DETECTION AND CHARACTERIZATION OF NOVEL STRUCTURAL ALTERATIONS IN TRANSCRIBED SEQUENCES

by

Deniz Yorukoglu

B.Sc., Sabanci University, 2009

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

APPROVAL

Name:	Deniz Yorukoglu
Degree:	Master of Science
Title of Thesis:	Detection and Characterization of Novel Structural Alterations in Transcribed Sequences

Examining Committee: Dr. Funda Ergun
Chair

Dr. Cenk Sahinalp, Senior Supervisor

Dr. Inanc Birol, Supervisor

Dr. Cedric Chauve, SFU Examiner

Date Approved: 9 August 2011

Abstract

One of the key problems in computational genomics is that of identifying structural variations between two sequences of genomic origin. Recently, with the advent of high-throughput sequencing of transcriptomes (RNA-seq), transcriptional structural variation studies also came into prominence.

This study introduces two novel frameworks for aligning transcribed sequences to the genome with high sensitivity to structural alterations within the transcript. (1) A pairwise nucleotide-level alignment model and (2) a faster lower-sensitivity solution based on chaining homologous substrings between the transcript and the genome.

A further contribution of this study is a stand-alone transcriptome-to-genome alignment tool, which can comprehensively identify and characterize transcriptional events (duplications, inversions, rearrangements and fusions); suitable for high-throughput structural variation studies involving long transcribed sequences with high similarity to their genomic origin.

Reported results include experiments upon simulated datasets of transcriptional events and RNA-seq assemblies of a human prostate cancer individual.

To my supporting family and my lovely nephew, Eren.

"By 2015, we will see the beginnings of a real transformation in the therapeutics of medicine, which by 2020 will have touched virtually every disorder, ... And the drugs that we give in 2020 will for the most part be those that were based on the understanding of the genome, and the things that we use today will be relegated to the dust bin."

— *Francis Collins*

Acknowledgments

First and foremost, I would like to thank my senior supervisor Dr. Cenk Sahinalp for his utmost support throughout my MSc studies, for introducing me to the field of Computational Biology and training me as an academic researcher within this field. I am very grateful for his guidance throughout this study, for his constructive comments and helpful suggestions.

I would also like to thank my thesis supervisor Dr. Inanc Birol, for suggesting this problem as a potential research topic. This study would not have been possible without the collaboration with his research group at Michael Smith Genome Sciences Centre.

I owe a significant part of my success throughout my MSc studies to the personal communications I have had with my co-workers and collaborators. Therefore, I would like to thank my friends in SFU Lab for Computational Biology and Trans-Abyss team at Michael Smith Genome Sciences Centre for their helpful discussions. I would especially like to thank Faraz Hach for his contributions and guidance throughout the software development process.

I would also like to thank Dr. Colin Collins at Vancouver Prostate Centre for providing the real-life data we have used for this study.

I am also very grateful to the Pacific Institute for the Mathematical Sciences, for their funding and support through the IGTC Mathematical Biology Fellowship Program.

Finally and most importantly, I would like thank my family for their invaluable support throughout these two years of my life that resulted in this study. I dedicate this work to family and my charmingly sweet 6-year old nephew, Eren. I sincerely hope that one day; if he wishes to pursue a scientific career, this study will act as a good reference to his academic endeavours.

Contents

List of Tables

Chapter 1

Introduction

In the past few years, next-generation sequencing has transformed genome studies. Sanger-based sequencing technology has been replaced with novel technologies (e.g. Illumina, 454, PacBio, SoliD) that produce typically shorter reads at an unprecedented speed and cost, enabling the realization of tera-base scale high-throughput genome sequencing projects such as the 1000 Genomes Project. Unfortunately new sequencing technologies not only provide reads that are short, but also have high sequencing error rate, both contributing to the level of ambiguity in determining the locus of each read on the reference human genome sequence. In order to resolve this ambiguity in an efficient and reliable manner, novel and sophisticated algorithms and software tools are needed. Even though recent developments of next-generation sequencing provide longer sequences (100-150bp) than initially obtained (20-30bp), the increase in read length is offset by the increase in the number and type of errors that need to be tolerated for analyzing these reads, keeping mapping ambiguity at a high level.

Longer read lengths are not only difficult to tackle due to higher sequencing or assembly errors but also due to structural variations in the donor genome with respect to the reference. These variations can be in the form of insertions, deletions, inversions and translocations as well as duplications in genomic sequences. The algorithms to be developed should thus handle increasing read lengths, with higher number of errors and different error structures in the form of not only mismatches and indels[1], but also these structural alterations that

[1]In this thesis, we denote indels as single nucleotide edit operations in a sequence that result in a single base insertion or deletion.

cannot be detected by nucleotide level analysis. The developed algorithms should also be fast and memory efficient enough to be able to handle very large datasets, which is a necessity for sequence analysis studies to catch up with the current explosion of sequence information in terms of both size and variety.

The advent of next-generation sequencing did not only affect the studies upon genomic sequences but also transcriptomics. From a mathematical point of view, transcriptomic sequences are relatively short polynucleotide sequences (consisting of 10s up to 10000s of nucleotides) that are copied (transcribed) from a substring of the genome (a gene). Apart from the transcription process; in order to take its form as a mature transcript (e.g. mRNA), some of these sequences also go under the process of splicing, which is the transcriptional procedure of cutting out several non-coding sub-strings (introns), and stitching back the remaining pieces, which are called exons.

Even though the scale of a whole transcriptome database is relatively smaller than a whole genome database[2], transcriptome datasets are more difficult to analyse due to their spliced nature and their possibility of containing structural alterations from its wild-type form, such as transcriptional duplications, inversions, fusions (trans-splicing/read-through) and rearrangements (non-co-linear transcription).

This thesis is a theoretical and experimental study upon efficient high-throughput identification of such structural alterations within transcriptome datasets with respect to a reference genome.

The remainder of the *Introduction* chapter is organized as follows:

- *Motivation*: This section contains a literature survey of research upon chimeric and non-co-linear transcripts as well as the significance of the current study for transcriptome research.

- *Methods Background*: In this section, already existing methods in the literature for finding structural alterations in transcriptomic sequences are described and their limitations are argued.

- *Contribution of Current Study*: This section details the deliverables of the current study in terms of theoretical results and software.

[2]Depending on the species and how deep the transcriptome database is constructed (including isoforms), from around %3 up to %20 of the size of the genome (according RefSeq and GenBank transcript databases).

- *Organization of Thesis*: The section breaks down the general structure of this thesis document for the remainder of the chapters.

1.1 Motivation

The phenomenon of structural alterations during transcription has been known for some time for simpler animals (e.g. nematodes [18]), yet only recently these transcriptional abnormalities have started to be investigated in higher mammals.

Even though there is an element of doubt with regard to whether these findings are authentic or experimental artifacts[19], there has also been growing evidence and interest upon this research area within the scientific community[13][43][28][3][16], and some of these studies suggest a correlation between the extent of these structural alterations and the abnormal activity of a cell (e.g. tumor cells)[13][43].

Due to the growing interest to the problem of detecting transcriptional variations, many computational tools were developed upon mining mappings of RNA-seq reads to the genome for detection and analysis of transcriptional abnormalities [35][26][20][12][4][29][38][31][30], most of which specifically focus on identifying gene fusions.

A common drawback of these methods is being limited upon RNA-seq short-read mappings (or alternatively single-split mappings) for variation discovery and thus their inability to scale well the progressing transcriptome sequencing technology. As next-generation sequencing currently provides longer reads (100-150bp) than before and is expected to get longer ($\geq$1000bp) with the emergence of new technologies such as PacBio sequencing; there is a much higher possibility of each transcript read to contain sequences from more than two exons, in contrast to the short read samples taken from the transcript. In such cases, semi-global short-read mapping and split-mapping methods will be rendered ineffective due to their inability to locate structurally important loci for structural variation discovery.

Furthermore, longer read sequencing technologies also allow more accurate de novo transcriptome assembly. Each additional length carrying exponentially higher information value; in near future highly accurate whole transcriptome assembly may not be a tough challenge as it currently is. With the emergence of such transcriptome assembly methods, it will be useful to have an efficient structural variation discovery method that can detect structural alterations in full transcript sequences within a spliced alignment framework.

Transcriptome to genome alignment problem has drawn a lot of interest from the biology community in the last decade. It has been of interest to find the spliced intron/exon structure of a transcript, given the contiguous transcript sequences themselves. Many well-known methods have been proposed for this transcriptome to genome alignment problem [21][46][41][33][2]. However, most of these methods assume the wild-type property of the transcript sequence, devoid of any structural modifications such as non-co-linear transcription, inversions within the transcript sequence, or very long or inter-chromosomal gaps between two exons[3].

With the next-generation transcriptional structural variation studies, there is a necessity for an efficient long transcribed sequence (long RNA-seq reads or assemblies) to genome aligner, that can address the limitations of the currently existing aligners and detect structural alterations in the form of duplications, inversions, fusions and rearrangements.

In this study we propose two novel frameworks for aligning transcripts to genome with structural changes as well as a new high-throughput transcriptome to genome aligner that can detect various structural alterations in transcripts.

Establishing a comprehensive mathematical model of structural alteration events in transcriptome sequences will lead to a better understanding of the novel transcription phenomena such as trans-splicing and non-co-linear transcription and their correlation with regulatory abnormalities of a cell. Thus, the availability of an efficient computational tool that can accurately identify such complex transcriptional events will be of great significance to personalized genome/transcriptome research, detecting complex variations that might lead to the characterization of personal biomarkers for cancer diagnosis and prognosis of tumours.

1.2 Methods Background

The problem of aligning a transcript sequence to genome (also known as spliced alignment problem), is a well investigated problem with many exact solution methods for different variants and a multitude of heuristic approaches based on homology search and fragment chaining. Theoretical foundations of this problem go as early as 80s with the investigation of convex gap cost functions[4] such as in the form $a + b * log(x)$, a,b being constant values and

[3]We should note among the referred methods, GMAP [46] has a limited support for chimeric transcript detection.

[4]In the cited article, convex gap cost function model is referred to as "concave" gap penalty model.

x representing the distance of the gap [45]. This more advanced gap penalty function can model RNA splicing and intron gaps in the alignment in a more realistic fashion than linear gap penalties. Until the late 80s, the best known solution method for this problem was the naive dynamic programming solution within $O(MN^2 + NM^2)$, M, N being the length of the genomic and transcriptomic sequences.

In the late 80s and early 90s, multiple results have been published upon speeding up the nucleotide-level alignment problem[5] for special gap penalty functions that satisfy certain convexity, concavity properties[32][11][23] and for functions that we can calculate their zeros in a simple way[11].

Also in the early 90s, BLAST[1] introduced a fast DNA to whole genome alignment heuristic method with reduced sensitivity as an alternative to the slower exact alignment solutions[36][42]. This method, instead of directly aligning matching nucleotides, utilized a seed-and-extend alignment strategy, which is based on searching common seeds of a certain length in both sequences and extending them in order to obtain the full alignment. In the same direction, there emerged faster methods for spliced alignment of a transcribed sequence to whole genome through a combination of homology search and fragment chaining methods.

From late 90s to mid-2000s, an ample amount of transcript to whole genome alignment tools emerged such as Gapped BLAST[2], BLAT[21], GMAP[46] and Exonerate[41]. Since solving this problem is computationally harder than aligning a DNA sequence to the genome, the proposed methods do not only utilize a seed-and-extend alignment strategy, but also combine it with a fragment chaining method that aims to stitch the fragments that yields a high-scoring or optimal fragment chain.

Even though currently available methods can align entire transcriptome datasets to whole genome within the order of days with a single processor, their support for detection of complex structural variation events is very limited. The primary reason behind this is the lack of knowledge about complex structural alteration events (novelties) in transcripts during the times these methods were developed, as biological studies on these are only recently becoming available to the scientific community.

In the realm of next-generation structural variation studies, these tools need to be modified towards the identification of complex structural variants. In this context, one needs to

[5]These solutions did not consider the direct application of the spliced alignment problem, but were mainly designed for genome-to-genome or protein-to-protein alignment purposes with a more realistic gap penalty model than affine gap penalties.

find the most likely alignment between a given transcript sequence and a genomic sequence under the possibility of the following genomic events: (1) Duplication events, in which two separate regions in the transcript sequence align to a shared region within the genomic sequence. (2) Inversions, in which an interval within the transcript sequence aligns to a different strand of the genome than the rest of the transcript. (3) Rearrangements that appear as non-linear ordering of the transcript alignment and (4) fusions transcripts, in which the transcript sequence is split into two separate genomic intervals that can be far apart on the same chomosome or on entirely different choromosomes sequences.

A practical approach to the problem of identifying novel transcriptional events was proposed by [37]; in which study, they parsed the alignment results of a regular spliced alignment method and mined the structural variation information such as folded inversions[6] and fusions hidden in the combination of alignments. Another method proposed in a similar direction, is to extract duplication and in-place inversion information that is indirectly represented in wild-type spliced alignment formations[44]. However, these approaches can provide only a limited level of analysis for detecting events involving complex structural variations, since they are highly dependent on the regular spliced alignment tools that were not originally designed to tackle these complex structural alterations. Frequently, transcripts containing structural alterations will appear in deformed formations when aligned with regular spliced alignment methods due to their over-sensitivity for the wild-type transcript structure, disregarding potential variations. Therefore, as an alternative initial alignment step for such studies; it will be useful to design a novel algorithm that can detect complex structural alterations during the alignment, rather than inferring them through a post-processing step.

There are several issues that need to be addressed for the purpose of designing and developing a spliced alignment method with the capability of detecting structural alterations in a personal transcriptome. First, the alignment framework (and the alignments it would provide) should be generalized to be able to handle complex alignment cases such as the alignment of different regions in the transcript to the same region in the genomic sequence (because of a duplication event during transcription), and the alignment of consecutive regions in the transcript to the genome with different directionality (due to an inversion).

[6]Inversions that cover a suffix or prefix of the full transcript. Note that these can be produced by only a single irregular transition between alignment bases.

Furthermore, the algorithm should allow backward junctions in the genome due to a rearrangement of two consecutive intervals (i.e., order of two consecutive blocks is switched when aligned to the genome while conserving their directionality). In addition (for the purpose of detecting a fusion event due to trans-splicing or genomic rearrangements), the algorithm should allow the alignment of different parts of the transcript sequence to two locations which are far apart in the genome (or even in two separate chromosome sequences).

To our best knowledge, apart from this study there is no single comprehensive long transcript[7] alignment method that can address all of the concerns noted above. A similar but a simpler genome alignment version of this problem was addressed in [7], in which a genomic sequence is aligned to another genomic sequence with the structural alterations (in the form of duplications, inversions and rearrangements) in a model of "1-monotonous" alignment that considers only one sequence indices to be increasing and the other sequence to be supplying copies of substrings with indices that are not necessarily increasing. The biggest drawback of this method, as also noted in the original paper, is that the alignment would inherently be asymmetric due to the detection of duplications in only one of the sequences rather than the other. However, this feature is not necessarily a drawback for transcript to genome alignment purposes, since the aim is to find the structural alterations in the transcript with respect to the genomic sequence and not vice versa. Therefore, finding the duplications of a string in the genomic sequence within the transcript would be of interest, yet reporting two alternative placements of a single transcript exon onto the genomic sequences less interesting in terms of finding transcriptional structural variants. However, the problem at hand is more complex than the genomic counterpart, due to the fact that transcriptomic gap penalties need to be modelled as more complex functions than the genome. Even though an affine gap penalty model can be satisfactory for the purposes of genome to genome alignment, more complicated gap penalty functions are necessary when designing a realistic transcriptome to genome alignment framework.

1.3 Contribution of Current Study

The first set of deliverables of this thesis project are two novel frameworks for transcriptome-to-genome alignment with structural alterations and the corresponding set of solutions for

[7]Number of splices in the transcript are not limited by a constant number.

different gap penalty models within each formulation.

Our first framework is a nucleotide-level alignment model that can detect duplications, rearrangements, inversions and fusions in the transcript as long as the sequence is represented as chain of unidirectional copies of segments taken from the genome, meaning that structural events are only detected on the transcript side and the duplications on the genome side are missed or ignored. For our formulations within this framework, we provide several dynamic programming solutions with different run-time complexities, depending on how complex the gap penalty of introns and other genomic gaps are modeled. The main theoretical result in this framework is the sparsification of the alignment matrix construction step using convex gap penalty model.

Our second framework aims to resolve the high run-time and memory cost of the initial the nucleotide-level alignment framework, by sacrificing sensitivity upon structures shorter than a minimum threshold. In this formulation we assume each alignment unit to be a fragment that represents a short homologous sequence between the transcript and the genome (that only contains mismatches), and try to find the fragment chaining that gives the best overall alignment score concordant with penalties given in the first framework. For the formulations within this framework; we do not present a sparse solution, yet examine the problem from different overlap resolution perspectives and propose alternative approaches.

The study also presents a novel transcriptome to whole genome alignment tool named **Dissect** (**DIS**covery of **S**tructural **E**vent **C**ontaining **T**ranscripts), suitable for high throughput transcriptome structural variation studies,. As far as we know, this is the first comprehensive stand-alone software pipeline for the purposes of detecting and characterizing novel transcriptional alterations, capable of direct global alignment of long transcriptomic sequences (such as long RNA-seq reads, ESTs and short-read assembly contigs[8]) that contain structural alterations to whole genome using realistic gap penalty models. This software pipeline is suitable for high-throughput analysis of large-scale transcriptome studies, allowing the processing of entire transcriptomes in the order of days with a modern single core processor.

Finally, we report experimental results using our alignment tool, upon a simulated mouse transcriptome database containing nucleotide-level noise and assembled RNA-seq reads of a human prostate cancer individual. Discovered structural alterations within the prostate

[8]Contiguous sequences

cancer dataset are presented, ready to be validated for analysis of their biological significance.

1.4 Organization of Thesis

This thesis document is structured as follows:

Methods

The methods chapter is composed of three separate sections.

The first two chapters present the two theoretical frameworks mentioned earlier. For both nucleotide-level alignment and fragment chaining approach, the problems are investigated in multiple variants and efficient solutions are proposed to tackle them.

In the third section of the methods chapter, we present our software aligner, Dissect. We present the multiple steps of the alignment and event detection process, but focusing on more practical problems than the theoretical concepts discussed in the earlier sections. Two of the practical problems that are investigated in detail in this section are: inferring putative gene regions that the transcripts are likely to be located and post-refinement of potential small-scale misalignments in the fragment chain due to the minimum fragment length threshold. In this section, we also talk about the software development aspects of Dissect, discussing the design considerations to make it more efficient in terms of memory and disk usage and improving the usability of the software.

Experiments

The experiments chapter reports experimental resuts in two main directions: experiments upon simulated datasets and real life biological datasets.

The simulation experiments fall into two main subcategories: (1) Showing evidence for the alignment accuracy and event detection specificity through tests upon wild-type transcriptome datasets comtaminated by with nucleotide-level noise and novel insertions. (2) Showing evidence for the event detection sensitivity through tests upon simulated structural events on wild-type transcripts.

In the second set of experiments, we use assembled RNA-seq contigs of a human prostate cancer individual and report the number of events we detect for each sub-type. Unfortunately there is not a reliable way to ascertain false negative rates in real life datasets and biologically relevant true positive rates, due to the fact that these sequences are generated by a separate assembly pipeline which will have its own rate of false assemblies that contain structural events. However, the purpose of the presented software is finding structural alterations assuming that the given sequence is correct and has a high rate of similarity to its genomic origin. For this reason, in order to evaluate the reliability of the software in real-life datasets, we use a wild-type spliced aligner as a surrogate for detecting false positive events.

Summary and Conclusion

This chapter contains an overall summary and a general discussion upon the reported theoretical results, design aspects of the presented alignment tool and obtained experimental results as well as the possible directions in which this study can be improved or extended in the future.

Chapter 2

Methods

2.1 A Theoretical Framework for Nucleotide-Level Spliced Alignment with Structural Alterations

Different from the alignment of two genomic sequences, aligning a transcribed sequence to a genomic sequence have several challenges even without the incorporation of structural alterations. These challenges mostly arise from the asymmetric gap models for the genome and transcript sides of the alignment and the nontrivial genomic gap penalty functions that are used to model long introns on the genome side. Furthermore, with the incorporation of splice signal scores to the alignment formulation, the penalty function of genomic gaps become position specific, thus traversing the constructed alignment graph becomes a challenge by itself.

The main concern when aligning two genomic sequences is whether to align them globally, forcing both sequences to be fully represented within the alignment or locally, giving both sequences the option to omit leading or trailing sequences that reduce the alignment score if they are included. In both cases the alignment would be symmetric (i.e. the same alignment is obtained when the order of the aligned sequences are changed within the pairwise alignment graph)

When we consider the alignment of a transcript to a genomic sequence, the two sequences do not have symmetric alignment properties (as opposed to genome-to-genome alignment). The aligned transcribed sequences would be contained within only a very small region of the full genomic sequence given, however the extent of this region would be larger than transcript's length due to the length difference between introns and exons. This property of aligned sequences creates an asymmetry in the transcript alignment problem, in which the transcript sequence should be considered fully within the alignment whereas its genomic counterpart should be very local and contain relatively long gaps within.

Moreover, since the discarded introns in a transcript sequence tend to be much longer than the retained exon sequences, constant or affine gap penalties that could act as plausible models of the gap length in the case of aligning two genomic sequences, would be inadequate to model the long gaps on the genome side in the case of transcriptome to genome alignments[1]. For this reason; in the following set of formulations, we assume genomic gap penalty models in the form of convex and log-scale cost functions.

For the sake of simplicity, the gap penalties on the transcript side are formulated within the constant gap penalty model, even though the algorithms described in this section are fully compatible with affine gap penalties with simple modifications. At the end of each subsection, how to modify the presented algorithms for affine gap penalties is briefly described.

The following two subsections provide two different problem formulations and algorithms to solve these problems exactly. The first problem definition presents a realistic model of the alignment of a wild-type transcript to the genome without involving any structural changes. Whereas the second problem definition expands the same model by involving structural alterations in the alignments; such as the alignment of two separate transcribed bases to a single genomic base (i.e. duplication), alignment of a base that comes later in the transcribed sequence to be aligned to a genomic base that comes earlier than the alignment of a previous base in the transcript (i.e. rearrangement), alignment of a substring of bases in the transcript to the reverse complement of the genomic sequence (i.e inversion) and the placement of a single special genomic gap that can span a much larger distance than usual or form a junction between two separate chromosome sequences (i.e. fusion). We further expand this model by incorporating additional scores (penalty reductions) for gaps that span an intron junction between genomic positions that represent a canonical splice signal pair (e.g. GT-AG). Even though this addition to the formulation does not represent a structural change; due to the additional steps required for integration, siplice signals are

[1]If we model an intron gap with a linear cost function $f(L) = c_0 + c_1 \times L$ (L representing the gap size and c_0, c_1 arbitrary constant values), gap penalty of long introns are likely to dominate the matching base pair scores obtained from other regions in the transcribed sequence.

investigated as part of the second problem formulation.

Provided algorithms for the first problem describe already existing methods in the literature. Whereas in the second subsection we describe our expanded problem definition and corresponding novel solution methods that extend the original methods to structural alterations staying within the same run-time complexity bound.

2.1.1 Problem 1: Wild-type transcript to genome alignment

Given an alphabet of $\Sigma = \{A, C, G, T\}$, let $T = t_1 t_2 \dots t_N$ be the transcribed sequence of length N and $G = g_1 g_2 \dots g_M$ be the genomic sequence of length M over Σ. The value function $v(t_i)$ returns the base character for the given position i in T, whereas $v(g_j)$ returns the base character for the given position j in G.

A wild-type transcript to genome alignment of sequence T to G, $A(T, G)$, is a mapping f from the set indices of T, $\{1, 2, \dots, N\}$ to the indices of G together with a special gap index, $\{1, 2, \dots, M\} \cup \{\phi\}$, ϕ representing a gap character '_' on the genome side, such that

$$\forall i, j \in [1, N] \ , \ i < j \ \wedge \ f(i) \neq \phi \ \wedge \ f(j) \neq \phi \ \Rightarrow \ f(i) < f(j) \tag{2.1}$$

holds[2]. This statement ensures that each base on the transcribed sequence should be either aligned to a gap, or should be aligned to a genomic position that comes later than all of the aligned positions of the previous bases in the transcript.

