



AltSplice computational pipeline

A technical intro and practical step-by-step

Jean-Jack Riethoven

ASD Project

ASD/ATD monthly meeting

February 2005



AltSplice Pipeline – Initial Notes

A technical document exists which will be made available soon in the CVS and the ASD internal pages where every point and step in this presentation is listed in more detail: this presentation is not a 'cut&paste' running of the pipeline.

This is **NOT** a talk on methodologies used in the pipeline
(see my Seqdbg Thursday seminar Oct 2003 for that
http://www3.ebi.ac.uk/internal/seqdb/organisation/meetings/thursday/riethoven_231003.htm).

This is **NOT** a code review of scripts/modules used in the pipeline
(another presentation later will deal with that).



AltSplice Pipeline

- ☞ Philosophy: to have an automated pipeline that can deliver a high-quality set of transcript confirmed alternative features, splice patterns, and events - together with appropriate annotation - in under two weeks from the baseline data sets.

- ☞ Design considerations:
 - **Quality control + speed.**
 - Each step in pipeline should have own module.
 - Every module has its own inherent quality checks and warning/error reporting.
 - Where possible, modules are run in parallel.
 - Where possible and appropriate, modules should have mile-stone markers.
 - As much as possible, independence of external modules.



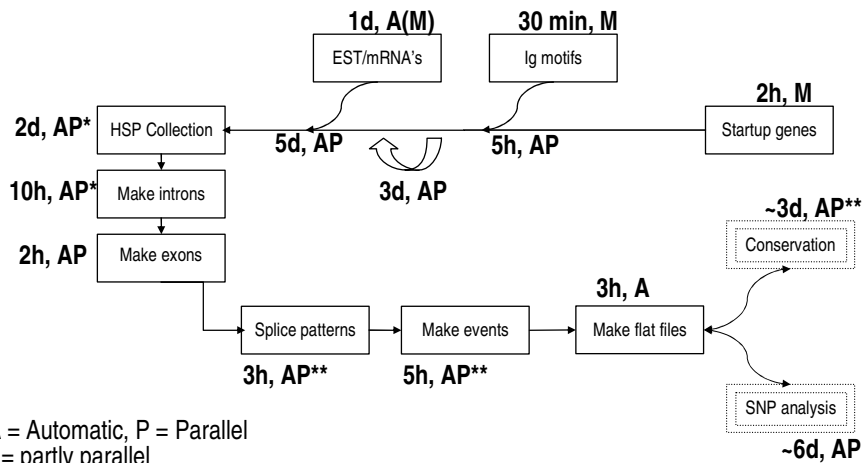
AltSplice Pipeline Building Blocks

- ☞ AltSplice (computational) pipeline: the total process from the start of retrieving our initial datasets to loading the database with all features and annotations is called the AltSplice pipeline.

- ☞ Several big building blocks:
 - *Initial Data retrieval*
 - *Creation of basic data (features, splice patterns, events, annotation) (peptide isoforms)*
 - *SNP mapping and annotation*
 - *Species conservation*
 - *Aedb integration*
 - *Database loading*



ASD Pipeline – Time by Step



AltSplice Pipeline – Environment 1

Software environment - 1

- ☞ Programmed in: Perl (5.6) [/sw/arch/bin/perl]
- ☞ Modules (external):
 - DB_File (EST/peptide indices, file indices)
 - DBI::Oracle (db loading)
 - Proc::Simple (child process handling)
 - Bio::Tools::BPlite/BPbl2seq (blast parsing)*
- ☞ Programs (external):
 - Ncbi-blast [/ebi/extserv/bin/ncbi-blast]
 - Fasta [/ebi/extserv/bin/fasta34t21b5b]



AltSplice Pipeline – Environment 2

Software environment - 2

- ☞ No clean special data build code directory (as of yet)
- ☞ Every script and module in 'asd' EBI CVS server.

- ☞ Pipeline scripts in ~/projects/asd/pipeline/bin
- ☞ Pipeline modules located in ~/projects/asd/pipeline/lib and ~/projects/asd/interfaces/lib*
- ☞ External required modules located in /ebi/sp/pro6/asd/external/perl/BioPerl-live-1.0.1/lib

* to-do is to collect them all under one module directory – initially they were different because of the diverse roles these two had. But since a year or so some modules are interwoven.



