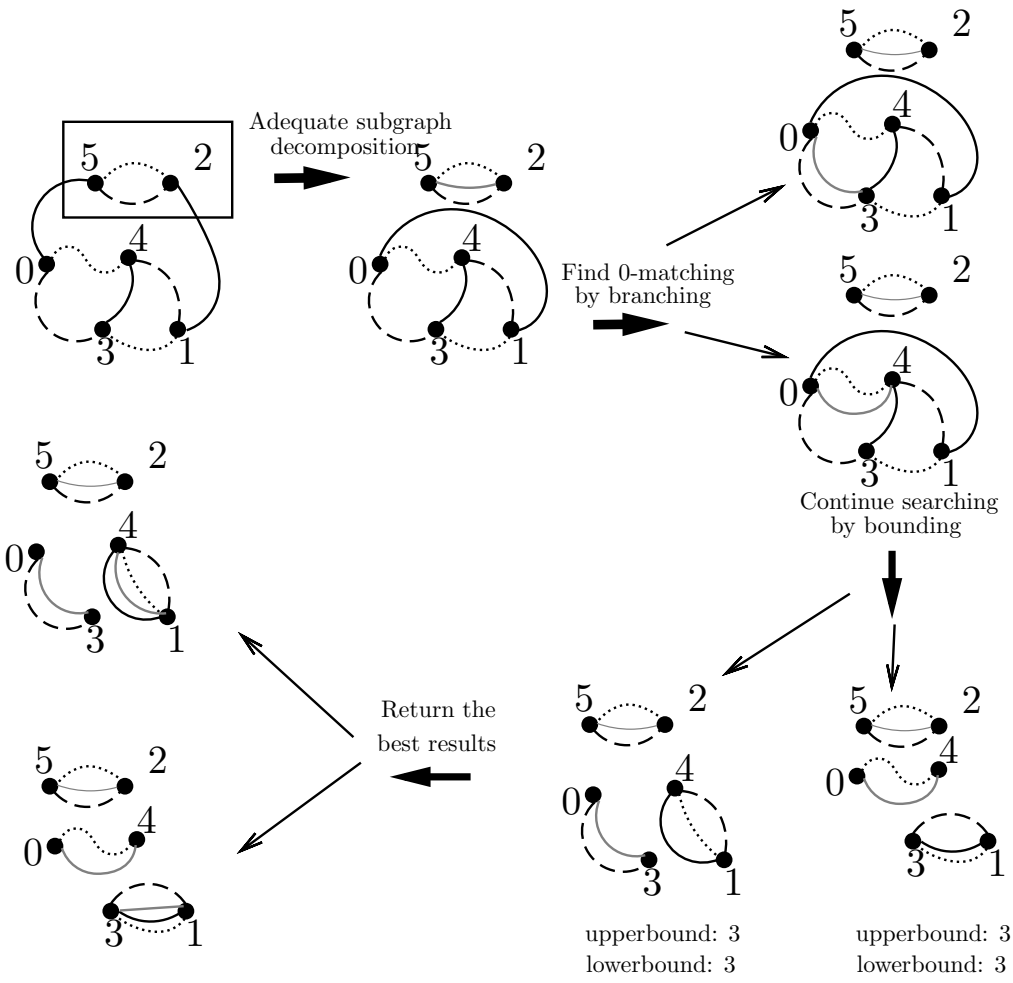


Figure 17: Example of multiple breakpoint graph (*MBG*) for gene order (1,2,3) which is represented by solid edges;(1,3,2) which is represented by dashed edges;(1,-2,-3) which is represented by dotted edges. The branch and bound (*BnB*) process for computing median genome is shown in the figure.

Sub-graphs (*ASs*) [146, 143, 142]. The decomposition is achieved by performing edge shrinking operation. In Fig 17, example of *AS* including vertex 2 and 5 is decomposed through edge shrinking; once an *AS* is decomposed, the according 0-*matching* is set.

The second step is to branch the remaining possible 0-*matching* edges. A new *MBG* is generated by shrinking one of possible 0-*matching* edges. The upper and lower bound for the new *MBG* are calculated by using the equation (3) and (4). Where N is half the number of vertices in the original *MBG*, n is half the number of vertices in the

new *MBG*, $cycles(m, i)$ is the number of cycles already formed between 0-*matchings* m and the input genome i, and $distance(i, j)$ is calculated using the new *MBG*. In Fig 17, when 0-*matching* (0; 3) is chosen, the corresponding $upperbound = 3$ and $lowerbound = 3$

$$upperbound = 3(N - n) - \sum_{i \in (1,2,3)} cycles(m, i) + \sum_{i,j \in (1,2,3)} distance(i, j) - \min_{i,j \in (1,2,3)} distance(i, j) \quad (3)$$

$$lowerbound = 3(N - n) - \sum_{i \in (1,2,3)} cycles(m, i) + \frac{\sum_{i,j \in (1,2,3)} distance(i, j)}{2} \quad (4)$$

Continue the *BnB* process, until $upperbound = lowerbound$, then the optimal 0-*matching* is found; the corresponding gene order representation is the output median genome. In Fig 17, there are two optimal median genome results, which are: (-1,-2,-3) and (1,-2,-3).

3.2 Using Streaming Breakpoint Graph Analysis Methods to Accelerate the Median Computation

The concept of κ is introduced in the the description of $A*BnB$. One very important point to notice is that, κ determines the performance of this $A*$ *BnB* algorithm. To begin with, the algorithm's branching factor is decided by κ. In addition, if there is no *ASs* in a search node, the third step in the $A*$ *BnB* will have complexity of $O(v^2)$, which makes the complexity for processing this search node $O(v^2)$. Last but not least, if κ is large there will be a huge number of *BPGs* generated for child search nodes, which makes this algorithm additionally bound by *I/O*. However, most of the operations on *BPG* is minor, which only involves change on one edge, by utilizing the

properties of streaming *BPG*, it is possible to reduce both time and space requirement for processing a search node.

3.2.1 Compressed Data Structures for Memory and I/O Efficiency

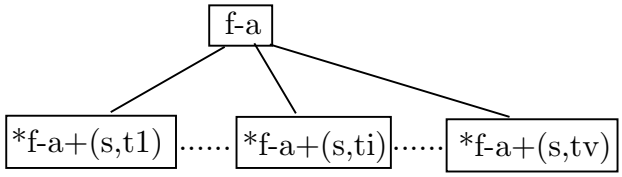


Figure 18: Child node i, just need to store a pointer to *f-a*, and an edge *(s, ti)*, need $O(1)$ storage.

In the current *DCJ* median algorithm, every time when there are new child search nodes expanded, new *BPGs* will be allocated and pushed into a search list. As a result, huge amount of memory allocation and *I/O* can not be avoided. Because each *BPG* of a child search node is generated from shrinking edges of the *BPG* of the root search node following several steps, we can store these edges as "footprints" instead of *BPGs* themselves. Here, because we only consider search nodes from the *BPG kernel* stage (because processing node before this stage is fast), every search node could only store footprints after the *BPG kernel* stage. In other words, there is only one *BPG* in the memory, when finished processing search node a with footprint *f-a*, and continue searching from another search node b with footprint *f-b*, edges of *BPG* in *f-a* are expanded then edges in *f-b* are shrunk to switch from node a to node b. In addition, if there is no *AS* found in the parent search node, the expanded $v-1$ child search nodes will all share the same footprint of their parent. Under such circumstance, the cost of storing a child search node can be additionally contracted to only 1 edge and a pointer to the footprint of their parent search node.

3.2.2 A Fast Method to Update Cycle/Path Numbers

In the current *DCJ* median algorithm, every vertex needs to be visited once to estimate cycle/path numbers for bounding. However, when the *BPG* of a child search node is generated by just shrinking one edge from the *BPG* of its parent node, we have the following observations (see Figure 19 for example.) to help design quick method to update cycle/path numbers. Suppose shrinking one *0-matching* edge connecting two vertices a and b (a and b are not neighbors, because if they are neighbors after shrinking, they will be bridged out from the *BPG* and there will be no change in the cycle number), the i and j color edge of a is connected to vertex x and y, the i and j color edge of b is connected to vertex w and z.

Observation 1 *If a and b are in different connected components (ij_c or* path*), and at least one vertex is in a ij_c, after shrinking, the number of ij_c will decrease by 1.*

Proof 1 i) *If a and b are both ij_c, To prove that the number of ij_c is reduced by 1 is to show that after shrinking, x, y, w and z are connected. Case 1, if x, y are the same vertex x' (overlapped) and w and z are the same vertex w', after shrinking both i and j color edge of x' will be joined with i and j color edge of w', x' and w' are connected. The same proof for when Case 2, if x and y (or w and z) are the same vertex x' (or w') and w and z (or x and y) are the different vertex. Case 3, if x, y, w and z are different vertex* **ii)** *suppose a (b) is in a ij_c and b (a) is in a path, after shrinking, x, y, w and z are connected, because x and y (w and z) are connected with* CAP/CUP *vertex, so the cycle number will decrease 1.*

Observation 2 *If a and b are in the same cycle, after shrinking, the number of ij_c could be changed by either increase 1, or stay the same).*

Proof 2 *To prove that the number of ij_c changed is ≤ 1 is equivalent to show that*

after shrinking, there are at most two connected component generated. Case 1, if x, y, w and z are different vertices, after shrinking, x and w will be connected with i color edge while y and z will be connected with j edge, there are at most two connected components. The same proof goes for when Case 2, if x and y (or w and z) are the same vertex x' (or w') and w and z (or x and y) are the different vertex. Case 3, if x, y are the same vertex x' (overlapped) and w and z are the same vertex w'.

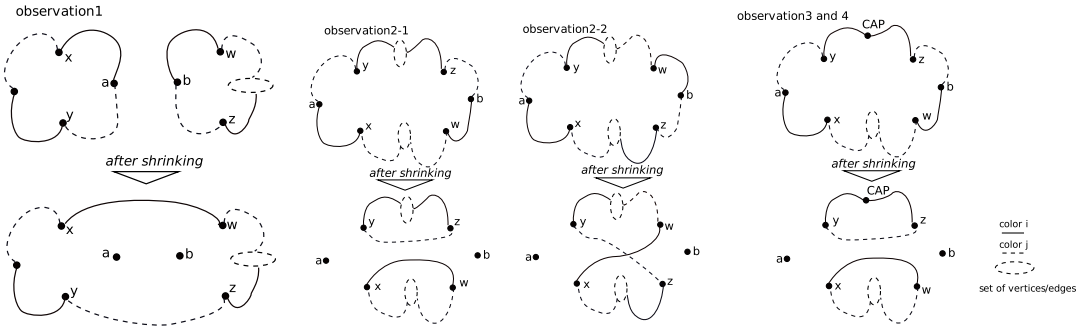
Observation 3 *If a and b are in the same path, after shrinking, the number of cycles will be increase by 1 if and only if x and w are in the same adjacent vertices set after the component is separated by a and b.*

Proof 3 *Sufficiency: if x and w are in the same separated component, after shrinking, i color edge of both x and w will be joined together, and there will be no other edge connecting to another component, and the cycle number will increase one. Necessity: if x and w are in the different separated component, after shrinking, these two separated component will keep connected, because i color edge of both x and w will be joined together, and the cycle number will not increase.*

Observation 4 *If at least one of a and b are in a path, after shrinking, the number of path stays the same.*

Proof 4 *Because a path starts from a* CAP/CUP *vertex and ends with a* CAP/CUP *vertex, the number of paths equals to half of the number of edges connected with* CAP/CUP*. 1) when a (b) is not connected with* CAP/CUP*, after shrinking the number of edges connected with* CAP/CUP *will not change. 2) when a (b) is connected with* CAP/CUP*, after shrinking the neighbor vertex (not* CAP/CUP*) of a (b) will connect with* CAP/CUP*, and the number of edges connected with* CAP/CUP *will not change. 3) when a (b) is* CAP*, after shrinking the neighbor vertex of a (b), will be connected with* CUP*, and the number of edges connected with* CAP/CUP *will not*

change. Based on 1-3) the number of path will not change after shrinking.



(a) An example of observation 1.

(b) examples of observation 2.

(c) An example of observation 3 and 4.

Figure 19: Examples of different observations when only one 0-matching is shrunk.

update_cycle_fast(a, b)

if a and b in different cycle **then**
  cycle number decrease 1;
else {a or b is in a cycle}
  x=neighbor$[a][i]$;
  w=neighbor$[b][i]$;
  if x and w are in the same
  adjacency vertices set **then**
    cycle number increase 1;
  end if
end if

update_path_type_fast(a, b)

if a and b in different path **then**
  update path types;
else {a or b is in a path}
  x=neighbor$[a][i]$;
  w=neighbor$[b][i]$;
  if x and w are in the same
  adjacency vertices set **then**
    cycle number increase 1;
    update path types;
  end if
end if

Based on Observation 1, 2 and 3, we can update the cycle number in $O(1)$ time if only one *0-matching* edge is shrunk. Based on Observation 4, it's possible to visit only the connected component of affected vertices to determine the change of path type. The summarized fast algorithm for updating cycle/path number is shown in *update_cycle_fast*(a, b) and *update_path_fast*(a, b) separately.

3.2.3 Heuristics for Reducing Search Space

In a recent paper [118], they designed approximation algorithm to compute *DCJ* median, though *DCJ* median problem is *APX-hard*, the algorithm achieved a good approximation rate for reverse median problem [30] (using *DCJ* median algorithm to approximate reverse median problem). This method can be served to provide initial tighter lower bound, even though it might not be helpful for reducing search space, it can be useful in reducing space requirement, because those branched search node whose upper bound is smaller than the global maximal lower bound will be discarded, and the initialization of a tighter global lower bound can increase this threshold. In this paper, we introduce a branching strategy, which is helpful to quickly exhaust search nodes with the maximal upper bound and approach lower bound faster. In [143], the author state that the upper bound is $upperBound = c + \frac{3}{n} + \frac{c_{1,2}+c_{1,3}+c2,3}{2}$ (actually, this formula is just for circular chromosomes, but we can extent it to linear chromosomes). Suppose vertex V_k is in ijc_k and we define *cycle rank sum of* V_k as $R(V_k) = \sum_{(i,j)\in(1,2,3),i\neq j} |ijc_k|$.

Proposition 1 *If there is no* AS *detected, and we select the vertex* V_k *in* BPG *such that* $R(V_k) = min(R(V_i))(V_i \in BPG)$*, and shrink this vertex with other vertices in* BPG *to generate* $|2v-1|$ *intermediate child* BPG*s, the number of child* BPG*s with upper bound value that is equal to the global maximal upper bound is minimized.*

Based on Observation 1, we know that if two vertices are in different component (cycle), after shrinkage the total cycle number will decrease by 1, since vertex V_k with $R(V_k) = min(R(V_i))(V_i \in BPG)$ has the least number of vertices that share component(cycle) with it, then shrink this vertex with all other vertices will cause the maximal number of cycles to merge, and the upper bound will decreased.

3.2.4 Experimental Results

We conduct our experimental tests of the streaming breakpoint graph based algorithm engineering methods on a machine using the linux operating system with 16 Gb of memory and an *Intel Xeon CPU E5530* 16 core processor, each core has 2.4GHz of speed. Because the existing *DCJ* algorithms (both for circular and linear chromosomes) are implemented using *JAVA* [146, 142, 145], we also implement our algorithm using *JAVA*, all of the source code are compiled with *JDK1.7* with -O option.

3.2.4.1 Time/space usage

In [146, 142, 145], the authors analyzed the performance of the original *DCJ* median algorithm with a simplified model, i.e. they only allowed reversal as the event and assumed that the *DCJ* distance between each of the three species is the same. In this paper, we generate simulated data using a model which is biologically more accurate. First we generate phylogenetic model trees of three genomes by using a birth-death model [89]. Then, based on the model tree, we execute *DCJ* operations on each genome for some random times, then we select the generated data of a specific κ, we collect 10 data samples for each κ. For the test of time/space complexity, we run the program to process only $10k$ search nodes then return. In the experiment, we have a parameter called *thresh*, which stands for the threshold for the number of graphs stored in the memory, if the number exceeds this threshold, these graphs will be stored back to the disk. We have tested the threshold for $2k$ and $10k$ nodes.

In Figure 20, combined with the total time, we also recorded four types of time: time to detect adequate sub-graphs, time to store and retrieve graphs, time for *I/O* and time to update lower/upper bounds. We can see from the figure that for the original algorithm, under all of cases, the time grows quadratic (since the time for the original

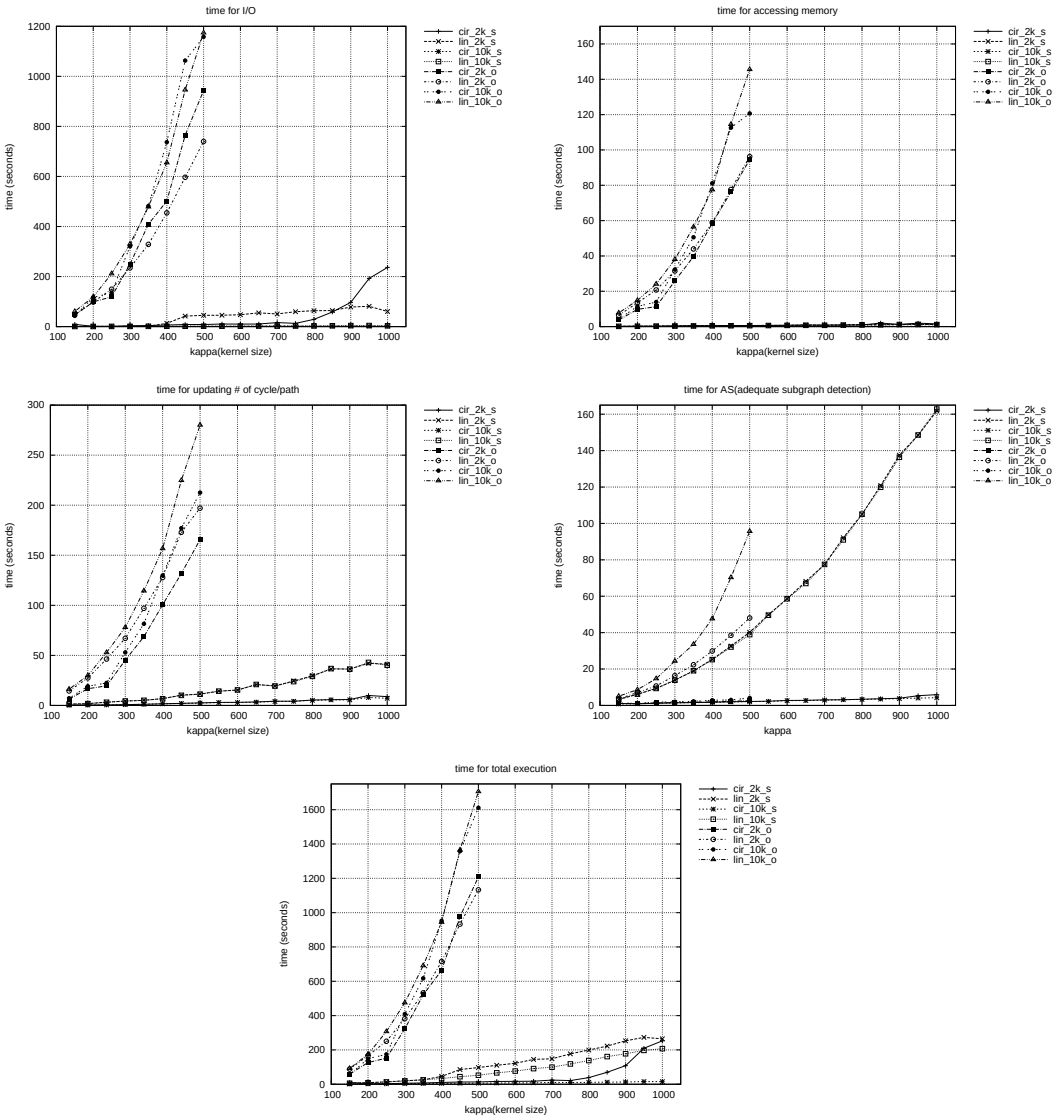


Figure 20: Time complexity comparison for our streaming algorithm (-s) and the original algorithm (-o).

algorithm to run larger data ($\kappa > 500$) is extremely huge, we do not include these results), and for our stream algorithm, the time grows linearly. This is also true for the separate time of *I/O*, memory access and bound update. Another thing is, when the threshold is increased from $2k$ to $10k$, our stream algorithm time reduces a lot on both circular and linear chromosome cases, but as for the original algorithm, it even takes more time. In addition, we have also noticed the reduce of time for detecting *ASs*, this could due to our code optimization, but also the improvement of memory

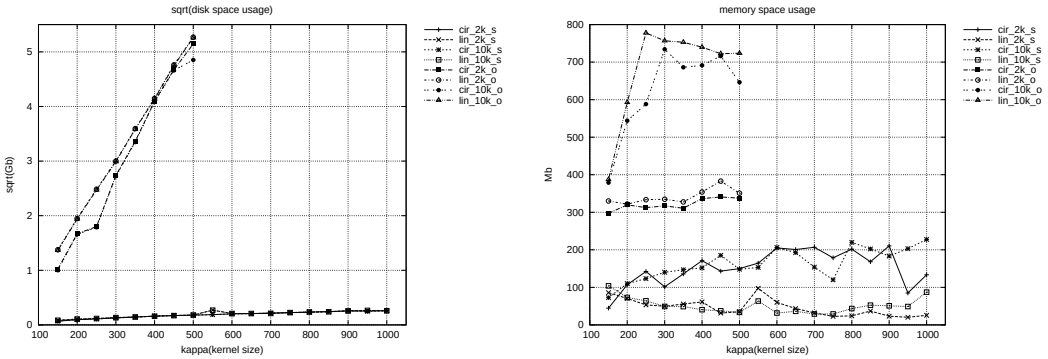


Figure 21: Space complexity comparison for our streaming algorithm (-s) and the original algorithm (-o).

locality, since there is only one graph stored in the memory. Last but not least, we can see that in the original algorithm, it spends more time on bound update than *ASs* detection, but our streaming algorithm reduce the bound update time to less than *ASs* detection time.

