Freddie: Annotation-independent Detection and Discovery of Transcriptomic Alternative Splicing Isoforms Using Long-read Sequencing (Supplementary Material)

Orabi et al.

October 21, 2022

S1 Supplementary Methods

S1.1 Details for pruning the candidate breakpoints in the segmentation stage.

Freddie selects a subset of breakpoints, $S \subseteq C$, to represent the finalized set of canonical exon boundaries inferred from the input LR alignments. Note that the breakpoint sets, C and S, divide the genome, respectively, into $|C| - 1$ and $|S| - 1$ non-overlapping genomic segments. We define a scoring function, f, which given a set of breakpoints simultaneously rewards individual reads for sharp changes in coverage between consecutive segments and penalizes them for having partial alignment on individual segments (see main text Figure 3B). Freddie selects the subset S which maximizes $f(S)$ over all the subsets of C, using a Dynamic Programming (DP) algorithm.

To formally define $f(S)$, we introduce the following notations. Let $cov(r, i, j)$ be the percentage of positions in genomic segment $[i, j]$ that are covered by the split-alignment intervals of read r. We also define three binary indicators, $Y_{r,i,j}$, $N_{r,i,j}$ and $P_{r,i,j}$, to respectively indicate if read r is covering, not covering, or partially covering the genomic segment $[i, j]$:

$$
\begin{aligned}
Y_{r,i,j} &= \begin{cases} 1, & cov(r,i,j) > 0.9 \\ 0, & \text{otherwise} \end{cases} \\
N_{r,i,j} &= \begin{cases} 1, & cov(r,i,j) < 0.1 \\ 0, & \text{otherwise} \end{cases} \\
P_{r,i,j} &= \begin{cases} 1, & 0.1 \le cov(r,i,j) \le 0.9 \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
\tag{S1}
$$

Next, we define the number of reads with partial split-alignment coverage over a genomic segment $[i, j]$:

$$
Partial(i,j) = \sum_r P_{r,i,j} \tag{S2}
$$

Finally, the number of reads with sharply contrasting coverage between two neighboring genomic segments, $[i, j]$ and $[j, k]$, $Contrast(i, j, k)$, is defined as follows:

$$
\begin{aligned}
&Contrast(i,j,k) = \\
&\sum_r \overbrace{Y_{r,i,j} \times N_{r,j,k}}^{\text{covering then not covering}} + \overbrace{N_{r,i,j} \times Y_{r,j,k}}^{\text{not covering then covering}}
\end{aligned}
\tag{S3}
$$

Using these functions, we define the scoring function f:

$$
\begin{aligned}
&f(S) = \\
&\sum_{(i,j,k) \in consecutive(S)} Contrast(i,j,k) - Partial(i,j)
\end{aligned}
\tag{S4}
$$

Where *consecutive*(S) is the set of all triplets of consecutive breakpoints in S.

We want to find S which maximizes $f(S)$ from all possible subsets of C. Although there are exponentially many subsets of C, we can efficiently find an optimal subset S because the structure of the scoring function lends itself readily to cubic time Dynamic Programming (DP). To describe the DP equations, let $\mathcal{C}$ be the list of the elements in C sorted by increasing genomic coordinate and let $\ell = |\mathcal{C}|$. We denote by $\mathcal{C}[i]$ the breakpoint at index i in $\mathcal{C}$ and by $\mathcal{C}[i..j]$ the sub-list of $\mathcal{C}$ starting at index i and ending at index j (inclusively on both ends). We fill in a DP table D defined as follows: for $1 \leq m < n < o \leq \ell$, $D[m, n, o]$ is the optimal score (as defined in Equation S4) among all subsets of C composed of the breakpoints in $\mathcal{C}[m..\ell]$ with the additional constraint that the first three breakpoints in the subset, according to the order defined by $\mathcal{C}$, occur at the respective indices m, n and o in $\mathcal{C}$. Then:

$$\begin{aligned} D[m,n,o] = & \\ & - \overbrace{Partial(\mathcal{C}[m], \mathcal{C}[n])}^{\text{first segment penalty}} \\ & + \overbrace{Contrasting(\mathcal{C}[m], \mathcal{C}[n], \mathcal{C}[o])}^{\text{first two segments reward}} \\ & + \overbrace{\max_{o<p\leq\ell} \Big(D[n,o,p]\Big)}^{\text{subproblem optimal score}} \end{aligned} \tag{S5}$$

The score of the optimal segmentation S is then:

$$\max_{1<n<\ell} \left(\max_{n<o\leq\ell} D[1,n,o] \right) \tag{S6}$$

Finally, to obtain S, we backtrack through the DP table D.

Speed-up heuristics. Since the cubic time complexity can be prohibitive if there is a large number of candidate breakpoints, we reduce the complexity by fixing breakpoints using some heuristics before running the DP formulation. Fixing breakpoints effectively breaks down the problem into independent subproblems, reducing the base of the cubic time complexity and thus dramatically speeding up the effective runtime. The first heuristics we use is to select any breakpoint that has extremely high smoothed signal value since it is highly likely to be in the optimal set of breakpoints. Specifically, we fix breakpoints with values higher than the mean plus three times the standard deviation of the $\mathcal{M}[i]$ for all i in C. The second heuristics we use is to introduce a limit on the maximum allowed problem size, $|C|$. If $|C|$, after applying the first heuristic, exceeds this limit, we split C uniformly into subproblems, each of size less than or equal to the limit. The limit is a user-set parameter with a default value of 50.