The score of the alignment $Score(A)$ is defined as follows:

$$Score(A) = \sum_{i=1}^{N} S_m(i, f(i)) - \sum_{\substack{1 \leq i < j \leq N \\ f(i) \neq \phi \wedge f(j) \neq \phi \\ \forall k, i < k < j, f(k) = \phi}} P_s(f(i), f(j))$$

where S_m represents the base match/mismatch score/penalty function and P_s representing the gap penalty function, defined as follows:

$$S_m(i, j) = \begin{cases} C_{match} & \text{for} \quad v(t_i) = v(g_j) \\ C_{mismatch} & \text{for} \quad j \neq \phi \wedge v(t_i) \neq v(g_j) \\ C_{gap} & \text{for} \quad j = \phi \end{cases}$$

[2]The conventional problem definition for alignment would involve a two-row matrix with gaps on either side. However, the given problem definition and the following score functions correspond to the exact same formulation with a two row matrix. Such a problem definition is provided for the sake of simplicity of the transition from wild-type transcript alignment to transcript alignment with structural alterations.

$$P_s(a,b) = h(b-a-1)$$

In the definition of the penalty score function[3], **h**: $\mathbb{Z}^+ \cup \{0\} \rightarrow \mathbb{R}$ is a convex gap cost function satisfying the following property:

$$h(x) - h(x-1) \geq h(x+1) - h(x) \geq 0 \tag{2.2}$$

Below, two different solution methods are given for solving the problem formulated above (**W**ild-**T**ype **T**ranstriptome **A**lignment). The first algorithm is a simple solution method that does not utilize the convex gap penalty function property, but is suitable for arbitrary gap penalties. Whereas the second solution provides a more complicated algorithm with a tighter run-time complexity that utilizes the convex gap penalty property.

Algorithm WTTA-1: Naive solution method

The general approach of this naive solution is to iteratively fill in the cells of the alignment matrix X row by row; each of the matrix cell $X[i,j]$ representing the best wild-type transcript to genome alignment of T's prefix $T[1..i]$ and G's prefix $G[1..j]$.

For each cell of the matix, the algorithm computes the cost all valid genomic gaps from the previous row to the cell or alternatively maps a particular cell to the gap character index ϕ.

The initialization step of the WTTA-1 algorithm is as follows:

$$\forall j \in [1,M],\ X[1,j].score \leftarrow max(S_m(1,j), S_m(1,\phi))$$
$$\forall j \in [1,M],\ X[1,j].parent \leftarrow \text{source}$$

where the source parent indicates that the alignment cell is the starting point for the alignment.

And for each row i, from 2 to N, and within each row; for columns j, from 1 to M, the following assignment is done:

$$X[i,j].score \leftarrow \max(X[i-1,j] + S_m(i,\phi), \max_{1 \leq k < j}(X[i-1,k] - P_s(k,j) + S_m(i,j)))$$

In the assignment above, the first parameter of the outer maximum function considers aligning the current base of the transcript (t_i) to a gap, whereas the inner maximum function

[3]A desirable property of the P_s function would be returning the value zero, if $b-a-1=0$

evaluates the gap transitions from all previous genomic positions (g_k). In the special case in which $j = k + 1$, the genomic transition does not represent a gap, but a consecutive alignment of two bases in the transcript to two consecutive bases in the genome. In this particular case, the function h would ideally return zero for $h(j - k - 1)$, therefore the value of the new cell would in this case be $X[i-1, j-1] + S_m(i, j)$, similar to pairwise genome alignment.

If the best score of the cell is determined by first parameter of the outer maximum function, $X[i, j].parent$ is assigned as j; alternatively if the score value is determined by one of the k values in the second parameter, the parent is assigned as k. Also note that; if the genomic gap to $X[i, j]$ is from a cell $X[i-1, k]$ that t_i is aligned to ϕ, the represented genomic gap is not from $X[i-1, k]$ but from the first ancestor of that cell that is not aligned to a ϕ or is *source*.

After the score and parent information of all cells in the matrix are filled, alignment step of the algorithm terminates by selecting the highest scoring base in the last row of the alignment matrix, which corresponds to the last base of the transcribed sequence. Therefore, the algorithm selects the best alignment of the entire transcript to a local region in the genomic sequence.

The backtracking step of the algorithm simply starts from the highest scoring cell in the last row and follows the direction of parents until it reaches the first row. The optimal alignment function f is constructed by the following scheme: whenever $X[i, j].parent = j$, $f(i) \leftarrow \phi$, otherwise if $X[i, j].parent = k \neq j$, $f(i) \leftarrow k$. When the top row is reached, $f(1)$ is assigned as the current column index (or a gap) and the backtracking step terminates.

The run-time complexity of the algorithm WTTA-1 is $O(NM^2)$ due to visiting every cell of the previous row for all cells in the matrix X, apart from the first row. Memory complexity of WTTA-1 is the size of alignment matrix X, thus $O(NM)$.

Algorithm WTTA-2: Sparse solution method

In order to reduce the time complexity of WTTA-1 from $O(NM^2)$ to $O(NMlog(M))$, we can utilize the property of the gap penalty function h being a convex cost function that has non-increasing growth with the length of the gap as defined in (2.2). The result described below is originally demonstrated by [32] and [11].

The first row of the matrix X is initialized in the same manner as in the algorithm WTTA-1, however generating $(i+1)^{th}$ row using the values from the i^{th} row can be performed in $O(Mlog(M))$ operations, using the property given in (2.2). The first step in obtaining this improvement would be to generalize the given inequality.

Lemma 1. *Given four integers $k \leq l \leq q \leq r$ and $\boldsymbol{h} : \mathbb{Z}^+ \cup \{0\} \rightarrow \mathbb{R}$; if h is a convex cost function with the property defined in* (2.2)*, then $h(r-l) - h(q-l) \geq h(r-k) - h(q-k)$.*

Proof. From (2.2), for $t \geq 0$ we have;

$$h(i) - h(i-1) \geq h(i+1) - h(i) \geq \ldots \geq h(i+t) - h(i+t-1) \tag{2.3}$$

And similarly, for $1 \leq m \leq i$;

$$\sum_{k=i-m+1}^{i} h(k) - h(k-1) \geq \sum_{k=i-m+2}^{i+1} h(k) - h(k-1)$$

$$\implies h(i) - h(i-m) \geq h(i+1) - h(i-m+1) \tag{2.4}$$

From (2.3) and (2.4)

$$h(i) - h(i-m) \geq h(i+t) - h(i+t-m)$$

Here if we rewrite $m = r - q$, $i = r - l$, and $t = l - k$

$$h(r-l) - h(q-l) \geq h(r-k) - h(q-k)$$

□

At this point, in order to make use of **Lemma 1**, we will modify the general dynamic programming scheme in WTTA-1. In WTTA-1, the algorithm searches for transitions from the previous row towards the current cell. However, in WTTA-2, each cell will perform a search for which cells of the following row it can affect by a transition. We will call this a forward assignment step as described follows:

Assume that, the initialization steps of WTTA-2 are the same as in WTTA-1 with the only difference that all cells other than the first row are initialized as having a score of $-\infty$. Then, for each row i, from 1 to $N-1$; and within each row, for each column j, from 1 to M.

$$X[i+1,j].score \leftarrow \max(X[i+1,j].score, X[i,j].score + S_m(i+1,\phi))$$

$$\forall k \in [j+1, M], X[i+1,k].score \leftarrow max(X[i+1,k].score, X[i,j].score - P_s(j,k) + S_m(i+1,k))$$

If the assignment in the first line above updates the score of the cell, $X[i+1, j].parent$ is set as j. Similarly for any k, that $X[i+1, k].score$ is altered during the assignments in the second line, $X[i+1, k].parent$ is set as j.

These operations during a full scan of the alignment matrix will produce the exact same results as in WTTA-1. However, these values are not filled in by each cell looking at previous values in the matrix; but by the forward assignment operations done by the previous matrix cells. However, this traversal also yields a run-time complexity of $O(NM^2)$. In order to perform these forward assignments in $O(NMlog(M))$ time we will make use of the following property of the assignment process.

Theorem 1. *During the forward assignment process described above, at any row i and column j, right before the forward assignments begin for that particular cell, the sequence formed by the current parents of the cells to be visited,*

$$I_p = (\ X[i+1, j+1].parent,\ X[i+1, j+2].parent,\ \ldots,\ X[i+1, M].parent\ )$$

is a non-increasing sequence[4].

Proof. If j' and k' are column indices such that, $1 \leq j' < j$ and $k < k' \leq M$; we can substitute j',j,k,k' in place of k,l,q,r in **Lemma 1**, obtaining the following inequality:

$$\begin{aligned} & h(k'-j) - h(k-j) \geq h(k'-j') - h(k-j') \\ \implies\ & P_s(j, k') - P_s(j, k) \geq P_s(j', k') - P_s(j', k) \\ \implies\ & P_s(j', k) - P_s(j, k) \geq P_s(j', k') - P_s(j, k') \end{aligned} \tag{2.5}$$

And from (2.5),

$$P_s(j, k) \geq P_s(j', k) \implies 0 \geq P_s(j', k') - P_s(j, k') \implies P_s(j, k') \geq P_s(j', k')$$

$$-P_s(j, k) \leq -P_s(j', k) \implies -P_s(j, k') \leq -P_s(j', k')$$

Thus,

$$\begin{aligned} & X[i, j].score - P_s(j, k) < X[i, j'].score - P_s(j', k) \\ \implies\ & X[i, j].score - P_s(j, k') < X[i, j'].score - P_s(j', k') \end{aligned} \tag{2.6}$$

[4] *Remember that $X[i, j].parent$ points toward a column k of a matrix cell in the previous row, $X[i-1, k]$.*

In order to prove **Theorem 1** by contradiction, we assume that I_p is not monotonously non-increasing:

$$\exists u, v \in [j+1, M] \text{ such that } u < v \land p_1 = X[i+1, u].parent < p_2 = X[i+1, v].parent$$

The statement $p_1 = X[i+1, u].parent$, would require the following statement to hold:

$$\forall t, 0 < t < j : X[i, p_1].s - P_s(p_1, u) \geq X[i, t].s - P_s(t, u) \tag{2.7}$$

Similarly for $p_2 = X[i+1, v].parent$:

$$\forall t, 0 < t < j : X[i, p_2].s - P_s(p_2, v) \geq X[i, t].s - P_s(t, u) \tag{2.8}$$

From (2.7) and (2.8), we obtain:

$$X[i, p_1].score - P_s(p_1, u) \geq X[i, p_2].score - P_s(p_2, u)$$

and

$$X[i, p_2].score - P_s(p_2, v) \geq X[i, p_1].score - P_s(p_1, v)$$

which contradict with the statement in (2.6). Hence, I_p needs to be monotonously non-increasing. □

Using the property in **Theorem 1**, we can skip the score comparisons for the majority of the row cells during the forward assignment process. An algorithm that utilizes this property during the forward assignments is described as follows.

- Assume that initialization step of the first row takes place in the same way as WTTA-1
- For each of the rows i, from 1 to $N-1$:
  - Initialize parents of all cells in the $(i+1)^{th}$ row as $\emptyset$ and define $X[i, \emptyset].score$ as $-\infty$[5].
  - Initialize a linked list structure B, that contains nodes B_1 to B_m that represent the block of columns in the $(i+1)^{th}$ row that has the same parent pointer[6]. It is sufficient for each node B_c to only hold the ending column ($B_c.ec$) of each block, and the parent index value ($B_c.pi$) each cell in the block B_c has[7].

[5]Thus, the score of having parent $\emptyset$ is $-\infty$ for any cell in the $(i+1)^{th}$ row

[6]The value m is determined by how many same parent blocks exists in the list. Thus initially $m = 1$

[7]Initially B will contain only a single node B_1, having $B_1.ec = M$ and $B_1.pi = \emptyset$

- Assign current block $B_c \leftarrow B_1$
- For each of the columns j, from $i+1$ to M
  - If the score transition from $X[i, B_c.pi]$ to $X[i+1, j-1]$ is larger than $X[i+1, j-1].score$, then update $X[i+1, j-1].score$ and assign $X[i+1, j-1].parent \leftarrow B_c.pi$
  - Check the score of aligning s_i to ϕ:

    $$X[i+1, j].score \leftarrow \max(X[i+1, j].score, X[i, j].score + S_m(i+1, \phi))$$

    If this assignment updates $X[i+1, j].score$, then $X[i+1, j].parent \leftarrow j$. At this point, we define another information type for the aligment matrix as *gap*, and assign $X[i+1, j].gap$ as *true*. We assume all *gap* values in the alignment matrix are initialized as $false$.
  - Update the current block if it is completed: If $B_c.ec \leq j$, then $c \leftarrow c+1$
  - Check if a transition from $X[i, j]$ can update $X[i+1, j+1]$'s score:

    $$X[i, j].score - P_s(j, j+1) > X[i, B_c.pi].score - P_s(B_c.pi, j+1)$$

    If it does not, from the observation (2.6), it is clear that $X[i, j]$ cannot update any one of the remaining cells. Thus, skip the following steps and continue this process with the next column $j+1$.
  - Check if a transition from $X[i, j]$ can update $X[i+1, k]$'s score, in which k is the ending column of the current block. In mathematical terms, check if:

    $$X[i, j].score - P_s(j, B_c.ec) > X[i, B_c.pi].score - P_s(B_c.pi, B_c.ec)$$

    If the inequality above holds[8], then assign $B_c.pi \leftarrow j$ then check if the same inequality holds for the next block's ending position, $B_{c+1}.pi$. If it holds, then merge the two blocks B_c and B_{c+1} by updating $B_c.ec \leftarrow B_{c+1}.ec$ and deleting B_{c+1}. As this inequality holds for the remaining blocks in the list, merge them one by one to B_c[9]. If all of the blocks are merged into one,

[8]Note that with this inequality, ties are broken favoring the block parent indicating an earlier cell. Even though either way may not represent a biological reasoning, the way this tie-breaking scheme is selected favors a higher quality alignment prior to a longer intron over a relatively lower quality alignment with a shorter intron.

[9]Since B is maintained by a linked list, shifting the remainder of the blocks whenever two adjacent blocks are merged, can be done in constant time.

return to the beginning of the inner loop and continue the same process with the next column $j+1$ and the resulting linked list B. If the first or any other block's inequality does not hold however, the block that does not satisfy the given inequality, $B_{c'}$, needs to be analysed in detail in the next step.

  * $B_{c'}$ in the $(i+1)^{th}$ row represents the column index interval $[(B_{c'-1}.ec + 1), B_{c'}.ec]$. If the inequality given above does not hold for $B_{c'}.ec$ but holds for $B_{c'-1}.ec+1$, then transitions from $X[i,j]$ can update only some of the cells within this block. However, due to the property proved in **Theorem 1** to be satisfied, the updated cells should form a sub-interval that starts at the beginning of the block. Here, we only need to find the position that this sub-interval ends. This can be achieved by employing a binary search within the block interval, trying to find the q^{th} cell in the interval that satisfies the equation where $(q+1)^{th}$ cell does not[10]. After this position q is found, the block $B_{c'}$ is split into two separate blocks with column index intervals: $[(B_{c'-1}.ec+1), q]$ and $[(q+1), B_{c'}.ec]$. Also the respective parent indices and ending positions are set. If $B_{c'}$ was the original B_c block, then after splitting B_c into two as described, we continue with the next column $j+1$. If $B_{c'}$ is one of the downstream blocks of B_c however, then all of the blocks in between have already been merged to B_c and the first block that is created during the split is merged with B_c as well.

- At this point we have the final parent and score values for all of the cells in the $(i+1)^{th}$ row. In order to fill the remainder of the rows, we go to the beginning of the outer loop.

• The backtracking step of this algorithm is employed in a similar way to the method described in WTTA-1. During the backtracking procedure, the current transcript base is aligned to a gap only if the *gap* value of the current alignment matrix cell is *true*. Otherwise, i^{th} base of the transcript is aligned to the j^{th} base of the genomic sequence as in the WTTA-1 algorithm.

Analysing the complexity of this method is more difficult than WTTA-1 algorithm,

[10]It is important to note that, throughout the binary search or block merging/splitting steps the actual parent and score values of the cells from $X[i+1, j+1]$ to $X[i+1, M]$ are not stored. All of the score calculations required for the inequality checks are performed on the fly, with the values not saved or reused.

however it is clear that the inner loop of the algorithm is called $O(NM)$ times throughout the algorithm, thus initialization and backtracking steps of the algorithm does not affect the run-time complexity. It can also be noticed that at each iteration of the inner loop; (1) a binary search is performed within an interval of $O(M)$ cells, (2) at most $\Theta(1)$ cells are split into two, and (3) at most $O(M)$ cells are merged. Step (1) adds an additional $O(logM)$ operations for each inner loop call and step (2) does not affect the complexity since it can be performed in constant time. In order to analyse the complexity of step (3), we can make use of the observation that before the first iteration of the inner loop starts, the number of blocks is exactly 1. Since at each step at most one block is split into two, there can at most be $O(M)$ blocks created in total. This observation indicates that there are at most $O(M)$ cells merged as well, giving an amortized complexity of $\Theta(1)$ for step (3) for each iteration of the inner loop. Therefore the overall complexity of the WTTA-2 algorithm is $O(NMlog(M))$.

Further improvements to WTTA-2

In the WTTA-2 algorithm, it was assumed that in the given penalty function genome-side gaps are modelled by a convex cost function satisfying the inequality given in (2.2). However, it is possible to achieve a further run-time complexity reduction if we consider a smaller set of functions that satisfy the "closest zero property" defined in [11].

The definition of this property for a given convex cost function, $h : \mathbb{R} \rightarrow \mathbb{R}$, is as follows:

- For every $x_1, x_2 \in \mathbb{Z}$, $x_1 < x_2$ and $r \in \mathbb{R}$, the smallest y integer value such that $y > x_2$

$$\text{and } h(y - x_1) - h(y - x_2) - a \leq 0 \text{ can be calculated in constant time} \qquad (2.9)$$

Theorem 2. *If the gap penalty function $P_s(a, b) = h(b - a - 1)$ in the **Problem 1** definition is a convex cost function satisfying* (2.2)*, and also has the "closest zero property" as defined in* (2.9)*. **Problem 1** can be solved in $O(MN)$ time; M, N being the transcript and genome lengths respectively.*

Proof. In the complexity analysis of the algorithm WTTA-1, we identified three steps within the inner loop: (1) binary search within the interval of a same parent block, (2) splitting of an interval, and (3) merging of one or more intervals. Steps (2) and (3) are shown to have $\Theta(1)$, and amortized $\Theta(1)$ runtime complexities respectively, whereas step (1) takes

$O(log(M))$ time. Due to the fact that the inner loop is iterated $O(MN)$ times, the overall running time was $O(MNlog(M))$. The goal of each binary search employed was to find the q^{th} column in the block that the following two inequalities are satisfied.

$$X[i,j].score - P_s(j,q) > X[i,B_c.pi].score - P_s(B_c.pi,q)$$

$$X[i,j].score - P_s(j,q+1) \leq X[i,B_c.pi].score - P_s(B_c.pi,q+1)$$

Given (2.9), we can find such a q index value in constant time.

This improvement will reduce the binary search step performed in $O(logM)$ time to $\Theta(1)$ time, giving an overall run-time complexity of $O(MN)$. $\square$

Log-scale functions in the form $h(x) = C_1 + C_2 * log_b(x)$, satisfy the 'closest zero property', given C_1 and C_2 are constant rational numbers. The calculation steps of y in a log-scale function is described as follows:

$$\begin{aligned} h(y-x_1) - h(y-x_2) - a &= 0 \\ \implies log(y-x_1) - log(y-x_2) &= \frac{a}{C_2} \\ \implies log(\frac{y-x_1}{y-x_2}) &= \frac{a}{C_2} \\ \implies \frac{y-x_1}{y-x_2} &= b^{\frac{a}{C_2}} \end{aligned}$$

Redefine $b^{\frac{a}{C_2}}$ as C_0:

$$\begin{aligned} \implies y - x_1 &= C_0 * y - C_0 * x_2 \\ \implies y &= \frac{C_0 * x_2 - x_1}{1 - C_0} \end{aligned}$$

At this point, if the calculated y is less than or equal to x_2, we infer that there is no such y that satisfies (2.9), otherwise return the smallest integer larger than or equal to y as the solution.

Note that, the calculation $b^{\frac{a}{C_2}}$ is an operation that can be done in constant time. The remaining of the steps can also be performed in constant time, if a unit operation is assumed to cover the basic arithmetic operations[11].

Another possible addition to the WTTA-2 algorithm is the extension of the transcript gap penalty model from constant to affine cost functions. In the affine gap penalty model,

[11]Such as addition, subtraction, multiplication, division and comparison.

a gap distance of t is not penalized by $C_{gap} * t$ as in the constant gap penalty model, but penalized as a linear function in the form: $C_{gap-open} + C_{gap-extend} * t$. In order to introduce this new gap penalty model, the function S_m in the definition of **Problem 1** should be changed to S_{ma} as follows:

$$S_{ma}(i,j) = \begin{cases} C_{match} & \text{for} \quad v(t_i) = v(g_j) \\ C_{mismatch} & \text{for} \quad j \neq \phi \wedge v(t_i) \neq v(g_j) \\ C_{gap-open} & \text{for} \quad j = \phi \wedge f(i-1) \neq \phi \\ C_{gap-extend} & for \quad j = \phi \wedge f(i-1) = \phi \end{cases}$$

In order to integrate S_{ma} definition to the new algorithm, we can add two more score matrices other than X: X^M and X^G. $X^M[i,j]$ represents the alignment of the i^{th} index of T to the j^{th} index of G as a match or mismatch, whereas $X^G[i,j]$ represents the same index alignment as a gap. Moreover, $X[i,j]$ represents the maximum of the cells in these two matrices.

In each forward assignment step in WTTA-2, a genomic transition score (from (i,j) to $(i+1,k)$) that ends with a match will be calculated from the base score of $X[i,j]$, and assigned to $X^M[i+1,k]$ and replace the value of $X[i+1,k]$, if larger. On the other hand; for a gap transition (from (i,j) to $(i+1,j)$) that ends with a gap, we have two different alternatives. The gap can either extend the gap in (i,j) or start a new gap in $(i+1,j)$. In the first case the transition score should be calculated as $X^G[i,j] - C_{gap-extend}$ and in the second case $X^M[i,j] - C_{gap-open}$. The maximum of these two values will be assigned to $X^G[i+1,j]$ and replace $X[i+1,j]$ if it is larger. Since the number of operations and additional number of cells are within a constant factor of the original WTTA-2 algorithm, the extension for affine gap penalties do not alter run-time and memory complexity.

2.1.2 Problem 2: Transcript to genome alignment with structural alterations and splice signals

Here we propose an alternative definition of the first problem, generalizing it for detecting transcriptional variations such as duplications, rearrangements and inversions that are represented as unidirectional copies of genomic substrings in the transcript. We further generalize the problem formulation for handling the special case of fusions that correspond to the alignment of a single transcript to two different genomic sequences[12] with the restriction that there can only be a single transition from the first sequence to the other and no transition back from the second to the first. Finally, we incorporate the additional scoring of canonical splice signals into our formulation of pairwise transcript-to-genome alignment with structural alterations.

Assume the original definitions for the alphabet Σ, transcribed sequence $T = t_1t_2 \dots t_N$ and the genomic sequence $G = g_1g_2 \dots g_M$, given in the previous problem definition.

Inclusion of duplications and rearrangements to the problem formulation will require the freedom to have backwards gaps/junctions in the genome, allowing the cases in which $i, j \in [1, N] : \; i < j \wedge f(i) \geq f(j)$. Therefore, in this alignment formulation the statement given in (2.1) is omitted.

Moreover, inclusion of inversions to the alignment framework will require the alignment of a base within the transcribed sequence to the complement of the base in the genome rather than the original. The base complement function for any sequence index is defined as follows:

$$Comp(c) = \begin{cases} c' : v(c') = A & for \quad v(c) = T \\ c' : v(c') = T & for \quad v(c) = A \\ c' : v(c') = G & for \quad v(c) = C \\ c' : v(c') = C & for \quad v(c) = G \end{cases}$$

For easier notation of the complement alignment of a transcribed base to the genome, we introduce a new genomic sequence $G' = g'_1g'_2 \dots g'_M$, such that for each i from 1 to M, $g_i = Comp(g'_i)$. An important point to note here is that, this new sequence does not represent the reverse complement genome sequence but a sequence derived from the original genome sequence by taking the complement of each base character in its original place[13].

Furthermore, for the incorporation of fusion events, a new secondary genomic sequence and its complement should be introduced. Let $S = s_1s_2 \dots s_L$ to be the secondary sequence completely independent from the sequence G, and $S' = s'_1s'_2 \dots s'_L$ be the complement of S such that for each i from 1 to L, $s_i = Comp(s'_i)$. An important condition that the designed alignment framework should enforce on these fusion model is that, there should be at least

[12]Or very far apart genomic regions such that any cost function cannot act as a realistic gap penalty model.

