VariantGrid: a graphical user interface for a scalable variant database

David M. Lawrence[1,2], Jinghua Feng[2], Andreas W. Schreiber[1,2], Joel Geoghegan[1]

[1] ACRF South Australian Cancer Genomics Facility, Centre for Cancer Biology, SA Pathology, Adelaide, South Australia

[2] School of Molecular and Biomedical Science, University of Adelaide, South Australia

Variants Are Important

Sequencing and variant calling is an increasingly popular means of detecting individual genetic variation, promising to deliver understanding, diagnosis and treatment of many genetic diseases such as inherited disorders and cancer. Variant calling is a leading driver of diagnostic adoption of high throughput sequencing.

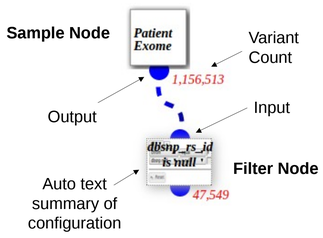
Problem

Individual variation ensures hundreds of thousands to millions of real variants in an exome/genome. Almost all are unrelated to a disorder.

Sorting through this requires expert knowledge of genes, pathways and phenotypes as well as the ability to manage enormous data sets. These skills are rarely found in the same person.

Solution

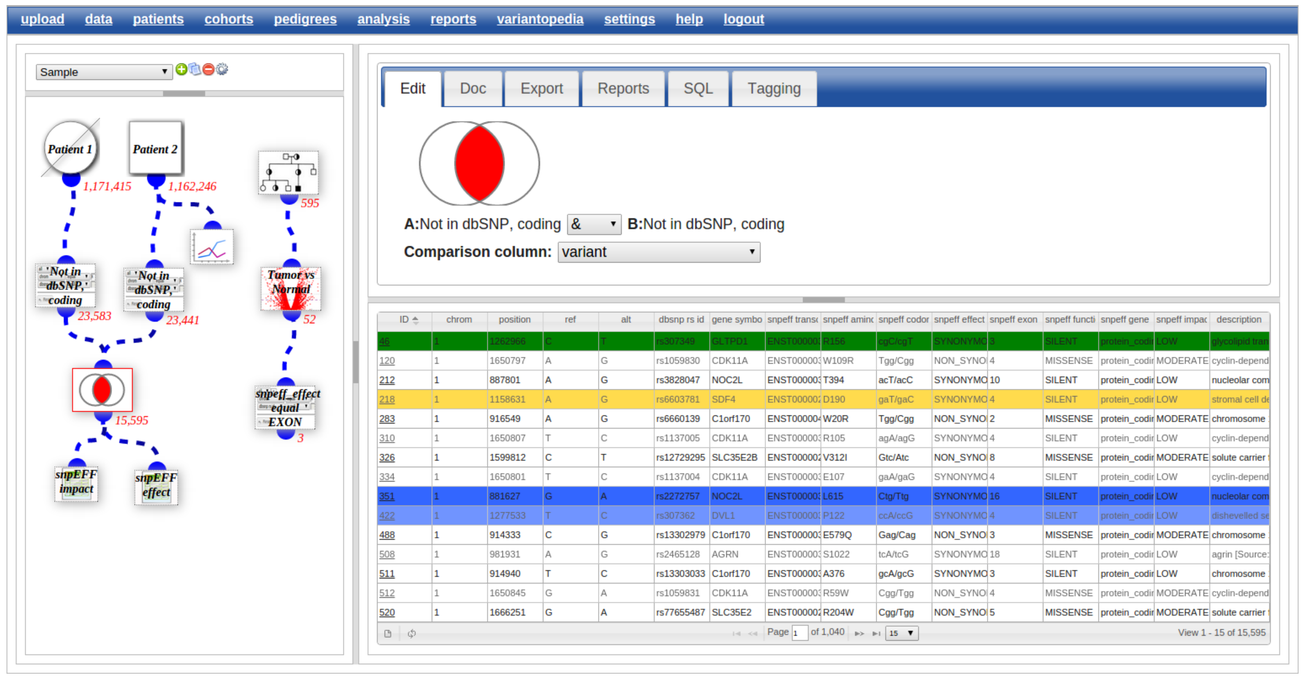
We have built a new user interface specifically designed to allow non-programming experts to search through millions of variants.



