LIPID: A LINEAR PROGRAMMING APPROACH FOR ISOFORM DETECTION AND ABUNDANCE ESTIMATION

by

Marzieh Bakhshi
B.Sc., Sharif University of Technology, 2009

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

SIMON FRASER UNIVERSITY
Spring 2012

APPROVAL

Name: Marzieh Bakhshi

Degree: Master of Science

Title of Thesis: LiPID: a Linear Programming approach for Isoform Detection and abundance estimation

Examining Committee: Dr. Kay Wiese
Chair

Dr. Cenk Sahinalp, Senior Supervisor

Dr. Martin Ester, Supervisor

Dr. Anna Lapuk, Supervisor,

Dr. Peter Unrau, SFU Examiner

Date Approved:

Partial Copyright Licence

Abstract

Alternative splicing of the premature mRNA is an important step of gene expression regulation affecting 75%-95% of human genes. Multiple studies have clearly demonstrated that compared to normal tissues, tumours shift splicing patterns of many cancer associated genes, which involves either complete switch from one isoform to another or change in ratio of isoforms. During the past decade many algorithms have been proposed to detect splice isoforms using high resolution microarrays and sequencing data. However, inferring relative abundance of detected isoforms remains a challenge. Here we present a Linear Programming method that infers splice isoforms expressed in a given sample and estimates their absolute abundance. The algorithm is applicable to any sub-gene level expression data from both micro-array and RNA-Seq technologies and requires exon annotation. We aim the optimization function at minimizing the deviation from the expression of the regions, and maximizing the lengths of the isoforms.

To my parent who always supported me no matter what, and to my husband who was always caringly by my side.

"Question everything. Learn something. Answer nothing."

— *Euripides*

Acknowledgments

I would first like to thank my senior supervisor, Dr. Cenk Sahinalp for all his great support for me during my masters; he gave me great instructions and motivation to first start working in the world of computational biology, and then walked with me step by step to accomplish what I aimed for. I am truly grateful for his great impression on my academic life.
I owe a significant part of my success in this project to Dr. Anna Lapuk for partly providing a good subject for my research, and also for helping me a great deal to widen my knowledge in biology.
I would also like to thank Dr. Martin Ester for agreeing to co-supervise my thesis.
My education and research was not possible without scientific, technical, and moral support of my lab mates, and other fellow graduate students of Computer Science at SFU; hence I am really thankful for being able to know them, and learn a lot from them.
The last but not the least, I would like to thanks my family for giving me invaluable moral support during these two years of my masters. I sure could not survive this period without their help.

Contents

Chapter 1

Introduction

1.1 Motivation

The introduction of several high-throughput sequencing platforms, replacing Sanger-based technologies, has hugely impacted genomics studies. Their ability to sequence genomes with low cost and high coverage has provided a whole new world of possibilities in many contexts, such as whole-genome sequencing, targeted re-sequencing, discovery of transcription factor binding sites, and non-coding RNA expression profiling [21]. In high-throughput sequencing technology, the reads are shorter than the ones produced by low-coverage Sanger-based sequencing methods, however they have much higher coverage, ranging from 10-100 fold [17].

For transcriptome studies, RNA-Seq a.k.a Whole Genome Shutgun Sequencing, is a tool that uses high-throughput sequencing technology to get information about RNA content by sequencing cDNAs. It has been first introduced in a series of studies [22], [34], [15], [20], [18], [3], [23], and it has been widely used since then. The data produced by this technique is invaluable in many contexts as in gene expression analysis and gene fusion detection[16].

1.1.1 What is alternative splicing?

In genetics, the term splicing refers to the connection of two ends of specific parts of RNA (ribonucleic-acid) molecule that are derived from a gene to give the final molecule of messenger RNA which is later translated into a protein.

Every cell in our body contains DNA (deoxyribonucleic-acid), RNA and thousands of

proteins and other chemicals. Different cells contain a different set of proteins.

The genetic information of every known organism is stored in long chains of DNA molecules. The functional units of the genome are genes, which are arranged successively on the DNA strands. Usually one gene codes for one protein, meaning that the sequence of the DNA determines the sequence of amino acids forming one specific protein. However, the information stored in the DNA cannot be translated into a protein directly; the DNA rather serves as a template which is copied into RNA molecules. This process is called transcription. The product, the messenger-RNA (mRNA), is the working unit of the genome. It provides the link between the code stored in the DNA or gene and the protein product. The messenger RNA is recognized by a big cellular machine, called the ribosome, which is able to decipher the information encoded by the RNA and translates it into a sequence of amino acids that form the protein molecule. This process is termed translation. The succession of events described is called gene expression; it is performed in the described order in every cell of every living organism known.

But now it gets a little more complicated; most genes are made up of pieces of DNA which code for proteins (exons) and pieces which dont (introns). When the gene is transcribed the messenger RNA also contains exons and introns. Before the messenger RNA can be translated all of the introns must be cut out and the pieces of exons joined together (this is where splice comes from) to form an isoform. In order to avoid mistakes in the final product (functional protein) the pre-mRNA must be spliced correctly. If mRNA is not pasted together properly it will result in the defective protein and might lead to disease or could even be fatal.

The gene expression pathway

The genome or DNA is located in the cellular nucleus where it is transcribed and the pre-mRNA is formed. After several RNA processing steps (including splicing) the isoform (mature mRNA) is transported to the cytoplasm where protein production proceeds (translation).

One of the big surprises when the human genome was finally sequenced was that there were only 25,000 30,000 genes. We thought that as very complex animals we would have a lot more genes than other animals or plants. In fact, we have the same number of genes as rice and other plants, but more than fruit flies or nematodes.

So how is it possible to run all the functions of a human body with all its functions

and abilities with this low number of only 25,000 30,000 genes? The answer is Alternative splicing.

Humans produce around 150,000 different proteins from their 25,000 30,000 genes. They do this by alternative splicing (AS). Alternative splicing means, that during the RNA splicing event different combinations of exons can be joined together to create a diverse array of isoforms from a single pre-mRNA. When these are translated they produce different proteins which do different things. The production of these proteins in the right cell, in the right amount and at the right time means that alternative splicing must be highly controlled. More than 70% of the human protein-coding genes are alternatively spliced. This explains the fact that the relatively small number of 25,000 genes can lead to over 100,000 of proteins. Alternative splicing is involved in all aspects of our growth and development and how our bodies work. It might be surprising but alternative splicing is responsible for many basic human characteristics, for example for hearing. Please refer to Figure 1.1 for a simple diagram demonstrating Alternative Splicing.

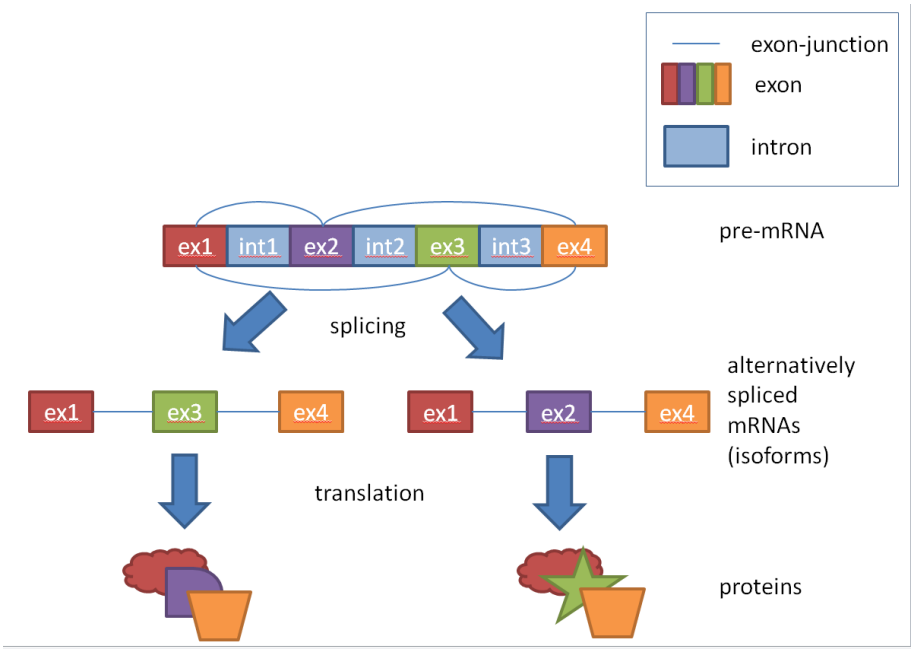

Figure 1.1: Alternative Splicing Simple Demonstration

Alternative Splicing and Health

With so many genes undergoing alternative splicing in such intricate and controlled ways it is not surprising that when things go wrong with splicing, diseases can occur. This can either be a mutation in a specific gene which changes the normal splicing or mutations which affect splicing of many genes. About 15% of mutations connected with disease affect splicing[13]. Alternative Splicing when it is changed can cause diseases like Cystic Fibrosis, Becker Muscular Dystrophy, Cancer and tumor development[11]. Multiple studies[29], [33], [30], [8], [35] have clearly demonstrated that compared to normal tissues, tumors affect splicing patterns of many cancer associated genes, which involve either complete switch from one isoform to another, or change in ratio of isoforms.

Understanding transcriptome is crucial to interpret the functional elements of the genome, yet one of the *major* steps to understand transcriptome is to assemble the transcript together with quantifying the expression level of each transcript.
Compared to array based expression profiling, RNA-seq technology has reduced the cost of sequencing and mapping the transcriptomes significantly, yet there exist some major challenges that need to be overcome in order to be able to assemble and estimate the abundance of the transcriptome.

In this thesis, we focus on the task of assembling the transcriptome and estimating its abundance based on the present gene segments(exons and exon-junctions) and their expression levels.

1.2 Methods background

There are some methods and tools out there that are developed to conduct the aforementioned task:

TopHat[27], SpliceMap[2], and MapSplice[31] find splice junctions by mapping the RNA-seq reads onto the transcriptome, and they discover the novel splice junctions considering the reads that span different locations of the genome.

Trans-ABySS[24], on the other hand, tries to assemble transcripts *de novo* from the short reads; it first uses ABySS[25] to assemble RNA-seq reads into full-length transcripts, then using BLAT[12] to map the assembled transcripts, it tries to find out if any transcript maps across fusion points discordantly.

Some other transcript assembly tools such as Cufflinks [28], Scripture [9], IsoInfer [6], and IsoLasso [14], map the reads to the whole genome, and based on the connectivity graph and the number of reads mapped to each region, and also by applying some algorithmic methods, predict the alternative splicing events, and quantify the predicted transcripts.

Cufflinks[28] converts the transcript assembly problem to finding a maximum matching (minimum path cover) in the overlap graph. It first aligns the reads onto the reference genome using TopHat[27]. It builds the overlap graph based on the reads mapped to the gene by splice fragments being nodes, and if two fragments are compatible, it connects them through an edge. Then it generates the transcripts by partitioning the overlap graph into minimum path cover, i.e. a set of disjoint maximal paths of the graph where all of the nodes and edges of the graph are covered by at least one path. After assembling the transcripts, Cufflinks[28] estimates the relative expression level of each isoform by counting the abundance of reads in each region, and in RPKM scale.

Scripture[9] also aligns the reads to the reference genome using TopHat[27]. Then considering the reads whose alignment to the reference genome contained a gap, it builds a transcript graph, where the nodes are bases and the edges connect each base to the next base in the genome as well as to all bases that it is connected to, through a spliced read. It then considers all possible paths in the transcript graph as possible alternative splicing events, and by scanning them with fixed-sized windows, it scores each path based on its significance. After this, by adding the paired-end information, it enhances the graph and removes the unlikely isoforms, or joins the closely related ones, and at the end it returns some scores which indicate the possibility of those isoforms.

IsoInfer[6] simulates the relationship between exons and splice junctions as a convex quadratic program, and then uses an algorithm to search for the isoforms. The valid isoforms are all those whose expression levels are more than a certain threshold. The algorithm tries to find the minimum number of valid isoforms that explains the read data, i.e. calculates the weighted set cover solution of the overlap graph. After that, it uses the minimum number of valid isoforms to cover the splice graph, and estimates the expression level of each isoform based on the read abundance for each region.