Figure 27 shows the space usage for the original algorithm and our streaming algorithm, it's very clear that our algorithm takes way less storage(memory and disk) than the original algorithm. One thing to notice is, for every graph in the original algorithm, it stores the according median genome for that graph, and the space usage is $O(g)$, we disable this storage to make the comparison fair, because our "footprint" based data structure and make sure that we don't need to keep the median genome information.

3.2.4.2 Problem space for complete search

In our previous discussion, we state that the performance of *DCJ* algorithm is directly dependent on *BPG kernel* size κ, and the figure shows that both the time/space complexity have been reduced, which makes our improvements well justified. However, the improvement of the time/space complexity can only introduce at most two orders

Table 1: The running time and search space for circular chromosomes

Search space for Circular Chromosome

κ	method	finished	avg time	avg search space
80	optimized	10/10	6.42	159784
	non-optimized	8/10	>336.61	> 9861818
90	optimized	10/10	27.66	669608
	non-optimized	6/10	N/A	N/A
100	optimized	10/10	850.33	17146636
	non-optimized	1/10	N/A	N/A

Search space for linear Chromosome

κ	method	finished	avg time	avg space
80	optimized	10/10	44.77	442855
	non-optimized	10/10	6576.07	71782336
90	optimized	10/10	201.48	1729328
	non-optimized	2/10	N/A	N/A
100	optimized	10/10	24236.91	138420207
	non-optimized	0/10	N/A	N/A

of magnitude of speedup, we can additionally reduce the search space to gain more performance by using heuristics. In Table 1, it shows the comparison between methods that performs the heuristic of upper/lower bound (optimized) and which does not use heuristics (non-optimized). For a given κ, if the non-optimized algorithm can not finish the search by searching the maximal number of search nodes of optimized algorithm, we marked it as "unfinished". From the table, it's very clear that our heuristic reduced search space for both circular and linear chromosomes by a factor of nearly two.

3.3 Using Genetic Algorithm to Solve Genome Median Problems

There are attempts of applying genetic algorithm to handle the DCJ median problem [62, 49]. Their approaches are relied on sorting. In this section, we will firstly introduce a method of recovering the ancesor gene orders (CAR), and its probabilistic alternation. This method form the basis of initialization of the gene orders for the genetic algorithm. In the genetic algorithm section, we also mention about our own approaches of performing evolutionary computing to get the median result.

3.3.1 Distance Space

Given three genomes g_1, g_2 and g_1, suppose there are some distance metric x, and the distance value is discrete. The distance between g_i and g_j is $dist_x(i,j)$. Given a median genome g_m, we have its distance between all other three genomes as ($dist_x(m,1)$, $dist_x(m,2)$, $dist_x(m,3)$), which is a triple, and it satisfies the following constraints:

$$\begin{cases} dist_x(m,1) + dist_x(m,2) \geq dist_x(1,2) \\ dist_x(m,1) + dist_x(m,3) \geq dist_x(1,3) \\ dist_x(m,2) + dist_x(m,3) \geq dist_x(2,3) \\ dist_x(m,1) + dist_x(m,2) \leq max_{i,j,k}^{i\neq j\neq k\leq 3}(dist_x(i,j) + dist_x(i,k)) \\ dist_x(m,1) + dist_x(m,3) \leq max_{i,j,k}^{i\neq j\neq k\leq 3}(dist_x(i,j) + dist_x(i,k)) \\ dist_x(m,2) + dist_x(m,3) \leq max_{i,j,k}^{i\neq j\neq k\leq 3}(dist_x(i,j) + dist_x(i,k)) \end{cases} \tag{5}$$

Any distance triple that satisfy the above constraint is a distance configuration (dc).

3.3.2 Median Genome Reconstructor

Given three genome sequences, we can use Ma's method to construct their predecessor graph (PG), then construct the intersection graph (IG) based on three predecessor graphs. To remove ambiguities in IG, we can use the following formula to weight an edge:

$$\begin{cases} weight(E_{s,t}) = \sum_{i,j}^{i\neq j\leq 3} \frac{dist_x(m,i)\cdot occur(E_{s,t},j) + dist_x(m,j)\cdot occur(E_{s,t},i)}{dist_x(m,i)+dist_x(m,j)} \\ occur(E_{s,t},i) = \begin{cases} \text{if } E_{s,t} \in PG(i) \\ \text{otherwise} \end{cases} \end{cases} \tag{6}$$

In the above formula, $E_{s,t}$ is the edge in IG that connect from vertex s to t. Once every edge in IG is weighted, they are sorted by their weight. And the edges with lowest weight are removed from IG iteratively until no ambulations exits. The result median genome generated by such way of a given dc is marked as $g_{m(dc)}$. Our object is, of all possible dc, find a dc that:

$$DC = \underset{dc}{argmin} \sum_{i \in (1,2,3)} dist_x(m(dc), i) \tag{7}$$

And the genome $g_{m(DC)}$ is the median we want.

3.3.3 Probability Based Method

Since there is also a probabilistic framework [92] to disambiguate the edge, we can also use this method to infer the median. Suppose D_s represents the observed edges in the IG starting from a given vertex s, and for an edge E_s^j, which means the edge start from s and is the part in PG_j. the probability using Bayes theorem of this edge to be in the median genome is:

$$P(E_s^i|D_s) = \frac{P(D_s|E_s^i)}{\sum_{j=1}^{3} P(D_s|E_s^j)} \tag{8}$$

Then the probability of $P(D_s|E_s^i)$ can be calculated by the following formula:

$$P(D_s|E_s^i) = \sum_{k=1}^{3} P(D_s^k|E_s^i) \tag{9}$$

In which,

$$P(D_s^k|E_s^i) = \begin{cases} \frac{1}{2n-1} + \frac{2n-2}{2n-1} e^{-(2n-1)\alpha(m,k)t(m,k)} & \text{if } D_s^k = E_s^i \\ \frac{1}{2n-1} - \frac{1}{2n-1} e^{-(2n-1)\alpha(m,k)t(m,k)} & \text{otherwise} \end{cases} \tag{10}$$

And $\alpha(m,k)t(m,k)$ can be calculated by:

$$\alpha(m,k)t(m,k) = -\frac{1}{2n-1} ln(1 - \frac{2n-2}{2n-1} \cdot \frac{dist_x(m,k)}{n}) \tag{11}$$

3.3.4 Genetic Algorithm Design

If we can find a good encoding and evolution strategy for genome rearrangement events. Genetic computing might help us to design a good heuristics for the median computation or even phylogenetic tree construction. Here we propose a encoding and evolution strategy:

Encoding: since there are $2g$ vertices in the BPG, there are $2g(2g-1)/2 = g(2g-1)$ possible edges, we can use a binary matrix M of length $2g(2g-1)$ to represent the selection of each edges under the constraint $\sum_{j=1}^{j<g} \sum_{i=1}^{i<g} M_{ij} \leq g$ and $\sum_{j=1}^{j<g} M_{ij} = 1 \forall i$.

Fitness Function: we can add three other arrays to represent the encoding of the input genomes, based on these arrays, we can design algorithms to calculate number of cycles/paths for the current generation encoding.

Mutation: We can prove that if two $0-matching$ edges are in the same $0-i$ cycle ($i \in 123$), if we use double cut and join operation to reorganize these two edges, the number of $0-i$ cycles will keep the same or increase by 1. In addition if two edges exists in more than two same $0-i$ cycles, after DCJ operation, the DCJ median distance will keep the same or decreased. We can use this feature to help design mutation function.

Selection: We can use the simplest roulette select as the selection function. However, there might be other ways to design the selection function to give different species more diversity. There are some problems in designing the selection function, for example, the difference between a "good" and a "bad" species is usually very small,

which means their probability of being chosen might be very close, therefore, we need to find a way to distinguish between "good" and bad candidates.

Crossover: Actually, there is no good cross over strategy yet. Because it's pretty hard to find a partition of edges between two graphs, and after exchanging these edges, the edges in two different graphs are still matchings. We designed a simple cross over strategy, that will only exchange the edges between two graphs that share the same head vertex. This method has the risk of reducing the diversity of the algorithm.

3.3.5 Experimental Results

We use the simulated data with the kernel size κ ranging from 100 to 1000. And compare the CAR median solver with the BnB based DCJ median algorithm. Since when κ is large, it takes huge amount of time for BnB algorithm to conclude to the global optimum. Considering that after searching for $10k$ search nodes, the BnB algorithm is already close to the optimal result, in this experiment, we only run BnB algorithm to search for $10k$ nodes. The experimental results are shown in Figure 22. We can see that the branch and bound based method is constantly outperforming CAR based method. This is because when κ is large, there are very few common ancestor adjacencies between three input genomes which makes the selection of CAR quite random and inaccurate.

We use the same data and test the Genetic algorithm using three different initialization methods: 1) randomized method: initiate every $0 - matching$ in random; 2) uniform method: in which vertex $2i$ is connected to vertex $2i + 1$; 3) CAR based method: in which we initialize the $0 - matching$ by using the CAR median algorithm mentioned before. The genetic algorithms run for 1000 iterations to terminate, and

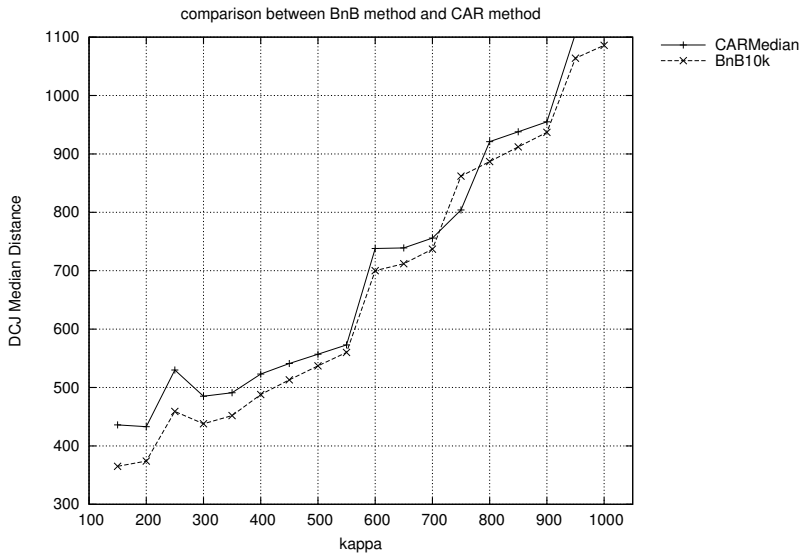


Figure 22: Comparison of *BnB* method and *CAR* based method.

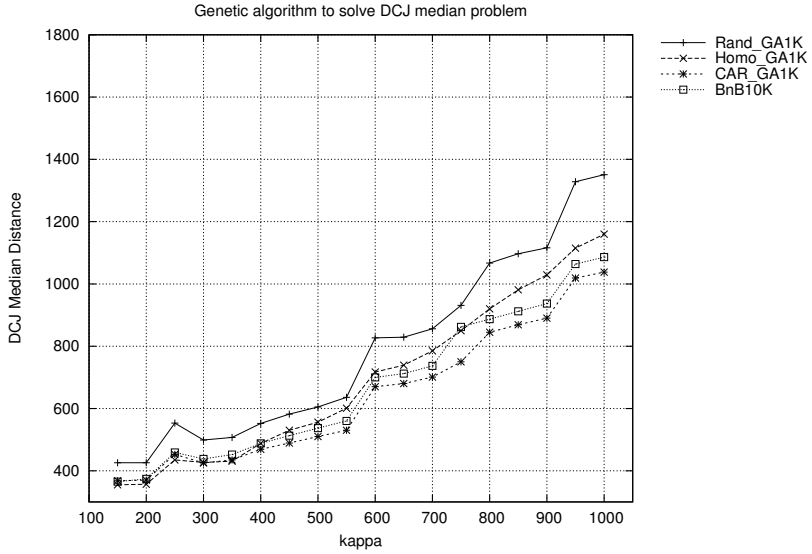


Figure 23: Comparison of different initialization methods in GA algorithm.

return the current best result. The experimental results are shown in Figure 23. We can see that in general the *BnB* method performs better than *GA* algorithm initialized using randomized and uniformed way. However, *GA* algorithm using *CAR* initialization method constantly perform better than *BnB* method and yield better medians. Therefore, we can conclude that *GA* algorithm is highly affected by the initialization method, and of which if it is properly selected, it can get pretty good

medians which is close or equal to the real median results.

3.4 Using Lin-Kernighan *Heuristic to Find* DCJ *Median with Genomes of Unequal Contents*

3.4.1 Problem Statement

Not surprisingly, finding the median genome that minimize the *DCJ-Indel-Exemplar* distance, is challenging. Because given three input genomes, there are infinite number of possible gene content selections for a median genome. Therefore, to make the problem easier, we can define a relaxed version of the median problem by providing known gene content.

DCJ-Indel-Exemplar median

Instance. Given the gene content of a median genome, and gene orders of three modern genomes.

Question. Find an adjacency of the genes of the median genome that minimize the *DCJ-Indel-Exemplar* distance between the median genome and the three input genomes.

DCJ-Indel-Exemplar median problem is *NP*-hard because there is no polynomial time algorithm to verify the results. Unfortunately, it's hard to design an exact *BnB* algorithm for *DCJ-Indel-Exemplar* median problem mainly because of two reasons: 1) the *DCJ-Indel-Exemplar* distance does not satisfy the property of triangular inequality. For example, given three genomes of $A(1, 2, 3)$, $B(-1, -1, 2, 3)$ and $C(-1, 2, 3)$, $distance(A, B) = distance(A, C) + distance(B, C) = 1$, which violate the triangular inequality rule, and make the bound computation incorrect. 2) when a *0-matching* edge is selected, edge shrinking is performed to generate the new *MBG*. The problem is, when there are duplicated genes in a genome, it's possible that there are multiple

edges of the same type connecting to the same vertex of a *0-matching*. This leads to ambiguity in the edge shrinking step, which makes the followed *BnB* search process very complicated and extremely hard to implement.

3.4.2 Design of *Lin-Kernighan* Heuristic

We use *Lin-Kernighan* (LK) heuristic to solve the *DCJ-Indel-Exemplar* median problem. The heuristic can generally be divided into two steps: initialization of 0-*matching* for the median genome, and LK search to get the result.

The initialization problem can be described as: given gene content of three genomes, find a median genome gene content that minimizes the sum of the number of *Indels* and duplications operations that transfer the median genome to other three genomes. In this paper, we design a very simple rule to initialize the gene content of the median genome, which is, given the count of one gene family of three genomes. If two or three counts are the same, we just select this count as the count of the gene family in the median genome. If all three counts are different, we select the median count as the count of the gene family in the median genome.

After fixing the count of genes, the next step is to set up the *0-matching* in the *MBG*, In this paper, we randomly set up the *0-matching*. Then we design the LK strategy by selecting two *0-matching* edges, and do a *DCJ* operation. We expand the search frontier by keeping all neighboring search solutions up until the search level $L1$. Then we only examine and add the most promising neighbors to the search list until level $L2$. The search is continued by the time when there is a neighbor solution yielding a better median score. This solution is then accepted and with it a new search is initiated from the scratch. The search will be terminated if there are no improvement on the results as the search level limit has been reached and all possible neighbors

has been enumerated. If $L1 = L2 = K$, the algorithm is called K-OPT algorithm.

3.4.3 Use of Adequate Sub-graphs to Simplify Problem Space

There are two categories of vertices in the *MBG*. One with exactly one edge of each edge type, is called "regular" vertices; another with less or more than one edges of each edge type, is classified as "irregular" vertices. A sub-graph in the *MBG* that only contains regular vertices, is defined as regular sub-graph [142]. By using the adequate sub-graphs [146, 142], we can prove that they are still applicable for decomposing the graph in *Indel-Exemplar* median problem.

Lemma 1 *As long as the irregular vertices do not involve, regular sub-graphs are applicable to decompose* MBG.

Proof 5 *If there are d number of vertices that contain duplicated edges in* MBG*, then we can disambiguate the* MBG *by generating different sub-graphs that contain only one of the duplicate edges (we call these sub-graphs disambiguate* MBG, d-MBG*). And there are* $\prod_{i<d} deg(i)$ *number of* d-MBG*s. Suppose a regular adequate sub-graph exists in the* MBG*, then it must also exist in every* d-MBG*. Suppose there is a global* 0-matching *solution for all* d-MBG*, then we can transform every* d-MBG *into completed* d-MBG *(*cd-MBG*) by constructing the optimal completion [37] between* 0-matching *and all the other 3 types of edges. After this step, the adequate sub-graphs exist in every* d-MBG *still exist in every* cd-MBG*. Which means, we can use these adequate sub-graphs to decompose* cd-MBG *for each median problem without losing accuracy.*

3.4.4 Search Space Reduction Methods

The performance bottleneck with the median computation is, in the exhaustive search step, for each search level, we need to consider $O(2g)^2$ possible number of edge pairs,

Algorithm 5: DCJIndelExemMedian

Input: *MBG* G, Search Level $L1$ and $L2$
Output: *0-matching* of G

Init search list L of size $L1$;
Init *0-matching* of G;
$currentLevel \leftarrow 0$ and $Improved \leftarrow true$;
while $Improved = true$ **do**
    $currentLevel \leftarrow 0$ and $Improved \leftarrow false$;
    Insert G into $L[0]$;
    while $currentLevel < L2$ **do**
        $G' \leftarrow$ pop from list $L[currentLevel]$;
        if G' *improves the median score* **then**
            $G \leftarrow G'$;
            $Improved \leftarrow true$ and break ;
        if $currentLevel < L1$ **then**
            for $x \in \forall$ 0-*matching pairs of* G **do**
                $G' \leftarrow$ perform DCJ on G' using x;
                if $num\_pair(x) > \delta$ **then** Insert G' into $L[currentLevel + 1]$
        else
            $G' \leftarrow$ perform DCJ on G' using $x = \underset{x}{argmax}\ num\_pair(x)$;
            if $num\_pair(x) > \delta$ **then** Insert G' into $L[currentLevel + 1]$
        $currentLevel \leftarrow currentLevel + 1$;
return 0-*matching of* G;

which is $O((2g)^{2L1})$ in total. In traveling salesman problem (*TSP*), it's cheap to find the best neighbor, but for *DCJ* operations, to evaluate a neighbor, we need to compute *NP*-hard *DCJ-Indel-Exemplar* distance, which makes this step extremely expensive to conclude. Noticing that if we search neighbors on edges that are on the same $0 - i$ color altered connected component (*0-i-comp*), the *DCJ-Indel-Exemplar* distance for genome 0 and genome i is more likely to reduce [152]. We can sort each edge pair by how many *0-i-comp* they share. Suppose the number of *0-i-comp* that an edge pair x share is $num\_pair(x)$. When the algorithm is in the exhaustive search step ($currentLevel < L1$), we set a threshold δ and select the edge pairs that satisfy: $num\_pair(x) > \delta$ to add into the search list. When it comes to the recursive deepening

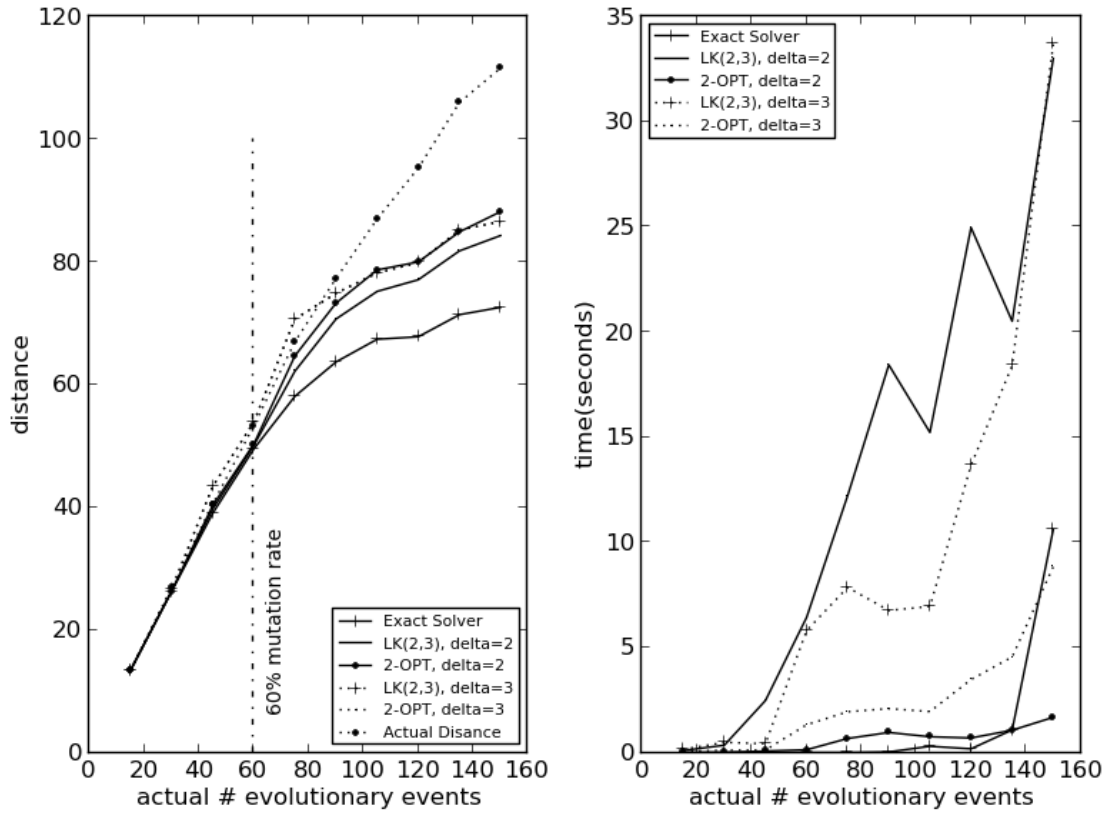


Figure 24: Median computation results for $\gamma = \phi = 0\%$ and θ varies from 10% to 100%.

step; we select the edge pair that satisfy $\underset{x}{argmax}\ num\_pair(x)$ to be added into the search list. This strategy has two merits, 1) some of the non-promising neighbor solution is eliminated to reduce the search space. 2) the expensive evaluation step which make a function call to *DCJ-Indel-Exemplar* distance is postponed to the time when a solution is retrieved from the search list. The skeleton of the algorithm is shown in Algorithm 5.