AltSplice Pipeline – Environment 3

Software environment - 3

- ☞ One (1) shell variable is required: PIPELINE_PERL_PATH which holds the directories for modules (replace /homes/pow/):

/homes/pow/projects/asd/pipeline/lib:
/ebi/sp/pro6/asd/external/perl/BioPerl-live-1.0.1/lib:
/homes/pow/projects/asd/altsplice/perl:
/homes/pow/projects/asd/interfaces/lib/.
 - some should be used if certain paths are not explicitly specified on the command line for the scripts (notable Blast DB and Blast temporary directories).

- ☞ Any shell can be used
 - Except one particular step in pipeline which requires a shell (and OS) that can handle files > 2Gb.



AltSplice Pipeline – Environment 4

Hardware environment - 1

- ☞ EBI's pcfarm1 linux compute farm
 - Avg. 135 blades: 2+ CPU 1-2Gb memory
 - ASD Project bought 15+1 blades
 - 15: 2 CPU 3.2Ghz 2Gb
 - 1: Xegony 4 CPU 3.2 Ghz 2Gb (Master node)
 - Use via LSF on queue production, project ASD (-P ASD)

- ☞ Oracle Server
 - Dec/Tru64, 4 CPU 9Gb (Swan) – cost-shared for 50% from ASD money.
 - Oracle 9.2.1 (APRO)



AltSplice Pipeline – Environment 5

Hardware environment - 2

- ☞ NFS NetApp storage (600Gb)
 - /ebi/sp/asd/ [/net/nfs6/vol6/sp-asd]
 - No snapshots; no backup for anything with 'tmp' or 'temp' (any case) in its name

- ☞ Setup
 - Core initial data: /ebi/sp/asd/data/ (ESTmRNA, ensembl_genes, lg)
 - Work (data builds): /ebi/sp/asd/work/ (human/mouse, database, integration)
 - Release data (new): /ebi/sp/asd/release/ (by species)
 - Services data: /ebi/sp/asd/SERVICES



Step by Step

AltSplice pipeline 101



AltSplice Pipeline – Step by Step 1

☞ Note: in the technical docs environment variables are used so the pipeline can be run by cut & paste (mostly). In the following practical a specific databuild is used.

☞ 1a. If required: get the AltSplice source code up to date (pre-req: a cvs checkout has been done for asd)

```
cd ~/projects/asd/pipeline
cvs update -d
cd ../interfaces/lib
cvs update -d
cd ~/projects/asd/pipeline
```

☞ 1b. Make the directories that are required. (see techdoc for a cut&paste).



AltSplice Pipeline – Step by Step 2

- ☞ 2. Retrieve Ensembl genes and their annotation from Ensmart*.
 - 2a. Creating peptide identifier and xref database with file pointers to the protein sequence
 - 2b. Create individual unflanked and flanked gene sequence files.
 - 2c. Merging the various annotation files and creating annotation files per individual gene. Requires 2a.

- ☞ Ad 2. Use Ensmart to select species, the whole genome, and all genes. This should result in more than 20k genes for human and mouse.

*Download manually since Ensembl schema/API changes too often for a production pipeline



AltSplice Pipeline – Step by Step 3

- ☞ Ad 2. The directory to where files should be downloaded is

```
/ebi/sp/asd/data/ensembl_genes/human/  
human_10022005_downloaded_flatfiles/
```

=> equals \${download_dir} in further slides

- All files downloaded compressed (gzip: .gz), sequence files as fasta format and others as tab delimited.
- Save sequence files as *.fasta.gz
- Save other files as *.tsv.gz
- (check via gunzip if files are not corrupted – sometimes download breaks off)



AltSplice Pipeline – Step by Step 4

- ☞ Ad 2. (techdoc for exact download instructions)
 - Tab Sequence: get unflanked genes, flanked (3000 bases both sides) genes, and peptide sequences.
 - Structure: gene structure (chrom, strand, gene id, transcript id, description, peptide id, exon id, exon start/end + exon cds start/end)
 - Features Tab: gene id, transcript id, peptide id, xrefs for Go, EMBL, UniProt/Swiss-Prot, InterPro, HUGO, PDB, RefSeq, Tr embl, species conservation.
NOTE: cannot download more than 3 xrefs per file.