[13]This notation is used for the sake of having a compatible index scheme between G and G'.

one prefix/suffix split of the transcript sequence such that all of the bases in the prefix are aligned to G, G', or ϕ, and all of the bases in the suffix are aligned to S, S', or ϕ. This property requires that the first and secondary genomic sequences are known prior to the alignment of a fusion transcript. Furthermore, this model is general enough to also include transcripts without a fusion; in which case, the sequences S and S' would be empty and the prefix/suffix split would occur after the last base of the transcript (the prefix contains the entire sequence). Or alternatively the split will occur before the first base of the transcript (the suffix contains the entire sequence).

Finally, we investigate the incorporation of splice signal scores to the alignment formulation. The direct incorporation of additional scoring for canonical splice signals appears to be violating the convex gap penalty scheme, due to the fact that the gap penalties are now affected by the splice sites. However, in the following problem solutions, we handle splice signal positions within separate genomic position subsets that do not violate the convexity property within their respective position set.

In mammalian genome the transcription splicing regulation is dominated by GT-AG splicing signals. Similarly, if the input transcript sequence is the reverse complement of an actual transcript, the majority of the splice signals throughout the transcript will have CT-AC signal. Even though, for higher accuracy, a splice signal score table can be used for different values of splice signal scores on either side; for simplicity purposes, in the following set of solutions we assume that the only type of valid splice signal is GT-AG. Thus the gaps spanning these signals in the genome are awarded with an additional score deducted from their respective genome transiton penalty. However, we also show that, with the condition that the splice signal length is constant; any scoring scheme upon multiple splice signals can be performed within the same run-time and memory complexity as in the case with the GT-AG signal scheme.

Splice signal-pair scoring scheme can be employed in two main forms: sums of partial signal scores or conjunction of signal scores. In the first scheme, the overall score for a single splice junction is represented as the summation of individual scores depending on the existence of canonical splice signals on either side. In the second scheme, the score is only considered as depending on the existence of canonical splice signals on both sides. In both schemes, however; splice signal scores are constant values independent from the gap length. For simplicity; in the current problem formulation, the first scheme will be considered, yet the extension for the second scheme is examined at the end of the problem solution subsections.

With the help of the clarifications given above, the problem of aligning a transcribed sequence to the genome (a single genomic sequence or alternatively two genomic sequences for fusion transcripts) with structural changes, is given as follows:

An alignment of a transcribed sequence T to genomic sequences G, G', S and S' with structural alterations, $A_s(T, G, S)$, is a mapping F from the bases of T, $\{t_1, t_2, \ldots, t_N\}$ to the bases of G and S and a special gap index, $\{g_1, g_2, \ldots, g_M\} \cup \{g'_1, g'_2, \ldots, g'_M\} \cup \{s_1, s_2, \ldots, s_L\} \cup \{s'_1, s'_2, \ldots, s'_L\} \cup \{\phi\}$, ϕ representing the gap character '_' on the genome side of the alignment[14], with the condition given below.

There is no such $i, j \in [1, N], q \in [1, M], r \in [1, L]$, such that:

$$(i < j) \wedge (F(t_i) = s_r \vee F(t_i) = s'_r) \wedge (F(t_j) = g_q \vee F(t_j) = g'_q) \tag{2.10}$$

The statement (2.10) ensures that the alignment of the transcribed sequence T can switch from the sequence G to S only once and after switch always stays on that side[15].

With the given alignment definition above, the score of such an alignment, $Score(A_s)$ is defined as:

$$Score(A_s) = \sum_{i=1}^{N} S_m(t_i, F(t_i)) - \sum_{\substack{1 \le i < j \le N \\ F(t_i) \neq \phi \wedge F(t_j) \neq \phi \\ \forall k, i < k < j, F(t_k) = \phi}} (P_s(F(t_i), F(t_j)) - J_s(F(t_i), F(t_j)))$$

with S_m defined as follows:

$$S_m(t_i, F(t_i)) = \begin{cases} C_{match} & \text{for} \quad v(t_i) = v(F(t_i)) \\ C_{mismatch} & \text{for} \quad F(t_i) \neq \phi \wedge v(t_i) \neq v(F(t_i)) \\ C_{gap} & \text{for} \quad F(t_i) = \phi \end{cases}$$

and P_s is defined below. Throughout P_s and J_s function definitions, the notation $F(t_i) = A \in S$ is used (implying $\exists k \in [1, length(S)] : A = S_k$), and $A.p$ stands for the position of the mapped base in its respective sequence, which is k in this case.

[14]Note that the mapping F is different from the previously defined function f, in the sense that it does not map positions between two sequences, but bases of the sequences themselves.

[15]However, in the cases that there is no fusion, one of these sequences will not be used therefore the whole alignment will take place in only one of these sequences. Yet such a formation will not be forced prior to the alignment but will emerge naturally from the best scoring alignment itself.

$$P_s(A,B) = \begin{cases} H_n(B.p - A.p - 1) & for \quad (A,B \in G \vee A,B \in S) \wedge A.p < B.p \\ H_n(A.p - B.p - 1) & for \quad (A,B \in G' \vee A,B \in S') \wedge B.p < A.p \\ H_b(A.p - B.p + 1) & for \quad (A,B \in G \vee A,B \in S) \wedge A.p \geq B.p \\ H_b(B.p - A.p + 1) & for \quad (A,B \in G' \vee A,B \in S') \wedge B.p \geq A.p \\ H_i(|A.p - B.p|) & for \quad (A \in G \wedge B \in G') \vee (A \in G' \wedge B \in G) \\ & \qquad \vee (A \in S \wedge B \in S') \vee (A \in S' \wedge B \in S) \\ C_f & for \quad (A \in G \vee A \in G') \wedge (B \in S \vee B \in S') \end{cases}$$

H_n, H_b and H_i are functions satisfying the convex cost property given in (2.2), and C_f is a constant penalty representing switching cost from the first sequence to the second as a transcriptional fusion. Even though there is a freedom in this formulation for the relative costs of H functions, in a realistic transcript to genome alignment setting, H_b function that represents backwards junctions in the genome (for detection of duplications and rearrangements) and H_i function that represents the junction between exons on different strands (for detection of inversions) are expected to be more costly than H_n function which corresponds to the regular intron spanning junction.

In the alignment score function defined above; the splice signal scoring function, J_s that returns an additional score depending on the existence of splice signals at the junction sites, is defined as follows:

$$J_s(A,B) = \begin{cases} 0 & for \quad (A,B \in G \vee A,B \in S) \wedge 1 \leq B.p - A.p < min\_int \\ 0 & for \quad (A,B \in G' \vee A,B \in S') \wedge 1 \leq A.p - B.p < min\_int \\ J_b(A) + J_e(B) & otherwise \end{cases}$$

given constant min_int value defined as the minimum possible length for an intron and J_b, J_e defined as follows:

$$J_b(A) = \begin{cases} 0 & for \quad (A \in G' \vee A \in S') \wedge A.p \leq 2 \\ 0 & for \quad (A \in G \wedge A.p > M - 2) \vee (A \in S \wedge A.p > L - 2) \\ \frac{C_{ss}}{2} & for \quad (A \in G \vee A \in S) \wedge v(A.p+1) =' G' \wedge v(A.p+2) =' T' \\ \frac{C_{ss}}{2} & for \quad (A \in G' \vee A \in S') \wedge v(A.p-2) =' A' \wedge v(A.p-1) =' C' \\ 0 & otherwise \end{cases}$$

$$J_e(B) = \begin{cases} 0 & for \quad (B \in G \vee B \in S) \wedge B.p \leq 2 \\ 0 & for \quad (B \in G' \wedge B.p > M-2) \vee (B \in S' \wedge B.p > L-2) \\ \frac{C_{ss}}{2} & for \quad (B \in G \vee B \in S) \wedge v(B.p+1) =' A' \wedge v(B.p+2) =' G' \\ \frac{C_{ss}}{2} & for \quad (B \in G' \vee B \in S') \wedge v(B.p+1) =' C' \wedge v(B.p+2) =' T' \\ 0 & \quad otherwise \end{cases}$$

In the following two subsections, we extend the initially described WTTA-1 and WTTA-2 algorithms to the new formulation of transcript to genome alignment with structural alterations (**T**ranscriptome to **G**enome **A**lignment with **S**tructural **A**lterations), staying within their original run-time and memory complexity.

Algorithm TGASA-1: Extending WTTA-1

The extension from the algorithm WTTA-1 to TGASA-1 within the complexity bound of $O(NM^2)$ is fairly straight forward if several key points are handled carefully: These are (1) four interdependent alignment matrices need to be defined with different properties (for fusions and inversions), (2) a cell in a row can update the values of the cells in the following row with both forwards and backwards genomic transitions, and (3) each gap will have a score based on its beginning and ending positions in the genome.

In the initialization step, we first identify the positions in G, G', S, S' that correspond to the beginning or ending of the GT-AG splice sites in accordance with the $J_s(A,B)$ function definition given above. This task only requires a linear scan in all four sequences, corresponding to $O(M+L)$ comparisons. In this and the following complexity analyses, we will assume that without loss of generality, $M \geq L$. Thus performing this initialization step will take $O(M)$ comparisons. The positions for canonical splice sites are stored in a constant time access table, so that whenever J_s score function is called, we can assume the result to be obtained instantly.

Furthermore, we define the four alignment matrices X_G, $X_{G'}$, X_S, $X_{S'}$, that correspond to aligning the transcribed sequence T with the four genome sequences G, G', S, and S'.

The initialization step of X_G is the same as in the WTTA-1 algorithm; however, the initialization step of $X_{G'}$ will be performed with the complements of the bases taken from T.

$$\forall j \in [1, M],\ X_G[1,j].score \leftarrow max(S_m(t_1, g_j), S_m(t_1, \phi))$$

$$\forall j \in [1, M],\ X_G[1, j].parent \leftarrow \text{source}$$

and for the complement sequence G'

$$\forall j \in [1, M],\ X_{G'}[1, j].score \leftarrow max(S_m(Comp(t_1), g'_j), S_m(Comp(t_1), \phi)$$

$$\forall j \in [1, M],\ X_{G'}[1, j].parent \leftarrow \text{source}$$

And similarly the initialization steps for X_S and $X_{S'}$ are:

$$\forall j \in [1, M],\ X_S[1, j].score \leftarrow max(S_m(t_1, S_j), S_m(t_1, \phi))$$

$$\forall j \in [1, M],\ X_{S'}[1, j].score \leftarrow max(S_m(Comp(t_1), S'_j), S_m(Comp(t_1), \phi))$$

$$\forall j \in [1, M],\ X_S[1, j].parent \leftarrow \text{source} \ \wedge \ X_{S'}[1, j].parent \leftarrow \text{source}$$

After the initialization, for each row i from 2 to N, the following steps are employed for the four alignment matrices:

- For each column j from, 1 to M
    - $X_G[i, j].score \leftarrow (X_G[i-1, j] + S_m(t_i, \phi))$ and $X_G[i, j].parent \leftarrow (X_G, j)$
    - Update[16] $X_G[i, j].score$ by $\max_{1 \le k \le M}(X_G[i-1, k] - P_s(g_k, g_j) + S_m(t_i, g_j) + J_s(g_k, g_j))$
        - If $X_G[i, j].score$ value is modified, $X_G[i, j].parent \leftarrow (X_G, k)$
    - Update $X_G[i, j].score$ by $\max_{1 \le k \le M}(X_G[i-1, k] - P_s(g'_k, g_j) + S_m(t_i, g_j) + J_s(g'_k, g_j))$
        - If $X_G[i, j].score$ value is modified, $X_G[i, j].parent \leftarrow (X_{G'}, k)$
    - $X_{G'}[i, j].score \leftarrow (X_{G'}[i-1, j] + S_m(t_i, \phi))$ and $X_{G'}[i, j].parent \leftarrow (X_{G'}, j)$
    - Update $X_{G'}[i, j].score$ by $\max_{1 \le k \le M}(X_{G'}[i-1, k] - P_s(g'_k, g'_j) + S_m(t_i, g'_j) + J_s(g'_k, g'_j))$
        - If $X_{G'}[i, j].score$ value is modified, $X_{G'}[i, j].parent \leftarrow (X_{G'}, k)$
    - Update $X_{G'}[i, j].score$ by $\max_{1 \le k \le M}(X_{G'}[i-1, k] - P_s(g_k, g'_j) + S_m(t_i, g'_j) + J_s(g_k, g'_j))$
        - If $X_{G'}[i, j].score$ is updated, $X_{G'}[i, j].parent \leftarrow (X_G, k)$
- For each column j from, 1 to L
    - $X_S[i, j].score \leftarrow (X_S[i-1, j] + S_m(t_i, \phi))$ and $X_S[i, j].parent \leftarrow (X_S, j)$

[16]The operation "Update A by X" stands for $A \leftarrow max(A, X)$.

- Update $X_S[i,j].score$ by $\max_{1\leq k\leq L}(X_S[i-1,k] - P_s(s_k, s_j) + S_m(t_i, s_j) + J_s(s_k, s_j))$
  - If $X_S[i,j].score$ value is modified, $X_S[i,j].parent \leftarrow (X_S, k)$
- Update $X_S[i,j].score$ by $\max_{1\leq k\leq L}(X_S[i-1,k] - P_s(s'_k, s_j) + S_m(t_i, s_j) + J_s(s'_k, s_j))$
  - If $X_S[i,j].score$ value is modified, $X_S[i,j].parent \leftarrow (X_{S'}, k)$
- Update $X_S[i,j].score$ by $\max_{1\leq k\leq M}(X_G[i-1,k] - C_f + S_m(t_i, s_j) + J_s(g_k, s_j))$
  - If $X_S[i,j].score$ value is modified, $X_S[i,j].parent \leftarrow (X_G, k)$
- Update $X_S[i,j].score$ by $\max_{1\leq k\leq M}(X_{G'}[i-1,k] - C_f + S_m(t_i, s_j) + J_s(g'_k, s_j))$
  - If $X_S[i,j].score$ value is modified, $X_S[i,j].parent \leftarrow (X_{G'}, k)$
- $X_{S'}[i,j].score \leftarrow (X_{S'}[i-1,j] + S_m(t_i, \phi))$ and $X_{S'}[i,j].parent \leftarrow (X_{S'}, j)$
- Update $X_{S'}[i,j].score$ by $\max_{1\leq k\leq L}(X_{S'}[i-1,k] - P_s(s'_k, s'_j) + S_m(t_i, s'_j) + J_s(s'_k, s'_j))$
  - If $X_{S'}[i,j].score$ value is modified, $X_{S'}[i,j].parent \leftarrow (X_{S'}, k)$
- Update $X_{S'}[i,j].score$ by $\max_{1\leq k\leq L}(X_{S'}[i-1,k] - P_s(s_k, s'_j) + S_m(t_i, s'_j) + J_s(s_k, s'_j))$
  - If $X_{S'}[i,j].score$ value is modified, $X_{S'}[i,j].parent \leftarrow (X_S, k)$
- Update $X_{S'}[i,j].score$ by $\max_{1\leq k\leq M}(X_G[i-1,k] - C_f + S_m(t_i, s'_j) + J_s(g_k, s'_j))$
  - If $X_{S'}[i,j].score$ value is modified, $X_{S'}[i,j].parent \leftarrow (X_G, k)$
- Update $X_{S'}[i,j].score$ by $\max_{1\leq k\leq M}(X_{G'}[i-1,k] - C_f + S_m(t_i, s'_j) + J_s(g'_k, s'_j))$
  - If $X_{S'}[i,j].score$ value is modified, $X_{S'}[i,j].parent \leftarrow (X_{G'}, k)$

After all the rows are filled with their best scoring values, the best alignment ending cell is searched among the last rows of all four matrices. The backtracking step is similar to WTTA-1, with the only difference that; the parent of a cell is a pair representing the parent matrix and the column number for the previous row of in that matrix. Similar to WTTA-1, backtracking stops when the top row is reached.

Since each of the individual steps above can be performed in $O(M)$ operations corresponding to a complete scan of the previous row, repeating them for each column in each row will give $O(NM^2)$ run-time complexity similar to WTTA-1. Since total number of cells that are held in memory is less than four times than that in WTTA-1, the memory complexity is still within the $O(MN)$.

More advanced splice signal scoring schemes can also be handled within this run-time complexity with little modification to the algorithm. If we were to change the scoring scheme from summation to conjunction, the only modification that needs to be added is checking if canonical splice signals exist on both sides during each J_s function call above. If the splice signal scoring is not based on canonical signals, but a matrix of scores based on the signals at the beginning and ending sites, the main part to modify in TGASA-1 algorithm would be the original splice site scanning of the G, G', S, S' sequences. We can use the row and column indices of this splice score matrix in order to indicate which score should be used from the matrix for each J_s function call. Since we initially assumed that splice signals are of constant length, such a matrix will have a constant number of rows and columns, thus the initial splice site scan procedure will stay within $O(M)$ complexity bound, and each J_s call be will performed in constant number of operations.

Algorithm TGASA-2: Extending WTTA-2

The extension from WTTA-2 to TGASA-2 is analogous to the extension from WTTA-1 to TGASA-1 in terms of the use of four interdependent matrices and backwards/forwards transitions in the genome. In the case of TGASA-2, the special data structure and algorithm that is used to lower $O(NM^2)$ run-time complexity to $O(NMlog(M))$ in WTTA-1 for convex gap cost functions should be applied in both backwards and forwards directions and also from different matrices. However, the main difficulty is to adapt the splice signal score functions to the convex gap penalty scheme; since in its current definition the final gap penalty function is not necessarily convex and **Theorem 1** does not hold. Thus the original speed-up cannot be used upon the alignment matrix in its current state and the matrices should be initially modified in order to apply this speed-up.

For the sake of simplicity, throughout the description of TGASA-2, the summation of signal scores scheme is used as in the original definition given for the J_s function. The extension for more general splice signal scoring schemes will be discussed after the algorithm is fully described.

A key point that is utilized in the TGASA-2 algorithm is that; if we consider only the subset of columns in a row that only consists of positions that correspond to canonical splice beginning sites (or only consists of positions that do not represent a canonical splice beginning sites), assuming that J_s is defined as sums of beginning and ending signal scores, the list of transition scores from the cells in the i^{th} row to the k^{th} column in the $(i+1)^{th}$ row show a convex gap cost property. Therefore **Theorem 1** applies to each of two sub-cases even though it does not apply to the entire row.

Assume that the initialization step of TGASA-2 for all four alignment matrices to be same as TGASA-1. In the following steps we describe how to fill the remaining values of the matrices X_G and $X_{G'}$. Since the explicit description of TGASA-2 operations would involve about 33 times as many operations as in the WTTA-2 algorithm, we give a sketch of the algorithm with references to the operations in WTTA-2 and TGASA-1.

For each of the rows i, from 1 to $N-1$:

- Initialize parents of all cells in the $(i+1)^{th}$ row of X_G and $X_{G'}$ as $\emptyset$ and define $X[i,\emptyset].score$ as $-\infty$.

- For each of the positions in the i^{th} row of X_G that represent canonical splice beginning sites, form an artificial row X_G^+ that consists of only such positions (by masking the scores of remaining positions as $-\infty$) and form X_G^- that only represent the remaining positions that do not have a canonical splice beginning site. Similarly create $X_{G'}^+$ and $X_{G'}^-$ for the i^{th} row of $X_{G'}$, considering the reverse complement splice beginning sites. We often denote these sets as $(+)$ or $(-)$ in the solution described below.

- Initialize sixteen linked list structures B^1 to B^{16} for X_G and $X_{G'}$ together, similar to the definition B in WTTA-2; each node representing the block of columns in the $(i+1)^{th}$ row that have the same parent pointer. The reason for handling sixteen separate linked lists is because each list differs by, (1) whether the current row belongs to X_G or $X_{G'}$, (2) whether the following row belongs to X_G or $X_{G'}$, (3) whether the junctions between the two adjacent rows are towards the left-side (backward for X_G and forward for $X_{G'}$) or towards the right-side (forward for X_G and backward for $X_{G'}$) of the alignment matrix, (4) whether the junctions are from $(+)$ or $(-)$ subsets of the columns in the i^{th} row.

- B_1 and B_2: In the case of the forward transitions (towards the right side) from X_G to X_G with the set of columns that have canonical splice signals $(+)$, the score values and parents in the $(i+1)^{th}$ row are calculated exactly as the WTTA-2 case, with the only difference that the left-to-right scan of i^{th} row for forward looking assignments only stop at the columns that are within the $(+)$ set. Instead of filling in the actual values of X_G, we store the resulting $(i+1)^{th}$ row in an artificial row $R_1[1..M]$. Similarly for the $(-)$ subset, we only stop at cells that do not represent a canonical splice beginning site and store the calculated $(i+1)^{th}$ row in the artificially constructed row $R_2[1..M]$.

- B_3 and B_4: The case for the forward transitions (towards the right side) from X_G to $X_{G'}$ involve the same calculations performed for B_1 and B_2, but making transitions from i^{th} row of X_G to the $(i+1)^{th}$ row of $X_{G'}$. The score calculation of this transition can also be calculated in constant time as described in H_i penalty scheme in P_s function definition. Resulting rows are stored in rows $R_3[1..M]$ and $R_4[1..M]$

- B_5, B_6, B_7 and B_8: The case of forward transitions (towards the left side) from $X_{G'}$ to $X_{G'}$ with $(+)/(-)$ subsets (B_5 and B_6), involve the same calculations used in B_1 and B_2 cases, with the only difference that the scan of the i^{th} row is done from right-to-left and all block operations in the linked list is performed in the reverse direction (as if each column index k is replaced with $M - k + 1$)[17]. The results from $(+)$ and $(-)$ sets are stored in artificially constructed rows $R_5[1..M]$ and $R_6[1..M]$ respectively. Similarly forward transitions (towards the left side) from $X_{G'}$ to X_G for $(+)/(-)$ subsets are calculated and stored in the same way in $R_7[1..M]$ and $R_8[1..M]$, with the difference of using H_i penalty scheme in P_s function definition.

- An important point to note here is that; for B_1, B_2, B_5, and B_6 linked lists, we only consider the transitions that are longer than *min_intron*. We compute forward transitions in X_G that are shorter than *min_intron* separately and merge the results with B_1 and B_2 cases. Similarly we compute forward transitions (towards left-side) in $X_{G'}$ that are shorter than *min_intron* separately and merge with the results that are obtained from B_5 and B_6 cases.

- B_9 to B_{16}: These cases correspond to the backward transitions instead of forward in each of the cases from B_1 to B_8. Cases B_9 to B_{12} represent backward transitions (towards the left side) X_G to X_G or $X_{G'}$ from $(+)/(-)$ column subsets. The direction to construct and handle the lists are right-to-left, similar to B_5 to B_8, but the transition penalties are calculated accoring to the H_b penalty scheme defined in P_s. The resulting rows are stored within artificially constructed arrays $R_9[1..M]$ to $R_{12}[1..M]$. Cases B_{13} to B_{16} correspond to the backward transitions (towards the left side) from $X_{G'}$ to X_G or $X_{G'}$ from $(+)/(-)$ column sets. The direction to handle the lists are left-to-right similar to the cases from B_1 to B_4 with the difference that gap penalties are calculated

[17]Therefore, the blocks that represent same parent blocks do not actually hold ending column indices but beginning column indices and all splits and merges are handled accordingly.

as described in the H_b penalty scheme of P_s function. The corresponding results are stored in R_{13} to R_{16}.

- After all of the rows R_1 to R_{16} are generated, the eight arrays that correspond to X_G (the cases that represent transitions from X_G or $X_{G'}$ to X_G) are merged into the $(i+1)^{th}$ row of X_G by selecting the cell with the maximum score from these arrays for each column. Similarly, the other eight arrays that correspond to $X_{G'}$ (the cases that represent transitions from X_G or $X_{G'}$ to $X_{G'}$) are merged into the $(i+1)^{th}$ row of $X_{G'}$ by selecting the maximum values among them for each column.

After the matrices X_G and $X_{G'}$ are constructed in the way described above, the construction of X_S and $X_{S'}$ take place as follows.

For each of the rows i, from 1 to $N-1$:

- Determine the highest scoring column Z_G amongst the i^{th} row of X_G according to the following function $X_G[i,Z] + J_b(Z)$.