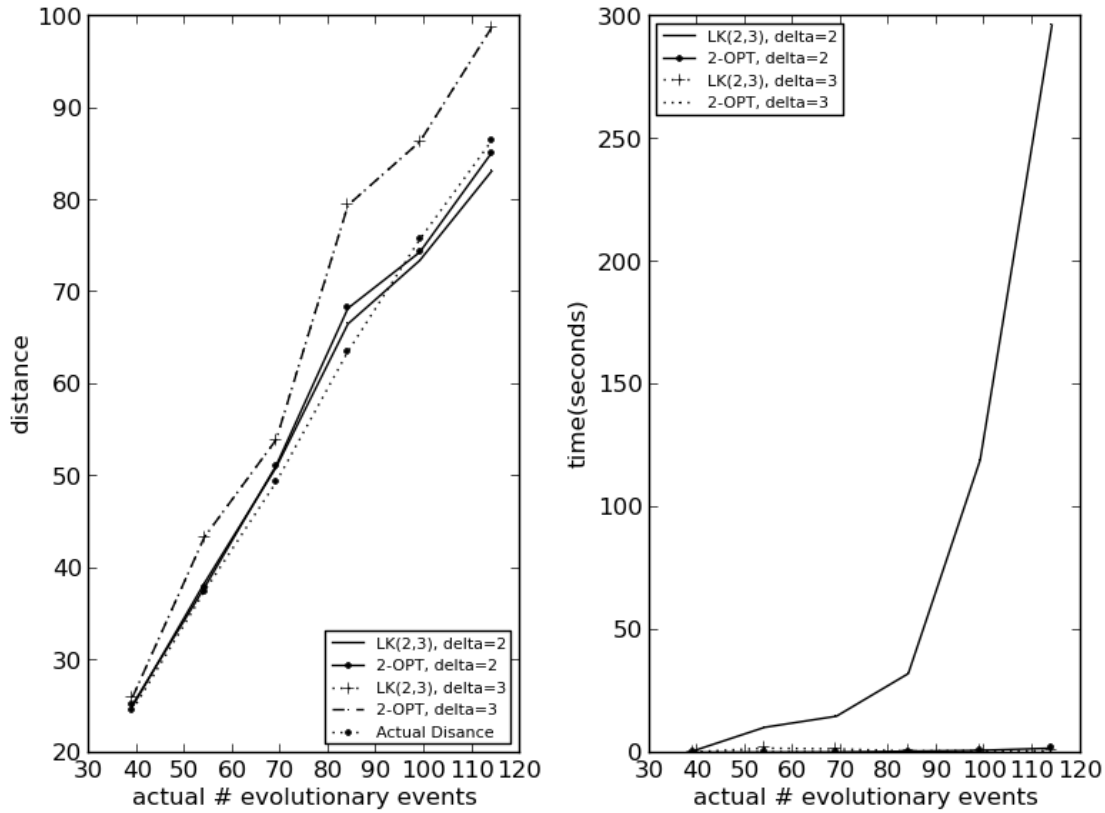


Figure 25: Median computation results for $\gamma = \phi = 5\%$ and θ varies from 10% to 60%.

3.4.5 Experimental Results

3.4.5.1 Results for LK solver using DCJ-Indel-Exemplar *distance*

We simulated the median data of three genomes using the same simulation strategy as in the distance simulation. In our experiments, each genome was "evolved" from a seed genome, and they all have the same evolution rate (θ, γ and ϕ). We compared the result of using LK algorithm with $L1 = 2$ and $L2 = 3$, and the K-OPT algorithm of $K = 2$. We used the search space reduction methods and set $\delta = 2$ and $\delta = 3$ respectively.

To test the accuracy of our LK and K-OPT methods, we first set both γ and ϕ to

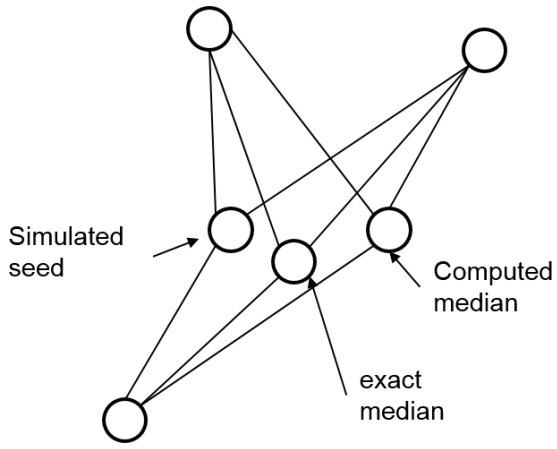


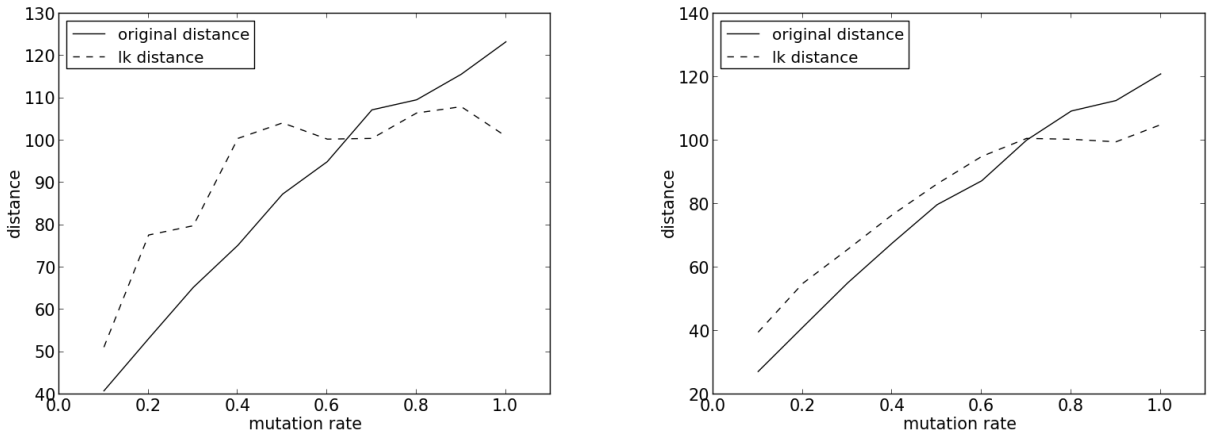
Figure 26: Methods for conducting median experiments.

0 and increased the mutation rate θ from 10% to 100%, so that each of the three genomes has the same gene content. We ran the exact DCJ median solver (we use the one in [152])to compare the time and accuracy of exact result with our heuristic result. As for computational time, since the exact solver is highly optimized by using streaming graph analytic methods, it's 2 orders of magnitude faster than the JAVA implementation by Xu [146]. In Fig 24, it shows the time and accuracy of our heuristic compared with the exact result. We can see that when $\theta \leq 60\%$, all results of the LK and $K\text{-}OPT$ methods are quite close to the exact solver. For parameter of $\delta = 2$, both LK and $K\text{-}OPT$ methods can generate exact results for most of the cases. The $K\text{-}OPT$ method, comparing with the exact method, consumes almost the same time when $\theta \leq 60\%$. Unfortunately, the LK cost much more time, since it needs to examine a much larger search space. The experimental process is shown in Figure 26.

As for the median results for unequal contents, we set both γ and ϕ to 5% and increased the mutation (inversion) rate θ from 10% to 60%. We compared our result with the accumulated distance of three genomes to their simulation seed, this method can not show the accuracy our method (since we do not have an exact solver), but can serve as the indicator of how close of our method was to the real evolution. Fig 25 shows the median results for unequal gene contents. We can see that when $\delta = 3$, both

LK and *K-OPT* algorithms get results quite close to the real evolutionary distance, while when $\delta = 3$ the result are not ideal. However, though *LK* method is slightly better than *K-OPT*, it takes orders of magnitude more time to conclude.

3.4.5.2 Results for LK solver using DCJ-Indel-CD *distance*



(a) $\gamma = 0.05$ and $\phi = 0.05$

(b) $\gamma = 0.1$ and $\phi = 0.1$

Figure 27: Median results for LK solver using *DCJ-Indel-CD* distance.

We use the same data as in the previous section, and the experimental results are shown in Figure 27(a) and Figure 27(b). Since the implementation of *DCJ-Indel-CD* median is more complicated than *DCJ-Indel-Exemplar* median. We use alternative way to implement this algorithm, which is slower than *DCJ-Indel-CD* media algorithm, therefore, we do not include the time result. We can see that in general, the new implementation is quite close to the real result when $\gamma = 0.05$ and $\phi = 0.05$ and slightly worse than real result when $\gamma = 0.1$ and $\phi = 0.1$.

Chapter IV

PHYLOGENY COMPUTATION

In this chapter, we describe the application of the previous mentioned distance and median algorithms to the topology and gene order inference of the phylogenetic tree. A software package called *DCJUC* (Phylogeny Inference using *DCJ* model to cope with Unequal Content Genomes) is developed. We test the *DCJUC* using gene order data. On simulated data set, our software can produce very curate trees using both *NJ* and *MP* methods. And can construct ancestor genomes with good parsimony score. Experiments on real gene order data (Yeast) show that our software package can help biologists discovering more complicated evolutionary patterns.

4.1 Phylogenetic Tree Topology Inference

Based on the given distance model and the way to compute median genome. We discuss the exhaustive way of inferring phylogenetic topology by using *BnB* algorithm. The algorithm is divided into the following four steps. And an example is given in Fig 28.

1) Initialize the root search node by selecting three species, and initialize a median for these three species. Then use the Neighbor-joining (NJ) method to get the upper bound ub and apply "twice around the tree" heuristic [100] to get the lower bound lb. After initial bound computation, allocate the search list of size $|ub - lb|$. Then insert the root search node into search list at position $|lb - lb|$. In Fig 28, root tree with species labeled by A, B and C and the median species labeled by number 1 is shown.

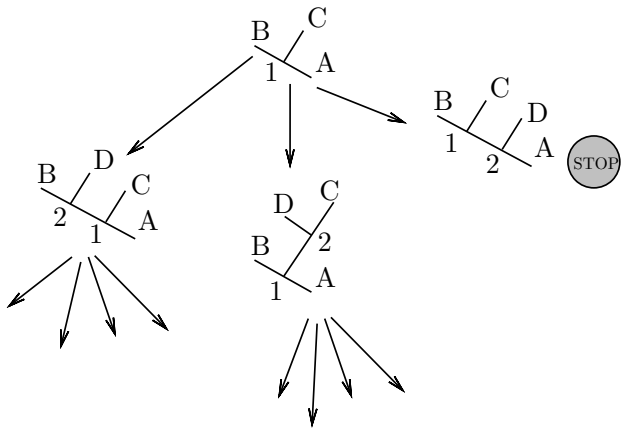


Figure 28: Example of using branch and bound for phylogenetic tree topology inference.

2) Expand the active search nodes in the *BnB* search tree by stepwise addition, trying to insert the next species into arbitrary branches of an incomplete phylogenetic tree. Evaluate the new search nodes by its upper and lower bound. If the lower bound of the new search node is larger than the global *ub*, abandon this search node.In Fig 28, there are 3 insertion positions for species D, and if it was inserted into the position between A and 1, its lower bound is larger than the global *lb*, which makes the proceeding of its searching process terminated.

3) Insert the new search node into the search list by its lower bounds, if the upper bound of the new search node is smaller than the global *ub*, set the global *ub* to this upper bound. Continue searching by retrieving saved search nodes with the smallest lower bound.

4) The program terminates when the upper bound meet the lower bound or there are no active search nodes, and return the tree with smallest upper bound.

The above branch and bound algorithm can ensure finding the maximum parsimonious tree given a specific objective function (distance model). However, this algorithm grows exponentially with the number of input modern species, which makes it

impossible to derive a *MP* tree with only tens of species.

Many heuristics, therefore, is proposed to help reducing the search space while keeping the accuracy of the tree as much as possible. There are multiple approaches to reduce the search space. One approach to help scaling the problem is the quartet tree method, which is relying on finding the optimal 4-leaf tree for each quartet. Once the quartet information is acquired, it is used to build the whole tree. Another is originated from searching on the "tree space" such as *NNI*, *TBR* and *SPR*. These methods are in hope of finding a optimal phylogenetic tree by searching in a promising local direction [151, 73] Many disk-covering methods (*DCMs*) are proposed as a divide and conquer heuristic to reduce the candidate tree search space [74, 80]. The *DCM* method can be viewed as two step process: 1) employ some methods to divide large number of species into some overlapped or un-overlapped subgroups; 2) apply the exhaustive search method to construct a sub-tree for each subgroup and merge these trees by incorporating various consensus tree algorithms [79].

4.2 Applying REC-DCM-Eigen *Method to Tree Topology Inference*

REC-DCM-Eigen Method [80] is one of the disk-covering method to reduce the tree search space. It shown to be more accurate and faster than other disk-covering methods. The basic idea, is to apply spectral partition method recursively to get overlapped disks. Each disk has a subset of genomes, construct a sub-tree for each disk. This process is shown in Algorithm 6.

After decomposition, these sub-trees are merged into a single phylogenetic tree using some consensus tree method. The consensus tree method Figure 29. In *DCJUC*, we do not use these consensus tree methods, because the consensus tree algorithms are not the research topic of this paper and which usually can not generate binary trees.

Algorithm 6: DecomposeDisk

Input: Disk D

Output: Partitions

```
while size(D) > size_threshold do
    Compute the similarity matrix (W) and the diagonal matrix (D) ;
    Compute the Laplacian matrix (L = D − W) ;
    Using Spectral method to compute the 2nd smallest eigenvalue of L and its
    eigenvector V;
    compute gap = V[last] − V[0] ;
    if gap > threshold then
        Return a non-overlapping disk decomposition ;
    else
        Return an overlapping disk decomposition ;
```

We design and implemented an algorithm to merge two sub-trees that can result in a unrooted binary tree.

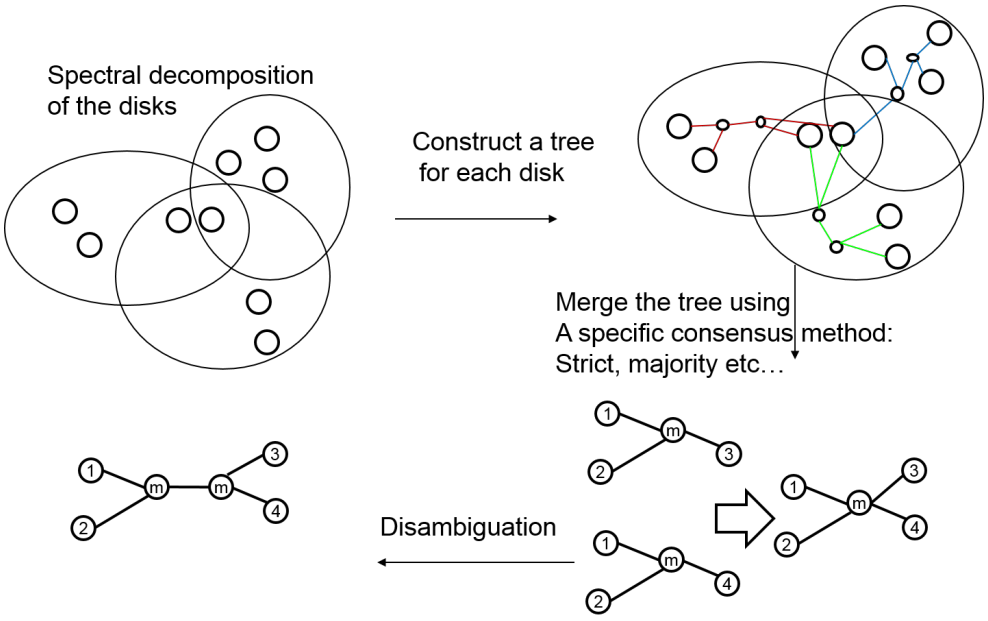


Figure 29: Example of using consensus tree methods to merge subtrees.

Suppose there are two trees T_1 and T_2, their genome sets are $Set(T_1)$ and $Set(T_2)$, the set of their overlapped genomes are $Set(overlap(T_1, T_2))$. If a tree is separated by one of its edge into two sub genome sets, we define a sub-tree formed by one of the sub genome sets as T_s and the according set as $Sub(T_s)$. The algorithm that is used

Algorithm 7: MERGETWOTREES

Input: T_1 and T_2
Output: merged tree T_0

```
if T1 ∩ T2 = ∅ then
    T0 ← Merge(T1, T2);
else
    T0 ← Merge(T1, ∅);
    while ∃Ts ∈ T2 and Ts ∉ T1 do
        T0 ← Merge(T0, Ts);
return T0;
```

to merge two sub-trees can be described using the algorithm 7 and 8. The algorithm can be generally described as a two step process, step 1, find a subtree T_s of T_2 that $Set(overlap(T_1, T_s)) = \emptyset$; step 2, merge T_s to T_1, which can be realized by: 1) compute the pairwise distance of all nodes in T_s and T_1, find the vertex pair with the minimum distance; 2) of all possible edge combinations of two selected vertices (maximum 9), find the edges pair that yields the minimum median score after connecting these two edges by inserting two internal nodes.

4.3 Ancestor Gene Order Reconstruction

Given the model tree topology, the approach for reconstruction of ancestor gene order in BPAnalysis [22], *GRAPPA* [100] and *GASTS* [145] is basicly iterative refinement, which is a two step process: initialization and refinement. In the phylogeny topology inference step, after the topology calculation, the gene orders of the internal nodes (ancestor genome) have already been initiated. However, the potential of the iterative refinement will not be putted into full play if the initial gene order are abounded local optima. In this section, we only introduce Xu's Generalized Adequate Sub-graph (*GAS*) method for initialization [145], because it utilized the global information of the tree and achieved the best results so far.

Algorithm 8: MERGE

Input: T_0 and T_s

Output: A merged tree T_0

```
1  disMin = MAX_INT ;
2  for i ← 1 to |Set(T_0)| do
3      for i ← 1 to |Set(T_s)| do
4          dis ← DCJIndelExem(T_0[i], T_s[j]) ;
5          if dis > disMin then
6              disMin ← dis;
7              min_0 ← i;
8              min_s ← j;
9
10 tScoreMin = MAX_INT ;
11 for i ∈ |neighbor(T_0[min_0])| do
12     for j ∈ |neighbor(T_s[min_s])| do
13         T_tmp ← Join(min_0, neighbor(T_0[min_0])[i], ;
14                min_s, neighbor(T_s[min_s])[j]) ;
15         if Score(T_tmp) < tScoreMin then
16             tScoreMin ← Score(T_tmp);
17             min_to_0 ← neighbor(T_0[min_0])[i];
18             min_to_s ← neighbor(T_s[min_s])[j];
19
20 Join(min_0, min_to_0, min_s, min_to_s) ;
21 return T_0;
```

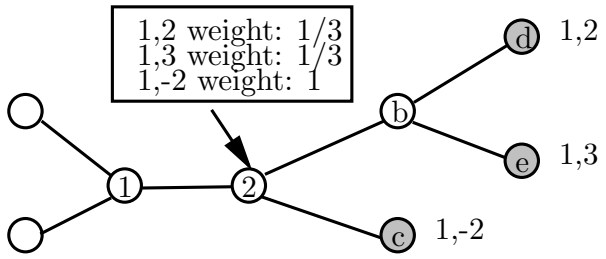


Figure 30: Example of *GAS* initialization of internal ancestor genomes, genome '2' is the one to be initialized, the perspective of 2 is all the nodes in the *BFS* route start from 2, and the directive nodes of the perspective are the nodes marked by gray color. In this example, the adjacencies of gene 1 are shown of how they are chosen and how they are weighted.