IsoLasso[14] borrows the LASSO[26] algorithm from statistical machine learning. Its objective function is to minimize the difference between the predicted expression of the isoform and the observed expressions of the exons and junctions. It also tries to minimize the number of isoforms by adding a regularization term to achieve sparsity. IsoLasso considers

all *maximal* paths possible, but then filters out the isoforms that are "infeasible" i.e. they cannot be assembled from the mapped reads. It is worth mentioning that IsoLasso calculates the expression levels of the isoforms at the same time that it discovers them.

1.3 Contribution: LiPID

In this thesis, a pipeline for detecting alternative splicing events and estimating their relative abundance was developed; LiPID takes as input RNA-seq short reads, aligns them to the reference genome using mrFAST[1], and then by counting the reads mapped to each segment, calculates the expression level of each gene segment, i.e. exons, and exon junctions. Based on splicing compatibility[10], and Dilworth theorem[5], "the number of mutually incompatible reads equals to the minimum number of transcripts needed to explain all the fragments", hence we devised a linear programming method, so that while the constraint on expression levels of gene segments holds, the parsimony principle (hence minimization of number of isoforms) is also satisfied. We eventually try to optimize the following function:

$$maximize \left[\sum_{i=1}^{K} w_i Iisoform_i - \sum_{j=1}^{N+M} w'_j err_j \right] \tag{1.1}$$

where

- K is the number of possible isoforms,
- N is the number of exons,
- M is the number of exon junctions,
- $Iisoform_i$ is the expression level of $isoform_i$ (the ith isoform among all possible ones) which should be determined by solving the linear program,
- err_j is the error we allow for divergence from the jth $exon_j/juc_j$
- w_i is the weight we give to $isoform_i$ expression level,
- w'_j is the weight we give to err_j to optimize it, and determine how far apart it can be from the calculated expression level of $exon_j/juc_j$

Please note that the weights w_i and w'_j are meant to normalize the range of isoforms' expression levels or errors regarding gene segments expression levels in the optimization function with respect to each other, and how much each value can oscillate.

The major advantage of LiPID and IsoLasso[14] over Cufflinks[28], Scripture[9], and IsoInfer[6] is that they predict the set of isoforms and estimate their expression levels simultaneously, instead of doing them in two separate steps. This is a better approach, because indeed considering expression levels of exons and exon-junctions contains a lot of important information that can help in determining the right set of isoforms. We predict the isoforms while we satisfy the constraints of the measured expression levels of the exons and junctions. Figure 1.2 shows an example which shows inability of these methods in detecting the isoforms when using only the presence of exons and junction, and how using the expression level information can help solve the ambiguity easily; Figure 1.2 (a) is the splice graph of a gene, where the expression levels and present exon connections are shown. If we consider the expression levels of the exons and junctions, Figure 1.2 (b) is the only answer with the best parsimony according to the splice graph in Figure 1.2 (a), which would be the answer that LiPID provides. However, Figure 1.2 (c) shows the answers that Cufflinks[28], Scripture[9], and IsoInfer[6] might come up with, this answer is equaly possible for them to come up with as the answer in Figure 1.2 (b); that's because they don't use the expression level of the gene segments information which makes it clear that the isoform $ex1 - ex3 - ex4$ cannot coexist with the isoform $ex1 - ex2 - ex3 - ex4$. It is worth mentioning that in this case, IsoLasso does not come up with the right answer either, because of its limitation with considering only maximal paths; we will discuss about this shortcoming more, later in this chapter.

The framework of our method is very similar to that of IsoLasso[14]. However, there are some differences; the major distinction is that, in our method, we consider all possible paths of the splice graph, no matter what their lengths are, while in IsoLasso they only consider the maximal paths as the candidate isoforms. In other terms, in IsoLasso, considering that u and v are nodes in the splice graph:

$$u \text{ can be the start of a path iff } deg_{in}(u) = 0 \tag{1.2}$$

$$and \tag{1.3}$$

$$v \text{ can be the end of a path iff } deg_{out(v)} = 0 \tag{1.4}$$

Where

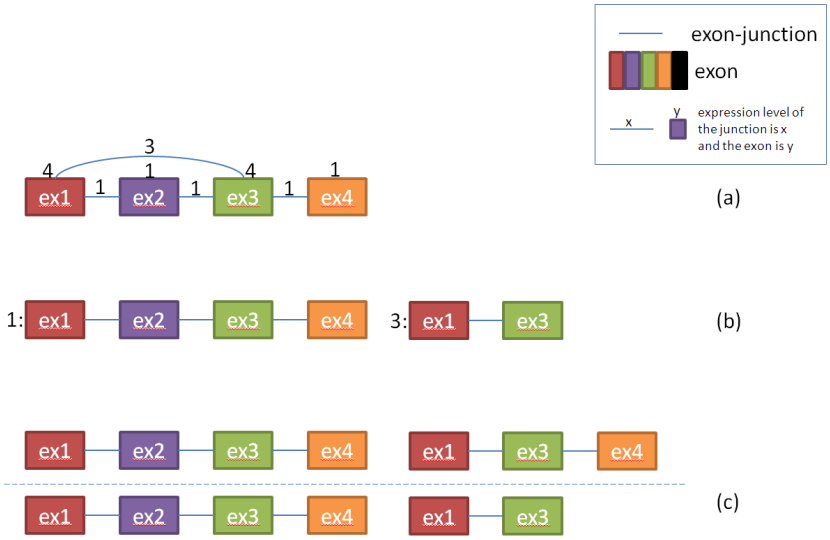

Figure 1.2: A simple example showing why the expression of exons and junctions is important to consider while detecting isoforms. An imaginary simple gene with 4 exons ex1, ex2, ex3, ex4 is shown; (a) the splice graph is depicted, with the nodes being the exons, and the edges being the present junctions. Note that the number on each edge or node is the expression level of that exon or junction. (b) the correct set of isoforms, where expression level of each isoform is stated to its left. (c) 2 of the possible solutions that programs like Cufflinks which predict the isoforms and estimate the expression level in two different steps predict.

- $deg_{in}(u)$ equals the number of edges whose right end is the node u,
- and $deg_{out}(v)$ equals the number of edges whose left end is the node v.

This makes our approach more flexible and complete, because in reality not only the maximal length isoforms are transcribed, but also the shorter isoforms may be transcribed in the gene, since there are cases that two transcribed isoforms both contain a sequence of exons, while one ends in the middle of the other one. Consider the simple example in Figure 1.3; given the splice graph in Figure 1.3 (a) and the answers in Figure 1.3 (b), it is shown that one of the two isoforms end at ex4, while $deg_{out(v)} \neq 0$; this means that the isoform $ex1-ex2-ex4$ is not in the set of possible isoforms for IsoLasso in the first place (IsoLasso, like LiPID, makes a set of possible isoforms from the splice graph, which is its reference for determining the present isoforms later), leading to IsoLasso not being able to predict its presence in the transcriptome.

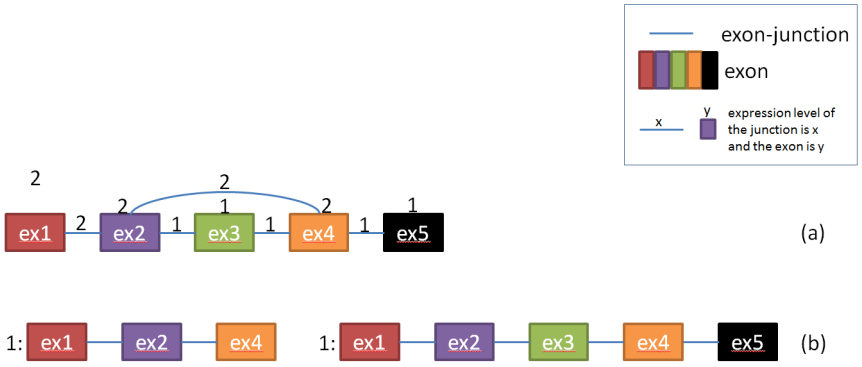

Figure 1.3: A simple example showing the shortcoming of IsoLasso. An imaginary simple gene with 5 exons ex1, ex2, ex3, ex4, ex5 is shown; (a) the splice graph is depicted, with the nodes being the exons, and the edges being the present junctions. Note that the number on each edge or node is the expression level of that exon or junction. (b) the correct set of isoforms, where expression level of each isoform is stated to its left.

1.4 Thesis Structure

The rest of this thesis is structured as follows:

- **Method** In chapter 2, we will describe the algorithms used in LiPID's implementation, and we will go over a comprehensive example for better understanding of the method.
- **Results** In chapter 3, we show the output of different experiments we did with simulated data, and discuss about LiPID's accuracy, sensitivity, and robustness compared to that of Cufflinks and IsoLasso. We also show the results of running LiPID on some validated real prostate cell lines to prove its efficiency.
- **Conclusion** Chapter 4 concludes this thesis, where we will give a summary of the work, together with ways of how to improve LiPID's performance and to make it incorporate other transcriptomic events in its prediction process.

Chapter 2

Method

Here we propose how our algorithm (LiPID) works, inputs and outputs of each section, and the complexity of the algorithm:

LiPID takes as input, RNA-seq reads of the region (gene) for which it is intended to predict the isoforms, together with an annotation file (bed or gtf file format[1]) which specifies the start and end coordinates of gene segments (exons), and a gene name. It is worth mentioning that for using LiPID, having access to reference is also necessary.

What LiPID outputs is the set of isoforms that it has predicted together with the relative expression level of each; isoforms are shown in the format of concatenation of the names that the input annotation provides for the exons.

In summary, here's how LiPID solves the isoform detection and abundance estimation problem:

- Based on the gene name and the coordinates present in the annotations, LiPID extracts the sequences of the gene's exons from the specified chromosome in the reference genome, then it creates a fasta file. In the first part of the file it adds each one of the exon sequences present in the annotation of the gene together with their genomic sequence, in the following lines every two exons and their sequences are concatenated to form an exon-exon junction. This fasta file will later on be used to map the reads onto.[2]

[1] for more information about BED and GTF annotation track format, you can visit http://genome.ucsc.edu/FAQ/FAQformat.html

[2] for more information about fasta file format, you can visit http://en.wikipedia.org/wiki/FASTA_format

- LiPID maps the reads given in the input onto the fasta file created in the previous step using mrFAST[1]. The output is the mapped file containing the mapping location and the exon or junction where each read maps to.
- In order to count the number of hits for each exon[3], LiPID counts the number of lines (i.e reads) that shows is mapping to a location within to each exon. To count the hits for each exon-exon junction, LiPID again counts the number of reads mapped to the concatenated corresponding exon sequences, but only the ones that span the boundary with some safe distance.
- Based on the present exons and junctions (the ones that have a reasonable number of read hits), LiPID calculates all possible isoforms.
- Considering all possible isoforms, LiPID builds an LP file[4] with constraints being summation of isoforms up to their common exons and junctions, and the optimization function being "maximizing the length of the isoforms and minimizing the error in expression levels due to ".
- LiPID feeds the above LP file to IBM ILOG CPLEX[4], which is a Linear Programming solver, and filters the output so that the best answer is the one with the least number of isoforms while it satisfies all the constraints and optimizes the optimization function.

Below comes detailed description for each step:

2.1 Extracting the Reference Sequence

The gene name, the chromosome, and the annotation file are all given. LiPID then searches for the gene name in the annotation file, finds the exons within that gene, and extracts the sequences for those exons from the corresponding chromosome. Later on in section 2.6, we will show an example with which we will go through all of the phases of running LiPID, and the way it works.

[3]hit here is the number of reads mapped to that exon

[4]for more information about lp file format, you can visit http://lpsolve.sourceforge.net/5.5/lp-format.htm

2.2 Mapping and Expression Calculation

In this step, LiPID uses mrFAST[1] to map the reads onto the reference sequence that was obtained from the previous step. mrFAST[1] is used because when comparing sequences, it considers edit distance instead of only mismatches. This is a useful property because in some cases of transcription, when two exons are to be concatenated, they might not be concatenated exactly at their boundaries, rather there might be a couple nucleotide shift in the boundary, which will cause the junction not to be the exact concatenation of the two exons, but that with some deletions or insertions.

