Graph Data Modelling for Genomic Variants

Sanna Aizad
College of Engineering and Technology
University of Derby
Derby, UK
s.aizad@derby.ac.uk

Ashiq Anjum
College of Engineering and Technology
University of Derby
Derby, UK
a.anjum@derby.ac.uk

***Abstract*—Genome variant analysis is performed on Variant Call Format (VCF) files. It can take days to process these files for genome analytics due to challenges such as loading the files for each user query and processing them to answer questions of interest. As data sizes grow, timely processing of this data is putting enormous pressure on the computational resources, leading to significant processing delays and may jeopardise the ultimate goal of bringing the genomic discoveries to masses. We believe this problem will not be solved until the underlying data structure to organise and process these files undergoes a transformation. To overcome this problem, we have proposed a graph based system to represent the data in VCF files. This allows the data to be loaded once in a graph model which is then subsequently queried and processed numerous times without any additional computational and data access penalties. This helps reduce data access time by giving a constant time access to any node and addresses performance and scalability challenges that have been a limiting factor for the mass scale adoption of genome analytics. It takes only 2ms to access any data node in our graph model and remains constant for any number of nodes.**

***Keywords*-VCF; genome; variant analysis; graph model;**

I. Introduction

Latest advancements in DNA sequencing technology have made it possible to sequence the human genome quickly and cheaply, generating exabytes of data. The *variations* or differences between the genomes of different individuals are known as *variants.* These are obtained from the process of *Variant Calling* and are stored in text files called *Variant Call Format* (VCF). The variant information from the VCFs are used for interpretation of different things such as diagnosing genetic disorders, population studies and personalised medicine etc. by running the variations through a series of analysis, along with clinical annotations, to bring out meaningful insights [1] [15].

VCF files are not easy to understand which is why existing variant analysis tools and pipelines have been developed for ease-of-use. However, the biggest drawback with these is the processing of large amounts of data. For example, a VCF file from Phase 3 of the 1000 Genomes Project[1] containing 84.4 million variants from 2504 individuals is over hundred gigabyte in size. Doing data analysis on this size means discovering differences, patterns and hidden information within the genome ranging from single nucleotide sites to large nucleotide changes. The data in the VCF files is repeatedly accessed and processes are run on them iteratively. Therefore, the main challenge is to overcome the Input/Output bottleneck, CPU and memory limitations as well as decreasing the computation time.

Here, we aim to introduce a graph model which maps the human genome variations (from VCF) onto a human Reference Genome. The proposed graph model is flexible enough to represent different categories of variations such as insertions and deletions. It also allows data to be quickly browsed, which helps speed up the analysis process significantly using intelligent data analysis and system optimisation approaches [20], [21].

This paper describes the contents of the VCF file in Section II, and then explains in Section III why using graphs is a better option than VCF files. It then describes the graph model for VCF in Section IV, and how different variations are represented on this model in Section V. It then talks about implementation and results in Sections VI and VII respectively. In the end, Section VIII concludes this paper and gives future directions for this work.

II. The VCF File and Variant Analysis

VCF[2] is a standard and well-defined text file for storing variations in the genome and allows for rich annotations [2] [3]. The maintenance and expansion of the standardised format is currently with the "Global Alliance for Genomics and Health Data Working group file format team"[3].

VCF consists of two parts - the header and the body. The header carries meta-data about variant fields. The

[1]http://www.internationalgenome.org
[2]http://samtools.github.io/hts-specs/VCFv4.2.pdf
[3]https://genomicsandhealth.org/working-groups/our-work/file-formats

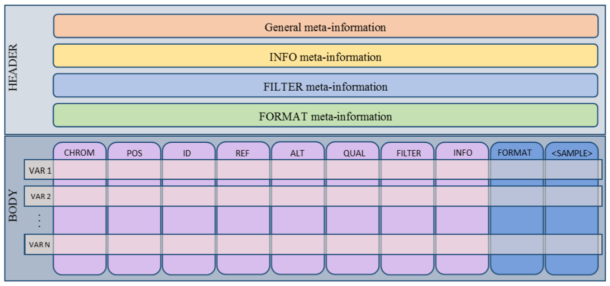


Figure 1. The VCF File data format. The header contains meta information about variants. The body has 8 mandatory fields (columns) and each row is a separate record.

body is in a tabular form with eight mandatory fields and can represent small-scale variations such as Single Nucleotide Polymorphisms (SNPs), insertion and deletions (indels) as well as other types of variations including variations observed in a population and structural variations. Each column within the body separates the fields and each row contains information about a single variation and is called a *VCF record*. Figure 1 shows the arrangement of the data within the VCF file. The VCF eliminates redundancy by not storing those portions of the genome which are the same as the Reference Genome.