Fig 30 shows an example of how to initiate an internal ancestor genome. There are two kinds of nodes in the figure, one is initialized nodes which is marked by gray color, another is uninitialized nodes with white color. When initiating a node (in figure denoted as number 1) that has two of its neighbor are initiated and one uninitiated (denoted number 2). The nodes that in the breath first search (*BFS*) route of the uninitiated neighbor are called the *perspective* of the neighbor. We use p to represent the perspective. In the perspective, the initiated nodes are called *directive* nodes, which we use g to represent these nodes. The adjacencies of the neighbor node is initialized by a weighting scheme that actually summarize all the adjacencies of directive nodes, and give each adjacency a weight using the equation (12). As the equation shows, $I_x(g)$ is set to 1 if adjacency x is existed in the directive node g, otherwise 0. d is the depth of the directive node g, which is equal to number of edges in the shortest path between neighbor node and the directive node. Fig 30 shows an example of how adjacencies of gene 1 are set up using the weighting scheme.

$$W_x = \sum_{g \in p} I_x(g) \cdot 3^{-d+1}. \tag{12}$$

Once the adjacencies of all three neighbors are initialized, a new median of internal node will be calculated. If the new median is better than the old one, it will replace

Table 2: The experiment result for phylogenetic tree construction

dataset	num genes	max κ	median κ	Heuristic tree length	Exact tree length	Heuristic time	Serial time	Parallel time
dros-5	9738	172	12	4395	4320	102.8	1449	394.5
dros-12	7332	234	60	5305	5244	547.1	7055	1933

the old median. And its neighbors if they are also internal nodes, because of having an update on their neighbors, will also try to update their genome value. This iterative refinement process, will terminate until convergence.

We modified the *GAS* method and reimplemented our initialization method. In our implementation, we do not evaluate every adjacencies in the perspective, alternatively, we introduced two threshold $t1$ and $t2$ for the selection of adjacencies. If the weight of an adjacency is larger than $t1$, it's selected, otherwise, if a vertex doesn't have an adjacency selected by the threshold $t1$, and its adjacency with the largest weight is larger than $t2$, select this adjacency. If no adjacency of a vertex with score more than $t1$ or $t2$, there will be no adjacency attached with the vertex, which means a insertion/deletion happened in the initialized genome. In our work, we set $t1$ as 1.0 and $t2$ as 0.5.

4.4 Experimental Results

4.4.1 Applying Streaming Breakpoint Graph Analysis Methods on Real Drosophila data for Phylogeny Inference

We plugged in our streaming breakpoint analysis based *DCJ* median algorithm into the state of art phylogenetic tree construction package *GASTS* [145] which uses a *DCJ* median heuristic [118], and conduct the experiment on the real Drosophila data set [19, 127], we designed two data sets with the number of species increased. The first data set is consisted of 5 species (which are *Dmel*, *Dere*, *Dana*, *Dpse* and *Dwil*), we choose these 5 species because they have a long diameter in the phylogenetic relations.

And the second data set contains all of 12 species of Drosophila. We deleted all of insertion, deletion and duplications to make each species has the same gene content, and the 5 species data set has more genes than 12 species data set (see Table 35).

The experimental results are shown in Table 35, we do not use a model tree in our experiment. We can see that during the process of constructing the phylogenetic tree, most of the median problems are easy to solve (with a small κ). However, there are a few median problems that is extremely hard to solve, which has a very large κ. For these problems, we set the program's threshold to search up to 1 million node then return the local minimum. On both data sets, our *DCJ* median plug-in helps to find a better phylogenetic tree which has smaller accumulated tree length, and comparing to the heuristics based median solver our serial code is just about 10 times slower, by utilizing the parallel algorithm, this gap reduces to about 4.

4.4.2 Phylogenetic Inference

For topology inference, we generate simulated data by the following steps: First we generate phylogenetic model trees of three genomes by using a birth-death model [89]. Then, based on the model tree, we start from the root of the tree and set it to have gene number 50. Following the branches of the tree, for each edge, the rate of *DCJ* operations is set to $0.2 \times \theta$ in which $\theta = \sqrt{\frac{\text{edge length}}{\text{max length}}}$, the insertion/deletion rate is set to $0.05 \times \theta$ and duplication rate is set to to $0.05 \times \theta$. We apply this strategy to generate two different types of trees which has the diameter of $1 \times$ (gene num) and $2 \times$ (gene num), we also generate trees that has uniform number of *DCJ* operations of 5 and number of insertion/deletion and duplications set to 1. We set the spectral partition parameters the same as in [80]. Since our tree merging methods generate only binary unrooted trees, we use accuracy to evaluate trees which is $\frac{total-FP}{total}$. We test the phylogenetic inference methods using both *NJ* method with our distance

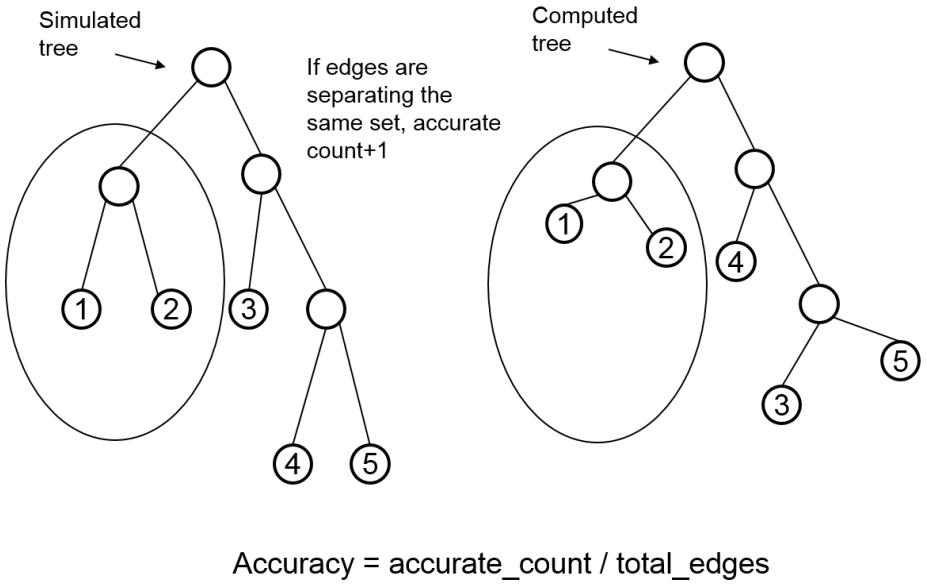
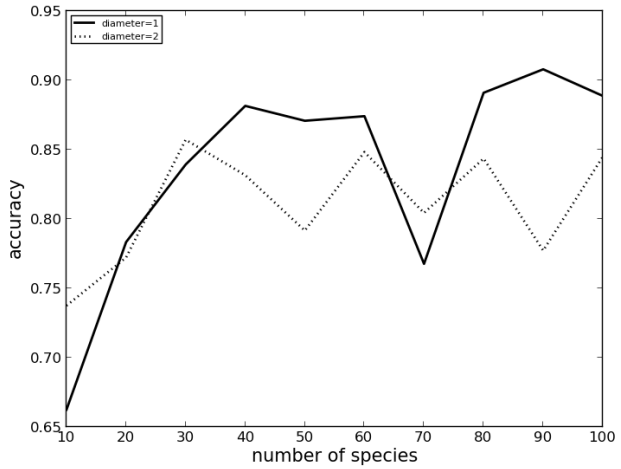


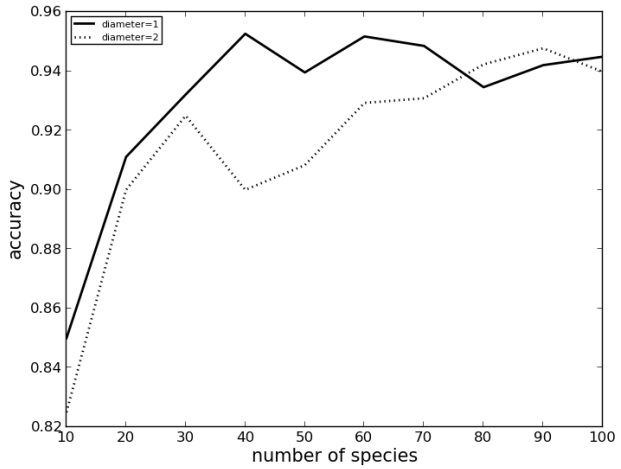
Figure 31: Methods for conducting phylogeny inference experiments.

estimator and *MP* method with our median calculator. The methods for conducting the experiments is shown in Figure 31. The experimental results are shown in Fig 32.

As for the *NJ* results, we can see from the result that, for both diameters are 1 and 2, our algorithm achieve about 90% of accuracy with number of species ranging from 10 to 100. And for many cases, the accuracies are more than 95%. As for the *MP* results, We can see from the result that, if the diameters are 1 and 2, our algorithm achieve about 80% of accuracy when there are 10 species, and with the growth of the number of species, the accuracy grows to about 85%. which is slightly worse than *NJ* method, which is usually not the truth. This is mainly due to the reason that we do not apply the consensus tree algorithm in our program. Which means, when merging two overlapping disks, error is highly possible to be introduced. Another reason might be because the median algorithm we used is a heuristic which might have accumulated error during the tree construction process.



(a) accuracy for super-tree method

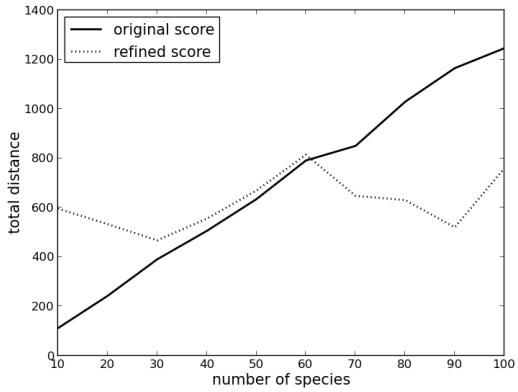


(b) Accuracy for Neighbor-Joining method

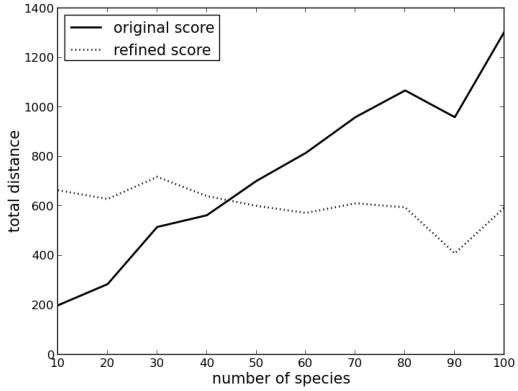
Figure 32: Results of tree topology construction accuracy, the x-axis is the number of species and the y-axis is the accuracy.

4.4.3 Ancestor Order Reconstruction

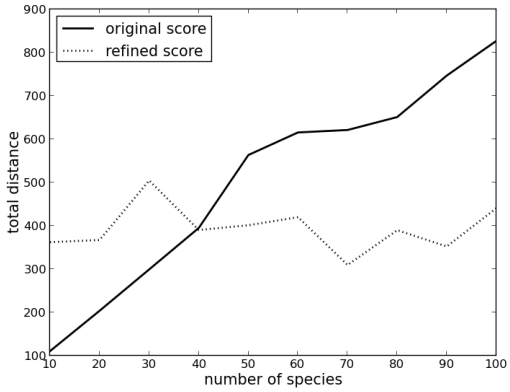
The ancestor order reconstruction experiment use the same data as in the phylogenetic tree topology inference experiment. After simulating the tree, we accumulate all the operations that happened in the path from root node to leaf nodes as the total score of real evolutionary operations. Then we unroot the tree by detracting operations happened along the edges between root node and its two children, and add



(a) Diameter=1



(b) Diameter=2



(c) Uniform edge length=5

Figure 33: Results for ancestor genome reconstruction.

the distance between two children. We then run experiments to compare the score of tree reconstructed by our method with the real score. The experimental result is

shown in Fig 33, we can see from the figure that when the number of genomes is small, the program will reconstruct ancestor genomes which might not be optimal comparing with the real number of operations. There are two reasons, when the number of species is small, there might not be enough global information to use as to initiate the third median genome, the initialization are more tend to to be started with a local optimum. Another reason is, when there are more species, the mutation rate of the child species might be more than less species. At this time, there will be more redundant operations just for which the distance estimator will under estimate the real distance.

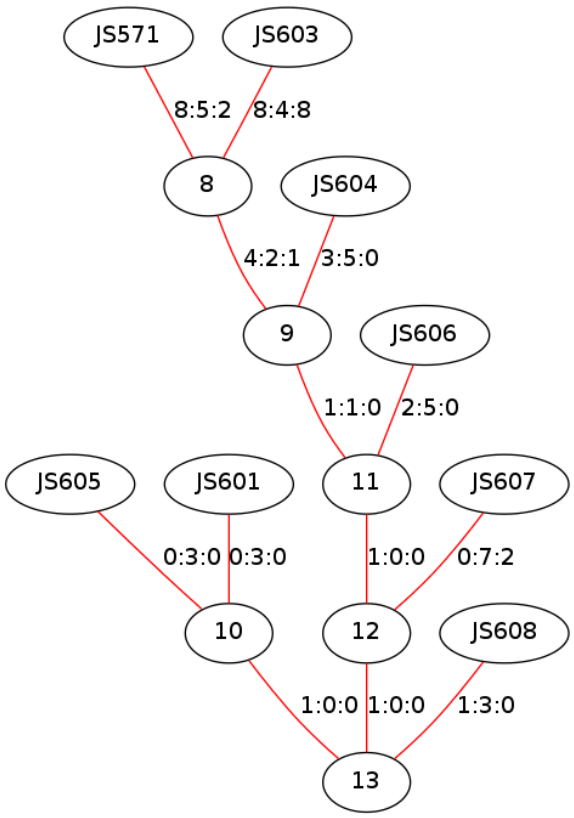


Figure 34: Example of the visualization of the phylogenetic tree using *DCJUC* with the input of a subset of yeast genome data.

4.4.4 Real Tree Construction Example

We conduct our phylogenetic tree simulation experiment using real data from the synthetic yeast genome project, Sc2.0 [50]. There are 44 genes in the genome data, and there are in total 53 species. we did not use a model tree in our experiment and construct the tree from the scratch.

The tree generated is shown in Fig 35. Along with the topology of the inferred phylogenetic tree, *DCJUC* can also generate three numbers of each edges in the phylogenetic tree. which are the (number of *DCJ*, number of insertions/deletions and number of duplications). These information are shown in the Figure reftreethree. A tree of subset of yeast genome is constructed using our software, and the visualized result is generated. And together with the topology of the phylogenetic tree, can help biologists better understand the rearrangement events that happened in the evolutionary path.

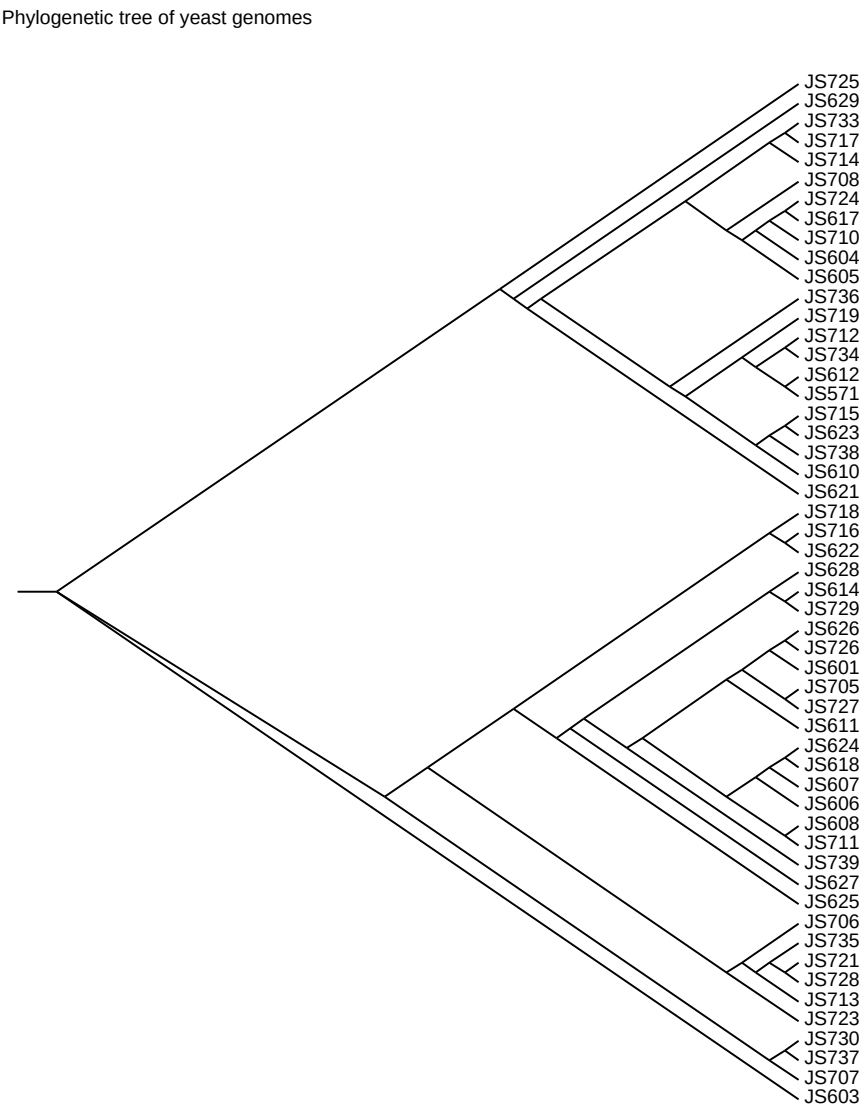


Figure 35: The phylogenetic tree of 53 species yeast genome. In the figure, each edge of the tree has three numbers, which are number of DCJ operations, number of insertion/deletion operations, and number of duplication operations.

Chapter V

USING EMERGING PARALLEL COMPUTING ARCHITECTURE TO ACCELERATE PHYLOGENETIC ALGORITHMS

5.1 Using GPGPU *to Accelerate* HMM *based Sequence Alignment Algorithm*

The *Viterbi* algorithm is the compute-intensive kernel in Hidden Markov Model (*HMM*) based sequence alignment applications. In this chapter, we investigate extending several parallel methods, such as the wave-front and streaming methods for the *Smith-Waterman* algorithm, to achieve a significant speed-up on a *GPU*. The wave-front method can take advantage of the computing power of the *GPU* but it cannot handle long sequences because of the physical *GPU* memory limit. On the other hand, the streaming method can process long sequences but with increased overhead due to the increased data transmission between *CPU* and *GPU*. To further improve the performance on *GPU*, we propose a new tile-based parallel algorithm. We take advantage of the homological segments to divide long sequences into many short pieces and each piece pair (tile) can be fully held in the *GPU*'s memory. By reorganizing the computational kernel of the *Viterbi* algorithm, the basic computing unit can be divided into two parts: independent and dependent parts. All of the independent parts are executed with a balanced load in an optimized coalesced memory-accessing manner, which significantly improves the *Viterbi* algorithm's performance on *GPU*.

Algorithm 9: DoWaveAlign

Input: seq_temp, seq_tar
Output: Aligned sequence

```
1  Initmatrix() ;
2  InitHMM() ;
3  for round_r ∈ all rounds do
4      forall block_mn ∈ blocks of round_r in parallel do
5          for stateS_i ∈ three states of Block_mn do
6              for stateS_j ∈ three states of dependent block do
7                  //when Si is Match,the dependent block is Block(m-1)(n-1) ;
8                  // when Si is Delete, the dependent block is Blcok(m-1)n ;
9                  //when Si is Insert, the dependent block is Blcokm(n-1) ;
10                 calculateδ(mn, i) ;
11                 //δ(mn, i) stands for the forward variable for the ith state of
                   Blockmn ;
12 seuqence = Traceback() ;
13 return sequence;
```

5.1.1 Wave-front Pattern to Implement the *Viterbi* Algorithm

The wave-front algorithm is a very important method used in a variety of scientific applications. The computing procedure is similar to a frontier of a wave to fill a matrix, where each block's value in the matrix is calculated based on the values of the previously-calculated blocks. On the left-hand side of Figure 36 we show the wave-front structure for parallelizing the *Viterbi* algorithm. Since the value of each block in the matrix is dependent on the left, upper, and upper-left blocks, in the figure, blocks with the same color are put in the same parallel computing wave-front round. The process for this wave-front algorithm is shown in Algorithm 9; we call it a simple implementation of the wave-front algorithm.

The right-hand side of Figure 36 shows the data skewing strategy to implement this wave-front pattern. This memory structure is useful because blocks in the same parallelizing group are adjacent to each other (they are marked with same color in Figure 37. In this way, because data accessed by neighbor threads are organized

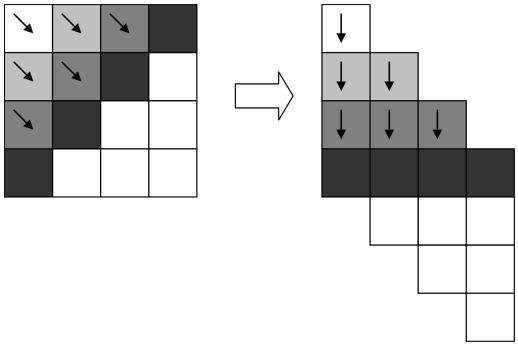


Figure 36: Wave-front structure of *Viterbi* Algorithm for Biological Sequence Alignment. The left-hand side of the figure shows the wave-front process, and right-hand side of the figure shows the memory data skewing to implement the wave-front algorithm. For the detail of this method, please refer to [6].