S1.2 Robustness of segmentation parameters

Since the segmentation includes a number of hyperparameters, we wished to justify our default values for them and ensure a degree of robustness for their choice. To do so, we changed one hyperparameter at a time (keeping the rest at their default values) and observed the effect on the accuracy of the segmentation (i.e. how far each detected segment boundary from an annotation boundary). The following results are from the real human dataset assuming the annotation as a ground truth. Note that the same values result in similar findings for the simulated human dataset.

For the first speed-up heuristic, the variation factor with default value of 3, we tested a number of values and which had very little impact on the accuracy of the segmentation:

Variation factor	% segment boundaries					
	Exact		Within +/-10bp		Outside +/- 10bp	
	Start	End	Start	End	Start	End
1.5	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
3.0	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
4.5	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
6.5	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%

For the second speed-up heuristic, the maximum problem size, we similarly observe almost no change in the results. This is because in practice, both heuristics select few breakpoints that were not going to be selected otherwise:

Maximum Problem Size	% segment boundaries					
	Exact		Within +/-10bp		Outside +/- 10bp	
	Start	End	Start	End	Start	End
50	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
100	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
150	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
200	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%

We also tested the following coverage threshold values (default 90%) and again observed little change:

Coverage Threshold	% segment boundaries					
	Exact		Within +/-10bp		Outside +/- 10bp	
	Start	End	Start	End	Start	End
0.75	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
0.80	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
0.85	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
0.90	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
0.95	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%

Finally, we tested the Gaussian filter size parameter. This time, we did observe a trade-off when selecting the parameter value:

Gaussian Filter Size	% segment boundaries					
	Exact		Within +/-10bp		Outside +/- 10bp	
	Start	End	Start	End	Start	End
1	95.8%	95.7%	1.6%	2.7%	1.6%	1.6%
3	95.3%	95.2%	3.4%	3.5%	1.3%	1.3%
5	91.7%	93.8%	5.2%	7.2%	1.0%	1.1%
8	92.2%	90.2%	6.7%	8.8%	1.0%	1.0%
15	82.5%	78.3%	15.6%	19.7%	2.0%	2.0%

Here, we opted for selected 5 as the default parameter value since it minimized the number of breakpoints that are not within 10bp of an annotation breakpoint and maximized the number of breakpoints that are exactly on top of an annotation breakpoint. Of course, we think that other values can be also reasonable choices as default values (e.g. 1 and 3) as long as they are similar in size to the expected erroneous indels generated by Oxford Nanopore LRs.

S1.3 Details for MErCI.

To formulate the MErCI problem, we start by introducing some preliminary definitions. Let A be an $N \times V$ binary matrix representing the input reads, i.e., N is the number of reads and $V = |S| - 1$ is the number of canonical genomic segments: $A[i][j] = 1$ if the i-th read covers $> 90\%$ of the j-th segment and $A[i][j] = 0$

otherwise. For a given subset I of the reads, we define the *induced isoform* of I, $\mathcal{I}_I$, to be a binary vector of size V consisting of column-wise disjunction of the reads in I:

$$\mathcal{I}_I[j] = \bigvee_{i \in I} A[i][j] \tag{S7}$$

Given an induced isoform, $\mathcal{I}_I$, we define the correction cost relative to $\mathcal{I}_I$ of read $i \in I$, $CC(i, \mathcal{I}_I)$, to be the number segments present in $\mathcal{I}_I$ but not in $A[i]$:

$$CC(i, \mathcal{I}_I) = \sum_{j=1}^{V} \mathcal{I}_I[j] \times (1 - A[i][j]) \tag{S8}$$

Finally, we define the optimization goal of MErCI to be finding a subset of reads I that minimizes:

$$\overbrace{\mu \cdot (N - |I|)}^{\text{Recycling bin penalty}} + \overbrace{\sum_{i \in I} CC(i, \mathcal{I}_I)}^{\text{Isoform bin penalty}} \tag{S9}$$

where μ is a user-defined threshold forbidding the inclusion of a read into the isoform bin if at least μ corrections are required. By default, we set $\mu = 3$, allowing for up to two error corrections per read.

Incorporating length of corrected segments constraint Formally, we denote a gap with a triplet (s, e, l) where s and e are the first and last covered (i.e. 1) segments of the gap, and l is the number of unaligned bases on the read between the s and e. Then, for each gap (s, e, l) of read i that is assigned to the isoform bin I, we add a restriction that the genomic length of the corrected segments between s and e must be close to l. More precisely, let $CGL(s, e, i)$ be the genomic length of the corrected segments within the gap (s, e, l) of read i:

$$CGL(s, e, i) = \sum_{s<j<e} \mathcal{I}_I[j] \times (1 - A[i][j]) \times GL(j) \tag{S10}$$

where $GL(j)$ denotes the genomic length of the j-th segment. Then the following length condition must hold:

$$(1 - \epsilon) \times CGL(s, e, i) - \delta \leq l \leq (1 + \epsilon) \times CGL(s, e, i) + \delta \tag{S11}$$

By default, we set ϵ to 0.1 and δ to 20 nucleotides.