After indexing the reference sequence, LiPID uses mrFAST[1] with $-e$ option to indicate the edit distance; edit distance can vary between 0 and 4, specified by the user, and that would be decided based on the type of data. For example for real data, one would use a higher edit distance, because of the susceptibility of the data to errors due to sampling and the natural transcription itself, however for simulation data, 0 edit distance would work, unless some sequences are altered on purpose to add errors.

Once the mapping is done, LiPID counts the number of reads that have been mapped to each exon and each junction as follows:

- A read will be counted as mapped to an exon, iff the whole read is mapped within the boundary of that exon.
- A read will be counted as mapped to a junction, iff it spans at least 15 nucleotides of each of its two adjacent exons. (15 is confirmed by biologist to gives us a certain amount of accuracy)

After mapping is done and the reads are counted for each region, we need to normalize the read count of each region based on their length; that's because more reads map to longer exons, while we need to be able to compare different regions in the same metric space. Here's the pattern:

$$el(exon_i) = \frac{count(exon_i)}{length(exon_i) - readLength + indel} \tag{2.1}$$

$$el(juc_i) = \frac{count(juc_i)}{readLength - spanSize + indel} \tag{2.2}$$

where

- $el(exon_i)$ is the normalized expression level of $exon_i$,

- $el(juc_i)$ is the normalized expression level of juc_i,
- $count(juc_i)$ is the number of reads that have been mapped to juc_i,
- $readLength$ is the length of the short reads (usual lengths are 36, 50, 75, 100),
- $spanSize$ is number of nucleotides that a read must span within the exon-exon boundary for it to be considered mapping their junction,
- and $indel$ is the number of allowed indels (i.e 0..4).

Gene segments might be of different lengths. The shorter they are, the more error-prone they can be with respect to the reads mapped to them. In order to fix the problem of low read hit to short-length exons, LiPID does a correction step. It categorizes the exons into three groups:

1. $exon_i$ is short-sized if: $length(exon_i) \leq readLength - indel$
2. $exon_i$ is medium-sized if: $readLength - indel \leq length(exon_i) \leq 1.5 \times readLength$
3. $exon_i$ is normal-sized if it is not included in the last two categories.

For an exon of case 1, no read would map to it, so LiPID concatenates it to its shorter adjacent exon (we explained above why this action is legitimate). For an exon of case 2, in addition to the reads that has been mapped within its coordinates, LiPID counts the reads that have been mapped to its corresponding junctions as its expression level (we take into consideration the normalization factor based on all the nucleotides that have been counted as a hit for that exon).

Please note that LiPID does the correction on short-sized exons before mapping, and map onto the new concatenated exon (excludes the sequences of the short-sized exon and its adjacent exon to which it has been concatenated, and includes the concatenation). And the correction on medium-sized exons' expression is done after mapping.

LiPID also does a pruning step in which it discards the junctions whose expression levels are less than $\frac{1}{10}th$ of the average of all exons expression levels. That's because these junction expressions came into existance merely because of noise, but their presence will complicate the problem afterwards, when the isoforms are created, and the LP problem is to be solved.

2.3 Building the set of possible isoforms

In order to predict the desired set of isoforms, we need to provide an initial set of isoforms for the program to choose from. LiPID uses a recursive algorithm to enumerate the set of all possible present isoforms given the expression levels of the exons and junctions. Unlike other algorithms in the context, not to lose a single possibility, we are interested in all the isoforms that are probable to exist with the obtained exons and junctions expression levels; no matter what the length of the isoform is, we build all of the possible isoforms, so that later the Linear Program can decide which ones would satisfy the conditions better. Here's how the algorithm works:

- First we consider the exons and junctions relationship as a graph, in which nodes are exons, and edges are junctions between two exons. Based on this graph, we build an adjacency matrix, M, and here's how we fill it:
  - If the junction between $exon_i$ and $exon_j$ is present(i.e has some reads mapped to it), then the matrix cell $M_{i,j}$ (and not $M_{j,i}$) will be set to 1. Please note that junctions can only be from a lower indexed exon to a higher indexed exon, so the adjacency matrix will become an upper triangular matrix. For example, for the graph of Figure 1.3(a), matrix M would look like the following:

$$\begin{array}{c} \\ ex1 \\ ex2 \\ ex3 \\ ex4 \\ ex5 \end{array} \begin{array}{c} \begin{array}{ccccc} ex1 & ex2 & ex3 & ex4 & ex5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

- Then the so called findAll method is called, which is a recursive method that is called in a loop for each node, and passes a buffer of the previous nodes in the path to the next recursive call, and in its call, it calls the recursive method for each node that the current node is connected to. Each time it is called, it prints out the sequence of nodes it has in the buffer, i.e an isoform. Hence, through the end of the adjacency matrix, all of the possible isoforms are enumerated and printed out.

Below you can see the signature of findAll:

$$void \text{ findAll}(int \text{ head}, vector < int > * \text{ buffer}, int \text{ level}) \tag{2.3}$$

We start by the $buffer$ being empty, and $level = 0$ and in a loop call findAll with $head = 0$ to $head = numOfNodes$, then at each step we print out the current node (head) attached to the buffer, and then add $head$ to the buffer, and call findAll for all adjacent nodes to $head$. This Recursive function prints out 2^n isoforms in case we have n exons and all of them are connected to each other. So the worst case time complexity is $O(2^n)$, which is exponential, however this is almost never the case, since only few isoforms are present in each sample and those few consist of a few number of junctions which will show up in sampling and then calculating the expression levels, leading in not many edges being present.

The following shows what findAll outputs when we call it for all 5 nodes of graph of Figure 1.3(a):

$$ex1 \tag{2.4}$$
$$ex1 - ex2 \tag{2.5}$$
$$ex1 - ex2 - ex3 \tag{2.6}$$
$$ex1 - ex2 - ex4 \tag{2.7}$$
$$ex1 - ex2 - ex3 - ex4 \tag{2.8}$$
$$ex1 - ex2 - ex4 - ex5 \tag{2.9}$$
$$ex1 - ex2 - ex3 - ex4 - ex5 \tag{2.10}$$
$$ex2 \tag{2.11}$$
$$ex2 - ex3 \tag{2.12}$$
$$ex2 - ex4 \tag{2.13}$$
$$ex2 - ex3 - ex4 \tag{2.14}$$
$$ex2 - ex4 - ex5 \tag{2.15}$$
$$ex2 - ex3 - ex4 - ex5 \tag{2.16}$$
$$ex3 \tag{2.17}$$
$$ex3 - ex4 \tag{2.18}$$
$$ex3 - ex4 - ex5 \tag{2.19}$$
$$ex4 \tag{2.20}$$
$$ex4 - ex5 \tag{2.21}$$
$$ex5 \tag{2.22}$$

2.4 Building the LP file

LiPID uses a Linear Programming Solver (IBM ILOG CPLEX[4]) to detect the isoforms and estimate their abundance; hence we need to make a linear programming input file based on our constraints and objective function, and translate it into an lp solver's recognized language. Here we use LP file format, which is of the form:

$< objectivefunction >$
$< constraint >$
$< declaration >$

- $< Objectivefunction >$ is a linear combination of variables and constants, ending with a semicolon, preceded by "max:" or "min:" to indicate whether you want it to be minimized or maximized.
- $< constraint >$ is an optional constraint name followed by a colon plus a linear combination of variables and constants, followed by a relational operator, followed again by a linear combination of variables and constants, ending with a semicolon.
- $< declaration >$ is to define a variable, and is of the form $typevar1, var2, var3, ...$; where *type* can be one of "int", "bin", ...

For example, the simple problem:
$x1 >= 1$
$x2 >= 1$
$x1 + x2 >= 2$
minimize $x1 + x2 (= -(-x1 - x2))$, $x1$ being integer

Can be written as following in lp-format:
$-x1 - x2;$
/* or min: $x1 + x2;$ */
$x1 >= 1;$
$x2 >= 1;$
$x1 + x2 >= 2;$
int $x1;$

Constraints

Based on how genes are transcribed and the fact that the samples are taken after transcription, there is a relationship between the expression level of the isoforms which are present in the gene and the expression level of the exons and junctions of that gene. That is:

$$\forall iso_j | exon_i \in iso_j \sum_{j=1}^{n} el(iso_j) = el(exon_i) \, and \tag{2.23}$$

$$\forall iso_j | juc_i \in iso_j \sum_{j=1}^{n} el(iso_j) = el(juc_i) \tag{2.24}$$

Where

- iso_j represents an isoform,
- juc_i represents a junction,
- $iso_1, ..., iso_n$ are all of the isoforms that contain $exon_i/juc_i$,
- and $el(x)$ is the expression level of exon/juntion/isoform x.

So the relation between the isoforms which contain an exon/junction and the corresponding exon/junction will be one part of the constraints of the Linear Programming. In each constraint, we consider an error because there might be some experimental errors (sampling and mapping errors), and the expression levels do not exactly match. So the constraints that we put in the Linear Programming would become:

$$\forall iso_j | exon_i \in iso_j \sum_{j=1}^{n} el(iso_j) + error_i = el(exon_i) \tag{2.25}$$

$$\forall iso_j | juc_i \in iso_j \sum_{j=1}^{m} el(iso_j) + error_i = el(juc_i) \tag{2.26}$$

Where $error_i$ is the corresponding error considered for $exon_i$ or juc_i.
Then we'll have another set of constraints like

$$Iisoform1 - MDisoform1 <= 0 \tag{2.27}$$

$$Iisoform1 - LDisoform1 >= 0 \tag{2.28}$$

Where

- $Iisoform_j$ is the expression level of $isoform_j$,

- $Disoform_j$ is a binary variable and indicates whether or not $isoform_j$ exists, i.e if $Disoform_j = 0$, then $Iisoform_j = 0$ as well, and if $Disoform_j = 1$, then $Iisoform_j$ has a value more than 1.
- and M is the upper bound and L is the lower bound for each isoform's expression level.

These two sets of inequalities are written in the constraints for all of the isoforms, and the reason they're there is for the LP solver not to assign very small or very large values to the isoforms. M and L are assigned very loosely, M is just equal to the maximum expression level of all the exons (no isoform can be expressed more than its exons are), and L is a the minimum difference among all exons and junction expression levels. They are set so loosely just to avoid extreme cases, especially L is for the isoforms that are present not to be 0 or negative.

Also note the subtlety in these set of inequalities; if $Disoform_j$ is equal to zero, then $Iisoform_j$ would have to be equal to zero, too, as one inequality obliges it to be less than or equal to zero, and the other more than or equal to zero. So somehow, this pair of inequalities represents the following inequality:

$$\begin{cases} Iisoform_j = 0 & \text{if } Disoform_j = 0, \\ L \leq Iisoform_j \leq M & \text{else.} \end{cases} \equiv \begin{cases} Iisoform_j - MDisoform_j \leq 0 \\ Iisoform_j - LDisoform_j \geq 0 \end{cases} \tag{2.29}$$

In fact, by this we are saving our program to become quadratic, yet keeping it simpler, and much more efficient. That's because if we had not come up with this formula, we had to use multiplication of $Disoform_j \times Iisoform_j$ in the objective function, so that when $Disoform_i = 0$, $Iisoform_i$ does not affect the objective function. However we have a more efficient formula now, where $Disoform_j$ and $Iisoform_j$ are correlated.