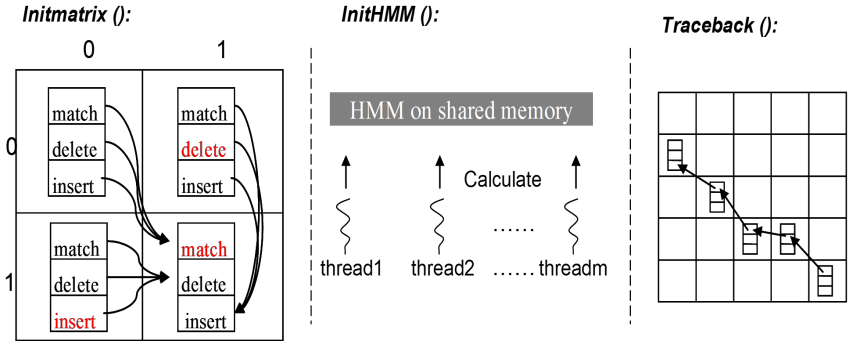


Figure 37: Example of three functions in the simple wave-front implementation.

adjacent to each other, threads could access memory in a more efficient manner [106]. There are three functions in the simple implementation, *Initmatrix ()*, *InitHMM ()* and *traceback ()*, and the process of these three functions are shown in Figure 37). *Initmatrix ()* sets up the dynamic programming matrix of the *Viterbi* algorithm for Biological Sequence Alignment. We use the color red to mark the three initial states as the termination for trace back, which can unify at each parallel round of computation. *InitHMM ()* sets up the probability model for template sequence. We use pseudo-count methods to train the *HMM* in parallel. *Traceback ()* is used to align the template to target according to the value of the matrix calculated. This process is executed on the *GPU* because the cost of transmitting the *DP* matrix back to main memory is high if the trace back is done on the *CPU*.

5.1.2 Streaming *Viterbi* Algorithm for Biological Sequence Alignment

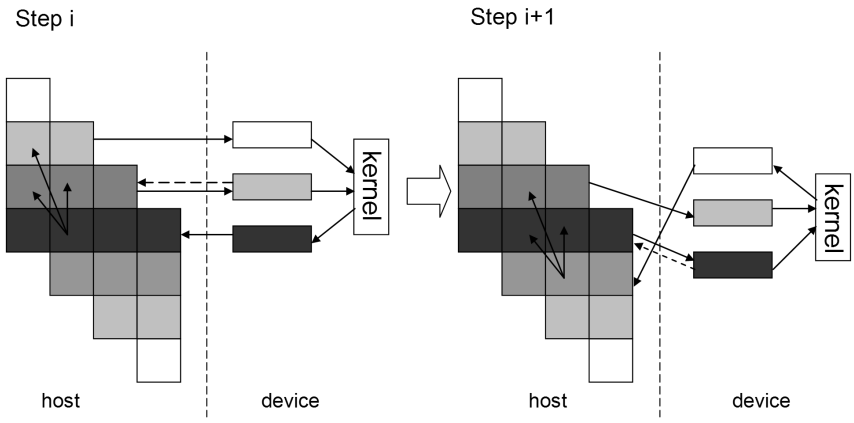


Figure 38: The *HMM* matrix is updated asynchronously at the host *CPU* and *GPU* device. The solid and dashed arrows represent the asynchronous execution between host *CPU* and *GPU* device.

A straightforward implementation of the wave-front pattern has one deficit – when the sequence length is too long. Since the size of the Dynamic Programming (DP) matrix is $O(len\_t \times len\_a)$, not all the data can be held in the *GPU*'s memory at one time. Here, we introduce a streaming method, which has already been used successfully in parallel *Smith-Waterman* algorithms. Because the *Viterbi* algorithm for biological sequence alignment has similar data dependencies, it is possible to apply this method as well to the parallel *Viterbi* algorithm to solve the *GPU* memory limitation problem.

In *CUDA*, a stream is a sequence of instructions that execute in order. Different streams, on the other hand, may execute their instructions asynchronously [106]. This feature ensures that the execution between the host and *GPU* device can be overlapped with each other. In the implementation of the *Smith-Waterman* algorithm, the sequence length supported by this mechanism is longer because only three rounds are needed to place the sequence into the *GPU*'s memory.

Figure 38 shows the process of computing the DP matrix for the streaming parallel *Viterbi* algorithm. Only three rounds of communication are needed to load the matrix

Algorithm 10: DoStreamingAlign

Input: seq_temp, seq_tar
Output: Aligned sequence

```
1  Initmatrix() ;
2  InitHMM() ;
3  for round_r ∈ all rounds do
4      stream[r]_forall block_mn ∈ blocks of round_r in parallel do
5          for stateS_i ∈ three states of Block_mn do
6              for stateS_j ∈ three states of dependent block do
7                  calculateÎt(mn, i) ;
8      stream[r]_do: transfer groupg-1 back to host
9  seuqence = Traceback() ;
10 return sequence;
```

block into the *GPU* memory and compute the appropriate kernel. The full *DP* matrix is stored in the host memory. This mechanism needs the values of the other two rounds to calculate the values of current round. When performing computation, data from the round which had been previously processed, will be transferred back into the host memory concurrently. Algorithm 10 describes this streaming approach. Note that procedures marked with the same stream[r] will execute instructions independently.

5.1.3 Tile Based Method to Harness the Power of *GPU*

One drawback of the streaming *Viterbi* algorithm is its high dependency on the computational resources of the host side, and the transition time between host and *GPU* device cannot be neglected either. The streaming *Viterbi* algorithm requires additional storage and communication bandwidth. Therefore, we introduce the new tile-based method, which simplifies the computational model and also handles very long sequences with less memory transmission.

The tile-based method can be described as follows: *If a matrix can be fully loaded into the* GPU *memory, do it; otherwise divide the large matrix into tiles to ensure*

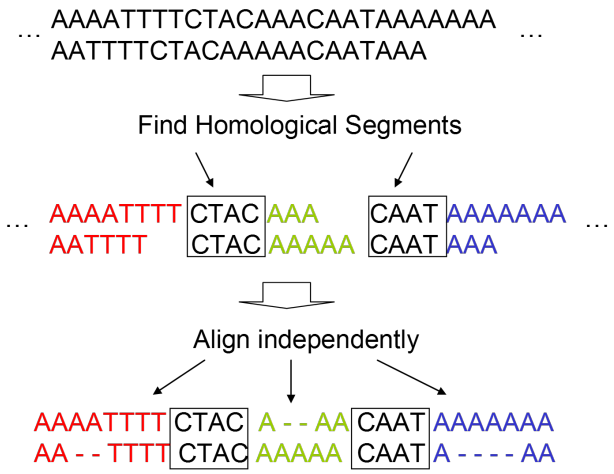


Figure 39: Using homological segments to divide long sequences.

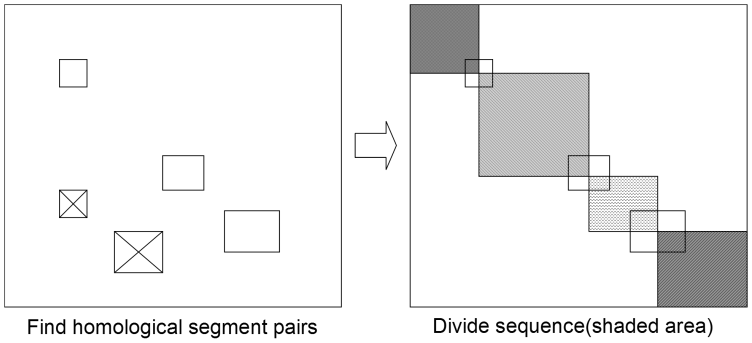


Figure 40: Example of finding homological segment pairs and using them to divide a large matrix into smaller, independent tiles. In the left-hand diagram, homological sequences form small pieces and are aligned using the Dynamic Programming method, and un-aligned homological tiles are marked with an "X". In the right-hand diagram, aligned tiles are used to divide large matrix into small sub-matrices.

that each tile fits in the GPU's memory as a whole and then calculate the tiles one by one or in parallel.

Cell-Swat [3] presents a tiling mechanism, but with this method, there are data dependencies among each tile. We introduce the homological segments concept into our tiling mechanism to eliminate the data dependency among different tiles. The biological meaning of homological segments makes them serve as separators very well (as shown in Fig. 6). To find homological segments, there are algorithms such as Fast Fourier Transform (*FFT*) [81] based algorithms and *k-mer* based algorithms.

We choose a *k-mer* based method for based algorithms. We choose a *k-mer* based method for segments are found, we use a dynamic programming algorithm to align these segments and cut the long sequence from the middle of these homological segments, as shown in Figure 40. For the details of homological segments, please refer to [48]. The process of finding homological segments and using them to divide independent tiles is as follows:

1) Find all homological segments whose score exceeds the threshold.
2) Using a dynamic programming algorithm to align these homological segments, ignoring those homological segments which are not aligned (such as in the left-hand side of Figure 40, the homological segments which are not aligned are marked with an "X").
3) Using homological segments to divide the full dynamic programming matrix into small tiles (marked with the dash area).
4) Calculate the tiles with the *Viterbi* algorithm.

5.1.4 Optimization Methods

Another deficiency of the wave-front *Viterbi* Algorithm for Biological Sequence Alignment is the unbalanced thread load; there are some idle threads at the beginning and the end of the algorithm. We solve this problem by transforming the following formula, which is used to calculate one block.

$$\begin{aligned} \delta_t(j) &= MAX_{1\le i\le N}[\delta_{t-1}a_{ij}(t)]b_j(O_t) \\ &= MAX_{1\le i\le N}\delta_{t-1}[a_{ij}(t)]b_j(O_t)\ 1\le j\le N \end{aligned} \tag{13}$$

In the transformed formula, the calculation of $a_{ij}(t)b_j(O_t)$ does not have any data dependency; therefore, we can calculate $a_{ij}(t)b_j(O_t)$ initially and store the results in

a temporary memory.

To express this idea more clearly, we present this method in Figure 41. In our method, we first perform the formula transformation. Before transforming the formula, the inner computation of one block is tightly dependent. This is because the current round is dependent on its previous rounds and only after the previous rounds have finished can it start. After formula transformation, the calculation is divided into two parts, the independent part and the dependent part. We collect all the independent parts together and let them run in parallel (the first step). At this stage, the threads are load balanced and the coalesced-memory optimization is employed. The dependent part of the computation is moved to step two, and it uses the data previously computed by the independent computation.

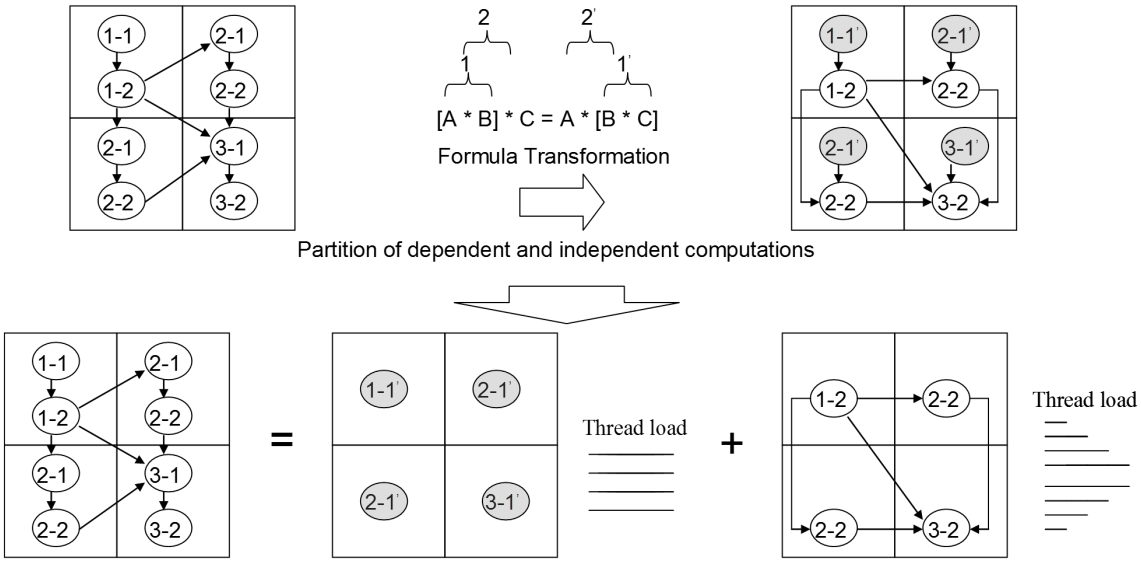


Figure 41: Thread load. The left side of this figure shows the wave-front *Viterbi* Algorithm for Biological Sequence Alignment and right side shows how the transformed formula can balance the thread load.

5.1.5 Experimental Results

The experiments are performed on the platform which has a dual-processor Intel 2.83GHz CPU with 4 GB memory and an *NVIDIA Geforce 9800 GTX GPU* with 8

streaming processors and 512MB of global memory. We tested using two operating systems: *Windows XP* and *Linux Ubuntu* version 10. To focus on the algorithmic efficiency in our study, we made two simplifications in our experiments, one is that we use a pseudo count method to train the *HMM*, and another is that we neglected the accuracy comparison with other methods and our parallel method has the same accuracy with the serial method. We employ the automatic sequence-generating program *ROSE* [42] to generate different test cases.

5.1.5.1 General Test

The general test was executed on four implementations of the *Viterbi* Algorithm for Biological Sequence Alignment. The first one is the serial implementation; the second is the simple wave-front implementation; the third is the streaming implementation; the last one is our tile-based implementation.

We have compiled and run our test programs under four different versions. The first is *Windows-Debug*; the second is *Windows-Release*; the third is *Linux-Debug* and the last is *Linux-release*. For long sequences, the simple wave-front version cannot load the entire dynamic programming matrix into the *GPU* memory. Therefore, we select groups of sequences which have lengths under 1000 to test all of the versions. The experimental results are shown in Table 3.

The results show that the best acceleration rate is achieved under the *Windows-Debug* mode, and we can see that the speedup under debug mode is better than release mode. This is because in the serial version, the compiler's optimization methods have a great effect on the manner of accessing memory; however, in *CUDA*, this is not true. Because *CUDA* has a special hierarchy of memory structure and the access time for different levels of the hierarchy greatly varies. For example, global memory

Seq-Length		Execution Time (Second)/Speedup						
		serial	simple wave-front		streaming		tile-based	
100	DW	0.73	0.37	1.97	0.38	1.92	0.28	2.61
	RW	0.017	0.007	2.42	0.02	0.85	0.006	2.83
	DL	0.063	0.008	7.87	0.023	2.74	0.007	9
	RL	0.027	0.007	3.86	0.025	1.17	0.007	3.86
200	DW	2.34	0.59	6	0.44	5.32	0.59	6
	RW	0.05	0.05	1.67	0.061	0.82	0.028	1.79
	DL	0.324	0.055	9.26	0.066	4.98	0.029	11.17
	RL	0.142	0.055	4.06	0.066	2.18	0.029	4.9
300	DW	5.89	0.42	14.02	0.46	12.8	0.45	13.7
	RW	0.12	0.068	1.76	0.1	1.2	0.055	2.18
	DL	0.647	0.07	9.26	0.112	5.78	0.054	11.98
	RL	0.283	0.068	4.16	0.116	2.44	0.064	5.24
400	DW	9.93	0.60	19.86	0.62	19.1	0.45	22.07
	RW	0.21	0.15	1.61	0.159	1.32	0.098	2.14
	DL	1.112	0.12	9.27	0.2	5.56	0.099	11.23
	RL	0.485	0.122	3.98	0.174	2.79	0.097	5
500	DW	15.9	0.54	29.44	0.54	29.44	0.62	30.58
	RW	0.34	0.19	1.78	0.259	1.42	0.174	1.95
	DL	1.783	0.198	9	0.262	6.8	0.155	11.5
	RL	0.783	0.191	4.1	0.251	3.12	0.153	5.12
1000	DW	62.1	0.99	62.73	1.1	56.45	0.86	72.21
	RW	1.34	0.64	2.09	0.686	1.95	0.554	2.42
	DL	6.98	0.64	10.91	0.725	9.63	0.53	13.17
	RL	3.07	0.635	4.83	0.62	4.952	0.512	6.0

Table 3: Performance comparison of for different *Viterbi* implementations. In the table, the first line of a group is the results for Debug-Windows mode (DW), second line, Release Windows (RW), third line, Debug Linux (DL), Fourth line, Release Linux (RL).

has an access time of about 500 cycles and the on-chip memory such as a register of only 1 cycle. Due to the reason that *CUDA* does not provide a good mechanism to optimize memory accesses at a compiler level, the speedup in *Debug* mode is better than in *Release* mode. Also we can see that among three versions of the algorithm, the tile-based method is the best and the streaming algorithm is the worst. In fact, since the length of the test sequences is not long, the only difference between simple- and tile-based algorithms is that tile-based algorithm introduces some optimization methods. The results show that our optimization methods are effective.

5.1.5.2 Test of Streaming Viterbi *Algorithm*

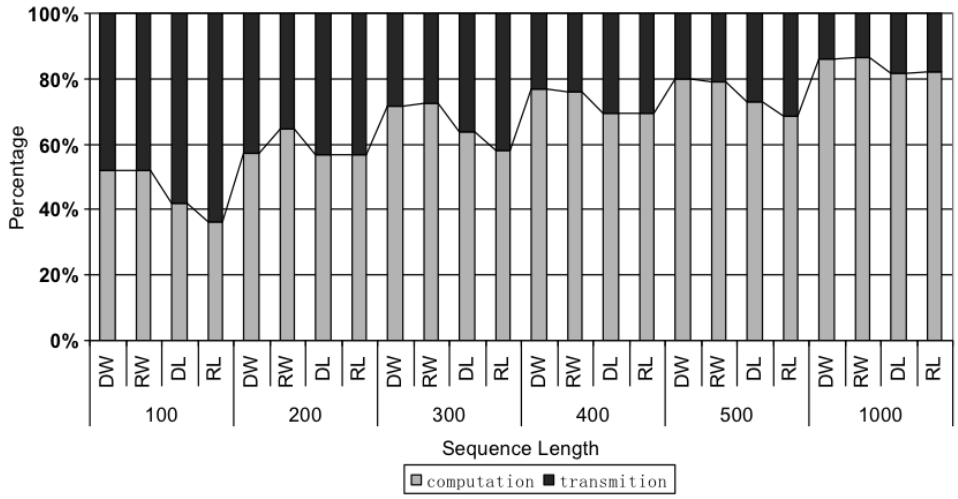


Figure 42: Results of the test on streaming *Viterbi* algorithm implementation for Biological Sequence Alignment.

Since there is data communicated between host and device memory, we must consider the communication cost. Here we test the time composed of computing and data transfer of the streaming implementation. The results are shown in Fig. 9. In Figure 42, we see that the longer the sequence is, the less percentage of time consumed in data transfer. This means that the computation and communication can be overlapped with each other better when the sequence length is longer. However,

these time of communication still can not be neglected.

5.1.5.3 Test of Tile-based Viterbi *Algorithm*

The selected homological segments and the size of the sub-sequence will significantly affect the algorithm's efficiency. We implement the serial algorithm of finding homological segments in *Java*. In our implementation, the length of the sub-sequences partitioned by homological segments is decided by the following three factors:

1) *K-mer* window length: When this is larger, the homological segment found will be better; however, this also means more computation and storage cost.
2) Homological-segment window length: With the growth of the homological-segment window length, there will be less homological segments detected, which means the sub-sequence length will be longer.
3) Homological-segment threshold: This threshold is used to score a segment of a sequence. The larger the threshold is, the fewer the number of homological segments will be detected.

In our test, we focus on how the average length of the sub-sequence will affect the final performance. Our tests are divided into two parts, the time of computing homological segments and the time of sequence alignment using our tile-based algorithm. The test results are shown in Figure 43. The time for computing homological segments is linearly increased. We do not include the time for computing homological segments in Figure 43 because we want to show the time changes for computing sub sequences separately.

From Figure 43, we see that with the growth of the average sequence length, the number of the segments decreases and the time for sequence alignment increases (there are some waves in the figure; this is because of the variation of sequence length

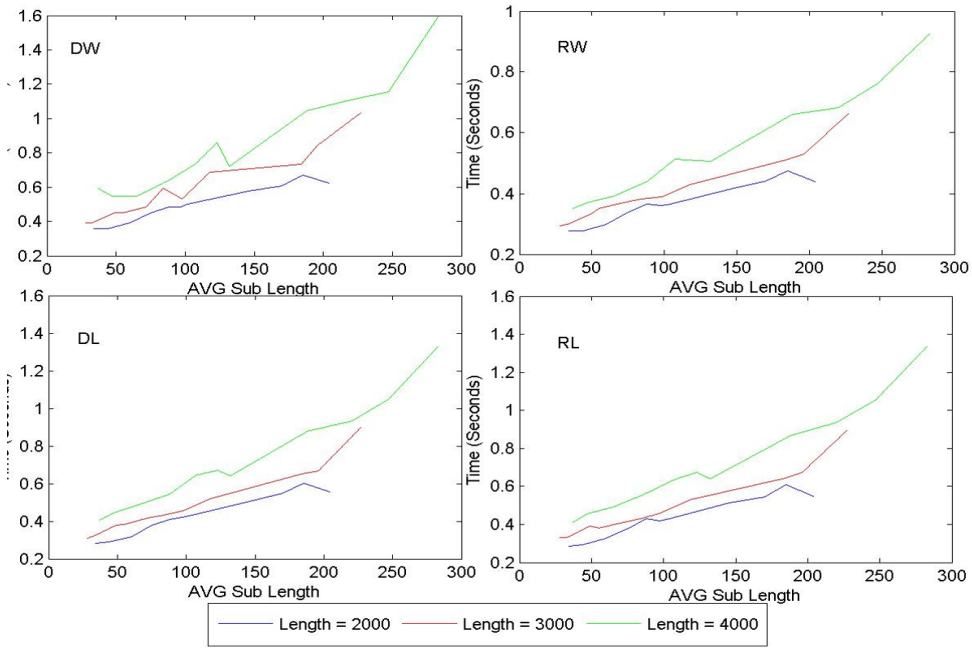


Figure 43: Results of testing the tile-based *Viterbi* Algorithm for Biological Sequence Alignment. Here we include only the time for computing sub sequences

for the given average length). We select the parameters, which are used to decide how to partition the sequence length of average length, to keep the residues length at 200. This is because when the average sequence length exceeds 200, the time of sequence alignment will increase much faster.