S2 Supplementary Figures

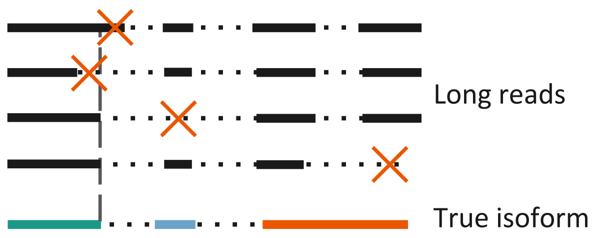

Figure S1: Long-read sequencing has a high error rate reaching up to 10-20%. Insertion and deletion (indel) of nucleotides dominate this error rate. These indel errors results in 1) a reduction in the resolution of exon boundaries and 2) missing small exons. Additionally, 3) full-length LR sequencing is not always successful, resulting in missing exons at the tail of the sequenced isoform. The figure illustrate these three hazards associated with LR sequencing of isoforms.

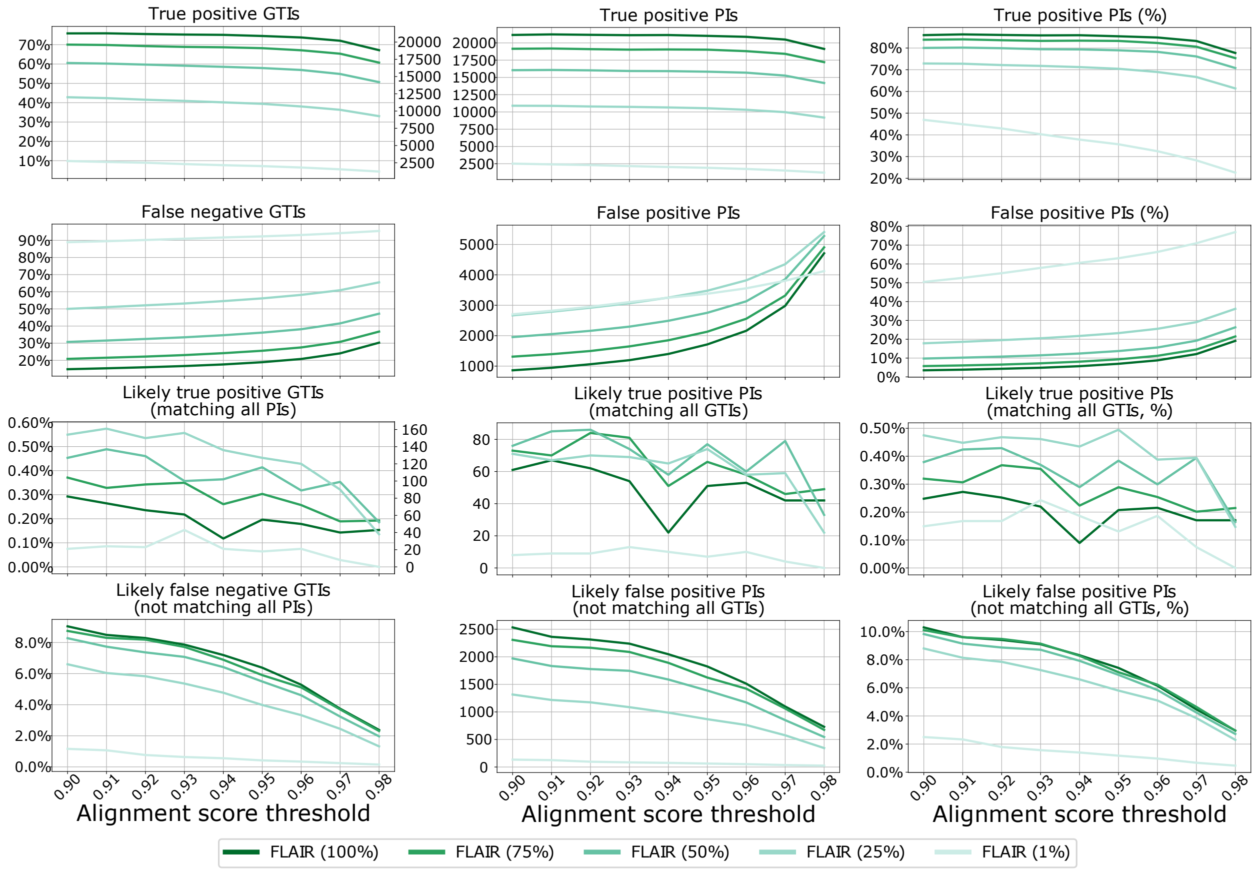

Figure S2: Graph-based analysis of FLAIR on the simulated human dataset when FLAIR is given varying sample sizes of the human annotations.