After we add these inequality pairs for the whole set of possible exons, we want to constrain errors, i.e er_i; since we want to add these errors to the optimization function, we would like them to be positive, but it would not make sense if we allow only positive errors, and negative errors are as likely as positive ones are. Hence, we should just put the absolute value of the errors in the optimization function. Intuitively, the less the absolute value of the error is, the more desirable it is for us, no matter whether it is positive or negative. In the following constraints, we introduce another set of variables err_i, that are equal to

absolute value of the errors used in the above equations for expression levels of isoforms, exons and junctions. So we will have:

$$\forall err_i \geq 0 \; err_i = |er_i| \tag{2.30}$$

We should consider another constraint which would be the maximum number of isoforms that we want to predict. We set this constraint because our ultimate goal is to discover the minimum number of isoforms (C) that can represent the splicing graph best, i.e the minimum path cover. We start setting C to 2, and run the solver to predict the isoforms. We repeat this process by incrementing C by one in each step, until there is slight change in the isoforms predicted. At this point, regardless of how big C becomes, the set of isoforms would be the same. Then we would choose the result of the experiment that has the least C as the minimum number of isoforms, and has little difference with its next step; meaning, for example if 3 isoforms with certain expression levels suffice the expression levels given by the exons and junctions, yet the 4 isoforms improve this difference just a little bit, we would still choose the 3 isforoms, because it is one of our goals to get as few isoforms as possible. Based on our experiments, we concluded that in the first event that the difference between sum of errors in step s is less than $\frac{1}{10}th$ of the sum of errors in step $s-1$, then we would terminate the process, and return the solution obtained from step $s-1$. More details about the determination of the best answer will be provided in section 2.4. The constraint would be:

$$\sum_{i=1}^{K} Disoform_i \leq C \tag{2.31}$$

Where K is the number of possible isoforms.

Declarations

As mentioned in the beginning of this chapter, there is a declaration part in lp file format, where you get to define your variables. The default type for variables are floating point numbers, so only if your variable is otherwise, you need to define it. In this part, we will specify our binary variables as:

$$BINARY \tag{2.32}$$

$$Disoform_i \tag{2.33}$$

Where for each $isoform_i$, with $1 \leq i \leq N$, (N is the number of all possible isoforms) we should add a separate line.

Optimization Function

Optimization function should be put in the beginning of the lp file, but since we needed to define the variables and constraints first, this section comes after the other two.

Based on what biologists have observed in gene transcription, the number of isoforms are parsimonious, and they are usually long. These reads that we have are sampled from the cell after transcription, and they have the information of the formed isoforms. Therefore, all we need is to set some ground rules in order to prioritize a set of satisfying isoforms over another. Based on the parsimony principle, Dilworth's law and many discussions with professionals, we came up with the following:

The most desirable set of isoforms is the one that has the least number of isoforms, which are as long as possible, and satisfy the expression levels of the exons and junctions as good as possible.

The following optimization function leads us to the above goal:

$$maximize \left[\sum_{i=1}^{K} w_i I isoform_i - \sum_{j=1}^{N+M} w'_j err_j \right] \tag{2.34}$$

where

- K is the number of possible isoforms,
- N is the number of exons,
- M is the number of exon junctions,
- w_i is the weight we give to $isoform_i$ expression level,
- w'_j is the weight we give to error we allow in calculating $exon_j/juc_j$.

It is apparent that both $Iisoform_i$ and err_j have the same unit; they both represent expression level. However, the weights are to decide how we want them to affect the optimization function. Also the scale of these errors matter, since the more different the errors are, the more variance we are allowing for the ones with smaller weights; since for the one with a large weight, a small change in value would scale to *weight times* more. At the same time, we should give more weight to the variables that are more desirable, to affect the optimization function more. Therefore, we should choose the weights in a way that both goals are fulfilled; the more desirable variables are emphasized more, and yet the weights are not that far apart to allow arbitrary value assignment to variables.

From a biological point of view, we do want to predict the longest isoforms, because the longer isoforms are more likely to be present in the gene. So we would prefer to give the longer isoforms a bigger weight compared to the shorter ones. For the isoforms' expression levels, we chose a polynomial function based on the length of the isoforms, as the following:

$$w_i = length(Iisoform_i)^z \ \forall z \in \mathbb{N}, z < 10 \tag{2.35}$$

For the errors' weights, we decided to go with a polynomial function of the number of isoforms in the whole gene, like N^2(with N being the number of isoforms in the whole gene). We would also tend to give some more flexibility to shorter exons, so if $exon_j$'s length is less than $readSize + 20$ ($exon_j$ is the medium exons category), then we would set its corresponding err_j as N. That is because of the fact that the shorter an exon/junction is, the more probable it is for the number of reads mapped to it to be erroneous.

This way, we are allowing smaller error for the longer exons. So the error weight would become:

$$w'_j = \begin{cases} N & \text{if } \ exJuc_j \text{ is an exon and } lengthN(exJuc_j) \leq readSize + 20\,, \\ N^2 & \text{if otherwise}\,. \end{cases} \tag{2.36}$$

Where

- $exJuc_j$ is the exon/junction that err_j corresponds to,
- and $lengthN(exJuc_j)$ returns the number of nucleotides that exon/junction err_j corresponds to, contains.

By this definition for w_is and w'_js, both of our concerns seem to be fulfilled; different categories of the optimization are not far apart, yet the more desirable variables are more stressed. Note that these values are obtained after several experiments to find the optimum parameters.

It is worth mentioning that this optimization function is decided on after several experiments with different functions; for example, we first tried to minimize the number of isoforms, and thought it would indirectly maximize the lengths of the isoforms. In some cases this worked just as well as the current function, but in others, especially in more erroneous samples it tended to come up with isoforms that consisted of single exons, and we ended up having several single-exon isoforms as the result. We tried several different approaches as well, but the current optimization function proved to be the most consistent and the most reasonable (in terms of how closely it simulates the biological problem) among all.

2.5 Solving the Linear Program and filtering output

Now that the lp file is ready, we can input it into CPLEX (the lp solver we are using) for it to detect the isoforms and their ratios.

Once the results are out, it's not always perfect, especially since for each time we try to solve the program, we need to specify the maximum number of isoforms, while we don't know how many isoforms there are to specify this max correctly. On the other hand, we cannot set this max to a large number and hope to get the least number of isoforms possible, because we have no other obligations in the optimization function to force least number of isoforms possible. In order to get the optimum set of isoforms, we would have to repeat the experiment for several values of max number of isoforms; the lp problem file would remain the same, just the constant C in the following constraint should change for each iteration:

$$\sum_{i=1}^{K} Disoform_i \leq C \tag{2.37}$$

We start with $C = 2$, and in each step increment C by 1; when the number of isoforms in the solutions for two consecutive steps are identical, we stop the iterations. It is now time to investigate the results; we start from $C = 1$, and in each comparison step, compare the solution of $C = c1$ to the solution of $C = c1 + 1$, there are two cases in this matter:

1. $\sum_{i=1}^{K} Disoform_i$ for $C = c1$ and $C = c1 + 1$ are equal.

   In this case, the solution for $C = c1$ is chosen and returned as the final solution; this ought to be the last step, since as indicated in the previous paragraph, we terminate the iteration after such condition is observed. So this indicates that even if we allow more isoforms, it still tends to detect the same number of isoforms, hence this solution tends to persist, and this shows that it probably fulfills the expression level constraints well enough, also since we are interested in as few number of isoforms as possible, we report this as the final solution.

2. $\sum_{i=1}^{K} Disoform_i$ for $C = c1 + 1$ equals $\sum_{i=1}^{K} Disoform_i$ for $C = c1$ plus 1. In this case we check how much each set of isoforms fulfills the expression levels, i.e, we just need to check the errors and see if the errors in the two sets have significant difference; i.e if $\sum_{i=1}^{L} err_i$ (L being the number of expressed exons and junctions) for $C = c1$ is less than $\frac{1}{10}th$ of $\sum_{i=1}^{L} err_i$ for $C = c1 + 1$, then we would terminate the process, and return the solution obtained from step $C = c1$, otherwise we go one step further.

2.6 Example

Here we show the solutions obtained from running the solver for the MYC gene[5], and what is input and output for each step:

We first build the lp file, and then in different iterations, we would increment C, and analyze the solutions. After getting the results, we filter the solution to extract just the information we need.

MYC is located in chromose 8, it consists of 6 exons, and here are its exons and their coordinates in the annotations file:

[5]MYC is a regulator gene that codes for a transcription factor. In the human genome, MYC is believed to regulate expression of 15% of all genes

Segment Name	Gene Name	Chromosome	Segment Start	Segment End
ex505	MYC	chr8	128816861	128817498
ex506	MYC	chr8	128817498	128818051
ex507	MYC	chr8	128819675	128820447
ex508	MYC	chr8	128821823	128821826
ex509	MYC	chr8	128821826	128821905
ex510	MYC	chr8	128821905	128822853

2.6.1 Extracting the reference sequence

Based on the above information, which is input to LiPID, the program searches chromosome 8 fasta file to extract the corresponding sequence for each exon.

Note that ex508 and ex509 are concatenated together, because one of them is short. In this step, based on the exons' coordinates, LiPID figures out the nucleotide length of each exon, and if an exon's length is less than the read length, it will concatenate that exon to its adjacent shorter exon. This is done because if the exon length is shorter than the read length, then no reads will map onto it.

Also note that, ex's in this example annotation are not real exons, but they are partial exonic regions, therefore we are allowed to concatenate them, and we can also treat them as if they are full exons.

2.6.2 Expression calculation and possible isoforms building

Let's assume that we already mapped MYC's reads to the reference gene that we built in the previous section, and after doing the short-exon and medium-exon fixing and counting all the expression levels of exons and juction, normalizing them, and filtering out the lowly expressed junctions, here's the result:

Segment Name	Expression Level
ex505	1.12
ex506	3.47
ex507	1.13
ex508Hex509	3.58
ex510	3.44
ex505_ex506	1.2
ex506_ex507	1
ex506_ex508Hex509	2.3
ex507_ex508Hex509	1.17
ex508Hex509_ex510	3.46

So the present junctions as we see in the table, are
ex505_ex506, ex506_ex507, ex506_ex508Hex509
, ex507_ex508Hex509, and ex508Hex509_ex510

and therefore the set of possible isoforms would become:

ex505
ex505–ex506
ex505–ex506–ex507
ex505–ex506–ex507–ex508Hex509
ex505–ex506–ex507–ex508Hex509–ex510
ex505–ex506–ex508Hex509
ex505–ex506–ex508Hex509–ex510
ex506
ex506–ex507
ex506–ex507–ex508Hex509
ex506–ex507–ex508Hex509–ex510
ex506–ex508Hex509
ex506–ex508Hex509–ex510
ex507
ex507–ex508Hex509
ex507–ex508Hex509–ex510
ex508Hex509
ex508Hex509–ex510
ex510

2.6.3 Building the LP file

Once the expression levels of exons and junctions are calculated and the set of possible isoforms is built, the lp file will be made.