5.1.5.4 Testing of Long Sequence

The last test shows the comparison of streaming and tile-based implementations on longer sequences. For the running time of tile-based algorithm, we included both the time for computing tiles and the time for the parallelized *Viterbi* algorithm. Figure 44 shows that the tile-based implementation is an order of magnitude faster than the streaming implementation, because it only needs to calculate a portion of the dynamic programming matrix. In addition, the time growth of the tile-based implementation is lower. From Figure 44, we see that as the residue length increases from 2000 to 5000, the time for tile-based method increases only about 2 seconds under both the *Linux*

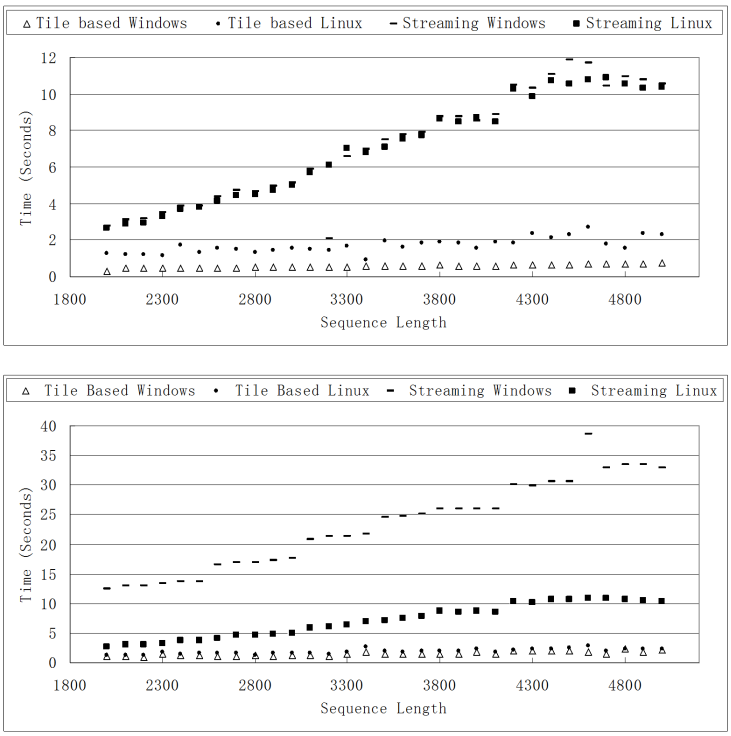


Figure 44: Results of testing on longer sequences, upper graph shows the results in Release mode and lower graph shows the results in Debug mode.

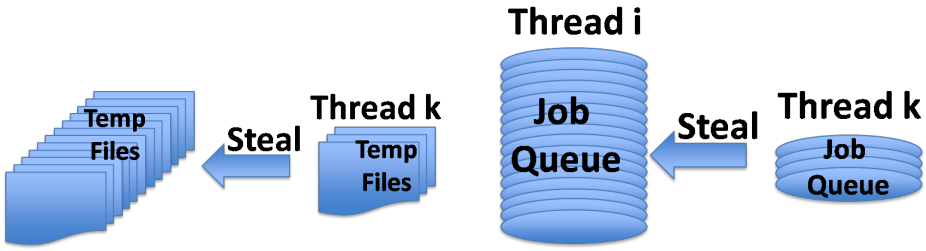
and *Windows* system, and the streaming-based algorithm grows at least 8 seconds. As the sequence length increases, the growth of the calculation time needed by our tile-based method is very low.

There are two special effects in the figure to explain. One is that for the tile-based method, the performance under *Windows* is faster than the *Linux* system, and it is different for streaming algorithm. The other is that the time for the streaming algorithm under the *Windows* system grows like a ladder. Both of these effects are because of the fact that, in *Windows*, the kernel initiated by *CUDA* cannot run too long consistently. Thus, we added some code to allow the kernel to sleep for some time for every 1000 kernel initiations.

5.2 *Parallelizing Branch and Bound Algorithms*

5.2.1 Parallel Speedup for *BnB DCJ* median algorithm

Since there are a lot of articles about parallel branch and bound algorithms [13], we will not dive into detail about the framework of the parallel algorithm. Here we discuss two load balancing strategies that we use in our shared memory multi-thread parallel algorithm. The first strategy is, when a thread has finished its work, it will check the intermediate files of other threads with the maximal upper bound value, if such files exits, it can "steal" the tasks of other threads by just renaming the files to its own, this strategy has very little overhead of synchronization. The second strategy is, when there is no files left of other threads, instead of "stealing" other threads' work, this thread just kill itself, and notify one of other threads with maximal expected work to do to fork their jobs to a new thread to continue searching.

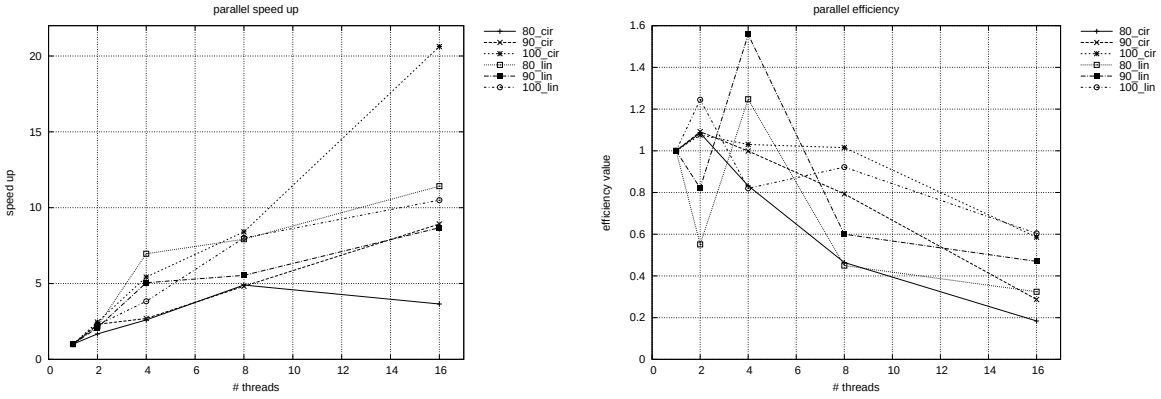


(a) load balancing on file (b) load balancing on job queue

Figure 45: Explanations of two different load balancing strategy.

We implement our parallel algorithm by using *JAVA* threads, and we perform the experiment on the same data set as we used in testing the streaming breakpoint graph method to solve the *DCJ* median problem, the speedup is averaged over 10 cases. We can see from Figure 46(a) that our algorithm achieved very good parallel speed up, especially on large data search space problems. For the circular chromosome, when $\kappa = 80$, the algorithm scales well up to 8 threads, and when $\kappa = 90$ we can achieve speed up close to 10 when using 16 threads. Surprisingly, we can see even

super-linear speed-ups for some cases such when $\kappa = 100$ and thread number is 16. The observation of super-linear speed up is due to the reduce of the search space when multiple thread is running. As for linear chromosomes, because the data we generated has larger search space and they are more evenly distributed, we can see that for all three kernel sizes, the algorithm scales well.



(a) parallel speed up (b) parallel efficiency

Figure 46: Parallel speed up.

Figure 46(b) shows the normalized efficiency for our parallel program which is calculated by $\frac{T_s}{T_p} \times \frac{W_p}{W_s}$ of which T_s/T_p is the serial/parallel execution time and W_s/W_p is the number of total processed nodes (total work) for serial/parallel program. In general, when increasing the number of threads, the parallel efficiency is reducing. There might be multiple reasons: 1) there is large overhead for Java thread scheduling, 2) the memory allocated is increasing with the number of threads growing, 3) there are overhead for synchronization when threads are doing load balancing.

5.2.2 Knowledge Learn from the Parallelization of Δ-Stepping Algorithm

Δ-stepping algorithm is the algorithm that is used to solve the single source shortest path problem (*SSSP*) [97]. Given a graph with V vertices and E edges. Traditionally the *Djkstra* algorithm has the complexity of $O(VlogV)$ which is optimial for this

problem, but it has no parallelism. And Belman-ford algorithm has the complexity of $O(V^2)$ while it has the parallelism of $O(V)$. Δ-stepping is the algorithm that take the advantage of both of these two algorithms, which trade off time complexity over parallelism. We can see it as a bucket implementation of Dijkstra's algorithm, and the parallelism is at the step of processing buckets in parallel.

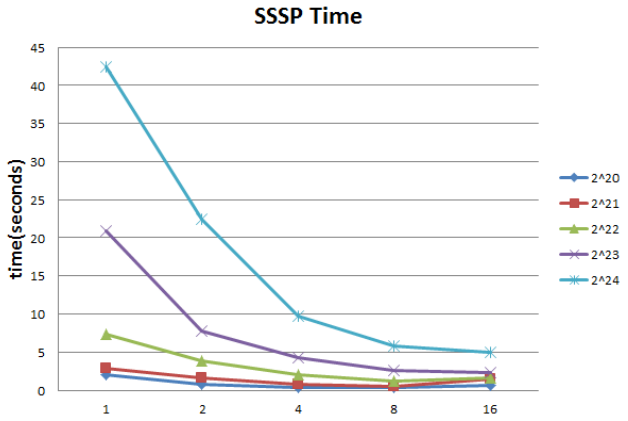


Figure 47: Results of the test on streaming *Viterbi* algorithm implementation for Biological Sequence Alignment.

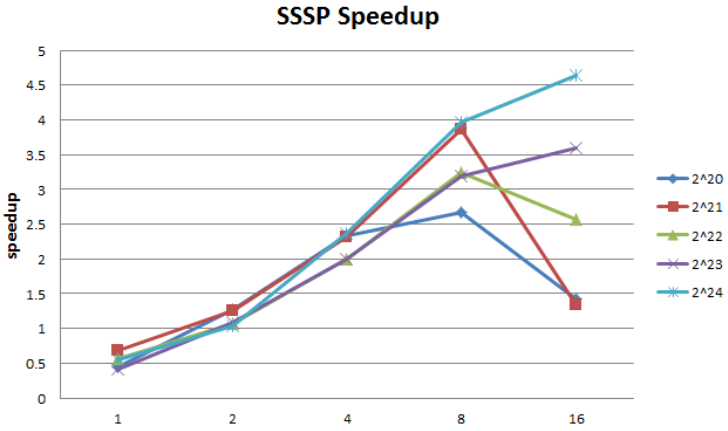


Figure 48: Results of the test on streaming *Viterbi* algorithm implementation for Biological Sequence Alignment.

In [95], the authors described the way of designing and implementing parallel Δ-stepping algorithm on *Cray-XMT* system. Following the similar strategy, we implement a parallel Δ-stepping on *Intel's Sandy bridge* system, with additional optimization methods as: 1) Since there is contentions when multiple threads are relaxing

edges that has the same end vertex. We use parallel partition method, partition edges to request array into 256 bins, and process the bins in parallel. 2) We use the bit array to store the buckets to save memory usage.

We conduct the experiment using different data sets generated by *Graph500 RMAT* data generator [32] which has the data size of $V = 2^{20}$ to 2^{24} with edge factor of 16. And the experimental results are shown in Figure 47 and Figure 48. We can see that in general there is an overhead of the same time spent on the computation of index and the time to put the vertices back to the right position. The lesson we learned from the parallel Δ-stepping is that, we can think irregular graph problems as "bucket" problems and achieve parallelism on processing the bucket. This strategy has been widely applied to other methods such as BFS. The question is, can we apply the bucket based method to the parallelization of BnB problems?

5.2.3 Design A Bucket Processing Based Parallel *BnB* Algorithm

With the advent of many core systems, the implementation of parallel BnB algorithm becomes more and more complicated. Meanwhile, because the traditional way of Parallelizing BnB algorithm requires a lot of load balancing, which is almost not affordable when there are hundreds of threads running at the different phases of BnB search. The lessons we learn from transferring irregular graph problems for *SSSP* into parallel bucket processing is that, at the extra cost of maintaining the bucket index, we can get the elegance of easy to program and scalability. These two features are very important because current many core architecture are based on the Single Instruction Multiple Data (*SIMD*). Which requires simplicity on the logic to process different data and big data level parallelism. Processing on a bucket can satisfy this requirement.

Algorithm 11: DoBucketBnB

Input: branch and bound problem
Output: branch and bound solution

```
1  Instance[num_t] = Init_instances() ;
2  bnb_seq(instance[0]) ;
3  upper_bound = instance[0].upper_bound ;
4  lower_bound = instance[0].lower_bound ;
5  list = init_search_list(upper_bound, lower_bound) ;
6  count[num_t][size_bucket] ;
7  start[num_t][size_bucket] ;
8  eliminate[num_t][MAX_BUCK_SIZE] ;
9  while upper_bound > lower_bound do
10     if list.num[current_score] > 0 then
11         //first step to calculate indexes ;
12         ComputePartition(list, Instance, count, start, eliminate) ;
13         //second step to calculate the start position of each thread at each bucket
           //which is based on prefix sum method. ;
14         calculate_start_position(count, start) ;
15         //third step to put the intermediate results generated by ;
16         //this iteration to the right bucket in the search list ;
17         ReorderList(list, Instance, start, eliminate) ;
18     else
19         increase lower bound or decrease upper bound ;
20 return solution with the best upper/lower bound;
```

In this section, we propose the design of a "bucket" based parallel BnB algorithm which is easy to implement and able to scale well to hundreds of threads. The basic idea behind this algorithm is, when we perform BnB search we put the intermediate results into different buckets based on its upper or lower bound. And we continue search by always picking up search nodes from the best (position at max upper bound or min lower bound) position, we call this bucket the best bucket. The idea is, we can process all the elements from the best bucket in parallel. And put the intermediate results into the destination bucket in parallel as well.

Challenges attached with this methods are, to begin with, to put the intermediate results into different buckets we need to know the exact index of which result node to

Algorithm 12: ComputePartition

```
Input: list, Instance, count, start, eliminate
Output: count, start, eliminate
1  forall search nodes encode[i] ∈ best bucket in parallel do
2      tid = get_thread_id() ;
3      to_search_node(Instance[tid], encode[i]) ;
4      num_branch = Instance[tid].get_branches(branch_code) ;
5      local_count = 0 ;
6      for j ∈ num_branch do
7          Instance[tid].to_branch(j, branch_code) ;
8          if Instance[tid]'s expect best score is worse than current best score then
9              eliminate[tid][local_count++] = false ;
10         else
11             eliminate[tid][local_count++] = true ;
12         count[tid][Instance[tid].bucket_pos]++ ;
13         Instance[tid].from_branch(j, branch_code) ;
```

be reside, otherwise, there will be contentions. Another problem is, the intermediate result may have the same destine bucket as the current best bucket, obviously reading and writing of the same position at the same time will cause data inconsistency. Last but not least, when we are processing a bucket, there might be the same intermediate results that is not qualified to be putted into a specific bucket because its sub-problems have been trimmed by bounds. Or, some intermediate results might yield better bounds. How to screen these unnecessary results and update these newly added information is another issue we need to tackle.

To solve these problems, firstly, we need to address the way to handle the computation of the indexes of different threads in the bucket. We use the parallel partition method to partition the work on the best bucket evenly to different threads, then each threads compute how much intermediate search nodes they will generate. After that, the work for each independent threads will be used to calculate a global index for each thread of the range of the index in every bucket range from global lower bound to global upper bound. Secondly, when the intermediate search node is going to be putted into

Algorithm 13: ReorderList

```
Input: list, Instance, start, eliminate
Output: reordered list
1  forall search nodes encode[i] ∈ best bucket in parallel do
2      tid = get_thread_id() ;
3      to_search_node(Instance[tid], encode[i]) ;
4      num_branch = Instance[tid].get_branches(branch_code) ;
5      local_count = 0 ;
6      for j ∈ num_branch do
7          Instance[tid].to_branch(j, branch_code) ;
8          if eliminate[tid][local_count + +] == false then
9              if Instance[tid].buck_id != current_bucket then
10                 list.add(Instance[tid].buck_id, start[tid][Instance[tid].buck_id]++) ;
11             else
12                 list.add(temporary_buck_id, start[tid][Instance[tid].buck_id]++) ;
13         if Instance[tid]'s evaluation score is better than current best score then
14             update upper or lower bound ;
15         Instance[tid].from_branch(j, branch_code) ;
16 copy data from temporary bucket to current bucket ;
```

the current best bucket, we put it into a separate temporary bucket first. After all the nodes in current bucket have been processed, all the intermediate nodes in the temporary bucket will be copied back. Lastly, to decide which intermediate search node to be eliminated during the bucket processing, we only use the current global upper or lower bound, no matter the number has been changed or not. To achieve this purpose, we add an additional bit array to keep track of which element has been eliminated. The detail of the method is shown in Algorithm 11, Algorithm 12 and Algorithm 13. And an example of using bucket based method to do parallel branch and bound search is as Figure 49 shows.

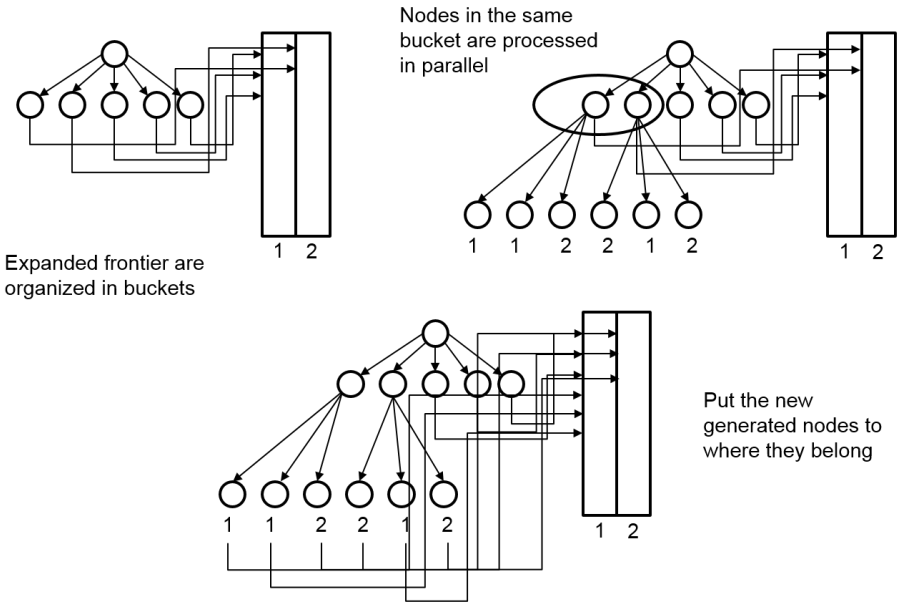


Figure 49: An example of using bucket based method to do parallel branch and bound search.

5.2.4 Algorithm Analysis

In this section, we will model the performance of two parallel *BnB* methods. Suppose the search space does not change when applying parallel methods. And for a sub-problem, it requires m amount of memory to store the information of this problem, and it takes c amount of computation to get the bound of this problem. If using bucket processing based algorithm, it requires m' amount of memory to store the bucket information of which bucket this nodes will be stored. If using thread based algorithm, there are o amount of overhead for load balancing. If we are using p number of threads, theoretically, the parallel model will be as equation 14 and equation 15 shows.

$$T_b = \frac{m + c + m'}{p} \tag{14}$$

$$T_t = \frac{m + c}{p} + o \tag{15}$$

We can see that there are two factors that affect the comparison of the two algorithms. The first factor is how intensive the computation is comparing with the memory cost. If the program is computational intensive, c will become the dominant part, and T_b and T_t will be almost the same. Another factor is how much does o cost for thread based method, since the load balancing method is not parallelizable and if o is large, it will strongly affect the time T_t.

5.2.5 Design and Implementation of *OPT-Kit*

OPT-Kit stands for Optimization Tool-kit for Parallelizing Discrete Combinatorial Problems in Emerging Platforms. It's design following such principles: 1) The user only need to consider how a instance is constructed, evaluated and branched, and all the other factors including the branch and bound strategy, parallelization will be hidden. 2) The polymorphism should not affect performance too much especially for scalability.

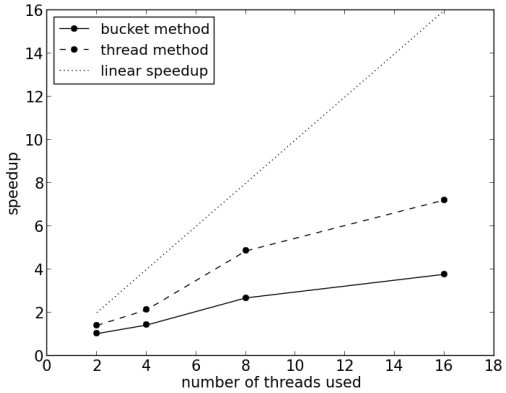
We implemented the bucket *BnB* algorithm using *CPP* and implemented two algorithms on top of it. The first algorithm is the well-known knapsack problem, and the second problem is the *DCJ-Indel-CD* distance algorithm. We run experiment using a machine that has two *Intel(R) Xeon(R) CPU X5680* @ 3.33GHz with 12 cores and *Intel Xeon Phi* with 60 cores. We compare our parallel bucket BnB algorithm with the traditional thread based method which is mentioned at the beginning of this chapter.