AltSplice Pipeline – Step by Step 5

- ☞ Ad 2a. Fast lookup peptide database creation:
 - ```
ln -s
${download_dir}/ensembl_27.35a.1.peptides.fasta
/ebi/sp/asd/work/human/27.35a.1/blastdb/Peptides_hu
man
```
  - ```
./createPEPTIDEindex.pl -  
blastdb=/ebi/sp/asd/work/human/27.35a.1/blastdb/Pept  
ides_human
```
 - **Note:**
``${work_dir} = /ebi/sp/asd/work/human/27.35a.1/`
`${data_dir} = /ebi/sp/asd/data/`
in further slides.`



AltSplice Pipeline – Step by Step 6

- Ad 2b. Create individual flanked and unflanked gene sequences from the master files

```
- ./split_Ensembl_mart_genes.pl  
  
-flanked_gene_file=${download_dir}/ensembl_27.35a.1.fasta  
  
-unflanked_gene_file=${download_dir}/  
ensembl_27.35a.1.no_flanks.fasta  
  
-output_dir=/ebi/sp/asd/data/ensembl_genes/  
human/human_27.35a.1/
```



AltSplice Pipeline – Step by Step 7

- Ad 2c. Merging the various annotation files and creating annotation files per individual gene. Requires 2a (peptides) for sanity checks.

```
- Does RefSeq/SPTR lookups for synonyms  
- ./make_annot_files.pl  
-input_dir=${download_dir}  
  
-output_dir=${data_dir}/ensembl_genes/human/human_2735a.1/  
  
-fasta_dir=${data_dir}/ensembl_genes/human/human_27.35a.1/  
  
-peptide_file=${work_dir}/blastdb/Peptides_human
```



AltSplice Pipeline – Step by Step 8

- ☞ 3. Run database queries on @REP to get list of EST and mRNA accession.version identifiers.
 - This step is done manually at the moment by doing 3 SELECT statements into temporary tables: a) one for EST's, b) one for mRNA's, and c) one non-species specific query for HTC/HTG entries. Combining (a+b) minus c yields us the final set of transcript id's, which are stored into:

```
`${data_dir}/ESTmRNA/human_17012005_acc.lst
```

- IIIa. Use the above list to retrieve EST/mRNA sequences, descriptions, lib id's (+sanity checks) from @REP and create a Fasta file of them:

```
./fetchFromEmbl.pl -dbname=REP -dbuser=corbadev  
-dbpassword=?????  
-log=${work_dir}/estdownload.log  
-species=human <  
`${data_dir}/ESTmRNA/human_17012005_acc.lst >  
`${data_dir}/ESTmRNA/human_17012005_est_mrna.fasta
```



AltSplice Pipeline – Step by Step 9

- ☞ 4. Download Ig (immunoglobulin) motif database.
 - <http://www.ncbi.nlm.nih.gov/igblast/showGermline.cgi>
 - Select species and cut&paste each locus into a file
 - Save file as `\${data_dir}/Ig/human_Feb2005.fasta
- ☞ 4a. Make blast database out of it via formatdb
 - ```
ln -s `${data_dir}/Ig/human_Feb2005.fasta
`${work_dir}/blastdb/humanIg
```
  - ```
/ebi/extserv/bin/ncbi-blast/formatdb -t  
"humanIg" -i `${work_dir}/humanIg -p F -o T
```



AltSplice Pipeline – Step by Step 10

- 5. Softlink main sequence + annotation files from the data directory to a mirror in the work directory

```
./copyMainGeneData.pl  
  
-from=${data_dir}/ensembl_genes/  
human/human_27.35a.1/  
  
-to=${work_dir}/genes/
```

- For each gene there will be a *_known.fasta, *.known_no_flanks.fasta, and an *.annotations file.



AltSplice Pipeline – Step by Step 11

- 6. Prepare for the Ig blast (checking for genes that match Ig motifs on large coverage): softlink unflanked gene files from mirror to work directory:

```
/preIgbblast.pl  
  
-from=${work_dir}/genes/  
  
-to=${work_dir}/all_unflanked_genes/
```

- For each gene there will be an *_known.fasta, *.known_no_flanks.fasta, and an *.annotations file.