As in the instructions in section 2.4, the constraints part would become:

$$Iisoform1 + Iisoform2 + Iisoform3 + Iisoform4 \tag{2.38}$$
$$+Iisoform5 + Iisoform6 + Iisoform7 + er0 \tag{2.39}$$
$$= 1.12 \tag{2.40}$$
$$Iisoform2 + Iisoform3 + Iisoform4 + Iisoform5 \tag{2.41}$$
$$+Iisoform6 + Iisoform7 + Iisoform8 + Iisoform9 \tag{2.42}$$
$$+Iisoform10 + Iisoform11 + Iisoform12 + Iisoform13 \tag{2.43}$$
$$+er1 = 3.47 \tag{2.44}$$
$$Iisoform3 + Iisoform4 + Iisoform5 + Iisoform9 \tag{2.45}$$
$$+Iisoform10 + Iisoform11 + Iisoform14 + Iisoform15 \tag{2.46}$$
$$+Iisoform16 + er2 = 1.13 \tag{2.47}$$
$$Iisoform4 + Iisoform5 + Iisoform6 + Iisoform7 \tag{2.48}$$
$$+Iisoform10 + Iisoform11 + Iisoform12 + Iisoform13 \tag{2.49}$$
$$+Iisoform15 + Iisoform16 + Iisoform17 + Iisoform18 \tag{2.50}$$
$$+er3 = 3.58 \tag{2.51}$$
$$Iisoform5 + Iisoform7 + Iisoform11 + Iisoform13 \tag{2.52}$$
$$+Iisoform16 + Iisoform18 + Iisoform19 + er4 \tag{2.53}$$
$$= 3.44 \tag{2.54}$$
$$Iisoform2 + Iisoform3 + Iisoform4 + Iisoform5 \tag{2.55}$$
$$+Iisoform6 + Iisoform7 + er5 = 1.2 \tag{2.56}$$
$$Iisoform3 + Iisoform4 + Iisoform5 + Iisoform9 \tag{2.57}$$
$$+Iisoform10 + Iisoform11 + er6 = 1 \tag{2.58}$$
$$Iisoform6 + Iisoform7 + Iisoform12 + Iisoform13 \tag{2.59}$$
$$+er7 = 2.3 \tag{2.60}$$
$$Iisoform4 + Iisoform5 + Iisoform10 + Iisoform11 \tag{2.61}$$
$$+Iisoform15 + Iisoform16 + er8 = 1.17 \tag{2.62}$$
$$Iisoform5 + Iisoform7 + Iisoform11 + Iisoform13 \tag{2.63}$$
$$+Iisoform16 + Iisoform18 + er9 = 3.46 \tag{2.64}$$

And here's how the isoforms' names are mapped to their exonic parts:

isoform1	ex505
isoform2	ex505–ex506
isoform3	ex505–ex506–ex507
isoform4	ex505–ex506–ex507–ex508Hex509
isoform5	ex505–ex506–ex507–ex508Hex509–psecondsr113510
isoform6	ex505–ex506–ex508Hex509
isoform7	ex505–ex506–ex508Hex509–ex510
isoform8	ex506
isoform9	ex506–ex507
isoform10	ex506–ex507–ex508Hex509
isoform11	ex506–ex507–ex508Hex509–ex510
isoform12	ex506–ex508Hex509
isoform13	ex506–ex508Hex509–ex510
isoform14	ex507
isoform15	ex507–ex508Hex509
isoform16	ex507–ex508Hex509–ex510
isoform17	ex508Hex509
isoform18	ex508Hex509–ex510
isoform19	ex510

Where

- $Iisoform_j$ is the expression level of $isoform_j$,
- and er_k is the error concerning in the expression level of $exon_k/juc_k$.

And here's another constraint on the maximum number of isoforms possible:

$$Disoform_0 + Disoform_1 + Disoform_2 + Disoform_3 + Disoform_4 \tag{2.65}$$

$$+Disoform_5 + Disoform_6 + Disoform_7 + Disoform_8 + Disoform_9 \tag{2.66}$$

$$+Disoform_10 + Disoform_11 + Disoform_12 + Disoform_13 \tag{2.67}$$

$$+Disoform_14 Disoform_15 Disoform_16 Disoform_17 + Disoform_18 \tag{2.68}$$

$$+Disoform_19 \leq C \tag{2.69}$$

Once the constraints are build, we need to add the optimization function:

$$Maximize \tag{2.70}$$

$$obj: \tag{2.71}$$

$$+1Disoform1 + 8Disoform2 + 27Disoform3 + 64Disoform4 \tag{2.72}$$

$$+125Disoform5 + 27Disoform6 + 64Disoform7 + 1Disoform8 \tag{2.73}$$

$$+8Disoform9 + 27Disoform10 + 64Disoform11 + 8Disoform12 \tag{2.74}$$

$$+27Disoform13 + 1Disoform14 + 8Disoform15 + 27Disoform16 \tag{2.75}$$

$$+1Disoform17 + 8Disoform18 + 1Disoform19 \tag{2.76}$$

$$-25err0 - 25err1 - 25err2 - 25err3 - 25err4 \tag{2.77}$$

$$-25err5 - 25err6 - 25err7 - 25err8 - 25err9 \tag{2.78}$$

2.6.4 Solving the linear program and filtering output

Now that out lp file is ready, we feed it into IBM ILOG CPLEX[4], note that LiPID does several iteration with different Cs and then compares the results in order to choose the one that best suits the goal.

CPLEX[4] returns one solution for us; the actual solution from contains more information than you see here, but here are the results after filtering extra information in the solution file:

for $C = 1$

ex506–ex508Hex509–ex510	2.3

for $C = 2$

ex505–ex506–ex507–ex508Hex509–ex510	1
ex506–ex508Hex509–ex510	2.3

for $C = 3$

ex505–ex506	1.12
ex506–ex508Hex509–ex510	2.3
ex507–ex508Hex509–ex510	1.13

For each iteration, we compare it with the previous one. For $C = 1$ and $C = 2$, it is case

2^6, where one has more isoforms than the other. In this case, we would have to compare the sum of errors assigned in each case. We can extract this information from the solution file, since the errors are among the variables and their values are assigned by solving the linear program.

In our case the error values are:

for $C = 1$

$$err0 = 1.12, err1 = 1.17, err2 = 1.13, err3 = 1.28, err4 = 1.14, err5 = 1.2,$$

$$err6 = 1, err7 = 0, err8 = 1.17, err9 = 1.16$$

for $C = 2$

$$err0 = 0.12, err1 = 0.17, err2 = 0.13, err3 = 0.28, err4 = 0.14, err5 = 0.2,$$

$$err6 = 0, err7 = 0, err8 = 0.17, err9 = 0.16$$

Please note that we look at err_i instead of er_i because the former is always positive, and we would like to compare the absolute values of the errors together.

As you can see above, for $C = 1$, $\sum_{i=1}^{9} err_i = 10.37$, and for $C = 2$, $\sum_{i=1}^{9} err_i = 1.37$. As we see here, the difference between the errors in the two cases are huge (we consider a difference between errors huge, if one is more than twice the other), and so we conclude that $C = 2$ is better than $C = 1$ so far. Now let's examine the errors in $C = 3$ case:

for $C = 3$

$$err0 = 0, err1 = 0.05, err2 = 0, err3 = 0.15, err4 = 0.01, err5 = 0.07,$$

$$err6 = 1, err7 = 0, err8 = 0.04, err9 = 0.03$$

We see that for $C = 3$, whose solution is not much different from the case of $C = 3$, the sum of errors is $\sum_{i=1}^{9} err_i = 1.35$, compared to $\sum_{i=1}^{9} err_i = 1.37$ for $C = 2$. Therefore, we conclude that $C = 2$ works better than $C = 3$, since it has fewer isoforms, yet its sum of errors is not much different from that of $C = 3$.

As a result, LiPID will return the following solution as the result of the simulated reads above for the MYC gene:

[6] refer to section 2.5 for more details about these cases

ex505–ex506–ex507–ex508Hex509–ex510	1
ex506–ex508Hex509–ex510	2.3

And here are the isoforms that we sampled the reads from in the first place:

ex505–ex506–ex507–ex508Hex509–ex510	1
ex506–ex508Hex509–ex510	2

Where the number beside each isoform shows its expression level.

Chapter 3

Results

There are plenty of RNA-seq samples from real tissues to use, however there are very few of them that are validated; that's because the validations are very costly, and yet not 100% accurate. With the new methods such as LiPID aiming for transcript detection and quantification, there will be more demand for the existence of real data, and eventually there will be more validated samples available for future use. For this thesis, we basically evaluated our method by the use of simulated data. We also tested LiPID on some validated real life data from prostate cancer tissues and cell lines, in order to prove LiPID's effectiveness on both simulated and real data. In this section, we compare the performance of LiPID with the two major available softwares in the area; IsoLasso and Cufflinks.

This section is written in the following way: first we explain how the simulated reads are generated, then we talk about the reasons for choosing different parameters for LiPID, then we will explain the result of running LiPID on simulated data, then LiPID's performance on some prostate cancel cell lines and tissues are demonstrated, and in the end, LiPID's performance is compared to that of Cufflinks and IsoLasso, by their results on the simulated data.

3.1 Simulator

The simulator that we developed works in the following manner:

Input: $geneName$, $chromosome$, $annotationsFileName$, $numberOfIsoforms$, $ratio$.
Output: reads.fa, isoforms.txt

- Given $geneName$, find all of its exons (gene parts) in the annotations file, and then based on their given locations, find their sequences in the reference genome, together with the junctions' sequences.
- Considering all of the exons within the gene, randomly generate $numberOfIsoforms$ distinct isoforms, whose number of exons are more than or equal to half of the whole number of exons for that gene. Save these isoforms together with their ratios in "isoforms.txt".
- Concatenate the sequences of the exons making up these isoforms, and from $isoform_i$ sample this many reads:
  $n = (lengthN(isoform_i) - readSize) \times coverage \times ratio$

  Note that sampling is unified random, meaning that, in n iterations, each time we use a random function to choose a location to sample a read from.

  Save the reads in reads.fa.

Here are the definitions for the names and variables used above:

- $geneName$ is the name of the gene we want to simulate reads for and sample from.
- $chromosome$ is the chromosome on which gene $geneName$ is located.
- $annotationsFileName$ is the path of the file where exons and their beginning and end coordinates are stored, together with the name of the gene they belong to.
- $numberOfIsoforms$ is the number of isoforms we want the simulator to generate.
- $ratio$ is a sequence of space delimited integers that indicates the relative expression levels of the isoforms we want to simulate; there should be $numberOfIsoforms$ integers in $ratio$.
- $lengthN(isoform_i)$ is a function that returns the number of nucleotides present in $isoform_i$.
- $coverage$ is the average number of reads representing one nucleotide.
- And n is the total number of reads that the simulator will generate.

3.2 Parameter Selection

There were several dilemmas in which we needed to choose the parameters' values in order to best match the purpose that LiPID was serving. Here we discuss these situations and why each parameter value was chosen:

3.2.1 Gene Coverage

We chose for the simulator to cover each base almost[1] 5 times. LiPID is supposed to work with RNA-seq data, and in the RNA-seq data, the coverage would be up to 100, however we did not need that high coverage, since we work on one gene at a time (much smaller area that the whole genome), and also we do not have significant sampling error. We tried coverage being 1, but it was too low, and the expression levels did not have high confidence with that low coverage. We also tried higher coverage (10), but since it did not add to the confidence of the expression levels, and the results were identical to that of coverage of 5, we chose 5 over 10. Also it's worth mentioning that the higher the coverage, the more time it'll take for the program to count the reads and calculate the expression levels, so we would like to stick to the lowest coverage possible, where accuracy is not sacrificed.

3.2.2 Read Size

We chose read size of 50 for all our simulations. For RNA-seq technology which is the main input for LiPID, the regular read size varies between 30 to 400 bp[2][32]. The lower the read size, the harder it gets to find a unique mapping for that read in the alignment step, therefore longer reads always align more accurate. However, since our intention was to simulate the reads as close to reality as possible, we chose read size of 50; most of the RNA-seq technologies can provide 50-base or longer reads, and we decided if LiPID could work with 50 bp reads, then it can work with higher read sizes for sure. So we chose this value to be on the safe side.

[1] almost, because the simulator chose the spots to sample from randomly, and hence not each location would be sampled the exact same number of times, however as the number of reads increases, these numbers become closer to each other

[2] base pair

3.2.3 Edit Distance

Edit distance indicates the number of bases you allow for the read to be different from the reference genome while mapping.

For simulation data, we do not need to consider erroneously sampled reads, because there is no error in the sampling here. The major errors in the real data sampling consist of two types:

1. **Transcription error** this occurs when the mRNA is being transcribed, i.e the transcription machinery is getting rid of the non-coding regions[3], and is concatenating the coding regions[4] together. In this process, some bases from either ends of an exon might be skipped, and/or some bases from either ends of an intron might be retained in the concatenation. And therefore, the genomic sequence of the junctions would not be exactly identical to the sequence of the junctions that are extracted from the reference genome, yet it has some base shift in the boundaries. As a result, the reads that will be sampled from these regions, would not map to the reference genome exactly, and so we have to allow some edit distance for these reads to map to where they belong within the reference genome.

2. **Sampling error** apart from the transcription machinery, the sequencing technology might make some errors when sampling reads from the genome, so as to skip some bases or record a certain base wrong.

Among different values that we tried for edit distance with real data, 4 seemed to be the most frugal for read size of 50, as it is not too large to have the reads map to the wrong location, yet it is not too small not to accommodate the transcription and sampling errors. Yet, for each read size, this value should be carefully set not to allow too many errors, or miss some valuable information.