The experimental results for the parallelization on *CPU* for knapsack problem and *DCJ-Indel-CD* distance problem are shown in Figure 50. We can see that for the knapsack problem both thread based method out performed our bucket method. This is mainly due to the reason that the elemental computation (such as the evaluation and branch process) is cheap, and manipulating the search list becomes the dominate part. Since in the bucket based method, there is overhead of computing the indexes which is almost the same time cost as the following reorder process, and in knapsack problem, this deficiency will be exemplified. And for *DCJ-Indel-CD* problem elemental computation is expensive, and the overhead for computing indexes will be negligible, under such circumstance we can see that our bucket based method performed better than thread based method because thread method spend much of the time for load balancing. And we can see that our methods scales pretty well.

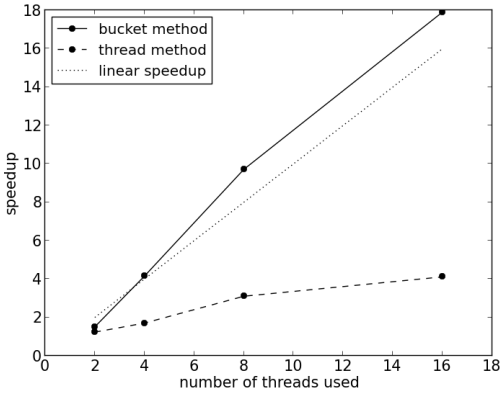
The experimental results for *Intel's MIC Xeon Phi* is also shown in Figure 50. We can see that in general, when running jobs on *Intel's MIC*, for the knapsack problem, the difference between thread based method and bucket based method is very close, while for the *DCJ-Indel-CD* problem, the bucket based method outperform the thread based method. To be specific, the bucket based method scales well on both of the problems except for the use of 128 threads in *DCJ-Indel-CD* problem, which is mainly due to the lack of work to do. And for thread based method, it stops scaling for the *DCJ-Indel-CD* problem when thread number is more than 16. As for the bandwidth, we can see that for the knapsack problem, the bucket based method achieved a better bandwidth while in the *DCJ-Indel-CD* problem, it perform better than thread based method until the thread number exceeds 64.

In general, for the problem with more memory intensive, the bucket based method

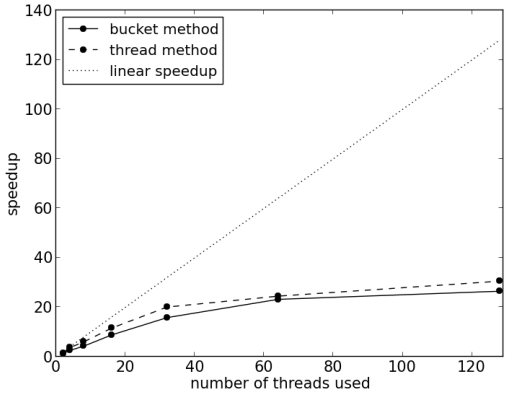
performs slightly worse than thread based method while for the computational intensive problem, the bucket based method outperform the thread based method. Considering that the bucket based method is much simpler than thread based method, it is promising to extend this method to the more complicated multi-node message passing parallelism.



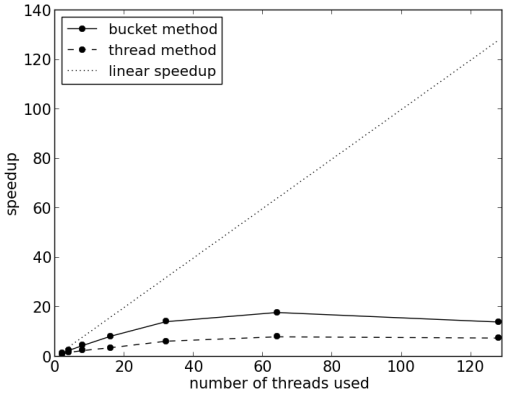
(a) parallel speed up *CPU* knapsack



(b) parallel speed up *CPU DCJ-Indel-CD* distance



(c) parallel speed up *MIC* knapsack



(d) parallel speed up *MIC DCJ-Indel-CD* distance

Figure 50: Parallel speed up for knapsack problem and *DCJ-Indel-CD* distance problem on *Intel*'s *Sandy Bridge* and *MIC* system.

Chapter VI

CONCLUSION AND FUTURE WORK

In this thesis we described the design and implementation of *DCJUC* which can help biologist infer the phylogenetic topology and reconstruct the ancestor gene order. We select the *DCJ-Indel-Exemplar* distance as the tool to estimate the dissimilarity between genomes, and adapt the *Lin-Kernighan* heuristic to solve the *DCJ* median problem with genomes of unequal contents. We proved that the use of the adequate sub-graphs is still sound with *BPG* of which not every vertex is 3-regular, therefore the search space can be dramatically reduced. We implemented the tree construction software using our new median solver, and applied the spectral partition method to help quickly build the tree. We incorporated the generalized adequate sub-graph (*GAS*) method to initialize the ancestor gene orders when the topology of the model tree is known. Nevertheless, there are still sheer amount of work to be done. To begin with, the divide and conquer method can be introduced in the distance computation for reducing the search space. Secondly, a good branch and bound strategy is needed to conquer the difficulty of triangular inequality and ambiguation problems. If these problems can not be solved, a more search space efficient and memory efficient version of *LK* algorithm is yet to be implemented. Thirdly, various of consensus tree algorithms are needed to help improve the phylogeny inference accuracy. Last but not the least, to deal with high resolution data, more algorithm engineering methods and high performance computing technologies are needed to improve the software's efficiency.

We also described the design and implementation of *OPT-Kit* which can help researchers to implement parallel branch and bound algorithms without paying too much attention to the parallel process and afraid of using new parallel architectures. The *OPT-Kit* scales well on both the traditional *Intel's CPU* architecture and emerging many core architectures, Which incorporated two parallel branch and bound methods such as thread based method and bucket processing based method. There are numorous things to be done in the future. The first thing is we need to port our algorithm into *GPU* to see how it performs. The second thing to do is to introduce the offload method into our software, because our software is limited by the size of the memory and the new many-core systems is short of memory. The third thing to do is extend our method to the *PGAS* programming model to see how it performed in the multi-node machine. The fourth thing to do is to add the *I/O* algorithm into our package so that it can process problems in a much larger scale. Last but not the least, since most of the combinatorial optimization problems can be mapped to linear programming problems, we should make our package support *LP* functions.

REFERENCES

[1] "Performance evaluation of load distribution strategies in parallel branch and bound computations," in *in Parallel Branch and Bound Computations Proc. 7th Symposium on Parallel and Distributed Processing (SPDP'95)*, pp. 402–405, Press, 1995.

[2] *2006 International Conference on Parallel Processing (ICPP 2006), 14-18 August 2006, Columbus, Ohio, USA*, IEEE Computer Society, 2006.

[3] Aji, A. M., chun Feng, W., Blagojevic, F., and Nikolopoulos, D. S., "Cell-swat: modeling and scheduling wavefront computations on the cell broadband engine.," in *Conf. Computing Frontiers* (Ramirez, A., Bilardi, G., and Gschwind, M., eds.), pp. 13–22, ACM, 2008.

[4] Alekseyev, M. A. and Pevzner, P. A., "Breakpoint graphs and ancestral genome reconstructions," *Genome Res*, vol. 19(5), pp. 943-57, 2009, May.

[5] Alekseyev, M. A. and Pevzner, P. A., "Colored de bruijn graphs and genome halving problem," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, pp. 98–107, 2006.

[6] Alekseyev, M. A. and Pevzner, P. A., "Are there rearrangement hotspots in the human genome?," *PLoS Computational Biology*, vol. 3, no. 11, 2007.

[7] Alekseyev, M. A. and Pevzner, P. A., "Multi-break rearrangements and chromosomal evolution," *Theor. Comput. Sci.*, vol. 395, pp. 193–202, Apr. 2008.

[8] Alexandrov, V. N., Lees, M., Krzhizhanovskaya, V. V., Dongarra, J., and Sloot, P. M. A., eds., *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013*, vol. 18 of *Procedia Computer Science*, Elsevier, 2013.

[9] Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., and Vialette, S., "A pseudo-boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes," in *Proceedings of the 2007 international conference on Comparative genomics*, RECOMB-CG'07, (Berlin, Heidelberg), pp. 16–29, Springer-Verlag, 2007.

[10] Angibaud, S., Fertin, G., Rusu, I., ThÃlvenin, A., and Vialette, S., "On the approximability of comparing genomes with duplicates.," *J. Graph Algorithms Appl.*, vol. 13, no. 1, pp. 19–53, 2009.

[11] Angibaud, S., Fertin, G., Rusu, I., and Vialette, S., "How pseudo-boolean programming can help genome rearrangement distance computation," in *in Comparative Genomics, RECOMB 2006 International Workshop, RCG 2006, ser. Lecture Notes in*, pp. 75–86, springer, 2006.

[12] Bader, D. A., Chandu, V. P., and Yan, M., "Exactmp: An efficient parallel exact solver for phylogenetic tree reconstruction using maximum parsimony," in *ICPP* [2], pp. 65–73.

[13] Bader, D. A., Hart, W. E., and Phillips, C. A., "Chapter 5 parallel algorithm design for branch and bound," in *Tutorials on Emerging Methodologies and Applications in Operations Research* (Greenberg, H., ed.), Oxford: Kluwer Academic Press, 2004.

[14] Bader, D. A., Moret, B. M. E., and Yan, M., "A linear-time algorithm for computing inversion distance between signed permutations with an experimental study," *Journal of Computational Biology*, vol. 8, pp. 483–491, 2001.

[15] Bafna, V. and Pevzner, P. A., "Sorting by reversals: Genome rearrangements in plant organelles and evolutionary history of x chromosome," *Mol. Biol. and Evol*, vol. 12, pp. 239–246, 1995.

[16] Bergeron, A., Mixtacki, J., and Stoye, J., "On sorting by translocations," in *Journal of Computational Biology*, pp. 615–629, Springer, 2005.

[17] Bergeron, A., Mixtacki, J., and Stoye, J., "A unifying view of genome rearrangements," in *WABI 2006. LNCS (LNBI)*, pp. 163–173, Springer, 2006.

[18] Bertrand, D., Gagnon, Y., Blanchette, M., and El-Mabrouk, N., "Reconstruction of ancestral genome subject to whole genome duplication, speciation, rearrangement and loss," in *Proceedings of the 10th international conference on Algorithms in bioinformatics*, WABI'10, (Berlin, Heidelberg), pp. 78–89, Springer-Verlag, 2010.

[19] Bhutkar, A., Schaeffer, S. W., Russo, S. M., Xu, M., Smith, T. F., and Gelbart, W. M., "Chromosomal rearrangement inferred from comparisons of 12 Drosophila genomes.," *Genetics*, vol. 179, no. 3, pp. 1657–80, 2008.

[20] Blanchette, M., Bourque, G., and Sankoff, D., "Breakpoint phylogenies," *Genome Informatics*, pp. 25–34, 1997.

[21] Blin, G., Chauve, C., and Fertin, G., "The breakpoint distance for signed sequences," in *Proc. CompBioNets 2004*, vol. Text in Algorithms, Volume 3, pp. 3–16, King's College London, 2004.

[22] Bourque, G. and Pevzner, P. A., "Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species," *Genome Res.*, vol. 12, no. 1, pp. 26–36, 2002.

[23] Bourque, G. and Pevzner, P. A., "Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species," *Genome Research*, vol. 12, pp. 26–36, Jan. 2002.

[24] Bourque, G., Yacef, Y., and El-Mabrouk, N., "Maximizing synteny blocks to identify ancestral homologs," in *Proceedings of the 2005 international conference on Comparative Genomics*, RCG'05, (Berlin, Heidelberg), pp. 21–34, Springer-Verlag, 2005.

[25] Braga, M. D. V., Willing, E., and Stoye, J., "Genomic distance with dcj and indels," in *Proceedings of the 10th international conference on Algorithms in bioinformatics*, WABI'10, (Berlin, Heidelberg), pp. 90–101, Springer-Verlag, 2010.

[26] Brown, M., Hughey, R., Krogh, A., Mian, I. S., Sjolander, K., and Haussler, D., "Using dirichlet mixture priors to derive hidden markov models for protein families.," in *ISMB* (Hunter, L., Searls, D. B., and Shavlik, J. W., eds.), pp. 47–55, AAAI, 1993.

[27] Bryant, D., "The complexity of calculating exemplar distances," in *Comparative Genomics* (Sankoff, D. and Nadeau, J., eds.), Kluwer, 2001.

[28] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P., "Brook for gpus: stream computing on graphics hardware," *ACM Trans. Graph.*, vol. 23, pp. 777–786, Aug. 2004.

[29] Budiu, M., Delling, D., and Werneck, R. F., "Dryadopt: Branch-and-bound on distributed data-parallel execution engines," in *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '11, (Washington, DC, USA), pp. 1278–1289, IEEE Computer Society, 2011.

[30] Caprara, A., "The reversal median problem," *INFORMS Journal on Computing*, vol. 15, 2003.

[31] Chauve, C., Fertin, G., Rizzi, R., and Vialette, S., "Genomes containing duplicates are hard to compare," in *Proc Int. Workshop on Bioinformatics Research and Applications (IWBRA)*, vol. 3992 of *LNCS*, (Reading, UK), p. 783ÃćâĆňâĂIJ790, Springer-Verlag, 2006.

[32] Checconi, F. and Petrini, F., "Massive data analytics: The graph 500 on ibm blue gene/q," *IBM Journal of Research and Development*, vol. 57, no. 1/2, p. 10, 2013.

[33] Chen, Z., Fu, B., and Zhu, B., "The approximability of the exemplar breakpoint distance problem.," in *AAIM* (Cheng, S.-W. and Poon, C. K., eds.), vol. 4041 of *Lecture Notes in Computer Science*, pp. 291–302, Springer, 2006.

[34] Chowdhury, S. A., Shackney, S., Heselmeyer-Haddad, K., Ried, T., Schaffer, A. A., and Schwartz, R., "Phylogenetic analysis of multiprobe fluorescence in situ hybridization data from tumor cell populations.," *Bioinformatics*, vol. 29, no. 13, pp. 189–198, 2013.

[35] Ciccarelli, F. and Miklós, I., eds., *Comparative Genomics, International Workshop, RECOMB-CG 2009, Budapest, Hungary, September 27-29, 2009. Proceedings*, vol. 5817 of *Lecture Notes in Computer Science*, Springer, 2009.

[36] Clark., A., "Drosophila 12 genomes consortium 2007 evolution of genes and genomes on the drosophila phylogeny," *Nature*, vol. 450: 203-218, 2007.

[37] Compeau, P. E. C., "A simplified view of dcj-indel distance," in *Proceedings of the 12th international conference on Algorithms in Bioinformatics*, WABI'12, (Berlin, Heidelberg), pp. 365–377, Springer-Verlag, 2012.

[38] Crainic, T., Gendron, B., and Centre for Research on Transportation (Montréal, Q., *Parallel Branch-and-bound Algorithms : Survey and Synthesis*. Publication (Centre for Research on Transportation (Montréal, Québec))), Centre for Research on Transportation = Centre de recherche sur les transports, 1993.

[39] Cunial, F. and Apostolico, A., "Phylogeny construction with rigid gapped motifs," *Journal of Computational Biology*, vol. 19, no. 7, pp. 911–927, 2012.

[40] da Silva, P. H., Braga, M. D. V., Machado, R., and Dantas, S., "Dcj-indel distance with distinct operation costs," in *Proceedings of the 12th international conference on Algorithms in Bioinformatics*, WABI'12, (Berlin, Heidelberg), pp. 378–390, Springer-Verlag, 2012.

[41] Dayhoff, M. O. and Schwartz, R. M., "Chapter 22: A model of evolutionary change in proteins," in *in Atlas of Protein Sequence and Structure*, 1978.

[42] Der, F., Informationstechnik, A., Stoye, J., Evers, D., Meyer, F., Herausgeber, I., Giegerich, R., Knoll, A., Ladkin, P., Ritter, H., Sagerer, G., and Wachsmuth, I., "Rose: Generating sequence families," 1997.

[43] Djerrah, A., Cun, B. L., Cung, V.-D., and Roucairol, C., "Bob++: Framework for solving optimization problems with branch-and-bound methods.," in *HPDC*, pp. 369–370, IEEE, 2006.

[44] Do, C. B., Mahabhashyam, M. S. P., Brudno, M., and Batzoglou, S., "Probcons: Probabilistic consistency-based multiple sequence alignment," *Genome Res*, vol. 15, pp. 330–340, 2005.

[45] Drosophila-12-Genomes-Consortium and Hultmark, D., "Evolution of genes and genomes on the drosophila phylogeny.," *Nature*, vol. 450, no. 7167, pp. 203–18, 2007. Drosophila 12 Genomes Consortium includes authors from 102 labs. Dan Hultmark contributed to the analysis of genes involved in immunity.

[46] Du, Z., Yin, Z., and Bader, D. A., "On accelerating iterative algorithms with cuda: A case study on conditional random fields training algorithm for biological sequence alignment," in *BIBM Workshops*, pp. 543–548, IEEE, 2010.

[47] Du, Z., Yin, Z., and Bader, D. A., "A tile-based parallel viterbi algorithm for biological sequence alignment on gpu with cuda," in *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Workshop Proceedings*, pp. 1–8, IEEE, 2010.

[48] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G., "Biological sequence analysis: probabilistic models of proteins and nucleic acids. cambridge univ," 1998.

[49] Durrett, R. and Interian, Y., "Genomic midpoints: Computation and evolutionary implications," 2007.

[50] Dymond, J. S., Richardson, S. M., Coombes, C. E., Babatz, T., Muller, H., Annaluru, N., Blake, W. J., Schwerzmann, J. W., Dai, J., Lindstrom, D. L., Boeke, A. C., Gottschling, D. E., Chandrasegaran, S., Bader, J. S., and Boeke, J. D., "Synthetic chromosome arms function in yeast and generate phenotypic diversity by design," *Nature*, vol. 477, pp. 471–476, Sept. 2011.

[51] Eckstein, J., Phillips, C. A., and Hart, W. E., "Pico: An object-oriented framework for parallel branch and bound," 2001.

[52] Eddy, S. R., "Multiple alignment using hidden markov models.," *Proc Int Conf Intell Syst Mol Biol*, vol. 3, pp. 114–120, 1995.

[53] Edgar, R. C., "Muscle: multiple sequence alignment with high accuracy and high throughput," *NUCLEIC ACIDS RES*, vol. 32, pp. 1792–1797, 2004.

[54] Edgar, R. C. and Sjolander, K., "Satchmo: Sequence alignment and tree construction using hidden markov models.," *Bioinformatics*, vol. 19, no. 11, pp. 1404–1411, 2003.

[55] Ediger, D., Riedy, E. J., Bader, D. A., and Meyerhenke, H., "Tracking structure of streaming social networks," in *IPDPS Workshops*, pp. 1691–1699, 2011.

[56] El-Mabrouk, N., Nadeau, J. H., and Sankoff, D., "Genome halving," in *Combinatorial Pattern Matching*, pp. 235–250, Springer, 1998.

[57] Farris, J. S., "Methods for Computing Wagner Trees," *Systematic Zoology*, vol. 19, no. 1, 1970.

[58] Felsenstein, J., "Evolutionary trees from DNA sequences: a maximum likelihood approach.," *Journal of molecular evolution*, vol. 17, no. 6, pp. 368–376, 1981.

[59] Fertin, G., Labarre, A., Rusu, I., Tannier, E., and Vialette, S., *Combinatorics of Genome Rearrangements*. The MIT Press, 1st ed., 2009.

[60] Fitch, W. M., "Toward defining the course of evolution: minimum change for a specific tree topology," *Systematic Zoology*, vol. 20, pp. 406–416, 1971.

[61] Fraenkel, A. S. and Lichtenstein, D., "Computing a perfect strategy for n*n chess requires time exponential in n.," in *ICALP* (Even, S. and Kariv, O., eds.), vol. 115 of *Lecture Notes in Computer Science*, pp. 278–293, Springer, 1981.

[62] Gao, N., Yang, N., and Tang, J., "Ancestral genome inference using a genetic algorithm approach," *PLoS ONE*, vol. 8, p. e62156, 05 2013.

[63] Gibbs, A. J. and Mcintyre, G. A., "The diagram, a method for comparing sequences, its use with amino acid and nucleotide sequences," *European Journal of Biochemistry*, vol. 16, no. 1, pp. 1–11, 1970.

[64] Goux, J.-P., Kulkarni, S., Yoder, M., and Linderoth, J., "An enabling framework for master-worker applications on the computational grid," in *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, HPDC '00, (Washington, DC, USA), pp. 43–, IEEE Computer Society, 2000.

[65] Grama, A. Y. and Kumar, V., "A survey of parallel search algorithms for discrete optimization problems," *ORSA JOURNAL ON COMPUTING*, vol. 7, 1993.

[66] Harding, S. and Banzhaf, W., "Fast genetic programming on gpus," in *Proceedings of the 10th European Conference on Genetic Programming, volume 4445 of LNCS*, pp. 90–101, Springer, 2007.