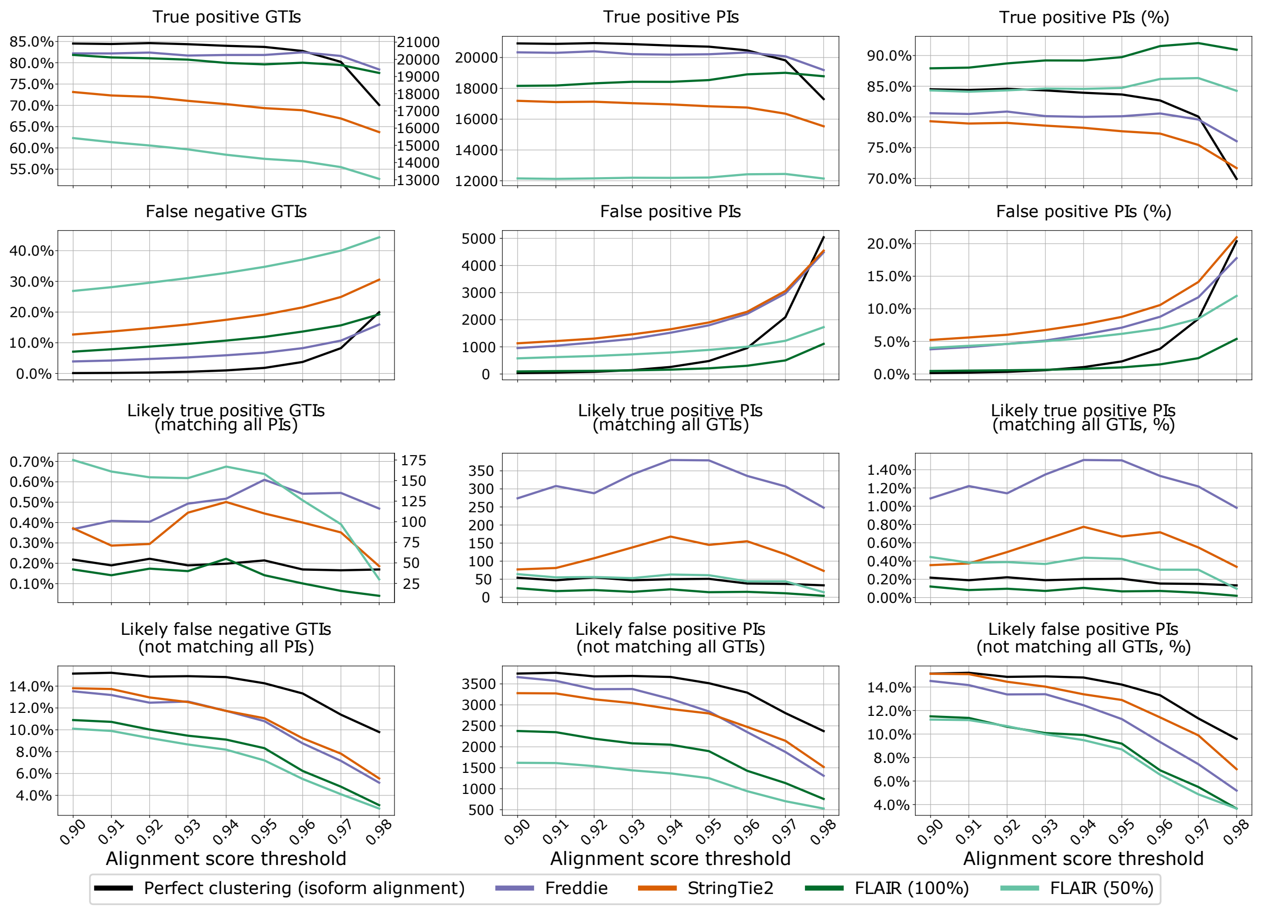

Figure S3: Graph-based analysis of different tools on the simulated fruit fly dataset.