3.2.4 Junction Span Size

Junction span size is the number of nucleotides a read should cover from each of its two end exons for the read to be considered as mapped on exons' corresponding junction. So

[3] introns

[4] exons

the read should cover at least a certain number of nucleotides from each exon, and as we talked to professionals in the area, they told us that 15 would be a reasonable number. We worked with different span sizes, from 4 to 20, to confirm that and found out that a number around 15 does works best. Another issue to consider here is the read size, meaning that this number does not necessarily work best for all read sizes. Since for 15, 30 nucleotides are fixed, and for the reads to be counted as mapped to that junction, they have 20 nucleotides for their mapping start point, i.e if the end location of $exon_l$ in $exon_l\_exon_r$ is n, only the reads mapped to $n-35$ to $n-15$ are acceptable for junction $exon_l\_exon_r$. So basically for shorter read sizes, we might have to work this around, try some examples, and see what value works best.

3.2.5 Short Exon Threshold

If an exonic region is shorter than the read size, then reads would not map to it, simply because it does not have enough bases for the reads to be compared upon. In these cases, before mapping, we would concatenate the short exon to its adjacent shorter exon, and consider the concatenation as a new exon.
Please note that in the annotation that we used, the exonic regions were not actual exons, but exon parts, so it is absolutely legitimate to concatenate them together if one region is very short. On the other hand, real exons are usually not this short, so basically in whole exon annotations there will be no need to concatenate short exons anyways, since there aren't any such cases present.

3.2.6 Medium Exon Threshold

Based on the same reasoning that we used for junction span size, the smaller exonic regions will get lower hits compared to longer ones; so based on the experiments that we did, it is not enough to normalize the hits just based on the length, and the smaller regions need further adjustments; for the exons with sizes less than $readSize+20$ (this value was obtained based on different experiments and observations), we also added a portion of the hits that were mapped to their corresponding junctions to their hits. This step was done after the mapping and counting was performed.

3.2.7 Normalizing expression levels

In order for the expression levels to be in the same scale, i.e comparable, we needed to normalize their read hits. Since during sampling(sequencing) each location in the gene is sampled almost equally to the rest, we should normalize the hits based on the exonic regions' lengths.

3.2.8 Optimizing Function Weights

In order for the LP solver to prioritize one solution to another, we need to give weights to different variables that appear in the optimization function. Remember the optimization function:

$$maximize \left[\sum_{i=1}^{K} w_i Iisoform_i - \sum_{j=1}^{N+M} w'_j err_j \right] \tag{3.1}$$

We would prefer longer isoforms to shorter ones, so we give the longer ones more weight, in the same time we need to have the least error possible, so for the errors to be less, we need to give them higher weights, that's because of maximizing their negation. We observed that any polynomial function the exonic length of the isoform would be sufficient to serve our purpose for w_i. And for w'_i, weight of the errors, we would choose a polynomial function of exponent of 1 less than that of w_is. For example if we choose n^3 for w_is, (n being the number of exons that $Iisoform_i$ consists of), then for w'_i it would be N^2 (with N being the number of isoforms in the whole gene). We would also tend to give some more flexibility to shorter exons, so if $exon_j$'s length is less than $readSize + 20$ ($exon_j$ is the medium exons category), then we would set its corresponding err_j as N.

3.3 Experiments

In this section we depict charts, that show the accuracy of LiPID compared to two methods of Cufflinks and IsoLasso. For each data set, there are two charts, one shows the accuracy of the three methods only in predicting the set of isoforms right, and the other considers accuracy of precision of both isoforms and their abundances. Precision, recall, and f-score is calculated for each method per data set, and the mean of those calculations for each method

is presented here. More detailed description of the genes that we tested on are attached in appendix A.

3.3.1 Variant exonic length

Using the simulator, we tested LiPID, Cufflinks and IsoLasso on 28 genes with exonic lengths ranging from 3 to 26. Below you can see and compare their accuracies:

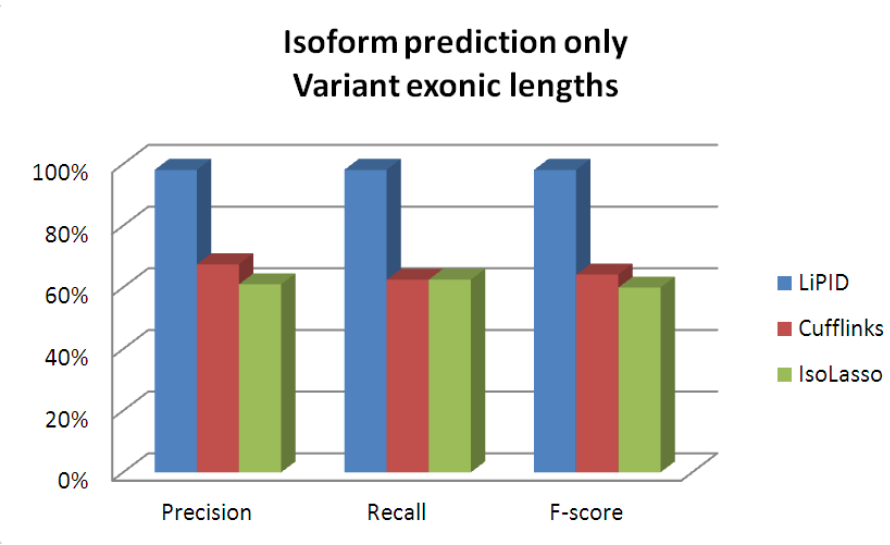

Figure 3.1: Comparison of LiPID, Cufflinks, and IsoLasso accuracy measures (their average precision, recall and f-score) for set of genes with different number of exons ranging from 3 to 26 exons, for isoform prediction only

As we see in Figure 3.1 and Figure 3.2, LiPID outperforms Cufflinks and IsoLasso, and especially in calculating the relative expression levels, it is by far better.

This experiment, bolds and proves LiPID's ***flexibility*** for different exonic lengths, and various exon sizes, and its advantage over the two competitor methods.

3.3.2 Variant number of isoforms

Using the simulator, we tested LiPID , Cufflinks and IsoLasso on different number of isoforms for an average length gene, say MYC. We devised sets of $\{2, 3, 4, 5, 6, 7, 8\}$ isoforms, and did the simulation on them, and ran the methods. The results come in the following charts:

As you can see in Figure 3.3, Cufflinks' and IsoLasso's performance is comparable to that of LiPID in predicting the set of isoforms, however Cufflinks performs very poorly in

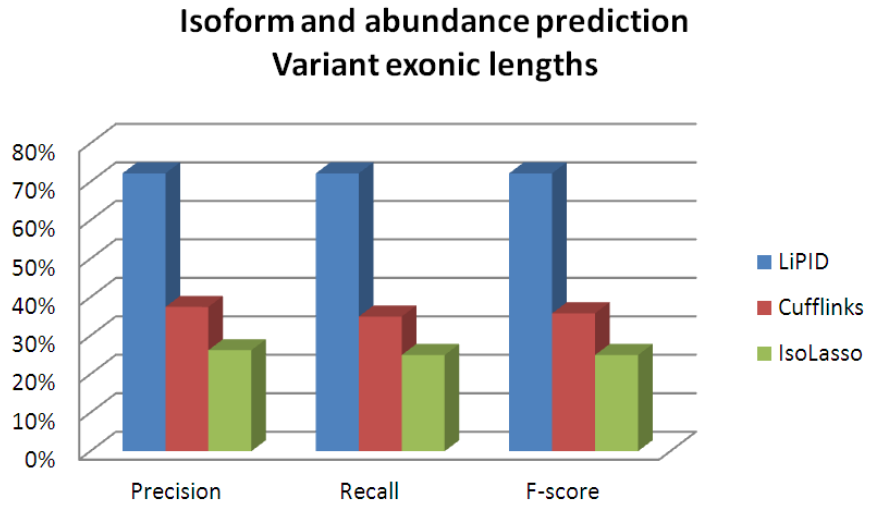

Figure 3.2: Comparison of LiPID, Cufflinks, and IsoLasso accuracy measures (their average precision, recall and f-score) for set of genes with different number of exons ranging from 3 to 26 exons, for both isoform and abundance prediction

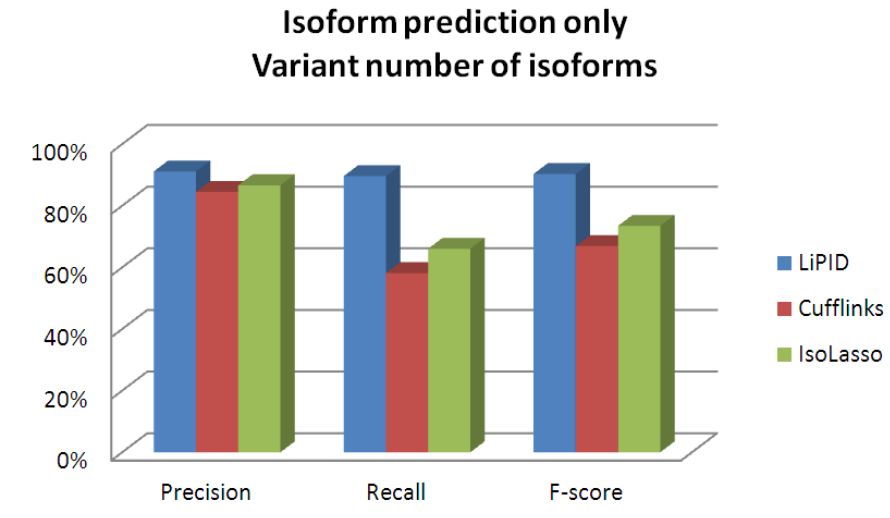

Figure 3.3: Comparison of LiPID, Cufflinks, and IsoLasso accuracy measures (their average precision, recall and f-score) for set of genes with different number of final isoforms, for isoform prediction only

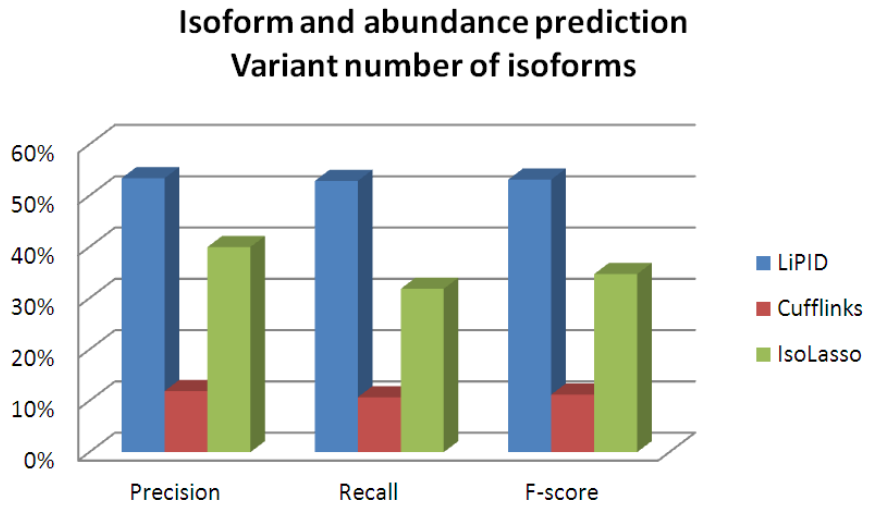

Figure 3.4: Comparison of LiPID, Cufflinks, and IsoLasso accuracy measures (their average precision, recall and f-score) for set of genes with different number of final isoforms, for both isoform and abundance prediction

calculating the relative expression levels.

It is worth mentioning that all methods' accuracies drop as the number of isoforms increases, and LiPID's and IsoLasso's performances are very similar for 7, 8 isoforms. Apparently, as says so in [14] it is much harder to find the correct set of isoforms when they increase in number, and like what happened in this experiment, in some cases there are more than one correct solution to a problem with certain expression levels for exons and junctions, with none or little preference of one over the other. Thus makes it even more difficult to predict the right isoforms and relative expression levels. Anyhow in reality, the number of transcripts in one gene will not exceed 4 or 5.