[67] Hart, P. E., Nilsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[68] Hartigan, J. A., "Minimum mutation fits to a given tree," *Biometrics*, vol. 29, pp. 53–65, 1973.

[69] Hartmanis, J., Immerman, N., and Sewelson, V., "Sparse sets in np-p: Exptime versus nexptime," *Information and Control*, vol. 65, pp. 158–181, May/June 1985.

[70] Heinecke, A., Karthikeyan, V., Mikhail, S., Alexander, K. V., S, D. R., Greg, H., Aniruddha, S. G., George, C., and Pradeep., D., "Design and implementation of the linpack benchmark for single and multi-node systems based on intel xeon phi coprocessor," in *Proceedings of the 2013 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '13, (Washington, DC, USA), pp. 1278–1289, IEEE Computer Society, 2013.

[71] Helsgaun, K., "An effective implementation of the lin-kernighan traveling salesman heuristic.," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[72] Henikoff, S. and Henikoff, J. G., "Amino acid substitution matrices from protein blocks.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 89, pp. 10915–10919, Nov. 1992.

[73] Hordijk, W. and Gascuel, O., "Improving the efficiency of spr moves in phylogenetic tree search methods based on maximum likelihood.," *Bioinformatics*, vol. 21, no. 24, pp. 4338–4347, 2005.

[74] Huson, D. H., Nettles, S., and Warnow, T., "Disk-covering, a fast-converging method for phylogenetic tree reconstruction.," *Journal of Computational Biology*, vol. 6, no. 3/4, pp. 369–386, 1999.

[75] Huson, D. H., Nettles, S., and Warnow, T., "Disk-covering, a fast-converging method for phylogenetic tree reconstruction.," *Journal of Computational Biology*, vol. 6, no. 3/4, pp. 369–386, 1999.

[76] Huson, D. H., Nettles, S. M., and Warnow, T. J., "Disk-covering, a fast-converging method for phylogenetic tree reconstruction," *JOURNAL OF COMPUTATIONAL BIOLOGY*, vol. 6, no. 3, pp. 369–386, 1999.

[77] Huson, D. H., Vawter, L., and Warnow, T., "Solving large scale phylogenetic problems using dcm2.," in *ISMB* (Lengauer, T., Schneider, R., Bork, P., Brutlag, D. L., Glasgow, J. I., Mewes, H.-W., and Zimmer, R., eds.), pp. 118–129, AAAI, 1999.

[78] Ishikawa, M., Toya, T., Hoshida, M., Nitta, K., Ogiwara, A., and Kanehisa, M., "Multiple sequence alignment by parallel simulated annealing.," *Computer Applications in the Biosciences*, vol. 9, no. 3, pp. 267–273, 1993.

[79] Jansson, J., Shen, C., and Sung, W.-K., "An optimal algorithm for building the majority rule consensus tree," in *RECOMB*, pp. 88–99, 2013.

[80] Kang, S., Tang, J., Schaeffer, S., and Bader, D., "Rec-dcm-eigen: Reconstructing a less parsimonious but more accurate tree in shorter time.," *PLoS One*, vol. 6, no. 8, p. e22483, 2011.

[81] Katoh, K. and Toh, H., "Recent developments in the mafft multiple sequence alignment program.," *Briefings in Bioinformatics*, vol. 9, no. 4, pp. 286–298, 2008.

[82] Korf, R. E., "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial Intelligence*, vol. 27, pp. 97–109, 1985.

[83] Korf, R. E., "Best-first frontier search with delayed duplicate detection," in *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pp. 650–657, AAAI Press, 2004.

[84] Korf, R. E., Zhang, W., Thayer, I., and Hohwald, H., "Frontier search," *J. ACM*, vol. 52, no. 5, pp. 715–748, 2005.

[85] Lafond, M., Swenson, K. M., and El-Mabrouk, N., "An optimal reconciliation algorithm for gene trees with polytomies," in *Proceedings of the 12th international conference on Algorithms in Bioinformatics*, WABI'12, (Berlin, Heidelberg), pp. 106–122, Springer-Verlag, 2012.

[86] Langdon, W. B. and Banzhaf, W., "A simd interpreter for genetic programming on gpu graphics cards," 2008.

[87] Lenne, R., Solnon, C., Stutzle, T., Tannier, E., and Birattari, M., "Reactive Stochastic Local Search Algorithms for the Genomic Median Problem," in *Eighth European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP)* (Carlos Cotta, J. v. H., ed.), LNCS, pp. 266–276, Springer, Mar. 2008.

[88] Lin, Y., Hu, F., Tang, J., and Moret, B., "Maximum likelihood phylogenetic reconstruction from high-resolution whole-genome data and a tree of 68 eukaryotes," in *Proc. 18th Pacific Symp. on Biocomputing*, PSB'13, (Washington, DC, USA), pp. 285–296, IEEE Computer Society, 2013.

[89] Lin, Y., Rajan, V., and Moret, B. M. E., "Fast and accurate phylogenetic reconstruction from high-resolution whole-genome data and a novel robustness estimator," in *Proceedings of the 2010 international conference on Comparative genomics*, RECOMB-CG'10, (Berlin, Heidelberg), pp. 137–148, Springer-Verlag, 2010.

[90] Liu, W., Schmidt, B., Voss, G., and Wittig, W. M., "Streaming Algorithms for Biological Sequence Alignment on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 9, pp. 1270–1281, 2007.

[91] Ma, J., *Reconstructing contiguous regions of an ancestral genome*. PhD thesis, University Park, PA, USA, 2006. AAI3248364.

[92] Ma, J., "A probabilistic framework for inferring ancestral genomic orders," in Park *et al.* [111], pp. 179–184.

[93] Ma, J., Ratan, A., Raney, B. J., Suh, B. B., Zhang, L., Miller, W., and Haussler, D., "Dupcar: Reconstructing contiguous ancestral regions with duplications," *Journal of Computational Biology*, vol. 15, no. 8, pp. 1007–1027, 2008.

[94] Maddison, D. R., Swofford, D. L., and Maddison, W. P., "Nexus: An extensible file format for systematic information," *Systematic Biology*, vol. 46, pp. 590–621, December 1997.

[95] Madduri, K., Bader, D. A., Berry, J. W., and Crobak, J. R., "An experimental study of a parallel shortest path algorithm for solving large-scale graph instances.," in *ALENEX*, SIAM, 2007.

[96] Manavski, S. and Valle, G., "Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment," *BMC Bioinformatics*, vol. 9, no. S-2, 2008.

[97] Meyer, U. and Sanders, P., "Delta-stepping: a parallelizable shortest path algorithm.," *J. Algorithms*, vol. 49, no. 1, pp. 114–152, 2003.

[98] Mingfu Shao, Y. L. and Moret, B., "An exact algorithm to compute the dcj distance for genomes with duplicate genes," in *RECOMB 2013, ser. Lecture Notes in CS*, pp. 1–10, springer, 2013.

[99] Moret, B. M., Wyman, S., Bader, D. A., Warnow, T., and Yan, M., "A new implementation and detailed study of breakpoint analysis.," *Pac Symp Biocomput*, pp. 583–594, 2001.

[100] Moret, B. M. E., Tang, J., san Wang, L., and Warnow, Y., "Steps toward accurate reconstructions of phylogenies from gene-order data," *J. Comput. Syst. Sci*, vol. 65, pp. 508–525, 2002.

[101] Moret, B. M. E., Wang, L.-S., Warnow, T., and Wyman, S. K., "New approaches for reconstructing phylogenies from gene order data.," in *ISMB (Supplement of Bioinformatics)*, pp. 165–173, 2001.

[102] Moret, B. M. E., Wyman, S., Bader, D. A., Warnow, T., and Yan, M., "A new implementation and detailed study of breakpoint analysis," 2001.

[103] Morgenstern, B., "Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment.," *Bioinformatics*, vol. 15, no. 3, pp. 211–218, 1999.

[104] Nguyen, C. T., Tay, Y. C., and Zhang, L., "Divide-and-conquer approach for the exemplar breakpoint distance," *Bioinformatics*, vol. 21, pp. 2171–2176, May 2005.

[105] Nickolls, J., Buck, I., Garland, M., and Skadron, K., "Scalable parallel programming with cuda," *Queue*, vol. 6, pp. 40–53, Mar. 2008.

[106] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, June 2011.

[107] O. Green, R. M. and Bader, D., "A fast algorithm for incremental betweenness centrality," in *4th ASE/IEEE International Conference on Social Computing*, 2012.

[108] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A. E., and Purcell, T., "A survey of general-purpose computation on graphics hardware," 2007.

[109] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., KrÃijger, J., Lefohn, A., and Purcell, T. J., "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[110] Park, J., Tang, P. T. P., Smelyanskiy, M., Kim, D., and Benson, T., "Efficient backprojection-based synthetic aperture radar computation with many-core processors," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, (Los Alamitos, CA, USA), pp. 28:1–28:11, IEEE Computer Society Press, 2012.

[111] Park, T., Tsui, S. K.-W., Chen, L., Ng, M. K., Wong, L., and Hu, X., eds., *2010 IEEE International Conference on Bioinformatics and Biomedicine,*

BIBM 2010, Hong Kong, China, 18 - 21 December 2010, Proceedings, IEEE Computer Society, 2010.

[112] Pe'er, I. and Shamir, R., "The median problems for breakpoints are np-complete," *Elec. Colloq. on Comput. Complexity*, vol. 71, 1998.

[113] Pevzner, P. A., *Computational molecular biology - an algorithmic approach*. MIT Press, 2000.

[114] Pham, S. K. and Pevzner, P. A., "Drimm-synteny: decomposing genomes into evolutionary conserved segments.," *Bioinformatics*, vol. 26, no. 20, pp. 2509–2516, 2010.

[115] Phillips, C. A. and Hart, W. E., "Pebbl 1.0 user's guide jonathan eckstein a."

[116] Poli, R., Langdon, W. B., and McPhee, N. F., *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[117] Przytycka, T. M. and Sagot, M.-F., eds., *Algorithms in Bioinformatics - 11th International Workshop, WABI 2011, Saarbrücken, Germany, September 5-7, 2011. Proceedings*, vol. 6833 of *Lecture Notes in Computer Science*, Springer, 2011.

[118] Rajan, V., Xu, A. W., Lin, Y., Swenson, K. M., and Moret, B. M. E., "Heuristics for the inversion median problem."

[119] Robson, J. M., "The complexity of go.," in *IFIP Congress*, pp. 413–417, 1983.

[120] Robson, J. M., "N by n checkers is exptime complete.," *SIAM J. Comput.*, vol. 13, no. 2, pp. 252–267, 1984.

[121] Roshan, U., Moret, B. M. E., Warnow, T., and Williams, T. L., "Rec-i-dcm3: A fast algorithmic technique for reconstructing large phylogenetic trees," in *In Proc. IEEE Computer Society Bioinformatics Conference (CSB 2004)*, pp. 98–109, IEEE Press, 2004.

[122] Saitou, N. and Nei, M., "The neighbor-joining method: a new method for reconstructing phylogenetic trees.," *Molecular biology and evolution*, vol. 4, pp. 406–425, July 1987.

[123] Sankoff, D., "Genome rearrangement with gene families.," *Bioinformatics*, vol. 15, no. 11, pp. 909–917, 1999.

[124] Sankoff, D. and Blanchette, M., "The median problem for breakpoints in comparative genomics.," in *COCOON* (Jiang, T. and Lee, D. T., eds.), vol. 1276 of *Lecture Notes in Computer Science*, pp. 251–264, Springer, 1997.

[125] Sankoff, D. and Haque, L., "Power boosts for cluster tests," in *Proceedings of the 2005 international conference on Comparative Genomics*, RCG'05, (Berlin, Heidelberg), pp. 121–130, Springer-Verlag, 2005.

[126] Sawa, G., Dicks, J., and Roberts, I. N., "Current approaches to whole genome phylogenetic analysis.," *Brief Bioinform*, vol. 4, pp. 63–74, Mar. 2003.

[127] Schaeffer, S. W., Bhutkar, A., McAllister, B. F., Matsuda, M., Matzkin, L. M., O'Grady, P. M., Rohde, C., Valente, V. L. S., Aguadé, M., Anderson, W. W., Edwards, K., Garcia, A. C. L., Goodman, J., Hartigan, J., Kataoka, E., Lapoint, R. T., Lozovsky, E. R., Machado, C. A., Noor, M. A. F., Papaceit, M., Reed, L. K., Richards, S., Rieger, T. T., Russo, S. M., Sato, H., Segarra, C., Smith, D. R., Smith, T. F., Strelets, V., Tobari, Y. N., Tomimura, Y., Wasserman, M., Watts, T., Wilson, R., Yoshida, K., Markow, T. A., Gelbart, W. M., and Kaufman, T. C., "Polytene Chromosomal Maps of 11 Drosophila Species: The Order of Genomic Scaffolds Inferred From Genetic and Physical Maps," *Genetics*, vol. 179, pp. 1601–1655, July 2008.

[128] Sforza, C. L. L. and Edwards, A. W. F., "Phylogenetic analysis: Models and estimation procedures," *American Journal of Human genetics*, vol. 19, pp. 223–257, 1967.

[129] Shao, M. and Lin, Y., "Approximating the edit distance for genomes with duplicate genes under dcj, insertion and deletion," *BMC Bioinformatics*, vol. 13, no. S-19, p. S13, 2012.

[130] Shinano, Y., Higaki, M., and Hirabayashi, R., "A generalized utility for parallel branch and bound algorithms," in *Proceedings of the 7th IEEE Symposium on Parallel and Distributeed Processing*, SPDP '95, (Washington, DC, USA), pp. 392–, IEEE Computer Society, 1995.

[131] Sjolander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I. S., and Haussler, D., "Dirichlet mixtures: a method for improved detection of weak but significant protein sequence homology.," *Computer Applications in the Biosciences*, vol. 12, no. 4, pp. 327–345, 1996.

[132] Sokal, R. R. and Michener, C. D., "A statistical method for evaluating systematic relationships," *University of Kansas Scientific Bulletin*, vol. 28, pp. 1409–1438, 1958.

[133] Sokal, R. R. and Michener, C. D., "A statistical method for evaluating systematic relationships," *University of Kansas Scientific Bulletin*, vol. 28, pp. 1409–1438, 1958.

[134] Stamatakis, A., "RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models," *Bioinformatics/computer Applications in The Biosciences*, vol. 22, pp. 2688–2690, 2006.

[135] Swenson, K. M., Doroftei, A., and El-Mabrouk, N., "Gene tree correction for reconciliation and species tree inference.," *Algorithms for Molecular Biology*, vol. 7, p. 31, 2012.

[136] Tannier, E., Zheng, C., and Sankoff, D., "Multichromosomal genome median and halving problems," in *Proceedings of the 8th international workshop on Algorithms in Bioinformatics*, WABI '08, (Berlin, Heidelberg), pp. 1–13, Springer-Verlag, 2008.

[137] Tantar, A. A., Melab, N., Talbi, E. G., Parent, B., and Horvath, D., "A parallel hybrid genetic algorithm for protein structure prediction on the computational grid," *Future Gener. Comput. Syst.*, vol. 23, pp. 398–409, Mar. 2007.

[138] Thompson, J. D., Higgins, D. G., and Gibson, T. J., "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice.," *Nucleic acids research*, vol. 22, pp. 4673–4680, Nov. 1994.

[139] Ullmann, J. R., "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, pp. 31–42, Jan. 1976.

[140] Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M., and Shringarpure, A., "On the limits of gpu acceleration," in *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, HotPar'10, (Berkeley, CA, USA), pp. 13–13, USENIX Association, 2010.

[141] Williams, S., Kalamkar, D. D., Singh, A., Deshpande, A. M., Van Straalen, B., Smelyanskiy, M., Almgren, A., Dubey, P., Shalf, J., and Oliker, L., "Optimization of geometric multigrid for emerging multi- and manycore processors," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, (Los Alamitos, CA, USA), pp. 96:1–96:11, IEEE Computer Society Press, 2012.

[142] Xu, A. W., "Dcj median problems on linear multichromosomal genomes: Graph representation and fast exact solutions," in Ciccarelli and Miklós [35], pp. 70–83.

[143] Xu, A. W., "A fast and exact algorithm for the median of three problem: A graph decomposition approach," *Journal of Computational Biology*, vol. 16, no. 10, pp. 1369–1381, 2009.

[144] Xu, A. W., "On exploring genome rearrangement phylogenetic patterns.," in *RECOMB-CG* (Tannier, E., ed.), vol. 6398 of *Lecture Notes in Computer Science*, pp. 121–136, Springer, 2010.

[145] Xu, A. W. and Moret, B. M. E., "Gasts: Parsimony scoring under rearrangements," in Przytycka and Sagot [117], pp. 351–363.

[146] Xu, A. W. and Sankoff, D., "Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem," in *Proceedings of the 8th international workshop on Algorithms in Bioinformatics*, WABI '08, (Berlin, Heidelberg), pp. 25–37, Springer-Verlag, 2008.

[147] Xu, Y., Ralphs, T. K., Ladanyi, L., and Saltzman, M. J., "Alps: A framework for implementing parallel search algorithms," in *In Proceedings of the Ninth INFORMS Computing Society Conference*, pp. 319–334, 2005.

[148] Yan, M., *High-performance algorithms for phylogeny reconstruction with maximum parsimony*. PhD thesis, 2004. AAI3129663.

[149] Yan, M. and Bader, D. A., "D.a.: Fast character optimization in parsimony phylogeny reconstruction," tech. rep., 2003.

[150] Yancopoulos, S., Attie, O., and Friedberg, R., "Efficient sorting of genomic permutations by translocation, inversion and block interchange," *Bioinformatics*, vol. 21, pp. 3340–3346, Aug. 2005.

[151] Ye, F., Guo, Y., Lawson, A., and Tang, J., "Improving tree search in phylogenetic reconstruction from genome rearrangement data.," in *WEA* (Demetrescu, C., ed.), vol. 4525 of *Lecture Notes in Computer Science*, pp. 352–364, Springer, 2007.

[152] Yin, Z., Tang, J., Schaeffer, S. W., and Bader, D. A., "Streaming breakpoint graph analytics for accelerating and parallelizing the computation of dcj median of three genomes," in Alexandrov *et al.* [8], pp. 561–570.

[153] Zhang, J., "Evolution by gene duplication: an update," *Trends in Ecology & Evolution*, vol. 18, pp. 292–298, June 2003.

[154] Zhang, W. and Korf, R. E., "Depth-first vs. best-first search: new results," in *Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, pp. 769–775, AAAI Press, 1993.

[155] Zhang, W. and Korf, R. E., "Performance of linear-space search algorithms," *Artif. Intell.*, vol. 79, no. 2, pp. 241–292, 1995.

[156] Zhang, y., Hu, F., and Tang, J., "A mixture framework for inferring ancestral gene orders," *BMC Genomics*, vol. 13(Suppl 1): S7, 2012.

[157] Zhou, R. and Hansen, E. A., "Breadth-first heuristic search," *Artif. Intell.*, vol. 170, pp. 385–408, Apr. 2006.

[158] Zola, J., Yang, X., Rospondek, A., and Aluru, S., "Parallel-tcoffee: A parallel multiple sequence aligner.," in *ISCA PDCS* (Chaudhry, G. and Lee, S.-Y., eds.), pp. 248–253, ISCA, 2007.

VITA

Zhaoming Yin was born in the city of Hengyang, the second largest city in Hunan Province. He spent 2 years studying Chemical Engineering at the Hunan University then transfered his major to Software Engineering. After his graduation, he spent three years to get his Master Degree from Peking University, where he was doing projects and research related to Nature Language Processing and High Performance Computing.

Since Aug 2010, Zhaoming moved to Atlanta, a beautiful city in United States, and started his PhD. studying Computational Science and Engineering at Georgia Institute of Technology under the supervision of Prof. David A. Bader. Working on genome rearrangement based phylogenetic tree construction algorithms, which is consisted of multiple *NPC* problems.

PUBLICATIONS

[1] Zhaoming Yin, Jijun Tang, Stephen Schaeffer, David A. Bader, A Lin-Kernighan Heuristic for the DCJ Median Problem of Genomes with Unequal Contents. (Submitted, COCOON 2014 : International Computing and Combinatorics Conference, Atlanta, USA)

[2] Satish Nadathur et. al Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets, SIGMOD 2014, Snowbird, USA 2014

[3] Zhaoming Yin, Jijun Tang, Stephen Schaeffer, David A. Bader, Streaming Breakpoint Graph Analytics for Accelerating and Parallelizing DCJ Median of Three Genomes.

International Conference on Computational Science, Barcelona, Spain, June, 2013

[4] Zhihui Du, Zhaoming Yin, Wenjie Liu, David A. Bader On Accelerating Iterative Algorithms with CUDA: A Case Study on Conditional Random Fields Training Algorithm for Biological Sequence Alignment Workshop on Data mining of Next-Generation Sequencing Data (In conjunction with BIBM 2010) Hongkong, China, Dec 17, 2010

[5] Zhihui Du, Zhaoming Yin, David. A. Bader A Tile-based Parallel Viterbi Algorithm for Biological Sequence Alignment on GPU with CUDA IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2010 HiComb Workshop, Atlanta USA.

[6] Zhaoming Yin, Huarui Zhang Research on Chinese n-gram Statistical Rule and its application 14th Youth Conference on Communication (YCC) 2009, Dalian, China. (ISTP: 000270587500121)