- Determine the highest scoring column $Z_{G'}$ amongst the i^{th} row of $X_{G'}$ according to the following function $X_{G'}[i,Z] + J_b(Z)$ (Note that J_b function in this case will check for reverse complement splice beginning signal).

- Initialize the parents of all cells in the $(i+1)^{th}$ row of X_S and $X_{S'}$ as the highest scoring cell between the two cell values[18], $X_G[i,Z_G]$ and $X_{G'}[i,Z_{G'}]$. Calculate and store the corresponding score values from these fusion transition functions (e.g. $X_G[i,Z] + J_b(Z) - C_f$ if $X_G[i,Z_G] > X_{G'}[i,Z_{G'}]$). The reason that all the cells are assigned the same parent is because fusion transitions are independent from genomic distance and alignment direction of the bases (as well as the direction matrix) on two sides of the fusion[19].

- Apply all of the steps described in the construction of X_G and $X_{G'}$ matrices apart from initialization and the final merging steps by replacing the terms X_G with X_S, $X_{G'}$ with $X_{S'}$, $R_1[1..M]$ to $R_{16}[1..M]$ with $R_1[1..L]$ to $R_{16}[1..L]$.

[18]Note that, the transition being representing an inversion does not affect the penalty of a fusion transition.

[19]This independence causes all of the sixteen different cases handled in the construction of X_G and $X_{G'}$ to merge into a single case.

- The difference in the final merging step is that, when merging the eight R arrays that correspond to the $(i+1)^{th}$ row of X_S, we also consider the actual $(i+1)^{th}$ row values of X_S that are initially assigned by fusion transitions. Similarly when merging the other eight R arrays we take the $(i+1)^{th}$ row of $X_{S'}$ into account together with the fusion transitions.

The backtracking step of TGASA-2 is same as TGASA-1.

It is clear that the number of operations for each specific case in TGASA-2 is in the same order as WTTA-2 algorithm. Since there are only a constant number of different cases to be considered, TGASA-2 algorithm also has the same overall complexity as WTTA-2.

Changing the splice signal scoring from the summation scheme to the conjunction scheme in TGASA-2, is not as straightforward as in the case with TGASA-1. In this particular case; since the splice site properties from both sides would affect the convexity property of the gap cost function, the $(+)/(-)$ cases that represent the column subsets of the i^{th} row with respect to the existence of canonical splice beginning sites should further be divided into three $(++)/(+-)/(-)$ cases ($(-+)$ and $(--)$ can be merged into $(-)$), that represent subsets from both i^{th} and $(i+1)^{th}$ rows, which are handled as separate B cases that satisfies the convexity property within each case.

In order to extend the splice sginal scheme to the most generalized case of a splice signal score matrix (of c_1 splice beginning signals and c_2 splice ending signals) would require the analysis of $c_1 \times c_2$ different B cases for each forward assignment in TGASA-2. However; since c_1 and c_2 are assumed to be constants, the run-time complexity of the extended algorithm is not more than the original TGASA-2 algorithm.

Further improvements to TGASA-2

Extending TGASA-2 for affine gap penalty model on the transcript side is analogous to the extension of WTTA-2.

We should modify the S_m function to S_{ma} as

$$S_{ma}(t_i, F(t_i)) = \begin{cases} C_{match} & \text{for} \quad v(t_i) = v(F(t_i)) \\ C_{mismatch} & \text{for} \quad j \neq \phi \wedge v(t_i) \neq v(F(t_i)) \\ C_{gap-open} & \text{for} \quad j = \phi \wedge F(t_{i-1}) \neq \phi \\ C_{gap-extend} & \text{for} \quad j = \phi \wedge F(t_{i-1}) = \phi \end{cases}$$

and split each of the matrices X_G, $X_{G'}$, X_S, $X_{S'}$ into three as X_G^M, X_G^G, X_G, $X_{G'}^M$, $X_{G'}^G$, and so on.

In the definition above, regardless of the original matrix; $X^M[i,j]$ represents the alignment of (i,j) pair as a match or mismatch, $X^G[i,j]$ as a gap, and $X[i,j]$ best of both. When calculating the scores of $X[i,j]$, $X^G[i,j]$, and $X^M[i,j]$ in the forward assignment step of TGASA-2; which matrix to use for the base score is as described in the case with WTTA-2, with the difference that the previous alignment pair can be from different genomic sequences.

The reduction from $O(MNlogM)$ run-time complexity to $O(MN)$ for convex gap penalty functions that satisfy "closest zero property" was described for the improvements for WTTA-2. A similar improvement can be made for TGASA-2, by replacing the binary search in each one of the cases B_1 to B_{16} for both G and S, with the exact calculation of the same parent block breakpoint. The difference in this case would be the binary search for the reverse ordered parent blocks (that hold beginning column parent index instead of the ending column), but this issue can also be easily solved by switching indices as if each column index k is replaced with $M - k + 1$.

2.2 Low-sensitivity Fragment Chaining Framework for Transcriptome to Genome Alignment with Structural Alterations

As described in the previous section, the problem of transcriptome to the genome alignment with structural alterations[20] can be optimally solved in polynomial time. However; for high-throughput transcriptome to genome alignment studies, run-time and memory requirements of TGASA-2 will be costly even with the improvements for log-scale gap cost functions.

In this section we propose a lower sensitivity solution for transcript to genome alignment with structural alterations by initially generating a set of homologous fragments between the transcript and genome sequences and stitching a list of fragments from this set afterwards in order to obtain a chain of substring alignments from the transcript to the genome.

Different than the previous section, we directly introduce the formulation of fragment chaining with structural alterations, without an introduction for the wild-type alignment

[20]As in the problem definition given in **Problem 1**.

version. However, we initially introduce a simpler version of the problem with no overlapping fragments in Problem 3 and provide a solution for this version. A more complex version of the problem with overlaps between chained fragments is investigated in subsection 2.2.2, which is a more realistic model of real-life fragment chaining applications.

In this section we do not describe methods for generating a comprehensive set of alignment fragments, but the method we use to efficiently construct a set alignment fragments is described in detail in the following section that introduces our transcriptome to genome alignment tool, Dissect. But for the following problem formulations to be complete, initially we mathematically define the properties of an alignment fragment and a valid set of fragments.

A pairwise alignment fragment F between a transcribed sequence T and a genomic sequence G is a data point that represents a homologous region of specific length between A and B. Each fragment holds the following pieces of information: starting position in the transcribed sequence, identity of the genome sequence[21], starting position in the genome sequence, direction of alignment in the genome[22], the length of the fragment, and the score of the fragment; respectively denoted by $F = (F.ts, F.gen, F.gs, F.dir, F.len, F.score)$. An important point to note here is that; $F.dir$ only represents the direction of the alignment on the genome side and the direction of all fragments on the transcript side are assumed to be $(+)$[23]. Another important property of an alignment fragment is that; there is only a single length value for both the genome and transcript sequences, which indicates that there cannot be any insertions or deletions within the transcript on either side[24]. This, however, does not mean that there cannot be mismatches within the fragment. Actually, the score of a fragment is directly related to the number of matches and mismatches within the fragment according to the scoring scheme defined below:

$$F.score = (C_{match} * (F.len - \text{ \#mismatches in F})) + C_{mismatch} * \text{ \#mismatches in F} \quad (2.11)$$

[21]This information will be useful in the fragment chaining method when two chained fragments are aligned to different genomic sequences.

[22]From here on, we often refer to the forward (downstream) alignment direction as $(+)$, and refer to the reverse direction alignment (upstream on the reverse complement strand) as $(-)$.

[23]This property comes without a loss of generality due to the fact that any fragment that is in the reverse direction in the transcript can be converted to a forward direction fragment by only changing the direction on the genome side and modifying fragment start positions accordingly.

[24]This property is not compulsory for the described algorithms below from a theoretical standpoint, yet due to its simplicity and closer representation of the fragment construction methodology adopted in our alignment tool Dissect; we also assume this property for the following set of problem definitions and solutions.

For the following set of problems we are given a set, $F_{set} = \{F_1, F_2, \ldots, F_K\}$, of K alignment fragments between a transcribed sequence and a genomic sequence. For our problem case, we have four separate sets for each of the genome sequences G, G', S, and S' as described in the previous section (the *gen* value for these would be 1, 1, 2, and 2 respectively, indicating the sequences' genomic origin as the first sequence or the second sequence). The sets for G and S only comprise of forward fragments, whereas the sets for the sequences G' and S' are in the reverse direction and the scores are calculated based on the complement similarity of each corresponding base pair. One key point to note about these fragment sets is that; they are concise in the way that the same fragment is not listed twice and no fragment is a sub-fragment[25] of another fragment in the set, thus all fragments are maximal. Also two fragments in F_{set} are called disjoint if they do not overlap in the transcribed sequence T and they are called overlapping fragments if they contain at least one common base in T.

It is also important for complexity concerns to mention that in this formulation described below, the maximum number of mismatches in a fragment is bounded by a constant number independent from the length of the fragment. This property is of particular importance due to the fact that transition penalty between two overlapping fragments will be related to the number of mismatches within the overlap interval for both fragments. Even though this property is not necessarily strict in the simpler Problem 3 formulation of disjoint fragment chaining, it is crucial for the complexity analysis of overlapping fragment chaining to have a well-defined mismatch model. At the end of each algorithm description, we discuss the effect of having an unbounded error rate to the complexity of the algorithm. Also we denote the list of mismatches of a fragment F as F^{mm}, and the position of its i^{th} mismatch in the transcript as F_i^{mm}.

2.2.1 Problem 3: Disjoint fragment chaining with structural alterations

In our problem formulation, a valid fragment chaining between T and [G, G', S, S'], of length N and [M,M,L,L] respectively, is a sequence of $k \leq K$ fragments $L_c = (F_1, F_2, \ldots, F_k)$, with each element $F_i \in F_{set}$ satisfying the following two constraints:

- $\forall i \in [1, k-1]$, $F_i.ts + F_i.len \leq F_{i+1}.ts$

[25]We define a sub-fragment as a fragment generated by removing leading or trailing matching/mismatching base pairs from a larger alignment fragment.

- $\forall i \in [1, k-1]$, $(F_i.gen = 2) \implies (F_{i+1}.gen = 2)$

The first constraint defined above ensures that the starting indices of the fragments on the transcript monotonously increase throughout the chain and it also prohibits any overlaps in between adjacent fragments. The second constraint ensures that when a fragment chain switches to the secondary fused sequence, all of the downstream fragments are also within the second sequence. This prevents multiple fusions within a transcript.

For conciseness in the following algorithm description, we will use the terms, F_i is chained to F_{i+1} and F_{i+1} is chained from F_i referring to the adjacency of the fragments F_i and F_{i+1} in a chain.

With the definition of a valid fragment chain as described above; our goal is to find the highest scoring chain L_b (of length $B \le K$ without loss of generality) over F_{set}, given the fragment chain scoring function F_{score} and the transition penalty function P described as below:

$$F_{score}(L_b) = \sum_{i=1}^{B} F_i.score - \sum_{i=0}^{B} P(i, i+1)$$

$$P = \begin{cases} P_t(F_i.ts - 1) & for \quad i = 0 \\ P_t(N - F_i.te) & for \quad i = B \\ P_t(F_{i+1}.ts - F_i.e - 1) + P_s(F_i.ge, F_{i+1}.gs) \\ \quad - J_s(F_i.ge, F_{i+1}.gs) & for \quad 0 < i < B \end{cases}$$

Even though $F.ge$ is not defined as part of fragment F, for simplicity purposes we assume that $F.ge$ denotes the ending index of a fragment in its respective genome considering the directionality of the fragment. Similarly $F.te$ denotes $F.ts + F.len - 1$.

In the transcript and genome gap penalties given above, the alignment is penalized globally on the transcript side considering trailing and leading gaps and locally on the genome side omitting trailing/leading gaps.

Similar to the formulations in the previous section, the transcript gap distance penalty, P_t is a linear function[26] defined as follows.

$$P_t(d) = C_{gap} \times d$$

The genomic transition penalty function P_s and canonical splice signal scoring function J_s are the same as their definition in *Section 2.1.2.* However; it is important to note that

[26] This function can be chosen as an affine gap penalty function without affecting the solution complexities throughout the section.

the value $F_k.gs$ sent to P_s or J_s functions does not represent an integer value of size $F_k.gs$, but rather the genomic base $(F_k.gen)_{(F_k.gs)}$. Sequence origin is crucial for the calculations of the genomic transition penalty function and the canonical splice signal score function.

We describe our method of solving the problem of disjoint fragment chaining with structural alterations as follows:

- Sort all fragments $F_i \in F_{set} = \{F_1, \ldots, F_K\}$, in a non-decreasing order based on their transcript starting positions $(F_i.ts)$. From here on, we refer to F_i as the i^{th} fragment in the sorted sequence $F_{list} = (F_1, \ldots, F_K)$. We also denote the best chaining score of F_i as $F_i.chain$ and the fragment that it is chained from as $F_i.parent$.
- Initialize all fragment chains F_i, as the beginning of their own chain.
    - $F_i.chain \leftarrow (F_i.score - P_t(F_j.ts - 1))$
    - $F_i.parent \leftarrow source$
- For each i from 1 to $K - 1$:
    - $j_{start} \leftarrow$ minimum k such that $F_i.ts + F_i.len \leq F_k.ts$
    - For each j from j_{start} to K
        * Transcript gap penalty, $tp \leftarrow P_t(F_j.ts - F_i.te - 1)$.
        * If $F_i.chain - P_s(F_i.ge, F_j.gs) + J_s(F_i.ge, F_j.gs) + F_j.score > F_j.chain$, replace $F_j.chain$ and update $F_j.parent \leftarrow F_i$.
        * Otherwise, do not update $F_j.chain$ or $F_j.parent$.
- Finalize each chain by subtracting the penalties for reaching the end of the transcript sequence from each fragment F_i:

$$F_i.chain \leftarrow (F_i.chain - P_t(N - F_i.te))$$

- Return the chain score of the fragment F_i with the highest $F_i.chain$ value. In order to reconstruct the chain, backtrack through parents of each fragment until source is reached.

Such an algorithm would correctly calculate best non-overlapping fragment chaining given the set of fragments F_{set}.

The sorting step at the beginning takes $O(min(Klog(K), N+K))$ operations depending on whether merge sort or counting sort is applied. However, this is dominated by the two nested loops that iterate along fragments, which has a run-time cost of $O(K^2)$, calculating the transition cost for each pair in constant time. Finally, the backtracking step will take $O(min(K, N))$ operations, yielding an overall algorithm run-time complexity of $O(K^2)$. The memory complexity of the overall algorithm will be $O(K)$, or alternatively $O(K+N)$ if we employ counting sort instead of merge sort.

We can also generalize this method to unbounded mismatch scheme[27] staying within the same run-time complexity. This is due to the fact that we do not consider any overlaps that would require detailed mismatch resolution for calculating transitions between fragments. However, due to the maximal extension property of the fragments; in real life transcriptome datasets, overlapping fragments can be caused by either over-extension of the fragments or existence of homologous genomic sequences in the leading and trailing flanking sequences of fragment alignments.

In the following set of problem formulations, we consider the cases in which the fragments that are chained can contain overlapping intervals.

2.2.2 Observations upon overlapping fragment chaining

In a real-life experimental setting, the set of maximal fragments is likely to contain fragments that are over-extended on either side. This might be due to a homology between the flanking sequences in the transcript and the genome, even though the extended region can be part of another fragment exhibiting higher similarity within the extended region. In *Problem 3* formulation given above, such two fragments are not allowed to be chained together. However, since each fragments are maximal[28], the ideal chaining might be missed due to the overlap between a fragment pair on the transcript side. The ideal chaining in the case described above could be the chaining to a suffix of the second fragment from a prefix of the first fragment, such that they are not overlapping. In this section, we investigate several possible problem formulations and solutions for chaining fragments that might overlap in the transcript sequence.

[27]in which each fragment F_i can hold $O(F_i.len)$ mismatches in its mismatch list F^{mm}.

[28]There can be no sub-fragments within the given set.

Defining the fragment chaining problem for overlap resolution is tougher than the original disjoint chaining problem defined above, due to the different approaches of overlap resolution schemes involving length of the overlap, mismatches within fragments and/or updated genomic distance of the newly constructed fragments after overlap resolution. Below we redefine the concept of valid fragment chains described in the previous subsection and investigate several transition scoring models for different overlap resolution schemes.

In this new problem formulation, a valid fragment chaining between T and [G, G', S, S'], of length N and [M,M,L,L] respectively, is a sequence of $k \leq K$ fragments $L_c = (F_1, F_2, \ldots, F_k)$, with each element $F_i \in F_{set}$ satisfying the following two constraints:

- $\forall i \in [1, k-1]$, $F_i.ts < F_{i+1}.ts \wedge F_i.ts + F_i.len < F_{i+1}.ts + F_{i+1}.len$
- $\forall i \in [1, k-1]$, $(F_i.gen = 2) \implies (F_{i+1}.gen = 2)$

Therefore, the fragments in a chain are allowed to overlap as long as the transcript starting indices of the fragments in the chain are monotonously increasing and no fragment is fully contained within another.

With the definition of a valid fragment chain with overlaps, the score of a given fragment chain L_b of length B is defined as follows:

$$F_{score}(L_b) = \sum_{i=1}^{B} F_i.score - \sum_{i=0}^{B} P(i, i+1)$$

$$P = \begin{cases} P_t(F_i.ts - 1) & for \quad i = 0 \\ P_t(N - F_i.te) & for \quad i = B \\ P_t(F_{i+1}.ts - F_i.e - 1) + P_s(F_i.ge, F_{i+1}.gs) & for \quad 0 < i < B \wedge F_i.te < F_{i+1}.ts \\ P_o(F_i, F_{i+1}) & for \quad 0 < i < B \wedge F_i.te \geq F_{i+1}.ts \end{cases}$$

$F_i.te$ indicating the ending position of F_i in T, $P_o(F_i, F_{i+1})$ representing the transition penalty for overlapping fragment pairs.

Note that, the splice signal score J_s is omitted from the definition of P function in this subsection. This is due to the complications that will be caused by the integration of splice junction scores into the formulation. This concern is addressed in detail in the final part of this subsection.

Before going into the details of different P_o function definitions, we first define the notion of overlap split position of a fragment pair overlapping in the transcript.

Given a valid fragment chain $L_c = (F_1, \ldots, F_k)$, an overlap split position between two overlapping fragments, F_i and F_{i+1}, is an index $r \in [F_{i+1}.ts - 1, F_i.te]$ in the transcript sequence indicating the transcript ending position of the updated F_i fragment, F'_i, and the position right before the transcript beginning position of the updated F_{i+1} fragment, F'_{i+1}.

Without detailing how the new score values are obtained, the exact properties of the updated fragments with respect to the split position r, are as follows:

$$F'_i \leftarrow (F_i.ts, F_i.gen, F_i.gs, F_i.dir, r - F_i.ts + 1, F'_i.score)$$

$$F'_{i+1} \leftarrow \begin{cases} (r+1, F_{i+1}.gen, F_{i+1}.gs + \beta, F_{i+1}.dir, F_{i+1}.len - \beta, F'_{i+1}.score) & F_{i+1}.dir = (+) \\ (r+1, F_{i+1}.gen, F_{i+1}.gs - \beta, F_{i+1}.dir, F_{i+1}.len - \beta, F'_{i+1}.score) & F_{i+1}.dir = (-) \end{cases}$$

$$\beta = r + 1 - F_i.ts$$

Hence, the overlap splitting procedure given above will return two new fragments adjacent in the transcript and these will be sub-fragments of the original fragment pair.

In the remainder of this subsection, we describe several different overlap resolution schemes, which utilize the overlap splitting procedure described above with different score values for the updated fragments. We also talk about the limitations of the overlap model and the overlap splitting scheme given above, such as the exclusion of the cases in which the overlap region covers one of the fragments entirely and the cases in which there are more than two fragments overlapping at a single transcript position.

Overlap resolution based on length of the overlapping interval and genomic distance

In this overlap resolution scheme, P_o penalty function is not specifically dependent on the overlap split position but only on the length of the overlap interval and the distance of the genomic gap[29] in between after the fragments are updated. Given an overlap split position q for overlap resolution of F_i and F_{i+1}, P_o is defined as:

$$\begin{aligned} P_o(F_i, F_{i+1}) = P_s(&genpos(F_i, q - F.ts), genpos(F_{i+1}, q + 1 - F_{i+1}.ts)) \\ &+ (F_i.te - F_{i+1}.ts + 1) \end{aligned}$$

[29]Note that there will be no gap in the transcript in between the updated fragments obtained from an overlap resolution.

The returned value from $genpos(A, x)$ is the base in $A.gen$ that corresponds to the x^{th} position of the fragment in the genome, namely:

$$genpos(A, x) = \begin{cases} Q_{(A.gs+x-1)} & for \quad A.dir = (+) \\ Q_{(A.gs-x+1)} & for \quad A.dir = (-) \end{cases}$$

Q representing the genome sequence the fragment is aligned to.

Therefore; in this overlap resolution model P_o function does not depend on how the updated fragments are formed, but only on the cut position q and the overlap length. Since the overlap length is set given F_i and F_{i+1}, the problem to solve for this model is to select the optimal q split position that minimizes P_o, thus $P_s(genpos(F_i, q-F.ts), genpos(F_{i+1}, q+1-F_{i+1}.ts))$.

It is important to note here that, if the fragment pair is very closely located in the genome, there might be a change of transition type when overlap resolution is applied. For instance, the original fragment pair might have a backward transition (which has a penalty determined by H_b function), whereas the updated fragment pair can have a regular forward transition (which has a penalty determined by H_n function). On the other hand, it is clear that an overlap procedure cannot convert an inverted transition into a regular forward transition or vice versa. Furthermore, it is also not possible to alter a fusion transition through overlap resolution into a non-fusion transition.

As described above, the only way an overlap resolution can convert a transition type is from a backward genomic transition into a forward one. Such cases can only occur when considering an overlap between two fragments in the same direction, hence the overlap resolution position will not affect the distance between the updated fragment pair. Due to the fact that this distance is constant with respect to the cut position, it requires only constant number of operations to check whether the transition type is altered. If so, the resulting penalty for the constant genomic distance can also be calculated in constant number of operations without affecting the run-time complexity of the overall algorithm.

Apart from genomic transition type concerns, it can be observed that; if $F_i.dir = F_{i+1}.dir \vee F_i.gen \neq F_{i+1}.gen$, the distance is not affected by the split position and any arbitrary split position q within the interval $[F_{i+1}.ts - 1, F_i.te]$ will yield the same score. To be consistent with the splitting scheme throughout the fragment chaining, the overlap split is chosen as the earliest possible index in the transcript, $q = F_{i+1}.ts - 1$. However; if $F_i.dir \neq F_{i+1}.dir \wedge F_i.gen = F_{i+1}.gen$, the split position will affect the size of the genome gap.

For the second case above, it might seem that the gap penalty function h needs to be calculated at many split positions. However; for any arbitrary gap penalty function that is non-decreasing with respect to the gap distance, there are only three potential cut positions to be considered. If the overlap suffix of F_i in the genome comes strictly after the overlap prefix of F_{i+1}, q should be selected as the position in the transcript that corresponds to the lowest position value in the genome[30]. Reversely, if it comes strictly before, q should be selected as the position in the transcript that corresponds to the highest position value in the genome[31]. Finally, if these two regions in the genome are overlapping, we can select q to be the position that makes the ending of F'_i meet with the beginning of F'_{i+1} in the genome. Such an overlap split selection guarantees the genomic gap distance between F'_i and F'_{i+1} to be 0 or 1, minimizing $P_s(F'_i, F'_{i+1})$. Calculation of q value is performed as follows:

$$q \leftarrow \begin{cases} \lfloor \frac{F_i.gs+(F_{i+1}.ts-F_i.ts)+F_{i+1}.gs}{2} \rfloor & for \quad F_i.dir = (+) \\ \lfloor \frac{F_i.gs-(F_{i+1}.ts-F_i.ts)+F_{i+1}.gs}{2} \rfloor & for \quad F_i.dir = (-) \end{cases}$$

Since P_o calculation can be done in constant time as described above for each fragment pair[32], the complexity bound will be again $O(K^2)$ if the solution described in the previous subsection is used, with the omission of splice signal scores and definition of the new P penalty function.

Extension for splice signal scores will be investigated later as a limitation of this problem definition. However, the exclusion of J_s score function from this problem formulation is mainly due to the requirement of scanning the positions within each overlap region (that is of size $O(N)$), significantly increasing the run-time complexity of this solution method.