This experiment proves LiPID's ***robustness***, by providing the exact set of isoforms for small enough numbers, and shows that Cufflinks performs much poorer than LiPID when there are more isoforms. IsoLasso's robustness is almost similar to LiPID's, yet LiPID works slightly better in such cases.

3.3.3 Variant expression levels

Using the simulator, and fixing the number of isoforms, we tested LiPID, Cufflinks and IsoLasso on different ratios for their expression levels to see how sensitive they are in recognizing the difference of expression levels. So we fixed the number of isoforms on 2, and tested the MYC gene for different isoforms by ratios $1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9, 1, 10, 1, 11$. Below you can see the comparison charts:

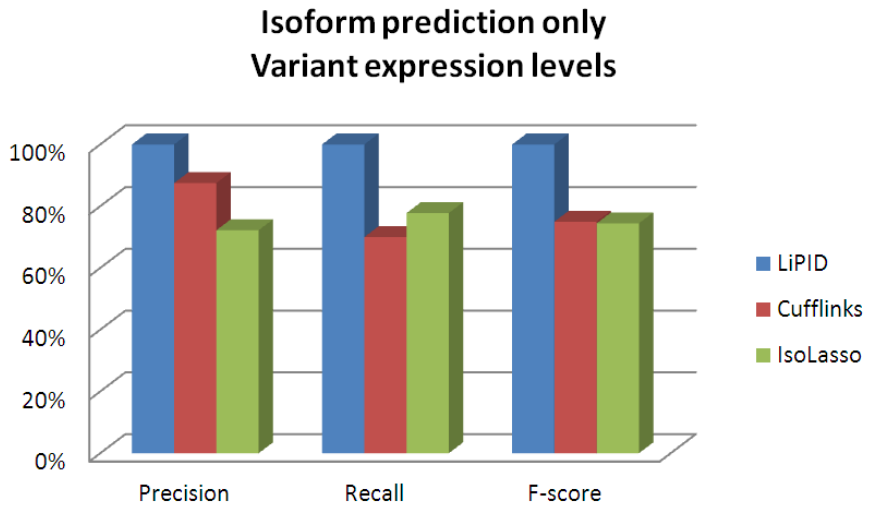

Figure 3.5: Comparison of LiPID, Cufflinks, and IsoLasso accuracy measures (their average precision, recall and f-score) for set of genes with different final expression levels, for isoform prediction only

In this section, LiPID's performance is quite impressive; it can predict the exact set of isoforms in all cases, and miscalculates the expression levels just in one case, however the competitor methods do not perform well, especially in calculating the expression levels, they have much lower accuracies compared to LiPID.

This experiment proves LiPID's ***sensitivity*** in differentiating expression levels, no matter how close they are together, or how far apart, compared to the two other methods.

3.3.4 Performance on Prostate cancer tissues

Our colleagues in Vancouver Prostate Center were able to validate 2 genes with an alternative exon for us, each with 6 different cell lines. We ran LiPID on the reads from samples

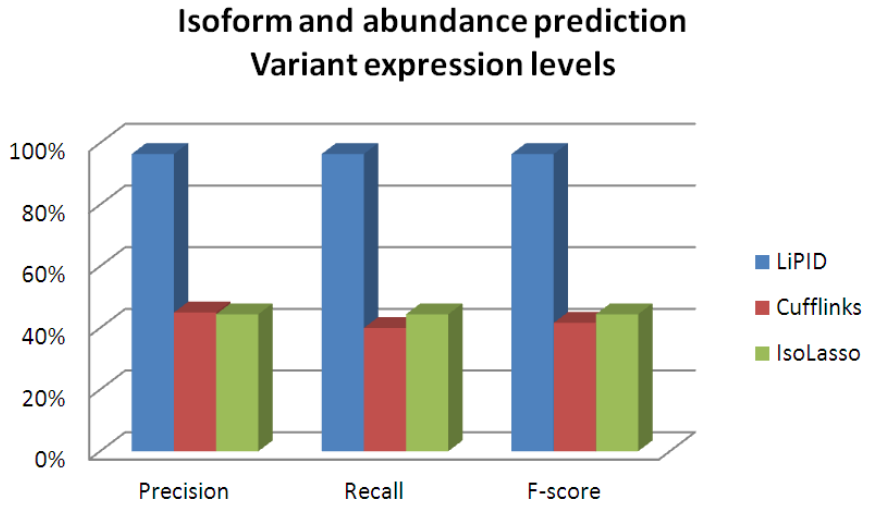

Figure 3.6: Comparison of LiPID, Cufflinks, and IsoLasso accuracy measures (their average precision, recall and f-score) for set of genes with different final expression levels, for both isoform and abundance prediction

to see whether it returns the same results as the validation; not surprisingly all of the alternative exons were identified correctly, although in 4/12 cases the expression levels were not predicted correctly, which is still a good performance for the algorithms of such kind.[5]

[5]This experiment was done on 2 genes, and cannot be used for scientifically showing the performance of any method due to its small size, hence we do not depict its performance chart, and comparison between the three methods.

Chapter 4

Conclusion

Alternative splicing is a vital inter-cell process in living beings. In this thesis we built a new tool, LiPID for detecting alternative splicing events, together with their relative expression levels, based on RNA-seq data. We simulated the relation between the expression level of the transcriptomes and the expression level of the gene segments (exons and junctions) by a Linear Program. The objectives of that Linear Program was to maximize the length of the isoforms predicted, minimize the number of isofoms predicted, and minimize the discrepancy of the exons and junctions expression levels with the set of final isoforms.

We compared the results of LiPID with the ones of two similar methods, Cufflinks and IsoLasso, to evaluate its performance; the comparison has been done for genes with different number of exons, genes with different number of isoforms, and genes with variant relative expression levels.

LiPID outperformed the two other methods in every single sample. Moreover, it showed two major advantages over its competitors:

1. It can predict the set of isoforms and their relative expression levels in genes with more than 8 exons at least 50% better than the other two.

2. with increasing the difference between relative expression levels, LiPID's accuracy increases in calculating the expression levels compared to the other two.

It is safe to say that LiPID offers more *flexibility* and *sensitivity* to the field of transcriptome discovery, compared to the present methods.

Applying LiPID on a limited number of validated real data demonstrated promising results regarding its efficiency in dealing with real data.

4.1 Future Work

In both normal and cancerous cells, there are some events where some parts of the introns[1] are retained during transcription, these are called intron retention events[7]. Since these events happen rarely, we did not design LiPID in a way that it can handle such cases. So for further improvements, we should add ability of intron retention inclusion in LiPID, not to miss these cases.

There are also some cases where some part of the exon is cut off during transcription, so that the exon is not included in the transcriptome wholly. By setting edit distance in LiPID to a reasonable value (we should set it relative to the $readLength$), we can address this issue to some extent, so that it will handle deletions; this method can also be helpful to address partial intron retention, since it can address insertion and deletion. However if the length of the inserted or deleted sequences is more than the edit distance, this method cannot help. So a more comprehensive approach to this issue is needed.

Gene fusions play an important role in developing some cancers[19]; gene fusion is the event where exons from two different genes attach and build the transcriptome. Currently, LiPID does not provide ability to detect isoforms built by such event, but it might be doable by using a gene fusion detection tool, and then considering all possible isoforms conformed of the two genes.

The most important improvement in LiPID's validity is to run LiPID on more real genome data, and to verify the data by RT-PCR, which would require a lot of time and cost, but would probably prove to be very helpful in making sure about LiPID's performance.

[1] non-coding regions

Appendix A

Simulated isoforms and expression levels

Here the types of isoforms, and their relative expression levels for each data set are shown.

A.1 Variant exonic length

ZBTB7A with 3 exons:

ex719–ex720	1
ex719–ex720–ex204721	2

GAST with 3 exons:

ex911Hex912	1
ex911Hex912–ex913	2

LEP with 3 exons:

ex595Hex596–ex597	1
ex595Hex596	2

MYOD1 with 3 exons:

ex571–ex572–ex573	1
ex571–ex572	2

IL2 with 4 exons:

ex800–ex803	1
ex800–ex802–ex803	2

CXCR4 with 4 exons:

ex944Hex945–ex947	1
ex944Hex945–ex946–ex947	2

SDHD with 4 exons:

ex833–ex836	1
ex833–ex834–ex835–ex836	2

CCR5 with 4 exons:

ex047Hex048	1
ex045Hex046	2

PIP with 4 exons:

ex903–ex904	1
ex902–ex903–ex904–ex905	2

IL10 with 5 exons:

ex266–ex267–ex268–ex270	1
ex267–ex268–ex270	2

BCL2 with 5 exons:

ex316–ex317–ex318	1
ex314–ex315–ex316–ex317–ex318	2

LIF with 5 exons:

ex163–ex165–ex166Hex167	1
ex163–ex166Hex167	2

RHO with 5 exons:

ex439–ex440–ex442	1
ex438–ex439–ex440–ex441–ex442	2

MYC with 6 exons:

ex505–ex506–ex508Hex509–ex510	1
ex505–ex508Hex509–ex510	2

SDHC with 6 exons:

ex940–ex941–ex942	1
ex938Hex939–ex940–ex941–ex942–ex943	2

OSM with 7 exons:

ex190–ex192–ex194Hex195	1
ex189–ex190–ex191–ex192–ex193–ex194Hex195	2

ABO with 8 exons:

ex445–ex446–ex447Hex448	1
ex445–ex446–ex447Hex448–ex449–ex450	2

KRT14 with 8 exons:

ex103–ex105–ex107–ex108	1
ex101–ex102–ex103–ex104–ex105–ex106 –ex107–ex108	2

SDHB with 8 exons:

ex586–ex587–ex588–ex589–ex590–ex591 –ex592	1
ex585–ex586–ex588–ex589–ex590–ex592	2

KRT5 with 9 exons:

ex124–ex125Hex126–ex127 –ex128–ex129–ex130–ex131	1
ex124–ex127–ex129–ex130–ex132	2

CD28 with 9 exons:

ex082–ex083–ex087Hex088–ex089	1
ex082–ex083–ex084–ex085Hex086 –ex087Hex088–ex089	2

CD4 with 10 exons:

ex829–ex830–ex833–ex834–ex836–ex837	1
ex829–ex830–ex831–ex832–ex833–ex834 –ex4836–ex4837–ex4838	2

BDNF with 11 exons:

ex990Hex991–ex992–ex993–ex994 –ex995Hex996–ex997–ex998–ex999 –ex1000	1
ex990Hex991–ex992–ex993–ex994 –ex997–ex998–ex999–ex1000	2

INS with 13 exons:

ex669–ex671–ex672Hex673	1
ex669–ex670–ex671–ex672Hex673–ex674Hex675 –ex676Hex677–ex678Hex679	2

MCM6 with 17 exons:

ex506–ex510–ex511–ex516–ex517–ex518 –ex519–ex521–ex522	1
ex506–ex24509–ex510–ex511–ex512–ex513 –ex514–ex516–ex517–ex518–ex519–ex521 –ex522	2

APC with 19 exons:

ex635Hex636–ex638–ex639–ex640Hex641 –ex642–ex645–ex646–ex647–ex648 –ex649–ex650–ex652–ex653	1
ex635Hex636–ex637–ex638–ex639 –ex640Hex641–ex642–ex643Hex644 –ex645–ex646–ex648–ex649–ex650 –ex651–ex652–ex653	2

APP with 19 exons:

ex899–ex900–ex901–ex902–ex903 –ex904–ex907–ex908–ex911–ex912 –ex913–ex914–ex915–ex917	1
ex899–ex900–ex901–ex902–ex903 –ex904–ex906–ex907–ex908–ex909 –ex910–ex911–ex912–ex913–ex914 –ex915–ex916–ex917	2

BRCA1 with 26 exons:

ex565–ex567–ex568–ex569–ex570–ex571
–ex572–ex573–ex574–ex575–ex576Hex577
–ex578–ex579–ex580–ex581–ex582–ex583
–ex584–ex585–ex587–ex588–ex589–ex590 1
ex566–ex567–ex568–ex569–ex570–ex571
–ex572–ex573–ex574–ex576Hex577–ex578
–ex579–ex580–ex581–ex582–ex583–ex584
–ex585–ex586–ex587–ex588–ex589–ex590 2