AltSplice Pipeline – Step by Step 12

- 7. Start the parallel Blast of the Ig sequences against the unflanked geneset

```
./BlastEngine.pl
-db=humanIg
-E=0.00000001
-log=${work_dir}/ig-blast.log
-dir=${work_dir}/all_unflanked_genes/
-out_dir=${work_dir}/all_unflanked_genes/
-runqueuesize=1000
-blastdb_dir=${work_dir}/blastdb/
-blast_tmp=${work_dir}/tmp/
```



AltSplice Pipeline – BlastEngine

- BlastEngine script:
 - Blasts individual sequences against a blast database
 - No mass-submit to LSF, has own internal queue for gradual submits
 - Using this queue keeps track of state of each job, can rerun on error
 - Continue from point of failure (LSF down, master node)
 - Defaults to optimal performance for jobs of 5-15 mins duration
- (similar functionalities in other parallel implementations in pipeline)



AltSplice Pipeline – Step by Step 13

- 8. Analyse Ig blast results and remove genes that bear resemblance to immunoglobulins

```
./postIgbblast.pl  
-pid=95  
-cover=95  
-fasta_dir=${work_dir}/all_unflanked_genes/  
-blast_dir=${work_dir}/all_unflanked_genes/
```

- Genes are removed directly from the 'fasta_dir'



AltSplice Pipeline – Step by Step 14

- 9. Collate remaining genes into one big Fasta file and make a proper Fasta database out of it.

```
find ${work_dir}/all_unflanked_genes/  
-name "*.fasta" -exec cat \{\} >>  
${work_dir}/blastdb/ensembl_27.35a.1 \;
```

```
/ebi/extserv/bin/ncbi-blast/formatdb  
-t "ensembl_27.35a.1"  
-i ${work_dir}/blastdb/ensembl_27.35a.1  
-p F -o T
```



AltSplice Pipeline – Step by Step 15

- 10. Do the genes vs. genes blast required for the redundancy analysis. Unflanked genes, ungapped blast.
- Note: generates 200+Gb of data.

```
./BlastEngine.pl -db=ensembl_27.35a.1  
-E=0.0000000000000001 -other="-g F"  
-log=${work_dir}/redundant-blast.log  
-dir=${work_dir}/all_unflanked_genes/  
-out_dir=${work_dir}/blast_results_redundant_tmp/  
-blastdb_dir=${work_dir}/blastdb/  
-blast_tmp=${work_dir}/tmp/
```



AltSplice Pipeline – Step by Step 16

- 11. Analyse the blast results and remove genes that show similarity over a long region (from any sim.pair keep the longer one)

```
./RemoveRedundantSeq.pl  
-fasta_dir=${work_dir}/all_unflanked_genes/  
-blast_dir=${work_dir}/blast_results_redundant_tmp/  
-redundant_dir=${work_dir}/redundant_genes/  
-log=${work_dir}/redundant.log  
-pid=99 -cover=90 -binsize=100  
-work_dir=${work_dir}/tmp/
```

- Remove large blast files:

```
- find ${work_dir}/redundant_genes/ -name  
  "*.fasta*" | xargs rm  
- mv ${work_dir}/blast_results_redundant_tmp  
  ${work_dir}/blast_results_redundant
```



AltSplice Pipeline – Step by Step 17

- 12. Replace gene files that are left with the flanked versions

```
mv ${work_dir}/all_unflanked_genes  
  ${work_dir}/genes_no_redundant  
./preESTblast.pl  
-log=/dev/null  
-from=${work_dir}/genes/  
-to=${work_dir}/genes_no_redundant/
```



AltSplice Pipeline – Step by Step 18

- 13. Do the genes vs. EST/mRNA's blast (will find HSP's which will be used to locate introns later on)

```
./BlastEngine.pl  
-db=ESTmRNAhuman -E=0.0000000001 -V=3000 -B=3000  
-log=${work_dir}/est-blast.log  
-dir=${work_dir}/genes_no_redundant/  
-out_dir=${work_dir}/blast_results_est/  
-delay=10 -blastdb_dir=${work_dir}/blastdb/  
-blast_tmp=${work_dir}/tmp/
```



AltSplice Pipeline – Step by Step 19

- 14. Collect HSP's from blast results into big master file (95% pid), generate a list of transcripts that match multiple genes, and split into individual gene files (98%).

```
./HSP-collector.pl  
-log=${work_dir}/HSP_Parse.log  
-blast_dir=${work_dir}/blast_results_est/ -  
fasta_dir=${work_dir}/genes_no_redundant/  
-gff=${work_dir}/hsp_pid95  
-pid=98  
-work_dir=${work_dir}/tmp/  
-binsize=190
```

- Binsize (number of genes per parallel job) can be optimised: number of genes / 120 if compute farm is not busy; number of genes / (60 to 80) if very busy
- Files created: **\$id.hsp.gff** (95%), **\$id.passed.gff** (98% + extra checks)



AltSplice Pipeline – Step by Step 20

- 15. Create fast lookup transcript database (for those ESTs/mRNAs that passed filters). Rev/compl any sequence as well if indicated by alignment.

```
./createESTindex.pl  
-fasta_dir=${work_dir}/genes_no_redundant/  
-blastdb=${work_dir}/blastdb/ESTmRNAhuman  
-log=${work_dir}/createESTindex.log
```

- Creates: `${work_dir}/blastdb/ESTmRNAhuman.ASD` and `${work_dir}/blastdb/ESTmRNAhuman.ASD.index`
- Next steps of pipeline `${work_dir}/blastdb/ESTmRNAhuman.ASD` is used as 'blast_db'/est_db'



AltSplice Pipeline – Step by Step 21

- ☞ 16. Create introns using the individual gene passed.gff HSP files.