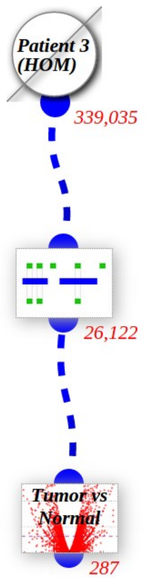
Our drag & drop interface allows users to create a Directed Acyclic Graph of variant filtering and analysis operations.

Nodes change to reflect configuration and results, allowing a high level view of the analysis.

These can be connected together to build custom analyses, which are converted into efficient queries and run on large databases.



Javascript



Python

```
class SampleNode(PipelineNode):
    def get_queryset(self):
        filters = [Q(observedvariant__sample=self.sample)]

        if self.zygosity:
            filters.append(Q(observedvariant__zygosity=self.zygosity))
        if not self.reference_alt:
            filters.append(Q(alt__isnull=False))

        return self.model.objects.filter(*filters)


class IntersectionNode(PipelineNode):
    def get_queryset(self):
        gic_id = self.genomic_intervals_collection.pk
        queryset = self.get_single_parent_queryset()
        intervals_queryset = queryset.extra(tables=["snpdb_locus", "snpdb_genomicinterval"],
                                            where=["snpdb_locus.id = snpdb_variant.locus_id",
                                                   "snpdb_genomicinterval.genomic_intervals_collection_id = %d" % gic_id,
                                                   "snpdb_locus.chrom = snpdb_genomicinterval.chrom",
                                                   "snpdb_genomicinterval.start - %d <= snpdb_locus.position" % self.left,
                                                   "snpdb_genomicinterval.end + %d >= snpdb_locus.position" % self.right])
        return intervals_queryset.distinct() # Select distinct as intervals may overlap


class ExpressionNode(PipelineNode):
    def get_queryset(self):
        cdr_qs = CuffDiffRecord.objects.filter(cuff_diff_file=self.expression_file)
        column_op = self._get_filter_op("log2_fold_change", self.comparison_op)
        cdr_qs = cdr_qs.filter(**{column_op : self.value})
        if self.significant:
            cdr_qs = cdr_qs.filter(q_value__lte=0.05)

        cdr_values = cdr_qs.values_list('gene', flat=True)
        queryset = self.get_single_parent_queryset()
        return queryset.filter(variantannotation__transcript__gene_id__in=cdr_values)
```

SQL

```
SELECT DISTINCT "snpdb_variant"."id",
                "snpdb_locus"."chrom",
                "snpdb_locus"."position",
                "snpdb_locus"."ref",
                "snpdb_variant"."alt",
                "snpdb_variantannotation"."dbsnp_rs_id",
                "snpdb_gene"."gene_symbol",
                "snpdb_variantannotation"."snpeff_transcript_id",
                "snpdb_variantannotation"."snpeff_amino_acid_change",
                "snpdb_variantannotation"."snpeff_codon_change",
                "snpdb_variantannotation"."snpeff_effect",
                "snpdb_variantannotation"."snpeff_exon_id",
                "snpdb_variantannotation"."snpeff_functional_class",
                "snpdb_variantannotation"."snpeff_gene_biotype",
                "snpdb_variantannotation"."snpeff_impact",
                "snpdb_transcript"."description"
FROM "snpdb_variant"
INNER JOIN "snpdb_observedvariant" ON ("snpdb_variant"."id" = "snpdb_observedvariant"."variant_id")
LEFT OUTER JOIN "snpdb_variantannotation" ON ("snpdb_variant"."id" = "snpdb_variantannotation"."variant_id")
LEFT OUTER JOIN "snpdb_transcript" ON ("snpdb_variantannotation"."transcript_id" = "snpdb_transcript"."ensembl_transcript_id")
LEFT OUTER JOIN "snpdb_gene" ON ("snpdb_transcript"."gene_id" = "snpdb_gene"."ensembl_gene_id")
LEFT OUTER JOIN "snpdb_locus" ON ("snpdb_variant"."locus_id" = "snpdb_locus"."id") , "snpdb_genomicinterval"
WHERE ("snpdb_observedvariant"."sample_id" = 7
       AND "snpdb_observedvariant"."zygosity" = 'O'
       AND "snpdb_variant"."alt" IS NOT NULL
       AND ("snpdb_locus"."id" = "snpdb_variant"."locus_id")
       AND ("snpdb_genomicinterval"."genomic_intervals_collection_id" = 1)
       AND ("snpdb_locus"."chrom" = "snpdb_genomicinterval"."chrom")
       AND ("snpdb_genomicinterval"."start" - 0 <= "snpdb_locus"."position")
       AND ("snpdb_genomicinterval"."end" + 0 >= "snpdb_locus"."position")
       AND "snpdb_transcript"."gene_id" IN
         (SELECT "expression_cuffdiffrecord"."gene_id"
          FROM "expression_cuffdiffrecord"
          WHERE ("expression_cuffdiffrecord"."gene_id" IS NOT NULL
                 AND "expression_cuffdiffrecord"."cuff_diff_file_id" = 9
                 AND "expression_cuffdiffrecord"."q_value" <= 0.05
                 AND "expression_cuffdiffrecord"."log2_fold_change" >= 1.0)))
```

Users add nodes, and drag & drop connections to construct a directed acyclic graph of Variant filtering operations.