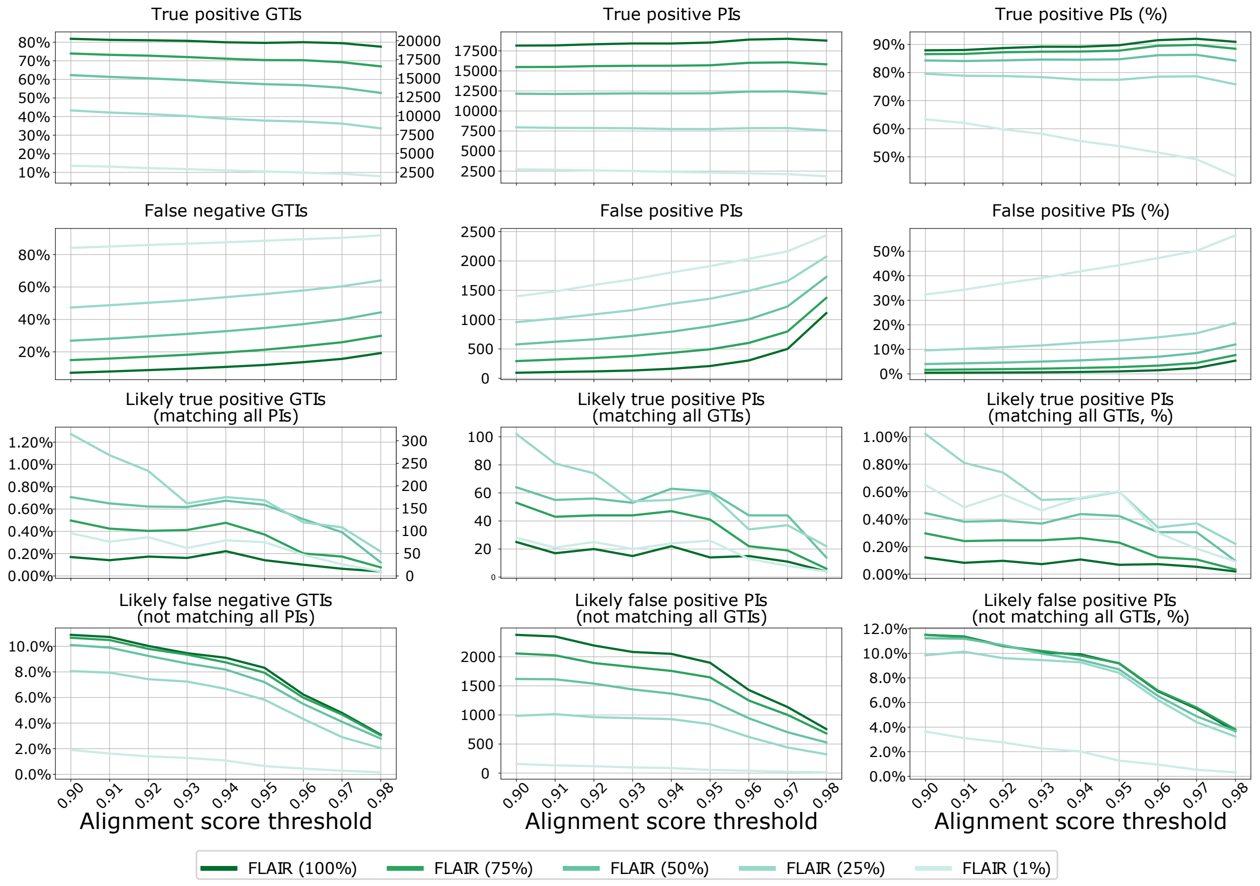

Figure S4: Graph-based analysis of FLAIR on the simulated fruit fly dataset when FLAIR is given varying sample sizes of the human annotations.

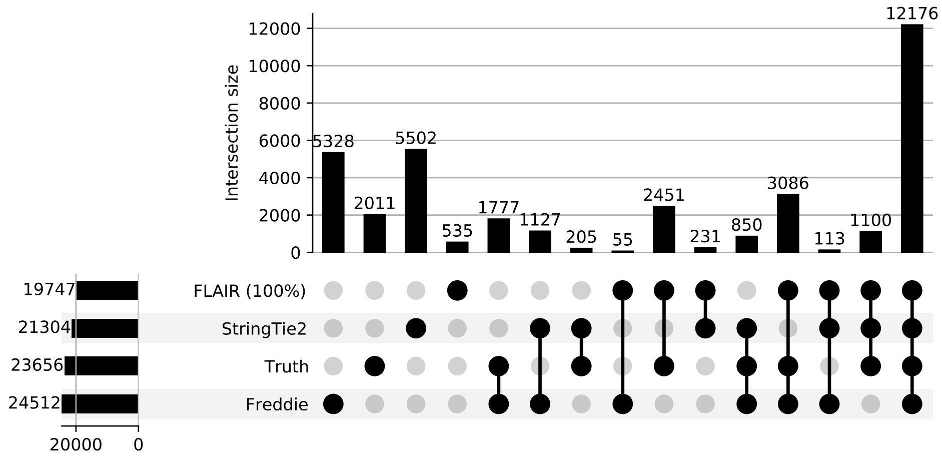

Figure S5: Simulated fruit fly dataset. Upset plot showing the intersection sizes between the sets of isoforms predicted by different tools or present in the simulated ground truth using their exon intervals as a condition of equivalence.

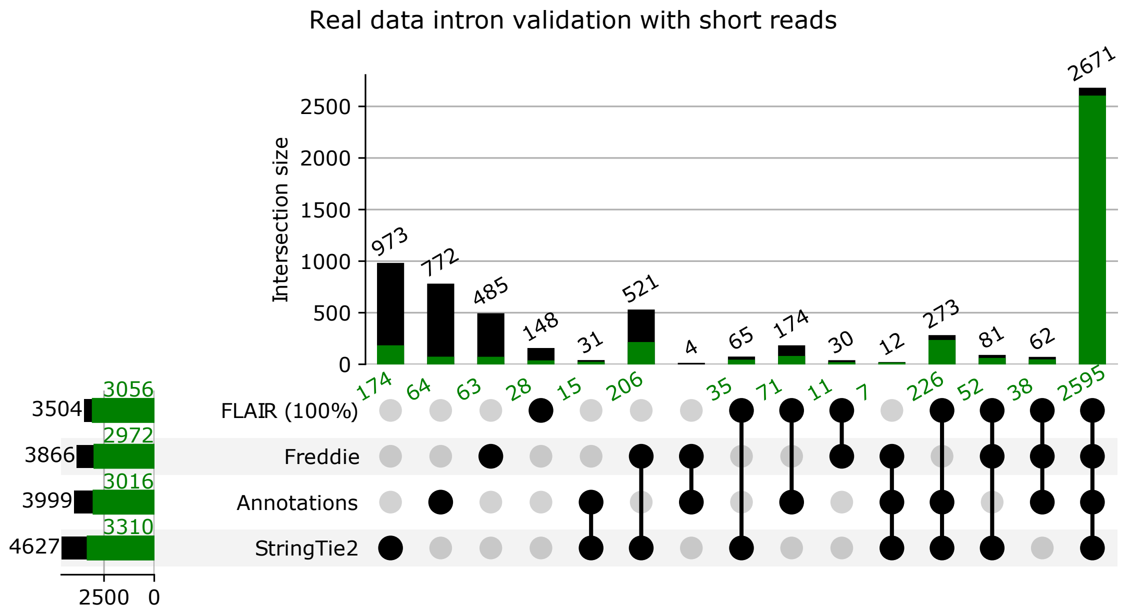

Figure S6: LNCaP real dataset. Upset plot showing the intersection sizes between the sets of introns predicted by different tools or present in ENSEMBL 97 annotation. For each intersection column, the number of introns that are validated using the STAR and the short-read dataset is indicated in green.

Supplementary Tables

Table S1: Human simulated dataset. CPU (user) time, real (clock) time, and memory use of FLAIR, StringTie2 and Freddie. Note that we report the results for StringTie2 using a single thread because it crashes when run with multiple threads (segfault).