- ☞

```
./HSP2intron.pl  
-log=${work_dir}/hsp2intron.log  
-out_file=${work_dir}/intron  
-fasta_dir=${work_dir}/genes_no_redundant/ -  
work_dir=${work_dir}/tmp/  
-blast_db=${work_dir}/ESTmRNAhuman.ASD
```
- ☞ **Creates:** \$id.passed-introns.gff, \$id.noncanon-introns.gff, \$id.keep-alt-introns.gff
- ☞ The \$id.passed-introns.gff file is used for next step in pipeline.



AltSplice Pipeline – Step by Step 22

- ☞ 17. Run the patching process (tries to fill gaps between introns and on terminal ends).

- ☞

```
../HSPPatcher.pl  
-log=${work_dir}/Patcher.log  
-fasta_dir=${work_dir}/genes_no_redundant/ -  
blast_dir=${work_dir}/tmp/  
-est_db=${work_dir}/blastdb/ESTmRNAhuman.ASD
```
- ☞ **Creates:** \$id.complete-introns.gff which is used for next step in pipeline.



AltSplice Pipeline – Step by Step 23

☞ 18. Create exons (using the introns)

```
☞ ./make_Exons.pl  
-log=${work_dir}/combined.log  
-est_db=${work_dir}/blastdb/ESTmRNAhuman.ASD -  
fasta_dir=${work_dir}/genes_no_redundant/
```

☞ Creates: `$id.exons.gff` and `$id.combined.gff` (the latter is used for next step in pipeline).



AltSplice Pipeline – Step by Step 24

☞ 19. [Optional] Run the sanity check on the feature data. This script is a central repository of checks that already have been done elsewhere in the pipeline.

- Normally run in parallel with the next step of the pipeline.

```
☞ ./do_sanity_check.pl  
-est_db=${work_dir}/blastdb/ESTmRNAhuman.ASD -  
fasta_dir=${work_dir}/genes_no_redundant/  
-log=${work_dir}/SANITY_CHECK.log
```

☞ Anything that hits a trigger will be logged. Result is overview of total warnings/errors, if any.



AltSplice Pipeline – Step by Step 25

- ☞ 20. Make the flat files.
 - Get features, do additional checks, make splice patterns and group patterns.
 - Use splice patterns to deduce events.
 - Annotate as much as possible, then write several output files (public, internal)

```
☞ ./make_ASD_flatfiles.pl
-log=${work_dir}/ASDFLAT.log
-est_db=${work_dir}/blastdb/ESTmRNAhuman.ASD -
fasta_dir=${work_dir}/genes_no_redundant/ -
output_file=${work_dir}/ASD_27.35a.1
-evoc_dir=/ebi/sp/asd/data/evoc/annotations/ -
species=human
-force_rebuild
```



AltSplice Pipeline – Step by Step 26

- ☞ 20. Make the flat files (continued)
 - `-force_rebuild` is only used when a fresh data build is made, or when the part of the data build (incl. Flat file generation) has changed which reflects in different transcripts to be present/absent. It regulates the transcript->library id fast lookup file.
 - Note: for Evoc-enabled species a new Evoc lookup has to be made before this script can be run (used for expression annotation).