Minimal mismatch penalty model for the resolution of overlapping fragments

In this overlap resolution scheme, updated F'_i, F'_{i+1} fragments are considered to have scores based on their length and the number mismatches retained in each fragment. With the updated scores of F'_i and F'_{i+1}, overlapping penalty function, P_o, is defined as:

$$\begin{aligned} P_o(F_i, F_{i+1}) = {} & P_s(genpos(F_i, q - F.ts), genpos(F_{i+1}, q + 1 - F_{i+1}.ts)) \\ & +(F_i.score - F'_i.score) + (F_{i+1}.score - F'_{i+1}.score) \end{aligned}$$

[30]This position is $F_{i+1}.ts - 1$ for $F_i.dir = (+)$, and $F_i.te$ for $F_i.dir = (-)$.

[31]This position is $F_i.te$ for $F_i.dir = (+)$, and $F_{i+1}.ts - 1$ for $F_i.dir = (-)$.

[32]Even when we consider overlapping fragments, there are $O(K^2)$ valid fragment pairs.

genpos function as defined earlier and q being the overlap split position.

The scores of the updated fragments F'_i and F'_{i+1} are calculated as in the following, given q:

$$F'_i.score = F_i.score - C_{match} * (F_i.te - F_{i+1}.ts + 1) + C_{mismatch} * mis(F_i, F_{i+1}.ts - 1, q)$$

$$F'_{i+1}.score = F_{i+1}.score - C_{match} * (F_i.te - F_{i+1}.ts + 1) + C_{mismatch} * mis(F_{i+1}, q + 1, F_i.te)$$

$mis(F, a, b)$ is defined as the number of mismatch positions $p \in F^{mm}$ that have a value within the interval from a to b, $a \leq p \leq b$.

With the score functions defined as above, the optimal cut position q is the value that minimizes both the genomic distance and the summation of the number of mismatches retained within the updated fragments F'_i, F'_{i+1}.

We can minimize the overlap penalty function applying the following solution steps:

- Merge the mismatch lists F_i^{mm} and F_{i+1}^{mm} in a new list L, in the increasing order of transcript positions, and note the fragment origin of each mismatch.
- Assign $r_i \leftarrow 0$ and $r_{i+1} \leftarrow |F_{i+1}^{mm}|$, which respectively indicate the number of mismatches retained in F_i and F_{i+1} for $q = F_{i+1}.ts - 1$
- Evaluate $scorediff = (F_i.score - F'_i.score) + (F_{i+1}.score - F'_{i+1}.score)$ for the current $q = F_{i+1}.ts - 1$ value and save as $L_0.score$
- For each mismatch position j in L:
  - If the mismatch is obtained from F_i, increase r_i by one,; if it is obtained from F_{i+1}, decrease r_{i+1} by one.
  - Evaluate $scorediff = (F_i.score - F'_i.score) + (F_{i+1}.score - F'_{i+1}.score)$ for the current $q = L_j$ value and save as $L_j.score$.
- Note that the $scorediff$ value for each $q \in [L_j, L_{j+1} - 1]$ is the same[33].
- For each same score interval calculate the lowest $P_s(F'_i, F'_{i+1})$ value similar to the previous solution[34] (by choosing the smallest possible q value if $F_i.dir = F_{i+1}.dir \vee F_i.gen \neq F_{i+1}.gen$ and evaluating the three possible q values described in the previous method if $F_i.dir \neq F_{i+1}.dir \wedge F_i.gen = F_{i+1}.gen$).

[33]for the last interval, assume that $L_{|L|+1} = F_i.te$

[34]Also considering the potential change of transition type during the overlap resolution.

- Among the lowest P_s positions in each interval, return the optimal split position that yields the lowest P_o value[35].

Run-time complexity of this solution depends on the number of mismatches in the fragment pair. Since we have a constant number mismatches in each fragment, we can calculate P_o in constant time for each fragment pair. Thus, the run-time complexity bound of the overall chaining solution is again within $O(K^2)$.

Limitations of the overlap resolution scheme

Even though the extension of the chaining formulation to the overlapping fragments do not change the complexity of the original disjoint chaining algorithm, there are certain limitations of this new model.

Firstly, the definition given above prevents a fragment in the chain to be contained within another. This property might cause the exclusion of the chaining of fragments, in which a low quality extension of the first fragment covers the second fragment entirely that has a relatively higher quality (or vice versa). However, we assume that the fragment error rate constraints for the maximal fragment set construction step will prevent such low quality extensions from occurring. Inclusion of such cases within the model would abolish the uni-directional monotonous increasing property, which we base our algorithm on, in order obtain a $O(K^2)$ solution. A naive solution method for the extended case will be $O(K^2N^4)$ run-time complexity, constructing all distinct sub-fragments and applying the non-overlapping chaining algorithm described in the previous subsection.

Another chain formation that is not addressed in the formulation given above is the case in which three or more fragments are overlapping on the same position in the transcript. Given that, F_i, F_{i+1}, F_{i+2} are three adjacent fragments lying on a fragment chain and F_i and F_{i+2} also overlap at some transcript position. Since the intermediate fragment is not allowed to be contained in any of the adjacent fragments, F_{i+1} should cover point where F_i and F_{i+2} meet in the transcript and thus should contain separate overlaps with both F_i and F_{i+2}, outside of their common overlap region. Below we investigate the conditions in which such a chain formation might occur.

If all three fragments are on a wild-type transcript (facing the same direction and have an ordered alignment in the genome) and the overlap resolution is performed regardless of

[35]The summation of P_s and $scorediff$ values.

the number of mismatches in the overlapping region according to the first formulation given above, such a formation cannot occur due to the fact that the chaining score directly from F_i to F_{i+2} is higher than through F_{i+1}, assuming that H_n distance penalty function is a convex gap cost function that return a positive value that increases with gap distance. This is also the case with transcripts that contain rearrangements if H_b is a convex gap cost function with the properties mentioned above.

H_n and H_b were already assumed to be proper gap cost functions at the beginning of the formulation. There is actually no strict rule defined for the relative values between H_n, H_b and H_i, but in a proper P_s function definition H_i is assumed to be more costly than both H_n and H_b for any gap distance. If it were not for this property, two fragments could be chained through a fragment facing the other direction in order to increase their chaining score (decrease their fragment transition penalty). But with such a property given, it will always be more costly to chain through an inverted fragment, in comparison to chaining directly from F_i to F_{i+2}.

With the cases analysed above, it is clear that with proper assumptions upon gap cost functions and their relative magnitude, in the first (simpler) overlap resolution scheme there will not be more than two fragments chained together that overlaps at a single position in the transcript. Below we investigate the same situation with the second overlap resolution scheme that takes mismatches also into account.

Within the overlap resolution scheme with mismatches, three adjacent fragments F_i, F_{i+1}, and F_{i+2} in a fragment chain can actually overlap at a common position in the transcript, even as a wild-type transcript alignment. A fragment formation like this could occur when the intermediate fragment shows similarity to both of the genomic flanking sequences of F_i and F_{i+2} but represents a higher similarity alignment (with less number of mismatches) than the extending regions of its adjacent fragments, so that the gap penalty cost of adding the intermediate fragment is dominated by the benefit of removing the mismatches in the overlapping region of F_i and F_{i+2}.

Even with a multiple overlap formation as described above; the updated fragments cannot contain contradicting indices if the overlap split position q from F_i, F_{i+1} overlap resolution comes earlier in the transcript, than the split position r from F_{i+1}, F_{i+2} overlap resolution; since this leaves a positive length fragment for the updated intermediate fragment F'_{i+1}. The q,r relation that satisfies this property is $q+1 < r$. Below we investigate whether an overlap resolution case in which $q + 1 \geq r$ might occur.

In a wild-type transcript, in which all fragments have the same direction and are ordered in the genome; a selected q position indicates that in any interval $[j, q]$ ($F_{i+1}.ts \leq j \leq q$), F_i contains less mismatches than F_{i+1} (or equal), since otherwise we could shift q to an earlier position in the transcript to further minimize $P_o(F_i, F_{i+1})$. Similarly r position selection indicates that within any interval $[q+1, j]$ ($q+1 \leq j \leq F_{i+1}.te$), F_{i+2} contains less mismatches than F_{i+1} (or equal), since otherwise we could shift r to a later position in the transcript in order to minimize $P_o(F_{i+1}, F_{i+2})$. From these constraints upon mismatches within the overlap interval, we can infer that within the interval $[r, q]$ ($q \geq r$), both F_i and F_{i+2} have less mismatches than F_{i+1}, therefore any cut position within the interval $[r-1, q]$ from F_i to F_{i+2} would reduce (or not increase) the number of mismatches in comparison to the summation of mismatches in the transitions from F_i to F_{i+1} and from F_{i+1} to F_{i+2}. Combined with the effect of convex gap penalty H_n in this situation, it is clear that the split positions $q+1 \geq r$ always have a better or same scoring alternative split selection $q'+1 < r'$. For the reasons described earlier, this condition similarly applies to transcripts with rearrangements if H_b is convex, and also to transcripts with inversions if H_i is selected to be more costly than both H_n and H_b gap cost functions. Hence, within the assumed gap cost model, the updated fragments will not be problematic if a pairwise overlap resolution is applied to F_i, F_{i+1} pair and F_{i+1}, F_{i+2} separately, instead of a joint overlap analysis of F_i,F_{i+1}, and F_{i+2} all together.

In the formulation given above for the overlapping fragment chaining problem, splice signal score function J_s is not included within the penalty function P. Due to the fact that each fragment's location in the genome is independent from the other fragments in the chain; there are $O(min(KN, M+L))$ positions in the genome[36] that needs to be checked for splice signal existence for the entire fragment set, F_{set}. However, since there is no particular way of determining the ideal cut position for each fragment pair without evaluating the combined splice signal score for each cut position, $O(N)$ operations need to be performed per transition penalty calculation. Therefore the overall run-time complexity for such an algorithm solving fragment chaining with splice signals is $O(K^2N)$.

In the following section we describe the set of methods and design preferences that are adopted in our high-throughput transcriptome to genome alignment tool sensitive to structural alterations in the transcript. Handling of splice junctions and fragment overlaps

[36] K,N,M, and L representing the size of F_{set}, T, G/G' and S/S' respectively.

in our aligner is described in detail in *Step 2* and *Step 3* subsections.

2.3 Efficient Whole Genome Search of Novel Transcriptional Structural Alterations

In the previous sections, we set up a theoretical framework for the transcriptome to genome alignment problem with structural alterations via base-to-base alignment and fragment chaining approaches. However, there are still other challenges to be considered when performing such an alignment in a high-throughput fashion from a large list of transcriptomic sequences (assemblies or long reads) to a whole genome sequence in the order of 10^9 bases. In this section we describe the detailed computational aspects of our transcriptome to whole genome alignment software, Dissect (**DIS**covery of **S**tructural **E**vent **C**ontaining **T**ranscripts), in terms of the approaches adopted to tackle these challenges.

The first challenge for such a method would be to reduce the search space within whole genome to the putative gene/gene pair that the wild-type or abnormal transcript is located. Therefore the initial problem to be addressed in a whole genome aligner is to efficiently locate the genomic region(s) that the given transcript is likely to be located.

In this study, we propose a general approach for tackling this challenge, based on extracting sample anchors from the transcript, mapping to the whole genome, and detecting regions that these anchors are densely distributed. This method does not assume a fixed length region size, but tries to infer the location and the length of the region together according to an optimization score. This method is described in the subsection *Step 1*.

Another challenge is to efficiently construct a comprehensive list of alignment fragments between the given transcript and the genomic region within given length and mismatch constraints. In the subsection *Step 2*, we describe the fragment generation method we have used in Dissect and further describe how these fragments are chained together in our fragment chaining method using the algorithms described in section 2.2.

Furthermore; after obtaining the best scoring fragment chain, due to the loss of resolution caused by mismatch and length constraints of the alignment fragments, there can be unwanted divergence in the exon boundaries from the optimal base to base alignment. In this final post-processing step we try to resolve small scale misalignments and divergence from an optimal alignment. In the subsection *Step 3* we describe these post-refinement methods in detail.

And in the final subsection, we describe the implementation details of our aligner that are not mentioned in *Steps 1-3*, mostly regarding disk and memory use optimizations throughout these steps and design considerations for functionality.

2.3.1 Step 1: Genomic region inference from anchor mappings

Genome region inference step of Dissect starts with sampling anchors from the transcribed sequence. Many of the commonly used methods in the literature of transcript to whole genome alignment employ some form anchor based genome region inference[21][46][41][2]. Even though they adopt the approach of sampling anchor sequences from the transcript and try to map them to the genome in order to infer the genomic subregion to perform the alignment, they have key differences in how to infer putative genes from anchor mappings. For instance, GMAP employs a method for finding the entire span of the transcript by sampling anchors from the beginning and end of the transcript, mapping them to the genome within a maximum region length threshold. However, in our problem with structural alterations, such an approach will cause rearrangements to be missed due to the long translocated sequences within the transcript that might be aligned to a location outside of this region. Alternatively, BLAT infers a genomic region by initially splitting the genome into constant size bins and measuring the density of anchor mappings in each bin that are sampled from the transcript sequence. After these measurements, they employ a heuristic method that identifies bins or combination of bins as valid regions based on several mapping density and gap distance constraints. However, this heuristic region inference method is aimed for locating the local alignment regions of the transcript and is not a global scale region inference for transcripts with structural alterations such as fusions, inversions and rearrangements. Even though the bin merging heuristic can be applied to infer homologous regions in our problem (other than fusion cases that should be located as two separate regions), diagonal gap constraints considered within the region inference heuristic will cause rearrangements or inversions to be missed or split into different local alignments.

To describe our version of the homologous region inference approach, we initially define a comprehensive model for singular/fusion regions and give a general scoring scheme with multiple parameters that can be optimized by users for specific structural variation studies. Afterwards; we describe our formulation and solution method for finding the optimal location and length of the genomic region, given a sample of anchors from the transcript with their multiple mapping locations in the genome.

An anchor is a substring of constant length L_A, of the transcribed sequence T of length N. Given a set of anchor mappings $S_{map} = \{m_1, m_2, \ldots, m_K\}$ of size K, each mapping m_i consists of the following set of information: starting position of the anchor in the transcript, beginning position of the mapping in the genome[37], a positive score of the individual anchor mapping corresponding to the similarity of the anchor sequence in the transcript to its aligned counterpart in the genome[38], and if the genome sequence is composed of multiple chromosome sequences, the chromosome identifier the anchor is mapped to. These values are denoted in their respective order as: $m_i = (m_i.t,\ m_i.g,\ m_i.score,\ m_i.chr)$

In a whole genome alignment setting, we have smaller disjoint anchor mapping sets $S^1_{map}, S^2_{map}, \ldots, S^{\#chr}_{map}$ contained within S_{map} that correspond to the set of mappings for each individual chromosome sequence even though the anchors themselves might be shared in multiple mappings across several chromosomes.

We define an inferred genomic region, $R_j = (R_j.b,\ R_j.e,\ R_j.chr)$, as a pair of positions in the genome that represents the beginning and ending positions of the region and the chromosome sequence that the region belongs to. Even though the number of potential regions to be considered appears to be in $O(N^2)$, most of these regions' scores are overshadowed by more compact regions that border on anchor mapping boundaries. Therefore, we restrict the definition of valid regions R_v, to only have $R_v.b$ indices that coincide with the beginning position of a mapping ($m_p.g$ for some p); and similarly we restrict $R_v.e$ indices to coincide with the ending position of a mapping ($m_r.g + L_A - 1$ for some r). With such a definition of a valid region, the number of regions in the entire genome is constrained by $O(K^2)$.

The genomic region inference problem aims to find the highest scoring region within the set of valid regions, according to the following parametric region scoring scheme:

$$Score(R_j) = \frac{C_N \times M(S_{map}, R_j)^{C\alpha}}{L(R_j) + C_L}$$

$$L(R_j) = R_j.e - R_j.b + 1$$

[37]This corresponds to the aligned position of the first base of the anchor if the alignment direction is forward, or aligned position of the last base of the anchor if the alignment direction is reverse.

[38]Note that, the length of these anchors and mappings are not included due to the fact that all of the anchors sampled from the transcript are of the same length and the mapping scheme does not allow indels within the anchor alignment.

$$M(S_{map}, R_j) = \sum_{k=1}^{N} \max_{\substack{m_i \in S_{map} \\ m_i.t = k \\ m_i \lhd R_j}} (m_i.score)$$

$$m_i \lhd R_j \iff m_i.chr = R_j.chr \land R_j.b \leq m_i.g \land m_i.g + L_A - 1 \leq R_j.e$$

In the definitions above, C_N, and $C_\alpha \geq 1$ are normalization parameters given by the user to adjust the relative importance of the number of mappings within the region with respect to its length. $C_L > 0$ is a length normalization parameter that prevents the bias for very short but densely populated regions that can otherwise dominate the score of sparser and larger regions that cover more anchor samples.

$M(S_{map}, R_j)$ is a score function correlated to the number of anchor samples taken from the transcript, which have a mapping onto the genome within the boundaries of the region R_j. However; instead of returning the number of these anchor samples, it returns the summation of the scores for the best scoring mapping of each anchor sample[39].

Below, we describe our anchor sample selection, mapping and genomic region inference methods used in our alignment software, Dissect.

Initially we sample a number of equally distanced anchors of length L_A (a user defined constant value), such that the first sample starts from the beginning of the transcript and the last sample taken ends at the last base of the transcript. Which value to select as L_A is a problem of sensitivity over speed. As L_A gets larger, less hits will be obtained from the mapping; thus less time will be spent for the downstream genomic region inference method. On the other hand, continuity of more anchor samples will be disrupted by exon-exon junctions, rendering the anchor sample unable to map to the genome as a single sequence. The number of sampled anchors can be determined in two separate schemes: (1) a user defined constant number of anchors are extracted from each transcript sequence or (2) a separate anchor is sampled from every constant number of bases. Thus, in the second scheme the number of anchor samples is correlated to the length of the transcript sequence, N. One of the main practical differences in these two schemes is that; in short sequences, the first

[39]Note that the outer summation iterates from 1 to N as the position in the transcript, yet only the positions that an anchor is sampled from will return proper score values. In order to prevent any inconsistencies, we can assume that a zero-score dummy mapping is created for the positions in the transcript that do not have an anchor sampled from them.

sampling scheme may create redundant samples per exon. This is the same problem that is caused for longer transcript sequences when using the second scheme, in which longer exons are sampled from many places. The second scheme is also a safer method for long transcript sequences that contain relatively short exons that are involved in rearrangements outside of the main span of the transcript; or similarly fusion instances, in which one side is significantly shorter than the other. However, this additional safety comes with significant run-time cost in the genomic region inference method, which is described later in the section. For the sake of simplicity, in the remainder of this section we assume that the first anchor sampling scheme is used.

The construction of the set S_{map} in Dissect is performed using a cache-oblivious short read mapper, mrsFAST [15]. mrsFAST employs a cache-oblivious seed matching algorithm for the optimization of cache usage during read mapping and finds all of the mappings of a given read within a constant number of mismatches but no indels. In Dissect, we allow this constant error rate per sampled anchor also to be a user defined parameter within certain bounds that will ensure a run-time and memory efficient mapping by mrsFAST. After each anchor is mapped to the genome within a constant error threshold, their score and position in the genome are determined by the mapping results and the set S_{map} is constructed together with its chromosomal subsets.

For the genomic region inference step in Dissect, we solve two different problem types: (1) inference of a single highest scoring region and (2) inference of two separate regions and a transcript cut position that will give the highest double region inference score. We define the double region score to be the summation of scores taken from the first and second region with the constraint that; the first region score is only calculated over the anchor samples taken from upstream of the transcript cut position and the second region score is only calculated upon the samples taken from downstream. Below we describe the solution method for these two problems that are used in Dissect and analyse their complexities. We describe the further details of parameter selection and the score comparison between single and double region results in *Implementation* part of this section.

As mentioned above, there are $O(K^2)$ possible regions in the genome that represent an interval between two mappings (including the mappings themselves) that both lie on the same genomic sequence (chromosome). A simple method to solve the single region inference problem would be to calculate the score of each one of these regions in linear time, yielding a $O(K^3)$ solution. We can reduce this complexity to $O(K^2)$ by visiting the regions in a particular order and dynamically reusing the calculated scores from the previous regions.

For following algorithm description, we assume that we are given a sorted list of sampled anchors from the transcript $A_1, \ldots, A_{C_A}$. The given list of anchors is sorted based on their beginning positions in the transcript, $A_i.b$, and thus their ending positions, $A_i.e$, as well. We define the set of mappings for an anchor sample, $S(A_i) = \{m_k \in S_{map} : m_k.t = A_i.b\}$. We also denote the index of the anchor sample of a mapping m_k as $a(m_k)$

- Initialize best scoring region, $BestRegion$, as an empty interval in the genome and the best region score, $BestScore$, as 0.

- For all chromosome sequences k, from 1 to $\#chr$.
  - Sort the mappings in S^k_{map} and obtain the sorted list $(m_1, m_2, \ldots, m_{|S^k_{map}|})$
  - For each $p_s = m_i.g$, i from 1 to $|S^k_{map}|$, representing the beginning position of a region:
    - Initialize $CurrentScore \leftarrow 0$, representing the current score of the visited genomic regions that start at p_s.
    - Initialize a best anchor table B (of size C_A). This table holds the information of whether for each anchor A_x, there is a mapping $m_y \in S(A_x)$ such that it is contained within the currently considered region, R_z ($m_y \lhd R_z$). In addition, the table cell B_x holds the $max(m_y.score)$ information for all such m_y. Initially all of the cells in the table are set to 0, representing the absence of any mappings.
    - For each $p_t = m_j.g + L_A$, j from i to $|S^k_{map}|$, representing the ending position of a region:
      - If $m_j.score > B_{a(m_j)}$, update $B_{a(m_j)}$ and add the difference between the new and old values of $B_{a(m_j)}$ to $CurrentScore$. This current longer region contains a mapping that is superior to the mappings of the same anchor within the previous region.
      - If $CurrentScore > BestScore$, update $BestScore$ and assign $BestRegion$ as an interval in chromosome k that starts at p_s and ends at p_t.

- Return $BestRegion$ and $BestScore$.

In the algorithm above, the values in the best anchor table and $CurrentScore$ are reused for larger regions that contain the previous smaller regions that start at the same position, p_s. If we assume that, the number of chromosomes and number of sampled anchors are constant values; there are two nested loops that iterate through all the mappings which yield a run-time complexity of $O(K^2)$ for the algorithm above. However, if we assume the anchors to be sampled as in the second sampling scheme described earlier (C_A depending on the length of T), the number of anchor samples would be $O(N)$. Since we create a best anchor table for each mapping, the complexity would increase to $O(K(K+N))$ or alternatively to $O(K^2 log(min(N,K)))$ if we do not initialize a static anchor table but maintain a balanced binary tree for only the anchor sample that has mappings within the region.

For the second problem (double region inference), there are $O(K^4)$ possible region pairs in the genome that represent a containment of anchor mappings in the genome for a particular fusion cut in the transcript. A naive method that calculates the score for each region pair and each anchor cut position can solve this problem in $O(K^5)$ time for a constant size anchor sample list. If the number of sampled anchors are dependent on the length of the transcript as in the second anchor sampling scheme, the complexity will be $O(NK^5)$ considering a linear number of possible transcript cut positions. However; since the overall optimization score is based on the summation of two independent region scores, we can apply the following method to reduce the complexity to $O(K^2)$ for the first anchor sampling scheme and to $O(NK^2)$ for the second[40].

- Best double region score, $D_{score} \leftarrow 0$.
- For each $i \in [1, C_A - 1]$
  - $P \leftarrow A_i.e$ and $S^-_{map} = \bigcup_{1 \le k \le i} S(A_k)$ and $S^+_{map} = \bigcup_{i < k \le C_A} S(A_k)$
  - $D^-_{score} \leftarrow$ score obtained by solving the single region inference problem for S^-_{map}.
  - $D^+_{score} \leftarrow$ score obtained by solving the single region inference problem for S^+_{map}.
  - If $D_{score} < D^-_{score} + D^+_{score}$

[40]Note that in these complexity calculations, we assume N and K to be independent variables due to the fact that there is no direct dependency relation in between; whereas if we assume a constant value for the number of mappings per anchor sample, a change in the number of samples would also affect the number of mappings. In practice, it would be reasonable to assume that the number of mappings and number of samples are linearly correlated. Therefore for relatively long transcript sequences, switching from the first anchor sampling scheme to the second is likely to cause a slowdown in the order of $O(N^3)$ for this problem.

    * Replace D_{score}, and set D^{-}_{reg} and D^{+}_{reg} as the region intervals returned from the single region inference calls.