In order to interpret called variants for downstream analysis, the VCF first undergoes pre-processing. It is then loaded, browsed, annotated, filtered and stored by variant analysis tools and pipelines to get meaningful insights.

Different tools have been developed to address the different challenges at the different stages of VCF analysis. BrowseVCF [2] and GEMINI [4] address the loading problem. Commercially available Ingenuity[4], Alamut[5], GoldenHelix SNP[6] and VariantStudio[7] as well as open source tools, such as, SNVerGUI [5], database.bio [6], gNOME [7], BrowseVCF [2] and VCFloci [8] aim towards providing a user-friendly interface to make data access easy while providing efficient ways to browse the data. Annotation can be added to variants in the VCFs using GEMINI [4], snpEff [9], Annovar [10] and VEP [11] [35] and Vcfanno [12]. Filtering of variants can be done using Vcffilter in vcflib[8] , Jvarkit[9] , GATK [13] and VCF-Miner [14]. Storage challenges are addressed by GEMINI [4].

The distributed analysis of large scale VCF data which is distributed geographically could be performed using HPC, grid and cloud-based approaches [16], [17], [18], [19].

III. Why graphs

A graph is an arrangement of data in the form of nodes and edges. The nodes represent the data properties and attributes, whereas the edges represent the relationships between the nodes. We have used Directed Graphs to create our graph model because the relationships between the data properties can be defined in different directions. Our graph model is made to reside inside a native graph database. Once the graph model is created and loaded to a native graph database, the data can be browsed quickly and iteratively due to the nature of the database.

On the other hand, the existing tools and pipelines browse the VCF files repeatedly to reach the different locations of interest within the files. This creates an I/O bottleneck because the files have to parsed from the top for every query. When represented as a graph, the required data node can be reached directly without having to parse through other parts of the data. This makes browsing and querying the graph easier than querying a VCF file.

IV. VCF Graph Model

In order to address the challenges associated with VCF discussed previously, we present an alternative data model which will allow for quick, efficient and iterative data analysis.

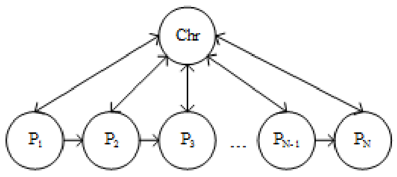


Figure 2. Graph Model of the Reference Genome: This models the nucleotides at any given locus in the Reference Genome.

The first step to create a graph model is to represent the human reference genome as a directed graph. In doing so, each nucleotide is marked where it resides on the genome by associating it with a locus. The locus can be described as the address of a nucleotide by marking the position and chromosome number. Having this address makes it easy to attach variations later.

The human Reference Genome data is read from a FASTA file which contains the chromosomes and the nucleotides at those chromosomes. In order to translate the Reference Genome to our graph model, two types of nodes are created using Algorithm 1: the chromosome node and the position node (as can be seen in Figure 2).

Each position is made into a node and holds the value “nucleotide base” taken from the same position within

[4]http://www.ingenuity.com/
[5]http://www.interactive-biosoftware.com/alamut-visual
[6]http://goldenhelix.com/SNP_Variation/index.html
[7]http://www.illumina.com/informatics/research/biological-data-interpretation/variantstudio.html
[8]https://github.com/vcflib
[9]http://lindenb.github.io/jvarkit

the FASTA file. Each chromosome node holds the value "chromosome number". Each position node is connected to a subsequent node as well as a chromosome node (using directed edges) indicating the position of that nucleotide on the given chromosome.

Algorithm 1: FASTA to Reference Genome Graph Model

```
Input: FASTA file
Output: Nodes and edges in Graph Query Language
create chromosome node $c$;
foreach char $m$ in FASTA file do
    find position $p$;
    create position node $n$;
    add attributes position $p$, nucleotide $m$ to node $n$;
    create edge from $n$ to $c$;
    if $n+1$ is not last node then
        create edge from $n$ to $n+1$;
```

Position and Chromosome nodes share the relation "at" and the two nodes can be traversed to and from each other. This allows for reading positions at any chromosome as well as reading chromosome numbers from any position. The position nodes can be only traversed in the forward direction from start to end. The relation between one position node and the next position node is "followed by".