AltSplice Pipeline – End Basic

- ☞ This concludes the pipeline run for the 'basic' data
 - Other modules (SNP, conservation, etc) will use the result files for further analysis and/or db loading
- ☞ Next Slides: intermediate & result files



AltSplice Pipeline – GFF files

- ☞ Real GFF3 file for HSP's (gene-transcript alignments)

```
ENSG00000001460 BLAST similarity 16641 16781 141 + .
signif "2e-71" ; FracId 141 ; Bits 280 ; SeqLength 64098 ; Group 0
; pid 100 ; QS
ggtgcttaggagaaacatgcagtagaacttttcatcaacaaatggaacacgtcacagaatTTTgctaacaatgga
caactctgcacagaaaaatgaacgcactggcaaacatcccagacgtgccagtgaaagtacagaaaggt ;
BG723366 BLAST similarity 194 334 141 + . signif "2e-71" ;
FracId 141 ; Bits 280 ; SeqLength 614 ; Group 0 ; pid 100 ; SS
ggtgcttaggagaaacatgcagtagaacttttcatcaacaaatggaacacgtcacagaatTTTgctaacaatgga
caactctgcacagaaaaatgaacgcactggcaaacatcccagacgtgccagtgaaagtacagaaaggt ;
```

- ☞ GFF-like file for features (exon, intron, both)

```
ENSG00000001461 ; intron 1 ; est BM468827 ; start 23694 ; end 25507 ;
strand 1 ; s_dinuc GT ; e_dinuc AG ; rel_end 2 ; shift 1 ; hsplqs 23624 ;
hsplqe 23693 ; hsplqstr 1 ; hsplss 86 ; hsplse 155 ; hsplsstr 1 ; hsplpid
100 ; hsplval 5e-29 ; hsp2qs 25508 ; hsp2qe 25679 ; hsp2qstr 1 ; hsp2ss
156 ; hsp2se 327 ; hsp2sstr 1 ; hsp2pid 100 ; hsp2eval 4e-91 ;
ENSG00000001461 ; exon 2 ; est BM468827 ; start 25508 ; end 25679 ; strand
1 ; hsplqs 25508 ; hsplqe 25679 ; hsplqstr 1 ; hsplss 156 ; hsplse 327 ;
hsplsstr 1 ; hsplpid 100 ; hsplval 4e-91 ;
```



Output Files - 1

- ☞ Output files that we generate at the moment:
 - Requires at least one confirmed feature:
 - **.GENE** – 'flanked' gene sequence + annotation.
 - **.TRANSCRIPT** – EST/mRNA id's, description, and concise alignment line.
 - **.REFERENCE-TRANSCRIPTS** – reference (Ensembl) transcript id's with mapping to local gene sequence
 - **.INTRON** – confirmed introns + further annotation.
 - **.EXON** – confirmed exons + further annotation.
 - **.EXON-LOADER** – same as above in special format for fast db loading (internal use)
 - **.CLASSES** – the classes (grouped transcripts) indicating distinct splice patterns for each class.
 - **.CLASSES-EXTRAS** - same as above but with extra data (internal use)
 - **.SPICEPATTERN-SEQUENCE** – sequences of individual splice patterns (all exons, no CDS check)



Output Files - 2

- ☞ Output files that we generate at the moment (cont'd):
 - Requires an alternative event:
 - **.GROUPS** – file with alternative events grouped by main type and main component.
 - **.LOADER** – same as above but easy format for fast db loading (internal use).
 - **.DISTINCT_GO** – list with GO ids that map to the genes with events. Used for db loading (internal use).
 - **.PATTERN_ANNOTATION** – expression profiles for event genes and splice patterns. Only for Evoc enabled species. (internal use)
 - **.BLASTDB-GENE** – fasta file with all gene sequences for Services Blast2 interface (internal use)



Example of .CLASS file

```
>ENSG00000179222
CLASS 1
BU845038-1 ~6040..6450,6581..6644,6920..6999,7168..7259,7487..7566,7658..~7701
EX367175-1 ~5732..6450,6581..6644,6920..6999,7168..~7256
BU155558-1,2 ~5780..6450,6581..6644,6920..6999,7168..~7219
BU185730-1,2 ~5780..6450,6581..6644,6920..~6967
BU190480-1,2 ~5780..6450,6581..~6643
CLASS 2
AL557885-2 ~5027..5133,5782..6450,6581..6644,6920..6999,7168..~7258
BG831556-2 ~5083..5133,5782..6450,6581..6644,6920..~6995
BQ709714-2 ~4957..5133,5782..6450,6581..~6631
BG334111-2,5 ~4519..5133,5782..~5885
CLASS 4
BQ062811-4 ~7656..7700,7954..8016,9555..9669,10926..11311,11519..~11725
BI195994-4 ~9592..9669,10926..11311,11519..~11727
Classes with staggered overlap + same structure : (2 & 1)
Classes with staggered overlap only : (4 & 1)
```

EST id has an indication of the class(es) number(s) into which it is (or can) be grouped. Multiple classes means the transcript is ambiguously added to a class.