- Return the best double region pair as D^{-}_{reg} and D^{+}_{reg}.

In the algorithm above; there is a loop over the indices of anchors and in each iteration, the algorithm makes two single region inference calls that are solved within $O(K^2)$ time, yielding a total run-time complexity of $O(C_A K^2)$. If C_A is a constant value as in the first anchor sampling scheme, the run-time complexity will be $O(K^2)$, whereas in the second sampling scheme in which $C_A = \Theta(N)$, the complexity will be $O(NK^2)$.

Even though, in the two algorithms described above the outputs contain only a single region or a single region pair; in order to allow lower scoring alternatives to be considered for the alignment, Dissect returns a constant number of different region/region pair results determined by a user determined parameter. Since the size of this resulting list is constant, an additional constant number of comparisons and updates upon the list do not affect the overall complexity of the algorithms described above. Further details of constructing these result lists are described in the *Implementation* subsection.

After the inference of a single region or a pair of regions using the algorithms described above, Dissect employs its homology search and fragment chaining methods upon these as described in the next subsection.

2.3.2 Step 2: Homology search and fragment chaining

The second step of Dissect, consists of two separate problems: (1) searching for homologous fragments between the transcript and the inferred genomic region in order to construct a fragment set and (2) finding the optimal fragment chaining within this given fragment set.

In order to find homologous regions between the transcript and the inferred genomic sub-region, we developed a modified version of mrsFAST aligner, mrsFAST-HS, which samples short constant size seeds from every few bases in the transcript and maps them to the genomic region using the cache-oblivious mapping method of mrsFAST. The implementation details of this extension and parameter selections are described in the *Implementation* subsection. However, the main difference between mrsFAST-HS and the original mrsFAST is the unbounded seed extension step, in which the seed mappings are extended towards both sides as long as the number of mismatches intercepted are lower than a constant threshold (and some other error rate constraints are satisfied, which are described in the *Implementation* subsection).

In short, given a transcript sequence T and a genomic sequence G; the fragment set F_{set} returned from mrsFAST-HS contains a comprehensive set of alignment fragments of different sizes and similarity scores between T and G, in both forward and reverse directions. In the case of a region pair, G_1 and G_2, instead of a single region, the same mapping is applied to both of these sequences, returning separate fragment sets F^1_{set} and F^2_{set} respectively.

After the fragment sets are obtained, we employ the overlapping fragment chaining algorithm based on the problem solutions described in *subsection 2.2.2*. If there is only a single genomic region, fragment chaining is applied to only that region, whereas in the case of two fusion regions, the chaining is applied considering the possibility of chaining from a fragment in F^1_{set} to F^2_{set} but not the other way around.

In *subsection 2.2.2* that defines separate formulations for the overlapping chaining problem; we addressed concerns upon overlaps between fragments and mismatch resolution schemes for determining the optimal split position within the interval. In Dissect, due to the heavy cost of evaluating optimal overlap split position for each transition (even though this cost does not affect the complexity of the algorithm); we only consider overlap resolution based on overlapping interval length without considering the optimal split position with respect to mismatches. However, a more advanced overlap resolution scheme is applied in *Step 3* as a post-processing step for the best scoring fragment chain.

The result of this step is a chain of fragments that represents the tentative exon structure of the transcript with potential structural alterations. In the next step, we describe Dissect's post-processing methods for alleviating small-scale misalignments that may be caused because of the minimum length and error rate thresholds in the fragment set construction step.

2.3.3 Step 3: Post-processing of fragment chains

In the post-processing step, Dissect applies modifications to some of the fragment pairs (adjacent fragments in the chain) in the final fragment chain that might potentially contain some form of minor misalignment. In order to fix these, Dissect employs several different procedures that can handle cases such as: (1) fragments overlapping in the transcript or (2) having short gaps in between, as well as handling situations such as (3) very short overlaps of two adjacent fragments in the genome and (4) fragments that are separated by an indel or a mismatch in their alignment.

Among the cases noted above, case (4) covers the instances of fragment pairs that are separated by one mismatching base in the transcript and in the genome. Or an alternative situation is that, the fragments are adjacent in the transcript but contain a single base gap in between in the genome or vice versa. Such situations can occur due to the error threshold of mrsFAST-HS described in the previous section and its inability to extend over indels. Even though the resulting fragment chain is not wrong in the case of indels (each fragment was initially assumed to represent homologous sequences that contain few mismatches and no indels); it is preferable to report these pair of fragments as a single fragment that contain an indel or a mismatch. The merging operation can be performed in constant time, allowing us to fix all such instances in a linear scan of each fragment pair.

Moreover, the case (3) noted above includes the fragment pairs in the list that are adjacent in the transcript, yet have a very short overlap in the genome not longer than several base pairs[41]. In a structural variation study, such short cases would usually be uninteresting due to the fact that they do not represent statistically significant aberrations from a wild-type transcript alignment. We encounter such instances frequently in cases that there is a single or a couple base pair insertion to the transcript sequence which at the same time shows similarity to the adjacent bases in the transcript. Removal procedure of such short overlaps is straightforward and performed as a linear scan of the fragment pairs in the chain. The finalized fragment pair becomes adjacent in the genome but contains a gap of the same length in the transcript, not implying any structural alterations.

Even though the cases described above does not require detailed computational methods, handling cases (1) and (2) will require some algorithmic perspective. These cases are described in detail below.

Refining overlapping fragments :

Case (1) mentioned above covers the fragment pairs that are disjoint in the genome but are overlapping in the transcript. Such cases can occur due to the appearance of an overlapping sequence twice in the genome, one as the trailing sequence of the first fragment in the genome and the other as the leading sequence of the second fragment. Such a fragment formation might indicate a duplication in the given genomic sequence, yet Dissect is aimed

[41]We allow this threshold to be user defined within a realistic range of values.

to find structural alterations on the transcript side, and such duplications on the genome side are not reported. Instead, Dissect employs an optimal junction finding procedure upon such overlapping fragment pairs as follows:

Given two overlapping fragments F_1, F_2, the length of their overlapping sequence is $L_O = F_1.ts + F_1.len - F_2.ts$. Dissect assumes that the optimal resolution of this fragment pair is another fragment pair F_1', F_2' such that; F_1' is contained within F_1 as a prefix fragment and F_2' as a suffix fragment of F_2, and the following transition scoring function is optimized[42]:

$$TScore(F_1', F_2') = F_1'.score + F_2'.score + FJ_s(F_1', F_2')$$

FJ_s function definition is similar to the S_j function defined in subsection 2.1.2, which returns $C_{ss}/2$ if there is a canonical splice beginning signal after the ending of F_1' in the genome (taking the direction of the fragment into account), and $C_{ss}/2$ if there is a canonical splice ending signal prior to the beginning of F_2' in the genome, or C_{ss} if both conditions are satisfied.

According to the fragment score defininition given in (2.11), we can determine the scores of F_1' and F_2' as:

$$F_1' = F_1.score - C_m * (F_1.len - F_1'.len) + C_{mm} * \#\text{mismatches in } F_1[(F_1'.len + 1)..(F_1.len)]$$

$$F_2' = F_2.score - C_m * (F_2.len - F_2'.len) + C_{mm} * \#\text{mismatches in } F_2[1..(F_2.len - F_2'.len)]$$

C_m representing the C_{match} and C_{mm} representing the $C_{mismatch}$ used in the previous sections.

It is clear from these equalities that; if we would like to maximize $F_1'.score + F_2'.score$, we should search for the cut position that maximizes the number mismatches in the discarded regions from both sequences. Or we can alternatively describe as the cut position that minimizes the number of mismatches retained by F_1' and F_2' within the overlapping region. Therefore, in order to find the optimum cut position from the fragment score perspective, it is enough to calculate the number of mismatches retained on both sides. However; in order to maximize $TScore(F_1', F_2')$, we should also take FJ_s into account. Since even in the most general splice signal scoring scheme, calculating FJ_s for a pair of fragments will be done in constant time, we can perform a linear scan over all cut positions across the overlapping

[42]A valid F_1', F_2' pair satisfies all of the following equalities: $F_1'.ts + F_1'.len = F_2'.ts$, $F_1'.ts = F_1.ts$, $F_1'.len \leq F_1.ts$, $F_2'.ts + F_2'.len = F_2.ts + F_2.len$, and $F_2'.ts \geq F_2.ts$.

region and calculate FJ_s for each pair on the go. A simple method for counting number of mismatches for each cut position while holding two counters for number of intercepted mismatches on both sides would be as follows:

- $BestCutScore \leftarrow -\infty$ and $BestCutPosition \leftarrow \emptyset$
- Number of retained mismatches of F'_1 before the cut within the overlap interval, $mm^{(-)} \leftarrow 0$
- Count the number of mismatches in F_2 within the overlap interval and assign to $mm^{(+)}$, which represents the retained number of mismatches in F'_2 after the cut within the overlap interval.
- For each cut position, i from 0 to L_O:
  - If $i = 0$, calculate the $TScore(F'_1, F'_2)$ with the current $mm^{(-)}$ an $mm^{(+)}$ values and calculate FJ_s for $F'_1 = F_1[1..(F_1.len - F_o)]$ and $F'_2 = F_2$. Store $BestCutScore$ and update $BestCutPosition$ as 0.
  - If $i > 0$
    - If $F_1[F_1.len + i]$ corresponds to a mismatch, then increase $mm^{(-)}$ by one.
    - If $F_2[i]$ corresponds to a mismatch, then decrease $mm^{(+)}$ by one.
    - Calculate $TScore(F'_1, F'_2)$ for $F'_1 = F_1[1..(F_1.len - F_o + i)]$ and $F'_2 = F_2[(i + 1)..(F_2.len)]$, and the retained number of mismatches in F'_1 and F'_2 within the overlap region ($mm^{(-)}$ and $mm^{(+)}$ respectively)
    - If the new score at the current cut position is higher than $BestCutScore$, replace it with the new best score and update $BestCutPosition$ as i.
- Return $F'_1 = F_1[1..(F_1.len - F_o + BestCutPosition)]$ and $F'_2 = F_2[(BestCutPosition + 1)..(F_2.len)]$ as the optimal overlap resolution.

The solution method described above iterates over a loop of size L_O for each fragment pair, which is bounded by $O(N)$. However; when an overlapping fragment pair is resolved, all consequent resolutions are performed over the updated fragments from the earlier overlap resolutions. For this reason, number of transcript positions iterated are bounded by $O(N)$ for the entire fragment chain (in contrast to a single fragment pair), yielding an overlap resolution complexity of $O(N)$ for the entire chain.

Refining gap boundaries :

A more advanced post-processing step in Dissect is for the case (2) mentioned above, which covers the fragment pairs that are apart from each other in the transcript sequence by a relatively short gap. Such a gap may be caused by a novel insertion in the transcript or alternatively it may be caused by an insufficient extension of the adjacent blocks (due to the possible existence of indels and/or frequent mismatches within the gap interval). A mismatch based scan across the region as in case (1), would not help in this particular case due to the possibility of intercepting indels.

For these cases, Dissect employs a gap refinement post-processing step based on a joint two sided pairwise alignment method extending neighbouring fragments towards the center of the gap region. This method used in Dissect is an adapted version of the "sandwich dynamic programming" described in [46]. The mathematical definition of the problem is as follows:

We are given a gap interval in the transcript, $I_G = T[p_b..p_e]$ located between a fragment pair F_1 and F_2. The trailing flanking sequence of F_1 in the genome is denoted as R_1 and the leading flanking sequence of fragment F_2 in the genome is denoted as R_2. The length of these flanking sequences will be described shortly, but currently assume that they are long enough to fully align I_G to them.

Given I_G, we would like to find p_1, p_2, ℓ_1, ℓ_2 values with given constraints that $p_b - 1 \leq p_1 < p_2 \leq p_e + 1$, $\ell_1 \geq 0$, and $\ell_2 \geq 0$; that maximizes the summation of Needleman-Wunsch global pairwise alignment [36] scores of $T[p_b..p_1]$ to the prefix of R_1 of length ℓ_1 and $T[p_2..p_e]$ to the suffix of R_2 of length ℓ_2.

The alignment score function for global pairwise alignment problem is not re-defined here; but for the problem solution given below, it would suffice to note that match scores, mismatch and indel penalties are all constant values that are symmetric on the transcript and the genome sides. In the solution below, we denote these constants respectively as C_m, C_{mm} and C_{in}.

This problem formulation covers the cases in which there is no extension from the F_1 side ($p_1 = p_b - 1$) and also the case in which there is no extension from the F_2 side ($p_1 = p_e + 1$). Therefore, different from the overlap resolution method described earlier, the gap resolution scheme allows the preservation of novel insertions in the transcript when there is no proper extension from either side or when the extensions do not meet within the gap region.

For our version of this two-sided extension alignment scheme, we create two alignment score matrices M_1 and M_2; one for aligning I_G with R_1 and the other for aligning I_G with R_2 in the reverse direction. The dimensions of both of these matrices are $(L_G + 1) \times (K_G + 1)$, with an additional 0^{th} row and column in M_1 and an additional $(L_G+1)^{th}$ row and $(K_G+1)^{th}$ column in M_2 for alignment initialization purposes.

K_G value above, is defined as the maximum sequence length that I_G can be aligned with a non-negative alignment score. This length can be calculated as $L_G + \lfloor \frac{C_m * L_G}{C_{in}} \rfloor$. We also denote K_1 as the prefix of R_1 of length K_G and K_2 as the suffix of R_2 of length K_G.

Each cell $M_1[x, y]$ represents the alignment of x^{th} base of I_G with y^{th} base of K_1, similarly $M_2[x, y]$ represents the alignment of x^{th} base of I_G with the y^{th} base of K_2. However, the score value representations are different between the two matrices: $M_1[x, y].score$ indicates score of the best scoring alignment of $I_G[1..x]$ with $K_1[1..y]$, whereas $M_2[x, y].score$ represents the score of the best scoring alignment of $I_G[x..L_G]$ with $K_2[y..K_G]$.

The gap-refinement algorithm employed in Dissect is as follows:

- For each column j, from 1 to K_G: $M_1[0, j] \leftarrow (-C_{in} * j)$ and $M_1[j, 0] \leftarrow (-C_{in} * j)$.

- Apply global pairwise alignment dynamic programming algorithm for filling in the values of each $M_1[i, j].score$ and $M_1[i, j].parent$ using C_m, C_{mm} and C_{in} for match, mismatch and indels respectively.

- For each column j, from 1 to K_G: $M_2[(L_G + 1), (K_G + 1 - j)] \leftarrow (-C_{in} * j)$ and $M_2[(L_G + 1 - j), (K_G + 1)] \leftarrow (-C_{in} * j)$.

- Apply global pairwise alignment dynamic programming algorithm in the reverse direction for filling in the cell values of M_2. The algorithm should be applied for each row i, from L_G to 1 and within each row for each column j, from K_G to 1. When determining the best score of $M_2[i, j]$, the match/mismatch transition score is calculated from $M_2[i + 1, j + 1]$ and indel transition scores are calculated from $M_2[i + 1, j]$ and $M[i, j + 1]$.

- At this step, in order to make an easier comparison between the two tables; we create arrays MR_1 and MR_2 that represent the maximum cells in the rows of both arrays. $MR_1[i]$ (for i from 0 to L_G) represents the maximum scoring cell in the i^{th} row of M_1, and similarly $MR_2[i]$ (for i from 1 to $L_G + 1$) represents the maximum scoring cell in the i^{th} row of M_2.

- Furthermore, we derive two more arrays from MR_1 and MR_2, forming BR_1 and BR_2. $BR_1[i]$ represents the best scoring cell within the interval $MR_1[0..i]$ and similarly $BR_2[i]$ represents the best scoring cell within the interval $MR_2[i..(L_G + 1)]$. A key point here is to select the cell with the lowest index value i in tie situations for BR_1 and select the cell with the highest index value i for equal scoring cells in BR_2[43].

- For obtaining the highest scoring combination of alignments, it would suffice to do a linear scan of BR_1 and BR_2 together in order to find the index $i \in [0, L_G]$ that maximizes $BR_1[i] + BR_2[i + 1]$

- After the optimum i and corresponding $MR_1[q]$ and $MR_2[r]$ cells are found. The indices of the cell $M_1[x_1, y_1]$ that $MR_1[q]$ is pointing towards, form p_1 and l_1 values, whereas the indices of the cell $M_2[x_2, y_2]$ that $MR_2[r]$ is pointing towards, form $p_2 = x_2$ and $l_2 = K_G - y_2 + 1$.

According to the returned p_1, l_1, p_2, and l_2 values, Dissect updates the fragments F_1 and F_2 adjacent to the gap region.

The final post-processing step in Dissect is the identification of different types of structural alterations existing in the finalized fragment chain. Since the methods used for this classification step are fairly straightforward, they are described in the following *Implementation* subsection together with description of structural alteration labels.

2.3.4 Implementation of Dissect

Earlier in this section, we described the three algorithmic steps that Dissect employs for an efficient whole genome alignment of a given transcript set that might contain structural variants. All of these stages described are implemented as decoupled modules in Dissect that can be run individually, and each stage generates an output that will be used as the input for the following stages. Since each of these sub-procedures are specialized in solving their corresponding problems, Dissect has an external main body that employs these runs and handles the input/output for the other steps[44].

[43]The reasoning behind this tie breaking scheme is to favor novel insertion cases over extension of blocks with low similarity. For example, if there is an extension to the center of the gap region with the alignment score 0, the algorithm will return $p_1 = 0$, $p_2 = K_G + 1$ representing an entire novel insertion in the gap region instead of a very low scoring extension reaching the center of the gap interval.

[44]Since Dissect's main program handles situations other than the algorithmic concerns noted in the previous section (such as indexing of the genome sequence or pre-sorting anchor mappings), the numbering for the

For the Dissect program to be executed, the following input files are required: (1) A genome file in FASTA format as a single file or multiple files for each chromosome. (2) A set of transcripts in FASTA format to be aligned to the genome. (3) A list of configuration parameters.

As the first step, Dissect reads the configuration file for identifying the following parameters (the pipeline stages in which each parameter is used is given in parentheses){and default values are given in curly brackets}:

- **Anchor index size**: (Stage 0){12} The seed size to index the input genome for mapping anchors.
- **#Sampled anchors**: (Stage 1){20} The number of anchors to be sampled from each transcript sequence.
- $\mathbf{L_A}$: (Stage1){24} Length of each anchor in the region inference step.
- **Anchor error rate**: (Stage 1 and 2){1} Number of substitution errors that can exist in an anchor alignment to the genome . This number should be less than or equal to (L_A/anchor index size - 1) for ensuring %100 detection of all mappings within this error rate constraint.
- **Max anchor hits**: (Stage 2 and 3){150} The maximum number of anchor mapping hits to be returned from the alignment of a single anchor. The reasoning behind this limit is described in *Stage 2*.
- $\mathbf{C_N}, \mathbf{C_L}, \mathbf{C}_\alpha$: (Stage 4){1,3,400000} Normalization parameters for the inferred region fitness function. These regulate the relative importance of the number of mappings within the region with respect to the region length.
- $\mathbf{C_R}$: (Stage 4){3} Number of regions that are maintained and returned in the highest scoring region list.
- **Region merging distance**: (Stage 4){400000} Highest distance threshold for merging two regions into one. This can either be applied in order to merge two fusion regions into one or merge two single regions in the high scoring list into one.

stages in Dissect can be different than the steps described in the previous section.

- **Marginal region score ratio**: (Stage 4){%90} The ratio of a region to the highest scoring region, which a region needs to maintain in order to get into or stay in the highest scoring list.
- **Transcript seed size**: (Stage 5){7} This is the size of the samples that are taken from the transcript in order to be mapped to the inferred genomic region and extended.
- **Transcript extension constraints** (Stage 5){%90,3} These two values represent the constraints that are enforced on the fragment extension scheme. They indicate the minimum similarity constraint and maximum consecutive errors constraints respectively.
- **Minimum fragment size**: (Stage 5){10} This threshold determines the minimum sized fragment that will be considered within the fragment set construction step.
- **Maximum fragment set size**: (Stage 6){10000} This is the maximum number of fragments to be kept in the final fragment set.
- **Transcript gap limit**: (Stage 6.2){150} This is an extra constraint that is used for a speed-up by pruning branches that contain longer insertions in the transcript than this value. This is not a required constraint; if such a limit is not desired, it can be set as a very large value.
- $\mathbf{C_n^1, C_n^2, C_b^1, C_b^2, C_i^1, C_i^2, C_f}$: (Stage 6.2){0, 1, 5, 2, 10, 1, 50} Penalty function parameters for wild-type, rearranged/duplicated, inverted and fusion transcripts.
- **Marginal alignment score ratio**: (Stage 6.5){%50} This value represents a marginal score boundary similar to *Marginal region score ratio*, but it is used for eliminating low scoring fragment chains from the output.

Each of the following stages in Dissect program are employed for all of the transcripts in the sequence at once (apart from Stages from 6.1 to 6.5, which are always done together without intermediate files generated in between). Each stage requires processing all transcripts from a certain aspect to produce the input for the next stage. Therefore; these stages are decoupled and can be employed in pieces instead of a single run, or one stage can be employed with different parameters without requiring the previous stages to be redone.

Stage 0: Indexing the genome

This is the step for generating the index file required for mapping anchor samples to the genome using mrsFAST[15]. Index file generated basically represents a hash-table of k-mers found in the genome, k being defined by the user parameter *anchor index size*.

An important difference of Stage 0 from Stages 1-6 is that, this step needs only to be run once for a given genome file. For each subsequent experiment that uses the same genome file with the same anchor index size, this step can be skipped. This is very useful due to the slow nature of the indexing step. One of the options in the user defined configuration file determines whether to skip Stage 0 or not.

Stage 1: Sampling anchors from transcript

This stage is the step in which Dissect samples the anchors from all transcripts in order to map them onto the genome as a batch short-read query set. L_A and *#sample anchors* are the user defined parameters in this step that determine the anchor sampling length and density. The first sample is taken from the beginning of the transcript and the last sample is taken from the end. All of the remaining anchors are sampled with equal distance from the beginning/end anchors and from each other.

In this stage, we also filter low quality anchors from the printed list. Low quality anchors are defined as strings that have Hamming distance of less than or equal to *anchor error rate*, from poly-A/poly-T strings or strings in the form $(C)GCGCGC\ldots$, $(C)ACACAC\ldots$, etc.

Stage 2: Mapping anchors to the genome

In this stage, Dissect maps the sampled anchors to the genome within the user defined constraints *anchor error rate* and *max anchor hits*. The purpose of using these parameters is mentioned in the configuration list above, however the main reasoning behind using a maximum threshold for the number of mappings for a single anchor is described below.

Due to the long repetitive regions in the genome, some anchor samples can map to significantly more places in the genome than others. Incorporating these large number of mappings in the region inference step will not help accuracy and will slow down the inference process, therefore mappings by these anchors should not be considered when searching for the optimal region. However, there is not an accurate way of determining if an anchor will be mapped to many places in the genome before actually performing the mapping (apart from the low quality mappings eliminated in the previous stage). Therefore, Dissect utilizes this maximum mapping threshold as an identifier that there might be more mappings of the same anchor if the mapping process is continued . For example, if this value is selected as 101; after all anchors are mapped, any anchor that contains 101 mappings will be removed. The remaining set of anchors with certainty will contain less than or equal to 100 mappings.

At the end of this stage the anchor mapping output is returned in SAM format [27]. This file is not ordered based on the order of queried anchors, but based on order of mappings in the hash table and order of chromosome file names in the genome. Therefore, a sorting step is required before this file can be used as an input to the region inference stage.

Stage 3: Sorting anchor mappings

Since the total number of the anchor mappings can be large, the following region inference stage cannot store all mappings in the memory. In addition, the generated file size is generally too large to seek back and forth in order to obtain all mappings of individual transcripts.

For this reason, Dissect sorts the anchor mappings in their respective order in the transcript dataset. However, since the file size can be very large (e.g. $> 15GB$ for a whole transcriptome assembly experiment of around 570000 contigs), any $O(nlogn)$ sorting algorithms (such as quicksort[17] or mergesort[25]) will be costly for this step in terms of run-time or disk/network bandwidth usage for data transfer.

In order to tackle this challenge; Dissect utilizes an iterative version of the counting sort solution [25], using the fact that it does not need to sort the given inputs according to the order of anchor samples but only in the order of transcript sequences in the original list.

As a first step, Dissect allocates an adequate space in the memory that can handle X of the Y mappings. In the following $\frac{Y}{X}$ iterations; in i^{th} iteration, Dissect loads the anchors corresponding to the transcripts from $X * i$ to $X * (i + 1) - 1$, to their allocated position in the memory and prints them in their sorted order. This method requires $\frac{Y}{X}$ scans of the mapping file and linear time to sort each batch of anchor mappings.