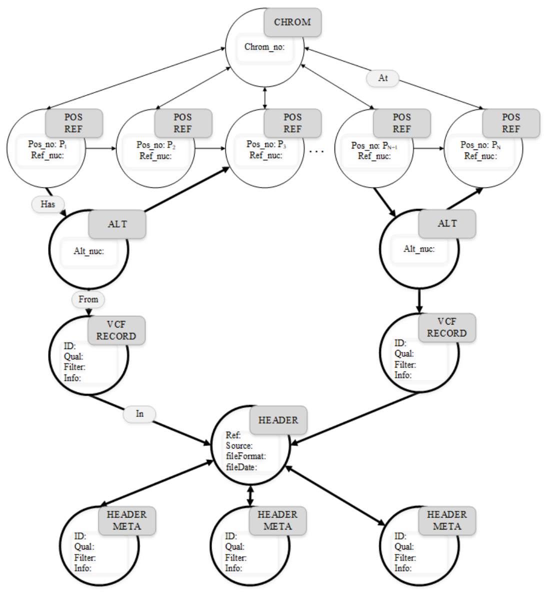


Figure 3. Variation Graph Model: The nodes and edges with bold lines are from the VCF file, where as the rest of the nodes and edges are from the Reference Genome FASTA file.

In order to map the variations from VCF to the Reference Genome, additional nodes are added to the Reference Genome graph model, using Algorithm 2, to get a Genome Variation graph model (Figure 3) to include the VCF variation records. The updated graph model now has:

- *Chromosome Node*: containing the Chromosome Number,
- *Position-Reference Node*: containing the Position Number of a nucleotide base on the chromosome and the Reference Nucleotide,
- *Alternate Node*: containing the alternate nucleotide at the given position within the chromosome from the VCF file,
- *VCF Record*: containing information about the variation from the VCF file,
- *Header Node*: linking the VCF Record to its VCF file, and
- *Header Meta Nodes*: containing meta information from the header part of the VCF file.

The information within the Alternate Node, VCF Record, Header and the HeaderMeta Node comes from the VCF file, whereas the information within the rest of the nodes comes from the FASTA file of the Reference Genome. The edges define the relationships between two nodes, and these relationships remain fixed.

Algorithm 2: VCF to Variation Graph Model

```
Input: VCF file, Reference Genome Graph Model
Output: Nodes and Edges in Graph Query Language
foreach record $m$ in VCF do
    find chromosome $c$;
    find position $p$;
    modify position node $p$ by creating an edge to the alt node $a$;
    add altNucleotide $an$ to node $a$;
    create edge from alt node $a$ to position node $pEnd$;
    create edge from alt node $a$ to vcfRecord node $v$;
    create edge from vcfRecord node $v$ to header node $h$;
    create edge from header node $h$ to headerMeta node $hm$;
```

V. Modelling the different types of variations in the graph model

VCF encodes different types of variations from Single-Nucleotide Polymorphisms (SNPs) to structural variations. Here, we talk about the different events (types of variations), how to identify them from the VCF record and how to model them onto a graph.

A. Substitution

A substitution occurs when a nucleotide base is replaced by another. In a VCF record, this would look like:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
20	3	.	C	T	.	PASS	DP=100

This VCF record shows a Single-Nucleotide Polymorphism (SNP) i.e. a variation in a single nucleotide at a given position. This means that this variation leads to two alleles. An allele can be described as a variant. In this case, the two alleles are:

Position:	P_1	P_2	P_3	P_4
Allele 1:	A	G	**C**	C
Allele 2:	A	G	**T**	C

When modelled to a graph, the path the nucleotides are following in the reference genome (indicated by the edges between the position nodes) is modified using Algorithm 2. It now includes the edges to the alternate nucleotide which is being substituted at a given position. This can be seen in Figure 4.

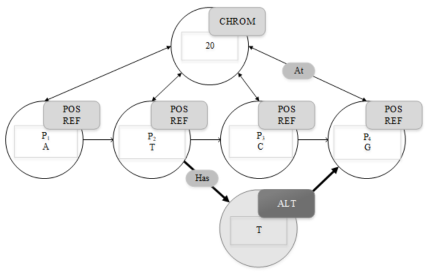


Figure 4. Substitution Model of SNP with single base substitution and two alleles.