[Note: various classes removed, a lot of transcripts removed for example purposes]



Example of .INTRON file (.EXON similar)

```
>ENSG00000167642 (3418..26294)
TYPE: GT-AG
ELM: i1(1..18628 18628)e2(1..171 171)i2(1..4078 4078)
NUMT: 4
FSDE:tttctcgccctgctgggategctgctctctctctggggtcctggcgccgaccgagaaacgcagcatccacg
FSDI:GTGAGGGCCGGGCGGGTAGGCTGGAGGGGGGCGCAGGGGGCAGAGGCTCGGGGGTCAACGGGGTCTGA
FSAI:CAGCTCATTCTTTGTCCCTGGCAGTCTCTCGAAAGCTTTTCACTGTGCTGTTTCTTTGTCCCTTGCGAG
FSAE:agaatgccacgggtgacctggccaccagcaggaatgcagcggattcctctgtcccaagtggtagttctt
CNTX:~2975..3417,26295..26354,27557..27610,28538..~28568
BPPPT: PPT(-67, -52), BP(-64,4.17), PPT(-35, -5), BP(-28,3.59)
SSIS:9.66,8.74 (U12 -12.30)
END
```



Example of .EXON-LOADER file

```
>ENSG00000179088 (20522..20620)
TYPE: GT-AG
STRUCTURE:      R
CONFIRMED:      Y
ELM:  e2(1..99 99)
NUMT: 1
FSAI: cattttttctctaataattatcattgcatcattgtaaacctgctcttctttctccattctttgtttatag
FSAE: GGAGTTGAACTTGCAAATTAATGCTACAGTGATATGTATGAAACAAA
FSDE: GGAAGAAGAATTCTTGCTAACCATCAGACCTTTTGCAAACAGGATGCAG
FSDI: gtaatgcataagagatactatattttcttacttttttaggttaaataggtttgctccatttcagccacta
CNTX: ~3001..3067,20522..20620,97294..97362,129971..130082,192624..192995,19
6410..~197026
END
```



Example of .TRANSCRIPT file

```
AK058052 [ENSG00000179088]
g(3001..3067)e(1..67),g(20522..20620)e(68..166),g(97294..9
7362)e(167..235),g(129971..130082)e(236..347),g(192624..19
2995)e(348..719),g(196410..197026)e(720..1336)

BF882018 [ENSG00000175220]
g(2919..3020)e(16..117),g(7415..7596)e(118..299),g(7816..7
911)e(300..395),g(15311..15398)e(396..483),g(21389..21452)
e(484..547)

BG720475 [ENSG00000175220]
g(2968..3020)e(6..58),g(7415..7596)e(59..240),g(7816..7911
)e(241..336),g(15311..15398)e(337..424),g(21389..21460)e(4
25..496)

BM721831 [ENSG00000175220]
g(7414..7596)e(20..202),g(7816..7911)e(203..298),g(15311..
15398)e(299..386),g(21389..21522)e(387..520)
```



Example of .REFERENCE- TRANSCRIPTS file

```
>ENSG00000179088  ENST00000315192  UFR(1..3000
3000),e1(3001..3067 67),i1(3068..20521
17454),e2(20522..20620 99),i2(20621..97293
76673),e3(97294..97362 69),i3(97363..129970
32608),e4(129971..130082 112),i4(130083..192623
62541),e5(192624..192995 372),i5(192996..196409
3414),e6(196410..197026 617),DFR(197027..200026 3000)

>ENSG00000175220  ENST00000311956  UFR(1..3000
3000),e1(3001..3020 20),i1(3021..7414 4394),e2(7415..7596
182),i2(7597..7815 219),e3(7816..7911 96),i3(7912..15310
7399),e4(15311..15398 88),i4(15399..21388
5990),e5(21389..21520 132),i5(21521..22191
671),e6(22192..22278 87),i6(22279..22461
183),e7(22462..22560 99),i7(22561..22823
263),e8(22824..22931 108),i8(22932..23007
76),e9(23008..23084 77),i9(23085..23288
204),e10(23289..23366 78),i10(23367..23794
428),e11(23795..23923 129),i11(23924..24102
179),e12(24103..24206 104),i12(24207..24345
139),e13(24346..26487 2142),DFR(26488..29487 3000)
```