Apart from sorting, the anchor elimination step described in the previous stage is actually performed during the counting sort step while the anchors are loaded to the memory.

Stage 4: Region inference

Given the anchor mapping for each transcript sequence, the region inference stage aims to solve the problem of finding the highest scoring C_R regions defined in *Step 1* earlier. However, there are some detailed aspects of the region inference stage that are not described in the algorithmic solution in *Step 1*.

One important point is about how the highest scoring region list of constant size C_R is maintained. At any point, this list contains at most C_R elements, in which all elements have a region score that is within the marginal score boundary with respect the highest scoring region. Therefore; if the highest region score within the list is S_c, then all regions within the list should also have a score higher than or equal to *Marginal region score ratio*$\times S_c$.

At every step of the region inference algorithm, the current region score is compared to the highest scoring list in order to check whether the region should be added to the list or not. As a first step, the current region score is compared to the highest scoring region in the list; if it is not within the marginal score boundary, it is disregarded. If it is within the marginal score boundary and there is an empty position in the list, current region is automatically added to the list. But if it is within the boundary but not higher than any of the regions within the list and there are no empty spaces in the list, it will be eliminated. If it is within the boundary and has a higher score than some other regions within the list, it will replace the lowest scoring region within the list. Finally, if the current region score is higher than the highest scoring region in the list, then the current region is automatically added to the list and all of the other regions in the list that are not within the current region's marginal score boundary are eliminated; if all are, then the lowest scoring region is eliminated if there is no empty space.

Since the highest scoring region list maintenance steps described above take at most $O(C_R)$ steps, they can be performed in constant time and can be integrated to the algorithm described in *Step 1* without affecting the original time complexity.

After C_R, or less number of, highest scoring regions are determined, there are additional post-processing steps applied to the list of regions.

Firstly, for any double (fusion) region in the list, a merge check is performed. If the distance between two fused regions inferred are shorter than *Region merging distance*, they are merged together with the intermediary region and reported as a single region. Furthermore, if there are regions in the highest scoring list that overlap or are within the *Region* *merging distance* in the genome, they are merged together as well (two fusion regions are not merged unless both sides representing the cuts on the same side coincide with each other). Resulting set is reported as the set of inferred regions. The number of regions reported here is denoted as N_R, to be referred to in the following stages.

Furthermore, another point that is not mentioned in the algorithm description in *Step 1* is the score comparison between fusion regions and single regions. Since a successful single region inference can be considered as a double region that consists of adjacent fused regions and the single region will always score higher than the fusion region in this case for $C_\alpha > 1$; fusion regions are only searched in the absence of a good quality single region. A good quality region is defined by a user defined parameter and is not mentioned above due to its elaborate purpose, and its default value is %87 representing the ratio of number of anchors that have a mapping within the region, to the total number of anchors that are mapped for the transcript and not eliminated in the previous stages due to excessive number of mappings. If a good quality region is not found within the highest scoring single region list, fusion region search is employed. A good quality region is also defined by the same score value, but considering both regions. If there is no good quality region for a double region selection either, no output is reported. If there are, they are reported as the inferred regions.

Stage 5: Mapping and extending fragments

Mapping and extending fragments are done by the homology search tool mrsFAST-HS modified from mrsFAST. According to the given parameter *transcript seed size*, γ, an index is created for the inferred genomic region. Afterwards, a seed of size γ is sampled from every $\lceil \frac{\gamma}{2} \rceil^{th}$ location in the transcript and mapped onto the genomic region. Instead of reporting mapping seed outputs directly, mrsFAST-HS applies the following extension and duplicate removal scheme.

According to *minimum similarity constraint* and *maximum consecutive error constraint*, the mapped seed is extended to both sides until either the first or second constraint fails.

After a seed is fully extended, it is verified if it has been reported earlier; depending on this, it is not printed twice in the maximally extended fragments list.

At this stage, Dissect also applies a minimum length threshold to the maximally extended fragments. If an extended fragment is shorter than *minimum fragment size* parameter, then it is not reported.

The reason that *minimum fragment size* is not used as *transcript seed size* is due to the fact that the inferred genomic region is shorter than the larger sequences mrsFAST is originally optimized to index (whole genomes). For this reason, we choose a smaller seed size (index size for mapping) than needed in order to keep the hash table used in the mapping step smaller.

Stage 6.1: Constructing fragment set

This is a precautionary step in Dissect, which prevents the execution of fragment chaining procedure for transcripts that have an overwhelmingly large amount of alignment fragments. This cut-off is determined by the user defined parameter *Maximum fragment set size.*

Since this parameter directly affects the memory and run-time cost of the chaining stage, it should be carefully chosen taking the memory size of the machine and the time to be spent per transcript sequence into account.

In this stage, Dissect iteratively removes smaller fragments in the maximally extended fragment list until the total number of fragments in the final fragment set is less than or equal to *Maximum fragment set size* parameter.

Stage 6.2: Fragment chaining

This step employs the chaining algorithm described in *Step 1.*

One point that is not mentioned earlier is the maximum transcript gap size considered when applying the chaining method.

Depending on the *transcript gap limit* parameter given, Dissect only considers chaining fragments only within this transcript distance limit.

Transition penalties for each fragment pair is calculated according to the C_n^1,C_n^2,C_b^1,C_b^2, C_i^1,C_i^2,C_f parameters given. Among these C_n^1, C_n^2 represent constant addend and the logarithmic coefficient respectively for wild-type transition pairs (towards downstream and on the same sequence). Similarly, C_b^1, C_b^2 represent the addend and coefficient for log-scale transition penalty function of rearranged/duplicated fragments (towards upstream and on the same sequence). Moreover, C_i^1, C_i^2 represents the same values for the transition penalty function of inverted fragments (between reverse complemented sequences). Finally C_f parameter determines the constant fusion transition penalty between fused genomic sequences.

Stage 6.3: Post-refining fragment boundaries

In this stage, Dissect employs its post-processing methods upon the highest scoring chain for refining potential small-scale misalignments at the transition boundaries of fragment pairs. These methods were described in detail in *Step 3*.

One point that is not mentioned earlier is reporting ambiguous transitions for overlap resolution. As described earlier in *Step 3*, the overlap resolution scheme with mismatches is iteratively applied to the fragment pairs in the highest scoring chain. During the overlap resolution process for a fragment pair; if there are multiple split positions that score equally well, Dissect selects one of them arbitrarily for the main result and reports the existence of the other equally scoring split positions in a special data field in the alignment output. Using this field, one can obtain all possible splits that Dissect might have reported as the main alignment result. These instances occur frequently when there are no mismatches and no canonical splice sites within the overlap interval of an overlapping fragment pair.

Stage 6.4: Classifying alignments into events

In this stage, Dissect identifies the structural event labels each alignment result belongs to in a straight forward manner. A single transcript can be identified as containing different structural alterations since it can be aligned to N_R different regions in the genome.

If the alignment contains two fragments belonging to separate fusion sequences, it is identified as a fusion transcript. Thereafter, a more detailed analysis can be made in order to verify if either side of the fused sequences contain further structural alterations.

If the alignment contains two fragments that belong to reverse complement sequences with respect to each other, it is identified as containing an inversion. It can further be analysed in order to determine if it is an inverted duplication, inverted rearrangement, or an in-place inversion of an exon or exon group.

If the alignment contains a back-jump between two fragment pairs in the resulting chain, this is identified as containing either a forward (non-inverted) duplication or a forward rearrangement. In order to differentiate between these two cases, Dissect checks whether the back-jump results in some fragments sharing at least several bases together. If it does, then the alignment is identified as containing a duplication event, otherwise a rearrangement event.

If some of the alignments that are identified as duplications have a duplication interval shorter than 5 bases, it is more likely to be a result of an insertion (exhibiting some homology to the flanking sequences) rather than a very short duplication caused by a genomic translocation or a novel splicing mechanism. However, since these cases can still be of interest, they are identified as a separate "ambiguous short insertion/duplication" label.

All of the remaining alignments that do not contain any of the structural events described above are denoted as "non-event" alignments. Furthermore, any transcript that does not contain any valid inferred regions or valid alignments is collected under the "no output" label.

Stage 6.5: Reporting alignment output

In this stage, Dissect employs a final filtering procedure among reported fragment chains for the same transcript (but for different inferred regions). Given *Marginal alignment score ratio* parameter, Dissect removes low scoring chains according to their score ratio to the highest scoring chain for the transcript. Furthermore, this is the stage Dissect prints the output for the alignment results.

Chapter 3

Experiments

In this chapter we report the alignment quality and structural event discovery accuracy results of our transcriptome to genome alignment tool, Dissect. Reported results include experiments upon simulation datasets derived from *RefSeq* transcript and *Known Gene* gene structure databases, which are subjected to nucleotide level substitution/indel noise at different frequencies, novel oligonucleotide sequence insertions and structural alterations at different size distributions; such as exon duplications, inversions, rearrangements and transcript-transcript fusions.

Apart from simulation, as a demonstration of our aligner's transcriptional novelty detection performance in real-life transcript databases, we aligned a whole transcriptome set of assembled 50bp RNA-seq reads sequenced from a human prostate cancer individual, assembled by de novo transcriptome assembler Trans-Abyss [6][37].

Due to the fact that there is no efficient way to accurately determine the positive predictive value of the structural events detected by our algorithm; for the analysis of experimental results, we consider a wild-type transcriptome to genome aligner, BLAT [21], as a surrogate for false positive event identification.

Furthermore, since the experiments performed upon real-life datasets also include the alignments of mis-assembled transcripts, the reported results cannot be assumed as a direct indication of structural alterations, but should go through the process of validating the assembled sequence before biological implications can be analysed. Conversely, comparison of validated results from other studies upon the prostate cancer RNA-seq dataset used, will indicate the combined accuracy of the assembly and alignment together. Since such validated datasets are not abundant, they will not serve as a large scale accuracy test; therefore such comparisons with validated transcriptional variations are omitted in this chapter.

3.1 Experiments upon Wild-type Transcripts with Novel Insertions and Nucleotide-level Alterations

In this set of experiments, we aim to evaluate the performance of Dissect in wild-type transcript datasets and several variant datasets altered with different types and levels of noise.

For the first experiment, we used the latest NCBI RefSeq mRNA annotation dataset for whole mouse transcriptome. We acquired this database from the latest build[1] in UCSC Genome Browser FTP server. Assuming that this annotation dataset is composed of wild-type transcripts that do not contain structural alterations, we evaluated Dissect's false event discovery rate by aligning these sequences to the mm9 build mouse reference genome. Since most of these sequences have very close matches to genes in human genome, we also used this dataset to evaluate the accuracy of Dissect alignments at a nucleotide-level resolution.

For the second experiment, we modified the original RefSeq sequences by adding nucleotide-level substitution/indel errors. These are performed uniformly at random for each base in a transcript and applied at different density levels in order to measure the increase in false event detection rate as the sequences diverge more from the original transcripts.

For the last experiment in this section, we used the latest RefSeq mRNA annotation dataset for whole human transcriptome. We acquired this database also from its most recent build in the UCSC Genome Browser FTP server[2]. The aim of this experiment is to evaluate Dissect's false event discovery rate in the existence of short-to-medium size novel insertions.

In order to simulate a realistic sample of novel human genome insertions, we sampled substrings of different sizes from the set of insertion sequences from a novel insertion characterization study by Kidd et al. [22], and inserted them to the transcript sequences at random exon breakpoints[3].

[1]as of the week of July, 18th.

[2]This dataset is also acquired on same week as the mouse build.

[3]We restrict the novel insertion simulations only to exon breakpoints due to the fact that, in a large set of transcripts containing mid-exon insertions, there frequently are cases in which the insertion displays a local similarity to the flanking sequences that would result in over extension of the maximal fragments of the adjacent exons. Fragment chaining algorithm of these over-extended fragments will result in short

Alignment results without noise

For the RefSeq mouse mRNA dataset; as a pre-processing step, we removed any sequences that contain base values that are different from A,C,G,T (such as n,y,w,r). We further removed any leading or trailing exact poly-A/poly-T sites that were longer than 5bp[4].

Remaining dataset after the pre-processing step contained 28058 transcript sequences of an average length of 2848 bases.

We aligned the full set of transcript sequences to the entire mm9 build mouse reference genome including 35 sequences; $chr1$ to $chr19$ and $chrX/Y$, as well as the random sequences[5] for 12 of these chromosomes, $chrUn\_random$[6] and $chrM$ representing mitochondrial DNA.

After aligning the entire set of sequences with the default parameter values as described in the previous chapter, we obtained the highest scoring alignment for each sequence. Within these resulting alignments, 27893 (%99.4) of them did not contain any positive structural alterations. 71 (%0.25) of the sequences were reported as containing a structural event. Among these 12 are detected as fusion transcription, 25 of them contained at least one inverted exon (including inverted duplications, inverted rearrangements and in-place inversions), 25 of them were same direction duplications and 9 of them were same direction rearrangements.

Apart from the 71 reported events, 40 transcripts were identified as containing a short ambiguous location (2-5bp) that cannot be clearly resolved whether they are short duplications or short insertions longer than a single nucleotide indel. In a reference transcriptome, these are more likely to be short insertion cases that show similarity to their flanking sequences in the transcript, causing the over-extended maximal fragments detecting a short duplication when they are chained together.

duplications between the leading and trailing regions of the inserted sequence (increasing the number of ambiguous insertion/duplication cases). In a large set of insertion simulations, such cases dominate the number of events and the main focus of this experiments is hindered, which is mainly the identification of the alignments that discover a false event by the alignment of the inserted sequence to the genome in a way that alters overall structure of the transcript alignment.

[4]Removing poly-A/poly-T sites is essential for Dissect, due to the fact that the transcript sequence is considered to be globally aligned to a local region in the genome. Since Dissect is also sensitive to structural alterations, a long poly-A/poly-T site at the end can cause structural alterations if there is a corresponding poly-A/poly-T site within a nearby genomic region.

[5]These represent the sequences that cannot be placed in a specific position in the corresponding chromosome with certainty.

[6]This represents the sequences that cannot be placed to any of the sequences with certainty.

The remaining 54 sequences did not produce any alignment results. Among which 7 reported low quality alignments but were discarded by the final low scoring alignment filtering scheme, and 47 of them could not locate a proper genomic region in the region inference step that satisfy the quality threshold.

Other than the alignment type detected, we also investigated the number of transcripts that Dissect was able to align accurately[7] within a nucleotide-level resolution.

Including the 54 transcripts that Dissect did not report any valid alignments, all sequence alignments other than 1844 are observed to be accurate at a nucleotide-level resolution. This corresponds to a %93.8 percentage of accurate nucleotide-level alignments for the full datasets. However, some of these 1844 alignments may contain gaps due to potential existence of novel insertions in the transcript, inaccurate removal of poly-A/poly-T sites from ends and/or absence of a proper region within the used build of reference genome for the alignment of the transcript sequence. In order to estimate the rate of such alignments, we used wild-type transcriptome to genome aligner BLAT (version 34x10) to align these 1844 sequences and find out for how many of these sequences there is an accurate[8] wild-type alignment that Dissect has missed. Among the 1844 aligned sequences, BLAT was able to align 690 of them to the genome as an accurate alignment at a nucleotide-level resolution. Therefore, since there is no clear evidence that the remaining sequences have a nucleotide-level accurate alignment, we consider the effective nucleotide-level sensitivity of Dissect for this experiment to be $\frac{26214}{26214+690} = \%97.4$.

In order to analyse the difference of alignment accuracies between BLAT and Dissect, we also performed the converse alignment experiment. We aligned the 26214 sequences that Dissect aligned accurately at a nucleotide level, using BLAT. Among 26214 transcripts, BLAT also aligned 25936 accurately.

With a modern single core processor, aligning the entire dataset of 28058 sequences took about 70 minutes with Dissect using default parameters, whereas it took more than 400 minutes for BLAT with its default parameters that do not apply additional speed

[7]Definition of an accurate nucleotide-level alignment here is a fragment chaining that do not contain any gaps within the transcript sequence including the end regions. Mismatches are allowed within the fragments as long as they satisfy the original fragment construction constraints.

[8]The definition of an accurate alignment for BLAT is parallel to the definition for Dissect. We considered any alignment that contains a gap within the transcript as not an accurate alignment. In order to verify the accurate property of a BLAT alignment, we checked whether "Q gap count" (number of exon-exon gap blocks within the transcript) is 0 and whether "Q start"/"Q end" (starting and ending positions of the alignment in the transcript) positions correspond to the transcript end-points.

optimization upon commonly repeated regions.

Simulation of nucleotide-level substitutions and indels

After applying the pre-processing steps to the 28058 mouse transcript sequences as described in the previous subsection, we applied a nucleotide-level error addition according to the following scheme, for a given error density value k.

- For each of the indices t_i, from 1 to $|T_j|$, within all transcript sequences T_j within the database:
  - With $\frac{1}{k}$ probability, decide whether to add an error to the current location.
  - If error will be added:
    * With $\frac{1}{2}$ probability, substitute the base value of $v(t_i)$, with another character and continue with the next character in the sequence.
    * With $\frac{1}{4}$ probability, remove t_i and continue with the current index in the updated sequence, in which the remaining sequences are shifted to the left.
    * With $\frac{1}{4}$ probability, insert a random nucleotide to the current index, and continue with the next character in the sequence.

For this nucleotide-level noise addition experiment set, we used two different density values, $k = 500$ and $k = 100$.

In the $k = 500$ simulation dataset, 27871 (%99.3) sequences out of 28058 were aligned as wild-type transcripts without any detected structural alterations. For 54 sequences, a proper alignment was not reported. 95 of the sequences were identified as containing structural alterations (12 fusions, 26 inversions, 49 duplications, 8 rearrangements). Apart from these, 38 sequences were identified as containing ambiguous short duplication/insertion sequences.

In the $k = 100$ simulation dataset, 27732 (%98.8) sequences were aligned as wild-type transcripts without structural alterations. 57 sequences, did not return any alignments. 232 sequences were identified as containing structural alterations (13 fusions, 29 inversions, 178 duplications, 12 rearrangements). Lastly, 37 sequences were identified as containing ambiguous short duplication/insertion sequences.

As it can ben seen from the rate of false event detection and distribution of different event types; the detected number of false events is correlated to the density of nucleotide-level noise, even though a large majority of the sequences stay within their wild-type alignment form. One noticable difference between the three experiments $k = \infty$, $k = 500$ and $k = 100$ is increase in the number of detected false duplications.

Simulations upon short-to-medium size novel sequence insertions in human transcriptome

The aim of this experiment is to evaluate the event detection specificity of dissect due to novel sequence insertion to wild-type transcripts. For this experiment we derived the simulations from RefSeq mRNA annotation dataset for whole human transcriptome.

After the preprocessing step that removes sequences with unwanted characters and poly-A/poly-T sites, number of sequences in the dataset is 37568 and the average number of bases per sequence is 3022.

We aligned these sequences with Dissect[9], and obtained 33652 sequences that are not reported as structural alteration containing transcripts and show nucleotide-level accurate alignment as described above.

Making use of the aligned fragment information reported in the Dissect output; we identified valid insertion locations in the transcript (apart from transcript end points) that represent an exon-exon boundary and represent an intron of longer than the inserted sequence[10].

We distributed this set of 33652 sequences into four equal sized disjoint subsets. For the first subset of transcripts, we inserted novel insertion sequences of length 6 to 20 bases, uniformly distributed within the length interval. For the second subset, we inserted novel sequences of length 21-35 bases. For the third subset, 36-50 and for the fourth subset, 51-65 base length sequences are inserted.

For obtaining realistic novel insertion sequences, we acquired 2363 characterized novel insertion sequences published by Kidd et al. [22]. For sampling the insertion sequences, we randomly located a sequence position among these novel insertion sequences and extracted the sequence of the required length.

The procedure of simulating novel insertions is described as follows:

[9]The full dataset alignment took less than 100 minutes.

[10]This constraint is for reducing contamination by short duplication events due to over-extension of homologous sequences as mentioned earlier.

- For each of the four subsets:
    - For each transcript sequence within the subset:
        * Randomly generate the size of sequence to be inserted according to the insertion length distribution of the subset.
        * Randomly select an end point of the transcript or one of the valid insertion locations identified earlier.
        * Extract the sequence from a randomly selected position within the dataset of 2363 novel insertions, and insert to the selected location in the transcript.

In the resulting alignment of the 33652 modified sequences that contain a novel insertion, 32874 (%97.7) returned a wild-type alignment without any structural alterations. 645 (%2.0) of the returned alignments contained at least one structural alteration. The insertions also caused 45 (%0.1) sequences overall to lose their originally inferred regions and resulted in 88 (%0.3) of the sequences to contain short ambiguous duplication/insertion regions.

However, the results shown above represent the overall simulation dataset. If we analyse each subset separately with respect to insertion length distributions, we can see that the distribution of events among sets clearly displays a bias upon the number of events detected in the presence of longer insertions. Details of this phenomenon can be viewed in Table 3.1, and the distribution of different structural alteration types are given in more detail in Table 3.2.

Insertion length	Total	Wild-type	All events	Amb. dup./ins.	No output
6-20 bases	8413	8367	26 (%0.3)	19	1
21-35 bases	8413	8298	87 (%1.0)	21	7
36-50 bases	8413	8189	186 (%2.2)	22	16
51-65 bases	8413	8020	346 (%4.1)	26	21

Table 3.1: Alignment results of Dissect for simulated wild-type transcriptome dataset with novel insertions. Rows represent the length interval of the novel insertion distributions (e.g. insertions reported in the first row are uniformly distributed between 6 and 20 bases). Columns indicate the output labels of Dissect: *All events* column represents the total number of transcripts that Dissect has identified a structural alteration, *Amb. dup./ins.* column represents the alignments that contain a short ambiguous interval that cannot be verified with certainty as an insertion or a duplication and *No output* column indicates the number of transcript sequences that Dissect did not return a valid alignment for.

Ins. length	All events	Fusion	Inversion	Forw. Dup.	Forw. Rear.
6-20 bases	26	2	6	16	2
21-35 bases	87	2	20	25	40
36-50 bases	186	0	59	43	84
51-65 bases	346	30	151	52	113
All intervals	645	34	236	136	239

Table 3.2: Alignment results of Dissect showing the number of structural alterations with respect to insert length distribution intervals and event types as fusion, inversion, forward duplication and forward rearrangement. *Inversion* column represents inverted rearrangements, inverted duplications, in-place inversions and single-breakpoint inversions altogether.

In Table 3.1 and Table 3.2, we can observe that as the length of the novel insertion increases, so do the false event detection rate of Dissect. Table 3.1 displays low false event detection rate for novel insertions shorter than 35 bases. However, we see a clear difference in false event detection rate of transcript sequences with insertions of length 6-35 bases, and sequences that contain insertions of length 36-65 bases. This high rate of false positives can be caused by Dissect's high sensitivity to structural events when there is strong evidence of sequence similarity. Since the insertion sequences are obtained from a real novel insertion study for human genome, in the neighbourhood of the aligned gene loci there might be sequences homologous to the insertion and these potentially increase the risk of detecting false positive rearrangements. As such, the correlation between the increase in the number of false inversions and false forward rearrangements in the results table is expected, due to the fact that the number of reported inversions also includes the inverted rearrangements.

3.2 Experiments upon Simulated Transcriptional Events

The experiments performed in this section are designed to determine Dissect's event detection sensitivity for various event types with various additional characteristics.

Below we describe 13 different simulation experiments upon mouse transcriptome derived from Known Gene gene structure annotation database.

In order to prepare the wild-type transcriptome dataset that the simulations are going to be applied upon, we extracted genome annotation files for each chromosome from chr1 to chr19 and chrX/Y. We obtained each wild-type transcript by extracting and concatenating exons in each gene in their respective order.

The dataset at this stage contained more than 50,000 transcripts. In order efficiently analyse a large amount simulation cases, we reduced the dataset to one tenth of its original size by sampling each tenth sequence for each chromosome. Afterwards, any transcript sequence that is shorter than 50 is removed due to the fact that they cannot properly express the structural alteration simulations performed in many cases.

At this stage, the dataset consisted of 5256 sequences. This wild-type transcript dataset is aligned with Dissect and 5234 transcripts that aligned successfully with no structural alterations were kept for applying the simulations described below.