B. Deletion

A deletion event occurs when nucleotide bases are removed from the DNA sequence altogether. These deletions could be small (usually only one to a few base pairs being removed within the region of a single gene) or large (deletion of an entire region, such as a gene or several neighbouring genes). In a VCF record, a small deletion would look like:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
20	2	.	TCG	T	.	PASS	DP=100

Here, the VCF Record shows that a deletion of two reference bases C and G has occurred. The REF field shows TCG and the ALT field shows T indicating that the reference allele TCG is being replaced by just the T (deleting CG). This deletion leads to two alleles:

Position:	P_1	P_2	P_3	P_4	P_5
Allele 1:	A	T	**C**	**G**	A
Allele 2:	A	T	-	-	A

Deletion is modelled in the graph by modifying the edges and indicating a deletion in the Alt node using Algorithm 2. This can be seen in Figure 5. The path followed by Allele 1 is through Position nodes $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$. Allele 2 follows the path $2 \rightarrow 4$. The ALT node indicates the presence of an allele and the "-" within this node indicates a deletion.

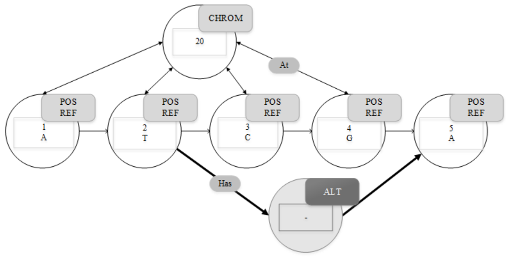


Figure 5. Deletion Model of two reference bases C and G.

C. Insertion

Insertion happens when extra nucleotide bases are added at any given position. These can be few or many, such as gene duplication, where whole genes are copied and added at a given chromosome location.

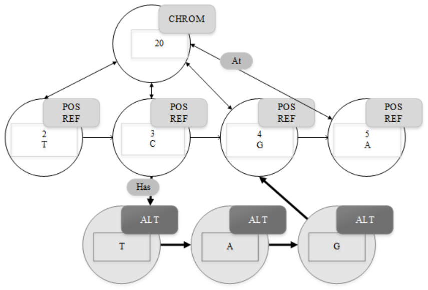


Figure 6. Insertion Model of three bases TAG.

The following VCF record shows that in the ALT column, three additional bases TAG have been inserted after the reference nucleotide base C (at the REF column):

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO
20	3	.	C	CTAG	.	PASS	DP=100

When mapped to the graph model, this VCF record shows two alleles:

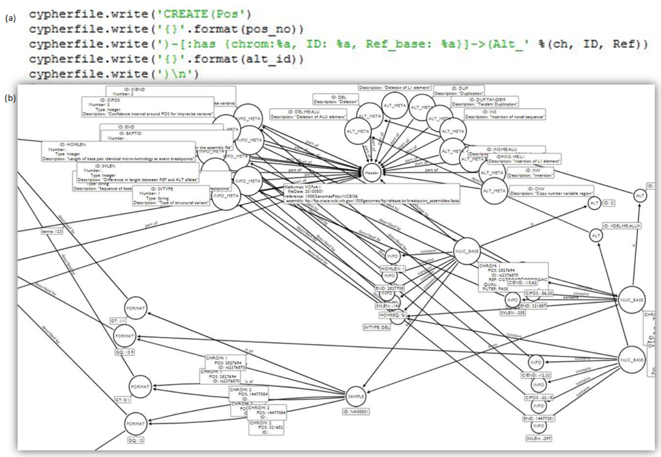
Position:	P_1	P_2	P_3				P_4	P_5
Allele 1:	A	T	C	-	-	-	G	A
Allele 2:	A	T	C	**T**	**A**	**G**	G	A

As can be seen in Figure 6, Allele 1 is represented by the path followed by the position nodes. Allele 2 follows the path through position nodes 3 and 4, but the additional bases TAG are inserted as ALT nodes between position 3 and 4. Algorithm 2 modifies the edges and adds the extra nodes in the variation to the graph model.

VI. Implementation

Algorithms 1 and 2 were used to convert the VCF file to graph. Python scripting was used to convert each VCF record to nodes and edges as Cypher queries. Cypher is a Graph Query Language used by Neo4j which allows creation of nodes and relationships. When these queries are pushed into Neo4j, they result in a graph.

Two text files, FASTA and VCF, were given as input. Each nucleotide in the FASTA file was used to write a CREATE position node query. The position number and nucleotide were made attributes to the node. Edges between the position nodes were defined as the relationship between two nodes. This was also written as a CREATE relationship query in Cypher.



```
cypherfile.write('CREATE(Pos')
cypherfile.write('{}'.format(pos_no))
cypherfile.write(')-[:has {chrom:%a, ID: %a, Ref_base: %a}]->(Alt_' %(ch, ID, Ref))
cypherfile.write('{}'.format(alt_id))
cypherfile.write(')\n')
```

Figure 7. (a) A CREATE node Cypher query being pushed into the output file. (b) Cypher queries being translated to a graph in Neo4j.