Example of .SPLICEPATTERN- SEQUENCE file

```
>ENSG00000158711  SP:1
STRUCTURE:~3120..3262,11003..11218,14056..~14916,~15818..15937,18
250..~18574  ELK4
AGAGTGTCTCCAGCTCCCGCGGAGGAGGCTGCGGGCGGCGCCCGGGATAGGGACCTGCAGCTCC
AGGGAGTTGAACTTGTCAAATTAATGTCTACAGTGATATGTATGAAACAAAGGGAAGAAGAAATTC
TTGCTAACCATCAGACCTTTTGCAAACAGGATGCAGAAATCCCTTGCTATATTTCCATTGTGAG
CAGTGCCACCCTGTGGATAGAAAGCACACCCAGTGCAAAGCACATCCCTTGTTATGAAAGAAGCTT
CAGTACCCTGCTCCAGATTCAATTAATCACATGAAGAATTTCTCTGAATCTCCTAAATTTTCGTAGT
CTACACTTTCTGAATTTTCCAGTATTTCCAGAAAGGACTCAAAATTC AATGGCGTGTAAGAACT
ACTTCATACTTGCCAGTACATAGTCCCGAGGTGTCTGTGAAGCACAGTTTCTTTTGATGAAGAAA
GCTATGAAGAATTCGTTCCCTCCTCC (etc)
```



Example of .GROUP file

```

Type      :      CASSETTE EXON (SCE,CCE-EB-5P)
Cassette exons: 16749..16845,17123..17189 [164 b]
Occurs in:      (5 & 2) is Complete CCE-EB-5P, (5 & 3) is Complete SCE,
                (5 & 1) is Complete SCE

(5 & 2) :
14855..19530 (intron) <=> 14855..16748,16846..17122,17190..19530 (introns)
e1 (14638..14854) <=> e1' (14596..14854) [42, 0]
e2 (19531..19647) <=> e2'' (19531..~19592) [0, -55]
1 Confirm. EST's <=> 2 Confirm. EST's

(5 & 3) (5 & 1) :
14855..19530 (intron) <=> 14855..16748,16846..17122,17190..19530 (introns)
e1 (14638..14854) <=> e1 (14638..14854) [0, 0]
e2 (19531..19647) <=> e2 (19531..~19647) [0, 0]
1 Confirm. EST's <=> 33 Confirm. EST's

TRPT-ISO1: ~13825..13977,14638..14854,19531..19647,20431..20597,21981..~22061
TRPT-ISO2 ~13928..13977,14638..14854,16749..16845,17123..17189,19531..19647,
20431..20597,21981..~22075
TRPT-ISO2: ~10471..10527,13826..13977,14596..14854,16749..16845,17123..17189,19531..~19592
TRPT-ISO2: ~10471..10527,13826..13977,14638..14854,16749..16845,17123..17189,19531..~19647

Type      :      EXON ISOFORM (EI-5P)
Struct    :      14596..14854 (exon) <=> 14638..14854 (exon)
Length change: -42, 0 (-42)
Occurs in:      (2 & 1) part of (none), (2 & 3) part of (none),
                (2 & 4) part of (CCE-EB-5P), (2 & 5) part of (CCE-EB-5P)

```



Example of .LOADER file

```

>ENSG00000161243
EVENT:IR
LOCATION:4020..4103
LENGTH_CHANGE:84
NBRE1:1
NBRE2:0
FORM:CIR-EB-5P
GRP_LEN_CHANGE:~196
COMPLETE:Y
PAIR-PATTERN:4-3
NUMT_ISO1:1
NUMT_ISO2:1
FEATURE_ISO1:EF:~3882..4019,I:4020..4103,EF:4104..4199
FEATURE_ISO2:IF:3365..3907,E:3908..4199,IF:4200..8222
FORM:SIR
GRP_LEN_CHANGE:196
COMPLETE:Y
PAIR-PATTERN:1-3,2-3
NUMT_ISO1:15
NUMT_ISO2:1
FEATURE_ISO1:EF:3908..4019,I:4020..4103,EF:4104..4199
FEATURE_ISO2:IF:3365..3907,E:3908..4199,IF:4200..8222

```



AltSplice Pipeline - Issues

- ☞ Trying to keep start to end within two weeks per species for basic data (excl. SNP/conservation, etc)
- ☞ Ever increasing size of EST/mRNA collection
 - Longer blast times per gene, esp. If blast index doesn't fit in memory (+file cache) anymore
 - Tried: resolve this by splitting blast db + parallel blast by gene (dblast/MPI) – not a speed increase
- ☞ I/O bound -> 120 processes accessing disk
- ☞ (Disk space – looks trivial but gene vs. gene/est blast fills up disk very fast)