Tool	Threads	Wall clock (min)	CPU time (min)	Memory (GB)
StringTie2	1	2.11	2.22	0.03
FLAIR (1%)	32	5.45	32.39	5.06
FLAIR (25%)	32	7.02	52.88	6.86
FLAIR (50%)	32	7.57	61.85	8.37
FLAIR (75%)	32	7.98	66.97	8.48
FLAIR (100%)	32	8.42	69.82	9.11
Freddie split	32	2.23	12.67	1.02
Freddie segment	32	1.09	26.76	0.90
Freddie cluster	32	8.87	83.91	1.36
Freddie collapse	32	0.24	1.21	0.06
Freddie (all stages)	32	12.43	124.55	1.36

Table S2: 22Rv1 real dataset. CPU (user) time, real (clock) time, and memory use of FLAIR, StringTie2 and Freddie. Note that we report the results for StringTie2 using a single thread because it crashes when run with multiple threads (segfault).

Tool	Threads	Wall clock (min)	CPU time (min)	Memory (GB)
StringTie2	1	0.23	0.23	0.03
FLAIR (1%)	32	1.72	5.55	0.81
FLAIR (25%)	32	2.05	6.76	0.97
FLAIR (50%)	32	1.94	7.09	1.03
FLAIR (75%)	32	1.94	7.23	1.07
FLAIR (100%)	32	1.94	7.41	1.11
Freddie split	32	0.46	0.69	0.38
Freddie segment	32	0.63	13.25	0.17
Freddie cluster	32	9.49	196.32	2.42
Freddie collapse	32	0.01	0.13	0.02
Freddie (all stages)	32	10.58	210.39	2.42

Table S3: Fruit fly simulated dataset. CPU (user) time, real (clock) time, and memory use of FLAIR, StringTie2 and Freddie.

Tool	Threads	Wall clock (min)	CPU time (min)	Memory (GB)
StringTie2	1	5.45	5.58	0.10
FLAIR (1%)	32	7.78	33.18	4.55
FLAIR (25%)	32	9.94	66.26	7.47
FLAIR (50%)	32	10.92	82.90	8.14
FLAIR (75%)	32	11.44	91.04	8.32
FLAIR (100%)	32	11.85	99.42	8.67
Freddie split	32	6.70	23.03	4.62
Freddie segment	32	2.61	48.32	1.49
Freddie cluster	32	6.41	89.97	3.40
Freddie collapse	32	0.15	2.34	0.09
Freddie (all stages)	32	15.87	163.66	4.62

Table S4: Accuracy statistics for simulated fruit fly dataset using the exon intervals to identify equivalent isoforms.

Tool	F1-score	Precision	Recall	True isoforms	False isoforms
Freddie	74.28%	72.98%	75.62%	17,889	6,623
StringTie2	63.75%	67.27%	60.58%	14,331	6,973
FLAIR (100%)	86.69%	95.27%	79.53%	18,813	934

S3 Selecting genes of DHT+ real dataset

In the real dataset experiments, we focused on a subset of the genes when evaluate the different tested methods. We first selected genes that have are the middle quintile (40th to 60th percentile) of genes in terms of their short- and long-read coverage. From these genes, we further selected genes with that fall into one of three splicing complexity levels. Specifically, we selected genes that, according to ENSEMBL database, have either: i) one isoform, ii) four to seven isoforms, or iii) ten to twenty isoforms. In total, this resulted in selecting 294 genes.

Selecting genes for real dataset experiment

In [1]:
```
# Reference ENSEMBL annotations
gtf = '/groups/hachgrp/annotations/GTF/97/Homo_sapiens.GRCh38.97.gtf'
# Short-read gene expression
sr_cov =
    '/groups/hachgrp/projects/col-dong-rna-splicing/'+
    'results/htseq/all.tsv'
# Long-read gene expression
lr_cov =
    '/groups/hachgrp/projects/col-dong-rna-splicing/'+
    'analysis/long-qc/stats/P003R000.gene-coverage.tsv'
```

Keeping a set of the middle quintile (40th to 60th percentile) of genes in terms of their short- AND long-read coverage

In [2]:
```
# SR gene coverage
sr_gid_cnts = list()
for l in open(sr_cov):
    if not l.startswith('ENSG'):
        continue
    l = l.rstrip().split('\t')
    gid = l[0]
    cnt = int(l[1])
    if cnt < 2:
        continue
    sr_gid_cnts.append((cnt,gid))
sr_gid_cnts.sort()

# LR gene coverage
lr_gid_cnts = list()
for l in open(lr_cov):
    if not l.startswith('ENSG'):
        continue
    l = l.rstrip().split('\t')
    gid = l[0]
    cnt = int(l[7])
    if cnt < 2:
        continue
    lr_gid_cnts.append((cnt,gid))
lr_gid_cnts.sort()
```

In [3]:
```
# intersection of the middle quintile of LR and SR covered genes
mid_cov_gids = set(
    [x[1] for x in sr_gid_cnts[int(len(sr_gid_cnts)*.4):int(len(sr_gid_cnts)*.6)]]
) & set(
    [x[1] for x in lr_gid_cnts[int(len(lr_gid_cnts)*.4):int(len(lr_gid_cnts)*.6)]]
)
len(mid_cov_gids)
```