Node classes return a queryset object, which represents an SQL query of Variants. This is passed down through the graph, applying more and more filters.

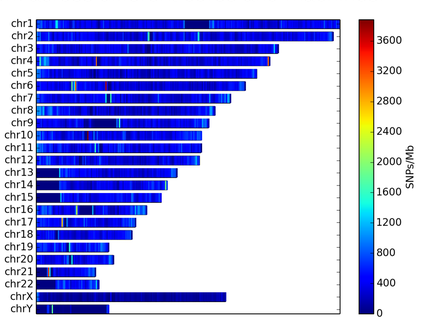
The user configured columns are added, and the queryset is converted into an SQL query, which is then efficiently executed against large datasets.

Asynchronous Tasks

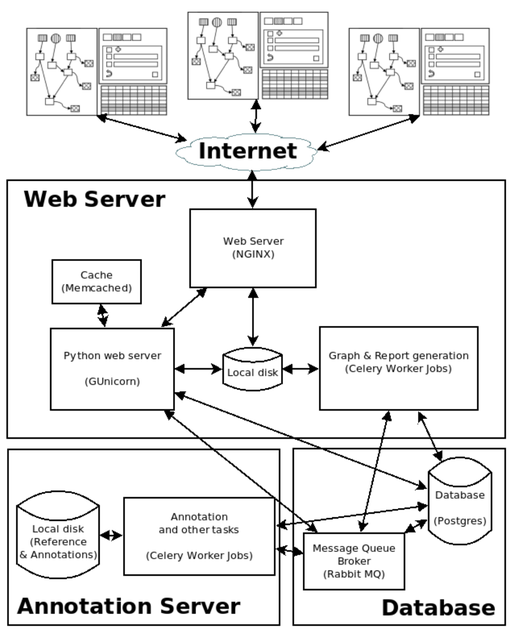
Jobs running for more than a fraction of a second are dispatched as asynchronous tasks, such as:

- Processing uploaded files
- Graphs & report generation
- Variant Annotation
- Counting variants
- Processing Pedigree inheritance models into a list of variants

This gives scalability, allows us to distribute work across different cores and machines.



Asynchronously generated graph of SNP density.



Database Design

We use standard relational database design as this is much more space efficient than VCF files.

Variants (unique values of chrom / pos / ref / alt) are stored in one table, and linked to by records from a VCF file (with sample & genotype)

This allows us to quickly perform operations such as "find all of the samples that have ever seen this variant"

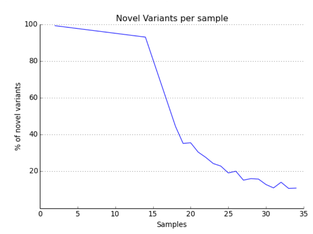
Annotations

Standard practice with VCF files is to annotate the entire file. This leads to duplicated variant level information across files, and duplicated gene information across multiple rows.

We store variant and gene annotation once and only once, joining to the appropriate tables in our queries.

After an import, automated tasks sync down novel variants to the annotation server, run annotation pipelines and re-upload them to the database. After a while, the database contains a large proportion of the variants in the population, reducing the amount of annotation required for each new sample to a much lower level.

We imported 34 exomes (43M vcf records) which contained 7.9M distinct variants. On the 34th exome, under 11% of variants were new and needed to be annotated, representing a significant space/cpu performance improvements compared to standard practices.



Final Thoughts

The thousand dollar genome does not include the cost of analysis. As sequencing throughput improves and costs drop, the relative cost of analysis will continue to grow, increasingly leaving bioinformatics as the bottleneck.

VariantGrid allows drag & drop analysis of millions of variants, helping research and diagnostics in their difficult task of extracting meaning from their data.

variantgrid.com

Centre for Cancer Biology · SA Pathology · The University of Adelaide · Australian Cancer Research Foundation · Government of South Australia · SA Health