A.2 Variant number of isoforms

MYC gene with 2 isoforms:

ex505–ex506–ex508Hex509–ex510 1
ex505–ex508Hex509–ex510 2

MYC gene with 3 isoforms:

ex507–ex508Hex509–ex510 1
ex505–ex508Hex509–ex510 2
ex505–ex507–ex510 3

MYC gene with 4 isoforms:

ex505–ex508Hex509–ex510 1
ex505–ex506–ex507–ex508Hex509–ex510 2
ex507–ex508Hex509–ex510 3
ex505–ex506–ex510 4

MYC gene with 5 isoforms:

ex505–ex506–ex507–ex508Hex509	1
ex505–ex508Hex509–ex510	2
ex507–ex508Hex509–ex510	3
ex505–ex506–ex507–ex508Hex509–ex510	4
ex505–ex506–ex508Hex509–ex510	5

MYC gene with 6 isoforms:

ex505–ex506–ex507–ex510	1
ex505–ex506–ex507–ex508Hex509–ex510	2
ex505–ex507–ex508Hex509–ex510	3
ex505–ex506–ex508Hex509–ex510	4
ex506–ex507–ex510	5
ex505–ex507–ex510	6

MYC gene with 7 isoforms:

ex505–ex506–ex507–ex510	1
ex506–ex507–ex508Hex509–ex510	2
ex505–ex506–ex507–ex508Hex509	3
ex505–ex506–ex510	4
ex505–ex506–ex508Hex509–ex510	5
ex505–ex507–ex508Hex509	6
ex505–ex506–ex507–ex508Hex509–ex510	7

MYC gene with 8 isoforms:

ex505–ex506–ex507–ex508Hex509–ex510	1
ex506–ex507–ex508Hex509–ex510	2
ex505–ex506–ex508Hex509	3
ex505–ex507–ex508Hex509–ex510	4
ex505–ex506–ex507–ex510	5
ex505–ex506–ex507	6
ex505–ex506–ex507–ex508Hex509	7
ex505–ex507–ex508Hex509	8

A.3 Variant expression levels

MYC gene with 2 isoforms of expression levels 1-2:

ex505–ex506–ex508Hex509–ex510	1
ex505–ex508Hex509–ex510	2

MYC gene with 2 isoforms of expression levels 1-3:

ex505–ex507–ex508Hex509–ex510	1
ex505–ex506–ex508Hex509–ex510	3

MYC gene with 2 isoforms of expression levels 1-4:

ex505–ex506–ex507–ex508Hex509	1
ex505–ex506–ex507–ex508Hex509–ex510	4

MYC gene with 2 isoforms of expression levels 1-5:

ex505–ex506–ex507–ex508Hex509–ex510	1
ex505–ex506–ex507–ex510	5

MYC gene with 2 isoforms of expression levels 1-6:

ex506–ex507–ex508Hex509–ex510	1
ex505–ex508Hex509–ex510	6

MYC gene with 2 isoforms of expression levels 1-7:

ex505–ex507–ex508Hex509	1
ex505–ex506–ex508Hex509–ex510	7

MYC gene with 2 isoforms of expression levels 1-8:

ex505–ex507–ex510	1
ex505–ex506–ex508Hex509–ex510	8

MYC gene with 2 isoforms of expression levels 1-9:

ex505–ex506–ex507–ex508Hex509–ex510 1
ex505–ex507–ex508Hex509–ex510 10

MYC gene with 2 isoforms of expression levels 1-10:

ex505–ex506–ex508Hex509–ex510 1
ex505–ex506–ex507–ex508Hex509–ex510 10

MYC gene with 2 isoforms of expression levels 1-11:

ex505–ex506–ex507–ex508Hex509–ex510 1
ex506–ex507–ex508Hex509–ex510 11

MYC gene with 2 isoforms of expression levels 1-12:

ex505–ex506–ex507–ex508Hex509–ex510 1
ex505–ex506–ex507 12

MYC gene with 2 isoforms of expression levels 1-13:

ex505–ex507–ex508Hex509 1
ex505–ex506–ex510 13

MYC gene with 2 isoforms of expression levels 1-14:

ex505–ex506–ex507–ex508Hex509–ex510 1
ex505–ex506–ex507–ex510 14

MYC gene with 2 isoforms of expression levels 1-15:

ex505–ex506–ex507–ex508Hex509–ex510 1
ex506–ex507–ex508Hex509–ex510 15

Bibliography

[1] C. Alkan, J.M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J.O. Kitzman, C. Baker, M. Malig, O. Mutlu, et al. Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature genetics*, 41(10):1061–1067, 2009.

[2] K.F. Au, H. Jiang, L. Lin, Y. Xing, and W.H. Wong. Detection of splice junctions from paired-end rna-seq data by splicemap. *Nucleic Acids Research*, 38(14):4570, 2010.

[3] N. Cloonan, A.R.R. Forrest, G. Kolle, B.B.A. Gardiner, G.J. Faulkner, M.K. Brown, D.F. Taylor, A.L. Steptoe, S. Wani, G. Bethel, et al. Stem cell transcriptome profiling via massive-scale mrna sequencing. *Nature methods*, 5(7):613–619, 2008.

[4] I.B.M.I. CPLEX. V12. 1: Users manual for cplex. *International Business Machines Corp., Armonk, New York, USA*, 2009.

[5] R.P. Dilworth. A decomposition theorem for partially ordered sets. *The Annals of Mathematics*, 51(1):161–166, 1950.

[6] J. Feng, W. Li, and T. Jiang. Inference of isoforms from short sequence reads. In *Research in Computational Molecular Biology*, pages 138–157. Springer, 2010.

[7] P.A.F. GALANTE, N.J.O. SAKABE, N. KIRSCHBAUM-SLAGER, and S.J. DE SOUZA. Detection and evaluation of intron retention events in the human transcriptome. *Rna*, 10(5):757, 2004.

[8] P. Gardina, T. Clark, B. Shimada, M. Staples, Q. Yang, J. Veitch, A. Schweitzer, T. Awad, C. Sugnet, S. Dee, et al. Alternative splicing and differential gene expression in colon cancer detected by a whole genome exon array. *Bmc Genomics*, 7(1):325, 2006.

[9] M. Guttman, M. Garber, J.Z. Levin, J. Donaghey, J. Robinson, X. Adiconis, L. Fan, M.J. Koziol, A. Gnirke, C. Nusbaum, et al. Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincrnas. *Nature biotechnology*, 28(5):503–510, 2010.

[10] B.J. Haas, A.L. Delcher, S.M. Mount, J.R. Wortman, R.K. Smith Jr, L.I. Hannick, R. Maiti, C.M. Ronning, D.B. Rusch, C.D. Town, et al. Improving the arabidopsis genome annotation using maximal transcript alignment assemblies. *Nucleic Acids Research*, 31(19):5654, 2003.

[11] J.M. Johnson, J. Castle, P. Garrett-Engele, Z. Kan, P.M. Loerch, C.D. Armour, R. Santos, E.E. Schadt, R. Stoughton, and D.D. Shoemaker. Genome-wide survey of human alternative pre-mrna splicing with exon junction microarrays. *Science*, 302(5653):2141, 2003.

[12] W.J. Kent. Blatthe blast-like alignment tool. *Genome research*, 12(4):656, 2002.

[13] M. Krawczak, J. Reiss, and D.N. Cooper. The mutational spectrum of single base-pair substitutions in mrna splice junctions of human genes: causes and consequences. *Human Genetics*, 90(1):41–54, 1992.

[14] W. Li, J. Feng, and T. Jiang. Isolasso: A lasso regression approach to rna-seq based transcriptome assembly. In *Research in Computational Molecular Biology*, pages 168–188. Springer, 2011.

[15] R. Lister, R.C. O'Malley, J. Tonti-Filippini, B.D. Gregory, C.C. Berry, A.H. Millar, and J.R. Ecker. Highly integrated single-base resolution maps of the epigenome in arabidopsis. *Cell*, 133(3):523–536, 2008.

[16] C.A. Maher, C. Kumar-Sinha, X. Cao, S. Kalyana-Sundaram, B. Han, X. Jing, L. Sam, T. Barrette, N. Palanisamy, and A.M. Chinnaiyan. Transcriptome sequencing to detect gene fusions in cancer. *Nature*, 458(7234):97–101, 2009.

[17] E.R. Mardis. Next-generation dna sequencing methods. *Annu. Rev. Genomics Hum. Genet.*, 9:387–402, 2008.

[18] J.C. Marioni, C.E. Mason, S.M. Mane, M. Stephens, and Y. Gilad. Rna-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome research*, 18(9):1509, 2008.

[19] A. McPherson, F. Hormozdiari, A. Zayed, R. Giuliany, G. Ha, M.G.F. Sun, M. Griffith, A.H. Moussavi, J. Senz, N. Melnyk, et al. defuse: An algorithm for gene fusion discovery in tumor rna-seq data. *PLoS computational biology*, 7(5):e1001138, 2011.

[20] R.D. Morin, M. Bainbridge, A. Fejes, M. Hirst, M. Krzywinski, T.J. Pugh, H. McDonald, R. Varhol, S.J.M. Jones, and M.A. Marra. Profiling the hela s3 transcriptome using randomly primed cdna and massively parallel short-read sequencing. *Biotechniques*, 45(1):81–94, 2008.

[21] O. Morozova and M.A. Marra. Applications of next-generation sequencing technologies in functional genomics. *Genomics*, 92(5):255–264, 2008.

[22] A. Mortazavi, B.A. Williams, K. McCue, L. Schaeffer, and B. Wold. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature methods*, 5(7):621–628, 2008.

[23] U. Nagalakshmi, Z. Wang, K. Waern, C. Shou, D. Raha, M. Gerstein, and M. Snyder. The transcriptional landscape of the yeast genome defined by rna sequencing. *Science*, 320(5881):1344, 2008.

[24] G. Robertson, J. Schein, R. Chiu, R. Corbett, M. Field, S.D. Jackman, K. Mungall, S. Lee, H.M. Okada, J.Q. Qian, et al. De novo assembly and analysis of rna-seq data. *Nature Methods*, 7(11):909–912, 2010.

[25] J.T. Simpson, K. Wong, S.D. Jackman, J.E. Schein, S.J.M. Jones, and İ. Birol. Abyss: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117, 2009.

[26] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[27] C. Trapnell, L. Pachter, and S.L. Salzberg. Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, 25(9):1105, 2009.

[28] C. Trapnell, B.A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M.J. Van Baren, S.L. Salzberg, B.J. Wold, and L. Pachter. Transcript assembly and quantification by rna-seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, 28(5):511–515, 2010.

[29] J.P. Venables. Aberrant and alternative splicing in cancer. *Cancer research*, 64(21):7647, 2004.

[30] J.P. Venables. Unbalanced alternative splicing and its significance in cancer. *Bioessays*, 28(4):378–386, 2006.

[31] K. Wang, D. Singh, Z. Zeng, S.J. Coleman, Y. Huang, G.L. Savich, X. He, P. Mieczkowski, S.A. Grimm, C.M. Perou, et al. Mapsplice: Accurate mapping of rna-seq reads for splice junction discovery. *Nucleic Acids Research*, 38(18):e178, 2010.

[32] Z. Wang, M. Gerstein, and M. Snyder. Rna-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009.

[33] Z. Wang, H.S. Lo, H. Yang, S. Gere, Y. Hu, K.H. Buetow, and M.P. Lee. Computational analysis and experimental validation of tumor-associated alternative rna splicing in human cancer. *Cancer research*, 63(3):655, 2003.

[34] B.T. Wilhelm, S. Marguerat, S. Watt, F. Schubert, V. Wood, I. Goodhead, C.J. Penkett, J. Rogers, and J. B
"ahler. Dynamic repertoire of a eukaryotic transcriptome surveyed at single-nucleotide resolution. *Nature*, 453(7199):1239–1243, 2008.

[35] Q. Xu and C. Lee. Discovery of novel splice forms and functional analysis of cancer-specific alternative splicing in human expressed sequences. *Nucleic acids research*, 31(19):5635, 2003.