Out[3]:
```
1562
```

Creating BED file of all gene intervals to remove overlapping genes. Any gene

that overlaps with other genes will be discarded

In [4]:
```
bed = open('all_genes.bed', 'w+')
for l in open(gtf):
    if l.startswith('#'):
        continue
    l = l.rstrip().split('\t')
    if not l[2] == 'gene':
        continue
    i = l[-1].find('gene_id "')+len('gene_id "')
    gid = l[-1][i:i+15]
    print(l[0], l[3], l[4], gid, sep='\t', file=bed)
bed.close()
!wc -l all_genes.bed
```

```
60617 all_genes.bed
```

Finding any gene that overlap with any other gene

In [5]:
```
!bedtools intersect -a all_genes.bed -b all_genes.bed -wb |\
    awk '$8!=$4{print $8;print $4}' |\
    sort -u > overlapping_genes.txt
overlapping_gids = {
    l.rstrip() for l in open('overlapping_genes.txt')
}
len(overlapping_gids)
```

Out[5]:
```
37827
```

Getting transcript counts per gene, for the genes that are NOT overlapping and that are in the middle quintile of coverage

In [6]:
```
gid_tid_count = dict()
for l in open(gtf):
    if l.startswith('#'):
        continue
    l = l.rstrip().split('\t')
    if l[0][0] in ['K', 'G']:
        continue
    if not l[2] == 'transcript':
        continue
    if not 'protein_coding' in l[-1]:
        continue
    i = l[-1].find('gene_id "')+len('gene_id "')
    gid = l[-1][i:i+15]
    if gid in overlapping_gids:
        continue
    if not gid in mid_cov_gids:
        continue
    gid_tid_count[gid] = gid_tid_count.get(gid, 0) + 1
print(len(gid_tid_count))
```

```
432
```