The thirteen simulation experiments described below aim to emulate the aberrant formations that can occur in transcripts due to structural alterations. These experiments are:

1. Forward tandem duplication of the full transcript sequence.
2. Forward tandem duplication of the longest exon in the transcript.
3. Forward tandem duplication of the shortest exon in the transcript
4. In-place inversion of the longest exon in the transcript
5. In-place inversion of the shortest exon in the transcript
6. Folding inversion of the transcript from a position close to mid-point. (Split interval: %36-%65)
7. Folding inversion of the transcript from a position close to the beginning/end. (Split interval: %16-%35 or %66-85)
8. Full split rearrangement of transcript. (Rearrangement of the full transcript sequence from a particular split position)
9. Rearrangement of adjacent exons.
10. Rearrangement of non-adjacent exons.
11. Fusion transcript with a well-balanced split ratio (shorter fused sequence is $\geq$ %60 the longer one)

12. Fusion transcript with a moderately balanced split ratio (short to long sequence ratio is $\geq$ %30 but $<$ %60)

13. Fusion transcript with an imbalanced split ratio (short to long sequence ratio is $<$ %30)

The distribution of transcript alignments according to event type labels for different event simulation experiments are given in Table 3.3.

	Tot.	No out.	Non-E.	Amb.	Tot-E.	Fusion	Inv.	F. Dup.	F. Rea.
Exp. 1	5234	8	85	2	5139	1	43	5095	0
Exp. 2	5234	5	58	2	5169	5	1	5163	0
Exp. 3	5234	0	139	3	5092	0	0	5092	0
Exp. 4	4788	0	27	0	4761	0	4761	0	0
Exp. 5	4788	0	507	0	4281	0	4272	0	9
Exp. 6	3188	1	12	0	3175	3	3171	0	1
Exp. 7	4654	2	101	0	4551	2	4547	0	2
Exp. 8	4788	2	166	6	4614	2	106	2	4504
Exp. 9	4788	1	165	6	4616	1	10	2	4603
Exp. 10	4316	0	41	2	4273	0	30	6	4237
Exp. 11	1312	16	12	0	1284	1279	5	0	0
Exp. 12	1558	25	31	0	1502	1500	1	0	1
Exp. 13	2363	15	1751	0	597	595	1	0	1

Table 3.3: Number of structural alterations detected by Dissect for the designed simulation datasets (*Tot.* = Total number of transcript sequences, *No out.* = The number of transcripts that Dissect did not return any valid alignment for, *Non-E.* = Number of transcripts Dissect returned a wild-type alignment for (with potential insertions), *Amb.* = Number of transcripts containing ambiguous short duplication/inversion regions, *Tot-E.* = Total number of discovered structural event containing transcripts, *Inv.* = Inversion events including inverted duplications, inverted rearrangements, in-place inversions, and single-breakpoint inversions, *F. Dup.* = Forward duplication events, *F. Rea.* = Forward rearrangement events.)

3.3 Searching Structural Alterations in Prostate Cancer Transcriptome Assemblies

For this experiment we obtained 50bp RNA-seq reads of a human prostate cancer individual.

These reads were assembled using short-read transcriptome assembler Trans-Abyss [6] version 1.2.0 that is based upon de novo short read assembly tool, Abyss [40] version 1.2.5.

For the assembly runs, we used two different k-mer sizes, 26 and 49[11]. Furthermore, for two separate contigs to be merged (regulated by n parameter), we required 10 pair mappings between the contigs.

We aligned this transcriptome assembly dataset to whole genome using Dissect with the default parameters described in the end of the previous chapter, and obtained the following alignment type distributions.

Among a total number of $576,381$ assembly contigs given input to our aligner: For $167,218$ transcripts Dissect did not return a valid output. For $391,504$ contigs, Dissect identified no structural alterations, with the exception of potential insertions to the contig sequence. In 4279 contigs, Dissect detected an ambiguous short insertion/duplication region. Finally for 13380 contigs, Dissect discovered a structural alteration.

Among the 13380 detected structural event containing contigs, 946 were identified as fusions, 1381 were identified as duplications, 513 were identified as rearrangements and 10540 were inversion events including inverted duplications, inverted rearrangements and in-place inversions.

In order to check the number of false positive events detected in this result, we again used BLAT alignments as a surrogate in order to determine whether there is a high scoring alignment.

Among 13380 transcripts BLAT returned a valid output for 13241. Considering the remaining 139 contigs as having 0 score value, we report the distribution of BLAT alignment similarity scores for event detected transcripts in Table 3.4, similarity score being calculated as the ratio of number of matching nucleotides to the overall length of the contig.

Among the 1576 contigs in the $[\%95, \%100[$ interval, 1088 of them contained at least one gap that is longer than an indel and for 602 of them total size of the gaps in the transcript was larger than or equal to 10.

[11]k-mer size parameter determines the size of the overlap required between two reads in order to connect them within a contiguous sequence.

Sim. interval	[%0,%50[	[%50,%65[	[%65,%80[	[%80,%90[	[%90,%95[	[%95,%100]
# contigs	418	7682	2204	920	580	1576

Table 3.4: Similarity score distribution of BLAT alignments for assembly contigs that Dissect detected a structural alteration. Each column in the first row indicates the percentage similarity distribution of the alignment and the second row represents the number of contigs that fall into the corresponding interval.

Chapter 4

Summary and Conclusion

In this thesis, we introduced two novel frameworks for the problem of aligning transcribed sequences to the genome in order to discover structural alterations such as duplications, inversions, rearrangements and fusions within the transcript. For both frameworks, the structural alterations are considered as copies of substrings from the genome forming the transcript sequence, allowing a problem formulation that can be solved in polynomial time.

The first framework we defined is based on nucleotide-level alignment model that involves transitions from each position pair (t_i, g_j) alignment between the transcript and the genome to any position pair (t'_i, g'_j) with the only condition that $t'_i > t_i$ forming a monotonously increasing set of indices on the transcript side of the alignment. However, this rule is not the same case for the genome side and positions throughout the alignment are not required to be monotonously increasing or decreasing. For this reason, instead of a regular 2-row alignment matrix we define the alignment formulation as a function from the positions in the transcript to the positions in the genome with an additional a gap character. We defined the best alignment as the optimal function definition that maximizes a certain alignment score model and we proposed solution methods for this problem definition. Our best algorithms for this formulation was $O(NMlog(M))$ for convex genomic gap penalties and $O(MN)$ for simple convex functions (including log-scale) that satisfy a certain zero-calculation property, M,N representing the size of the transcript and genome sequences respectively.

The second framework we defined is a low-sensitivity problem formulation that allows a faster alignment in a whole genome alignment setting. Given a set of homologous fragments between the transcript and the genome (defined as intervals in the transcript corresponding to an interval in the genome with a particular direction and similarity score), we described our problem as finding a chain of fragments that maximizes a certain fragment chain score model with the constraint that within the fragment chain the transcript start positions are monotonously increasing. Since this problem can be reduced to the first problem, it can also be solved in polynomial time and we describe $O(K^2)$ solutions for two versions of this problem (disjoint and overlapping fragment chaining), K being the number of fragments in the given fragment set.

Furthermore, in this thesis we presented an efficient transcriptome to whole genome alignment tool, Dissect, which is formed as a pipeline of the following steps; reducing the solution space from the whole genome to a smaller putative gene region, constructing a comprehensive set of homologous fragments between the transcript and the genomic region, applying a fragment chaining algorithm that is sensitive to structural alterations and finally refining potential small-scale misalignments that could have been caused by length and similarity constraints of the fragment construction step.

We evaluated the accuracy and efficiency of our aligner upon simulation datasets and assembled RNA-seq reads of a human prostate cancer individual.

We split the simulation tests into two separate directions: (1) estimating false event detection rate of Dissect in wild-type transcripts with simulated noise and novel insertions and (2) evaluating event detection sensitivity of Dissect in event simulation datasets constructed from gene structure annotation.

We estimated the false detection rate of Dissect to be very low, less than %1.2 up to a substitution/indel noise frequency of 1/100 bases. We also estimated the nucleotide-level alignment accuracy of Dissect, using BLAT wild-type alignments as a surrogate and estimated this accuracy to be %97.4 in a wild-type transcriptome dataset with no noise addition.

Furthermore, in novel insertion simulations we observed the false event detection rate to be less than %2.2 up to novel insertions of size 50 bases. However, for novel insertions of sizes longer than 51 bases, we detected a higher error rate up to %4.1 of the overall transcript sequences (mostly due to rearrangements).

In the second simulation dataset, we evaluated event detection sensitivity of Dissect to be over %94 for all the experiments performed apart from short in-place inversions and fusion transcripts with an imbalanced ratio of the fused sequences.

In the final experiment, we reported event detection results of Dissect in an assembly dataset. Even though there is no comprehensive event annotation for this dataset, we estimated that for more than %88 of the reported events, corresponding transcripts do not have a global high-quality alignment of the transcript when aligned with a wild-type spliced aligner.

In the remainder of this section, we discuss some potential improvements to this study from theoretical, biological analysis and software engineering standpoints.

4.1 Future Works

In this thesis study the best algorithms given for nucleotide level alignment algorithms are within the run-time complexity of $O(NMlog(M))$, N,M representing transcript and genome sequence lengths respectively. The algorithm for the wild-type transcript alignment with convex functions is known to be reduced from $O(NMlog(M))$ complexity to $O(NM\alpha(M))$, α representing very slowly growing inverse Ackermann function, using the matrix searching method introduced by [23]. However, as also noted in [10], this method has a high constant in comparison to the WTTA-2 algorithm due to the heavy cost of matrix searching methods. This cancels the advantage gained by the reduction from $O(M)$ to $\alpha(M)$ for genomic purposes, especially when considering mouse/human genomes of size 2.5-3Gb[1]. Therefore for practical purposes, the use of the lower complexity solution is not encouraged. For this reason, we did not investigate the possibility of improving our "alignment with structural alterations" solution in TGASA-2, using the matrix searching method in [23]. However; for theoretical purposes, it might be worthwhile to investigate this problem as a future study.

Furthermore; in this thesis study, the solution given to the chaining problem does not consider dynamic programming sparsification techniques in order to reduce chaining solution complexity. The version of the chaining problem with no structural alterations, no overlaps and perfect fragments is known to have a lower complexity solution for several transition penalty functions slightly different than described in our formulation[8][10][24].

There are further improvements upon the affine (linear) gap penalty version of the chaining problem, useful for genome-to-genome alignment purposes in which a gap penalty is described in an affine gap penalty setting. In this case, for an overlapping fragment definition that is slightly different than the one described in our formulations, a workaround has been proposed by [39] that sparse solution methods can be applied upon. Moreover,

[1]Gigabases.

in [7] a genome-to-genome alignment fragment chaining method is proposed involving genomic structural variations such as inversions, duplications and translocations, extending the sparse chaining solution given in [9] for linear gap penalties.

As of this thesis study, we are not certain whether sparse solution methods can be applied to our chaining formulation; which involves structural alterations, overlapping fragments and an advanced genomic gap penalty scheme that involves a convex gap penalty function combined with splice signal scores as well as an additional penalty based on mismatch resolution of overlapping fragments. However; as a future study, it might be worthwhile to investigate the potential ways to extend this in order to gain speed and provide an improved theoretical framework for transcriptome-to-genome fragment chaining problem involving transcriptional structural alterations.

Another possible improvement to the fragment chaining formulation and the corresponding implementation would be to extend the currently given fragment definition to maximal fragments involving mismatches and indels within similarity and minimum length constraints. There two challenges that need to be addressed before attempting this new problem: (1) a homology searching tool is needed that can find maximally extended fragments containing mismatches and indels within certain error rate constraints and (2) the overlap resolution scheme should be changed in order to allow resolution of two fragments that contain mismatches and indels. Adding this functionality will increase nucleotide-level accuracy of the aligner, due to the fact that the majority of small-scale misalignments are caused by inadequate/improper extension of maximal fragments that are not allowed to contain indels.

There are several other improvements that can be made to this study from a downstream analysis standpoint. A joint analysis can be made for gene structure annotation and transcripts that contain structural alterations identified by Dissect, in order to evaluate biological significance of detected events and prioritize events that are more likely to be authentic. Furthermore, for the experiments with assembled RNA-seq reads, it is possible to further estimate the authenticity of the detected events by evaluating the score of the assembled contigs with the analysis of the original reads that are used to construct the contigs. A parallel study to ours, [44] (unpublished) addresses this aspect of identifying transcriptional novel events, which can be integrated to Dissect in order to obtain alignment results prioritized with respect to the biological relevance of identified events and RNA-seq read support analysis of assembled transcript sequences.

Also from a software engineering perspective, Dissect can be further improved in terms of usability, accessibility and visualization. In this thesis study, the main focus was on developing an efficient alignment tool that follows the guidelines given by the alignment frameworks described. However, contribution of an alignment tool to the transcriptomics research community is also correlated to accessibility of the software. Upgrading this software to a server-client architecture requires a lot of modifications on both Dissect and mrsFAST_HS implementations, but setting up a server that already has the genome and mrsFAST index loaded in the memory will allow client processes make instantaneous queries to the server returning a single transcript alignment with structural alterations. In the current version each transcript query takes around 10-200 milliseconds on average with a modern processor. However, it will also take around 10 minutes of loading time per input dataset, regardless of the number of transcripts in the input. Within a server-client framework, this overhead time can be minimized allowing faster access of users for small number of queries. Similarly an associated visualization tool for Dissect will improve the user experience by real-time visualization of the transcript alignments containing structural alterations. This addition will improve manual evaluation of biological relevance of the structural variations to the applied study.

Bibliography

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

[2] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997.

[3] Alissa M. Anderson and Jonathan P. Staley. Long-distance splicing. *Proceedings of the National Academy of Sciences*, 105(19):6793–6794, 2008.

[4] Yan W. Asmann, Asif Hossain, Brian M. Necela, Sumit Middha, Krishna R. Kalari, Zhifu Sun, High-Seng Chai, David W. Williamson, Derek Radisky, Gary P. Schroth, Jean-Pierre A. Kocher, Edith A. Perez, and E. Aubrey Thompson. A novel bioinformatics pipeline for identification and characterization of fusion transcripts in breast cancer and normal cell lines. *Nucleic Acids Research*, 2011.

[5] Steven A. Benner, Mark A. Cohen, and Gaston H. Gonnet. Empirical and structural models for insertions and deletions in the divergent evolution of proteins. *Journal of Molecular Biology*, 229:1065–1082, 1993.

[6] Inan Birol, Shaun D. Jackman, Cydney B. Nielsen, Jenny Q. Qian, Richard Varhol, Greg Stazyk, Ryan D. Morin, Yongjun Zhao, Martin Hirst, Jacqueline E. Schein, Doug E. Horsman, Joseph M. Connors, Randy D. Gascoyne, Marco A. Marra, and Steven J. M. Jones. De novo transcriptome assembly with abyss. *Bioinformatics*, 25(21):2872–2877, 2009.

[7] Michael Brudno, Sanket Malde, Alexander Poliakov, Chuong B. Do, Olivier Couronne, Inna Dubchak, and Serafim Batzoglou. Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(suppl 1):i54–i62, 2003.

[8] David Eppstein. Sequence comparison with mixed convex and concave costs. *J. Algorithms*, 11:85–101, February 1990.

[9] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming i: linear cost functions. *J. ACM*, 39:519–545, July 1992.

[10] David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming ii: convex and concave cost functions. *J. ACM*, 39:546–567, July 1992.

[11] Zvi Galil and Raffaele Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theor. Comput. Sci.*, 64:107–118, April 1989.

[12] Huanying Ge, Kejun Liu, Todd Juan, Fang Fang, Matthew Newman, and Wolfgang Hoeck. Fusionmap: detecting fusion genes from next-generation sequencing data at base-pair resolution. *Bioinformatics*, 2011.

[13] Thomas R. Gingeras. Implications of chimaeric non-co-linear transcripts. *Nature*, 461(7261):206–211, September 2009.

[14] Xun Gu and Wen-Hsiung Li. The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment. *Journal of Molecular Evolution*, 40:464–473, 1995. 10.1007/BF00164032.

[15] Faraz Hach, Fereydoun Hormozdiari, Can Alkan, Farhad Hormozdiari, Inanc Birol, Evan E. Eichler, and S. Cenk Sahinalp. mrsFAST: a cache-oblivious algorithm for short-read mapping. *Nature Methods*, 7(8):576–577, August 2010.

[16] Roberto Hirochi Herai and Michel E. Beleza Yamagishi. Detection of human interchromosomal trans-splicing in sequence databanks. *Briefings in Bioinformatics*, 11(2):198–209, 2010.

[17] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[18] Takayuki Horiuchi and Toshiro Aigaki. Alternative trans-splicing: a novel mode of pre-mrna processing. *Biol. Cell*, 98(2):135–140, 2006.

[19] Jonathan Houseley and David Tollervey. Apparent non-canonical trans-splicing is generated by reverse transcriptase *In Vitro*. *PLoS ONE*, 5(8):e12271, 08 2010.

[20] Koichiro Inaki, Axel M. Hillmer, Leena Ukil, Fei Yao, Xing Y. Woo, Leah A. Vardy, Kelson F. Zawack, Charlie W. Lee, Pramila N. Ariyaratne, Yang S. Chan, Kartiki V. Desai, Jonas Bergh, Per Hall, Thomas C. Putti, Wai L. Ong, Atif Shahab, Valere Cacheux-Rataboul, Radha K. Karuturi, Wing-Kin Sung, Xiaoan Ruan, Guillaume Bourque, Yijun Ruan, and Edison T. Liu. Transcriptional consequences of genomic structural aberrations in breast cancer. *Genome Research*, 21(5):676–687, May 2011.

[21] W. James Kent. Blat—the blast-like alignment tool. *Genome Research*, 12:656–664, 2002.

[22] Jeffrey M. Kidd, Nick Sampas, Francesca Antonacci, Tina Graves, Robert Fulton, Hillary S. Hayden, Can Alkan, Maika Malig, Mario Ventura, Giuliana Giannuzzi, Joelle Kallicki, Paige Anderson, Anya Tsalenko, N. Alice Yamada, Peter Tsang, Rajinder Kaul, Richard K. Wilson, Laurakay Bruhn, and Evan E. Eichler. Characterization of missing human genome sequences and copy-number polymorphic insertions. *Nature Methods*, 7(5):365–371, April 2010.

[23] Maria M. Klawe and Daniel J. Kleitman. An almost linear time algorithm for generalized matrix searching. *Siam Journal on Discrete Mathematics*, 3:81–97, 1990.

[24] James R. Knight and Eugene W. Myers. Approximate regular expression pattern matching with concave gap penalties. *ALGORITHMICA*, 14:67–78, 1992.

[25] Donald E. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition)*. Addison-Wesley Professional, 2 edition, May 1998.

[26] Joshua Z. Levin, Michael F. Berger, Xian Adiconis, Peter Rogov, Alexandre Melnikov, Timothy Fennell, Chad Nusbaum, Levi A. Garraway, and Andreas Gnirke. Targeted next-generation sequencing of a cancer transcriptome enhances detection of sequence variants and novel fusion transcripts. *Genome biology*, 10(10):R115+, October 2009.

[27] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[28] Xin Li, Li Zhao, Huifeng Jiang, and Wen Wang. Short homologous sequences are strongly associated with the generation of chimeric rnas in eukaryotes. *Journal of Molecular Evolution*, 68:56–65, 2009. 10.1007/s00239-008-9187-0.

[29] Yang Li, Jeremy Chien, David I. Smith, and Jian Ma. Fusionhunter: identifying fusion transcripts in cancer using paired-end rna-seq. *Bioinformatics*, 2011.

[30] Andrew McPherson, Fereydoun Hormozdiari, Abdalnasser Zayed, Ryan Giuliany, Gavin Ha, Mark G. F. Sun, Malachi Griffith, Alireza Heravi Moussavi, Janine Senz, Nataliya Melnyk, Marina Pacheco, Marco A. Marra, Martin Hirst, Torsten O. Nielsen, S. Cenk Sahinalp, David Huntsman, and Sohrab P. Shah. defuse: An algorithm for gene fusion discovery in tumor rna-seq data. *PLoS Comput Biol*, 7(5):e1001138, 05 2011.

[31] Andrew McPherson, Chunxiao Wu, Iman Hajirasouliha, Fereydoun Hormozdiari, Faraz Hach, Anna Lapuk, Stanislav Volik, Sohrab Shah, Colin Collins, and S. Cenk Sahinalp. Comrad: detection of expressed rearrangements by integrated analysis of rna-seq and low coverage genome sequence data. *Bioinformatics*, 27(11):1481–1488, 2011.

[32] Webb Miller and Eugene Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50:97–120, 1988. 10.1007/BF02459948.

[33] Richard Mott. Est_genome: a program to align spliced dna sequences to unspliced genomic dna. *Computer applications in the biosciences : CABIOS*, 13(4):477–478, 1997.

[34] Gene Myers and Webb Miller. Chaining multiple-alignment fragments in sub-quadratic time. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, SODA '95, pages 38–47, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[35] Serban Nacu, Wenlin Yuan, Zhengyan Kan, Deepali Bhatt, Celina Rivers, Jeremy Stinson, Brock Peters, Zora Modrusan, Kenneth Jung, Somasekar Seshagiri, and Thomas Wu. Deep rna sequencing analysis of readthrough gene fusions in human prostate adenocarcinoma and reference samples. *BMC Medical Genomics*, 4(1):11, 2011.

[36] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[37] Gordon Robertson, Jacqueline Schein, Readman Chiu, Richard Corbett, Matthew Field, Shaun D. Jackman, Karen Mungall, Sam Lee, Hisanaga Mark M. Okada, Jenny Q. Qian, Malachi Griffith, Anthony Raymond, Nina Thiessen, Timothee Cezard, Yaron S. Butterfield, Richard Newsome, Simon K. Chan, Rong She, Richard Varhol, Baljit Kamoh, Anna-Liisa L. Prabhu, Angela Tam, YongJun Zhao, Richard A. Moore, Martin Hirst, Marco A. Marra, Steven J. Jones, Pamela A. Hoodless, and Inanc Birol. De novo assembly and analysis of RNA-seq data. *Nature methods*, 7(11):909–912, November 2010.

[38] Andrea Sboner, Lukas Habegger, Dorothee Pflueger, Stephane Terry, David Chen, Joel Rozowsky, Ashutosh Tewari, Naoki Kitabayashi, Benjamin Moss, Mark Chee, Francesca Demichelis, Mark Rubin, and Mark Gerstein. FusionSeq: a modular framework for finding gene fusions by analyzing paired-end RNA-sequencing data. *Genome Biology*, 11(10):R104+, October 2010.

[39] Tetsuo Shibuya and Igor Kurochkin. Match chaining algorithms for cdna mapping. In *In Proc. Workshop on Algorithms in Bioinformatics (WABI), volume 2812 of LNCS*, pages 462–475. Springer, 2003.

[40] Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven J. Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123, June 2009.

[41] Guy Slater and Ewan Birney. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, 6(1):31+, 2005.

[42] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, March 1981.

[43] Philip J. Stephens, David J. McBride, Meng-Lay Lin, Ignacio Varela, Erin D. Pleasance, Jared T. Simpson, Lucy A. Stebbings, Catherine Leroy, Sarah Edkins, Laura J. Mudie, Chris D. Greenman, Mingming Jia, Calli Latimer, Jon W. Teague, King W. Lau, John Burton, Michael A. Quail, Harold Swerdlow, Carol Churcher, Rachael Natrajan, Anieta M. Sieuwerts, John W. M. Martens, Daniel P. Silver, Anita Langerod, Hege E. G. Russnes, John A. Foekens, Jorge S. Reis-Filho, Laura van /'t Veer, Andrea L. Richardson, Anne-Lise Borresen-Dale, Peter J. Campbell, P. Andrew Futreal, and Michael R. Stratton. Complex landscapes of somatic rearrangement in human breast cancer genomes. *Nature*, 462(7276):1005–1010, December 2009.

[44] Lucas Swanson, Karen Mungall, Gordon Robertson, Readman Chiu, Shaun D Jackman, Jenny Q Qian, Sam Lee, Deniz Yorukoglu, Rong She, Yongjun Zhao, Richard Moore, Marco A Marra, Steven JM Jones, Aly Karsan, Pamela A Hoodless, S Cenk Sahinalp, and Inanc Birol. Browsing assembled rna for chimera with localized evidence, July 2011.

[45] Michael S. Waterman. Efficient sequence alignment algorithms. *J. Theor. Biol*, 108:333–337, 1984.

[46] Thomas D. Wu and Colin K. Watanabe. Gmap: a genomic mapping and alignment program for mrna and est sequence. *Bioinformatics/computer Applications in The Biosciences*, 21:1859–1875, 2005.