Selecting genes from different ranges of isoform complexity (according to ENSEMBL database)

```
In [7]: import random
        random.seed(42)

        #Keeping genes with the # of ENSEMBL-reported isoforms falling in these ranges
        ranges = [(1,1), (4,7), (10,20)]
        chosen_gids = set()
        for x,y in ranges:
            chosen_gids.update(
                [gid for gid,cnt in gid_tid_count.items() if x<=cnt<=y],
            )
        len(chosen_gids)
```

```
Out[7]: 294
```

```
In [8]: chosen_gids
```

```
Out[8]: {'ENSG00000004660',
         'ENSG00000005448',
         'ENSG00000008277',
         'ENSG00000010318',
         'ENSG00000010704',
         'ENSG00000011478',
         'ENSG00000015153',
         'ENSG00000040275',
         'ENSG00000040487',
         'ENSG00000064012',
         'ENSG00000065057',
         'ENSG00000065621',
         'ENSG00000066651',
         'ENSG00000067365',
         'ENSG00000071243',
         'ENSG00000073605',
         'ENSG00000076650',
         'ENSG00000076716',
         'ENSG00000077152',
         'ENSG00000081870',
         'ENSG00000083544',
         'ENSG00000083750',
         'ENSG00000085185',
         'ENSG00000086730',
         'ENSG00000087088',
         'ENSG00000087586',
         'ENSG00000088035',
         'ENSG00000088826',
         'ENSG00000089177',
         'ENSG00000089195',
         'ENSG00000089685',
         'ENSG00000090447',
         'ENSG00000090924',
         'ENSG00000091127',
         'ENSG00000091138',
         'ENSG00000092208',
         'ENSG00000097046',
         'ENSG00000099860',
         'ENSG00000101247',
         'ENSG00000101276',
         'ENSG00000101888',
         'ENSG00000101928',
         'ENSG00000102078',
         'ENSG00000102218',
         'ENSG00000102221',
         'ENSG00000102312',
         'ENSG00000102543',
         'ENSG00000102870',
         'ENSG00000104356',
         'ENSG00000104953',
         'ENSG00000105497',
         'ENSG00000105708',
         'ENSG00000105771',
         'ENSG00000107833',
         'ENSG00000109536',
         'ENSG00000110002',
         'ENSG00000110031',
         'ENSG00000110723',
         'ENSG00000111196',
         'ENSG00000111554',
         'ENSG00000111790',
         'ENSG00000111981',
         'ENSG00000112290',
         'ENSG00000112312',
         'ENSG00000115289',
         'ENSG00000115350',
         'ENSG00000115421',
         'ENSG00000115750',
         'ENSG00000117399',
         'ENSG00000117569',
         'ENSG00000118965',
         'ENSG00000119397',
         'ENSG00000119906',
         'ENSG00000119965',
         'ENSG00000120798',
         'ENSG00000121417',
         'ENSG00000121671',
         'ENSG00000122694',
         'ENSG00000122873',
         'ENSG00000124374',
         'ENSG00000124508',
         'ENSG00000124613',
         'ENSG00000124784',
         'ENSG00000125144',
         'ENSG00000125352',
         'ENSG00000125703',
         'ENSG00000126787',
         'ENSG00000126870',
         'ENSG00000126953',
         'ENSG00000128394',
         'ENSG00000128534',
         'ENSG00000128604',
         'ENSG00000128944',
         'ENSG00000129534',
         'ENSG00000130349',
         'ENSG00000130772',
         'ENSG00000131015',
         'ENSG00000132275',
         'ENSG00000132323',
         'ENSG00000132541',
         'ENSG00000133247',
         'ENSG00000133627',
         'ENSG00000133739',
         'ENSG00000133874',
         'ENSG00000134056',
         'ENSG00000134152',
         'ENSG00000134531',
         'ENSG00000134602',
         'ENSG00000134716',
         'ENSG00000135211',
         'ENSG00000135315',
         'ENSG00000135838',
         'ENSG00000135919',
         'ENSG00000136235',
         'ENSG00000136603',
         'ENSG00000136824',
         'ENSG00000136936',
         'ENSG00000137413',
         'ENSG00000137747',
         'ENSG00000138036',
         'ENSG00000138231',
         'ENSG00000138376',
         'ENSG00000138399',
         'ENSG00000138604',
         'ENSG00000139083',
         'ENSG00000139826',
         'ENSG00000139832',
         'ENSG00000141665',
         'ENSG00000143033',
         'ENSG00000143061',
         'ENSG00000143147',
         'ENSG00000143502',
         'ENSG00000143847',
         'ENSG00000144034',
         'ENSG00000144048',
         'ENSG00000144120',
         'ENSG00000144134',
         'ENSG00000144199',
         'ENSG00000144559',
         'ENSG00000144589',
         'ENSG00000145088',
         'ENSG00000145220',
         'ENSG00000145241',
         'ENSG00000145349',
         'ENSG00000145632',
         'ENSG00000146072',
         'ENSG00000146263',
         'ENSG00000146757',
         'ENSG00000146918',
         'ENSG00000148057',
         'ENSG00000148225',
         'ENSG00000148690',
         'ENSG00000148814',
         'ENSG00000148908',
         'ENSG00000149503',
         'ENSG00000149636',
         'ENSG00000151332',
         'ENSG00000151657',
         'ENSG00000151715',
         'ENSG00000151876',
         'ENSG00000152086',
         'ENSG00000152219',
         'ENSG00000153015',
         'ENSG00000153140',
         'ENSG00000153574',
         'ENSG00000153896',
         'ENSG00000153898',
         'ENSG00000153975',
         'ENSG00000155330',
         'ENSG00000155636',
         'ENSG00000155961',
         'ENSG00000156345',
         'ENSG00000156469',
         'ENSG00000156603',
         'ENSG00000156804',
         'ENSG00000156869',
         'ENSG00000157343',
         'ENSG00000157796',
         'ENSG00000158122',
         'ENSG00000160124',
         'ENSG00000160193',
         'ENSG00000160256',
         'ENSG00000160298',
         'ENSG00000162366',
         'ENSG00000162385',
         'ENSG00000162396',
         'ENSG00000162639',
         'ENSG00000162664',
         'ENSG00000162897',
         'ENSG00000163002',
         'ENSG00000163312',
         'ENSG00000163328',
         'ENSG00000163378',
         'ENSG00000163898',
         'ENSG00000164109',
         'ENSG00000164188',
         'ENSG00000164291',
         'ENSG00000164611',
         'ENSG00000165030',
         'ENSG00000165496',
         'ENSG00000166024',
         'ENSG00000166788',
         'ENSG00000166860',
         'ENSG00000166881',
         'ENSG00000167637',
         'ENSG00000167900',
         'ENSG00000168228',
         'ENSG00000168298',
         'ENSG00000168393',
         'ENSG00000168661',
         'ENSG00000168876',
         'ENSG00000168993',
         'ENSG00000169016',
         'ENSG00000169087',
         'ENSG00000169155',
         'ENSG00000169252',
         'ENSG00000169598',
         'ENSG00000169599',
         'ENSG00000170191',
         'ENSG00000170260',
         'ENSG00000171100',
         'ENSG00000171295',
         'ENSG00000171345',
         'ENSG00000171421',
         'ENSG00000171611',
         'ENSG00000171903',
         'ENSG00000172006',
         'ENSG00000172428',
         'ENSG00000172878',
         'ENSG00000174428',
         'ENSG00000174599',
         'ENSG00000174842',
         'ENSG00000174951',
         'ENSG00000175189',
         'ENSG00000176125',
         'ENSG00000177076',
         'ENSG00000181467',
         'ENSG00000182173',
         'ENSG00000182208',
         'ENSG00000182378',
         'ENSG00000182405',
         'ENSG00000182504',
         'ENSG00000182518',
         'ENSG00000182700',
         'ENSG00000183856',
         'ENSG00000183891',
         'ENSG00000184481',
         'ENSG00000184979',
         'ENSG00000185163',
         'ENSG00000185418',
         'ENSG00000185842',
         'ENSG00000186458',
         'ENSG00000186470',
         'ENSG00000186787',
         'ENSG00000187257',
         'ENSG00000187609',
         'ENSG00000187626',
         'ENSG00000188295',
         'ENSG00000189057',
         'ENSG00000189164',
         'ENSG00000189369',
         'ENSG00000196372',
         'ENSG00000196646',
         'ENSG00000196724',
         'ENSG00000196787',
         'ENSG00000196793',
         'ENSG00000197016',
         'ENSG00000197223',
         'ENSG00000197444',
         'ENSG00000197461',
         'ENSG00000197808',
         'ENSG00000197841',
         'ENSG00000198298',
         'ENSG00000198417',
         'ENSG00000198478',
         'ENSG00000204104',
         'ENSG00000204899',
         'ENSG00000204991',
         'ENSG00000205269',
         'ENSG00000205423',
         'ENSG00000205643',
         'ENSG00000221923',
         'ENSG00000226124',
         'ENSG00000228278',
         'ENSG00000235750',
         'ENSG00000241945',
         'ENSG00000262152',
         'ENSG00000263002',
         'ENSG00000269556',
         'ENSG00000273559',
         'ENSG00000275111',
         'ENSG00000276966',
         'ENSG00000277075',
         'ENSG00000277224'}
```