Next, each record from the body of the VCF was extracted. Different regular expressions were used to find the different fields in the columns. These were used as attributes to the nodes. Here, a new node type, ALT (containing information about the variation), was defined. The position node from the FASTA file was appended to the ALT nodes by using the MODIFY relationship command in Cypher, which allows creation of a new relationship to an existing node. This way, the different variations, such as insertion, deletion and substitution were incorporated into the graph model.

The output file contained Cypher queries to create and modify the nodes and edges. These were then pushed into Neo4j where the graph was generated. Figure 7 (a) shows a Cypher query being pushed into the output file using Python. Figure 7 (b) shows the nodes and edges in the Neo4j environment.

VII. Results

A. Data Translation and Correctness

The main goal of the Graph Model is to translate the data from VCF into a graph. Each line in the body of the VCF makes a single VCF record. Each record has eight mandatory fields, and may have additional sample fields. The graph model works around these fields and incorporates the information within.

The VCF record below translates to the graph model in Figure 8:

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	NA0001
20	10696	rs6040	A	G,T	67	PASS	NS=2	GT:GQ:DP	1|2:21:6

The following Cypher queries were used to translate the VCF record into a graph model with nodes and edges:

Nodes:

CREATE (cr20:CHROM {id:20}), (p2:POSRES {p:10965, r:'G'}),
(p3:POSRES {p:10966, r:'A'}), (p4:POSRES {p:10967, r:'C'})

Edges:

CREATE (cr20)-[:HAS]->(p2), (cr20)-[:HAS]->(p3),
(cr20)-[:HAS]->(p4), (p2)-[:TO]->(p3), (p3)-[:TO]->(p4)

These queries create four nodes. One of these nodes is of type chromosome with label "CHROM" and property "id". The other three nodes are of type position-reference with label "POSRES" and properties "p" and "r" representing position and reference nucleotide. This information was extracted from the VCF using python scripting.

Once the graph was generated, the same Query Language was used to update the graph by appending nodes with variation information. The following queries modify the graph above to include additional nodes and relationships describing variations:

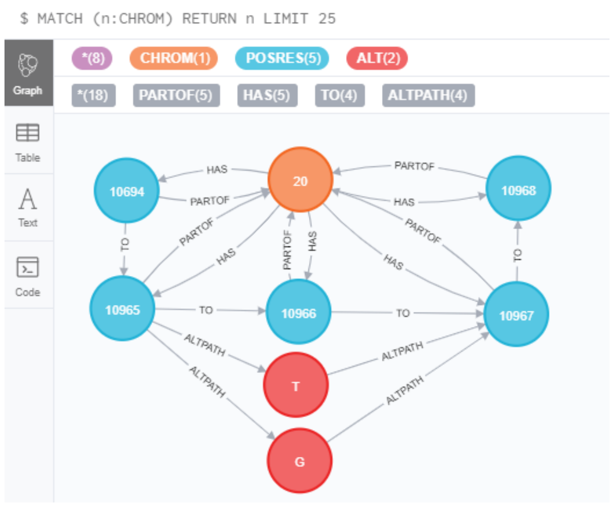
New Nodes:

CREATE (a1:ALT {a:'G'}), (a2:ALT {a:'T'})

New Edges:

MODIFY (p2)-[:ALTPATH]->(a1), (p2)-[:ALTPATH]->(a2),
(a1)-[:ALTPATH]->(p4), (a2)-[:ALTPATH]->(p4)

The Cypher queries were passed to the native graph database Neo4j (Figure 8) which generated the graph model.



```
$ MATCH (n:CHROM) RETURN n LIMIT 25
```

Figure 8. Nodes and Edges in Neo4j

As shown in Figure 8., the CHROM field from the VCF record is converted to the node type "chromosome" with label CHROM and has node property set to the value of the CHROM from the VCF record. The second node type "locus" is created with two labels POSRES. The properties of the node type "locus" are set to the position value and the reference nucleotide from the VCF record. The third node type "variation" is created with the label ALT and its node property is set to the alternative nucleotide(s) from the VCF record.

The node types "chromosome" and "locus" follow a one-to-many relationship. This is because a chromosome is indexed to have many nucleotide positions, whereas any given position is only associated with one chromosome. On the other hand, the relationship between the "locus" nodes is one-to-one in the forward direction, since the locus is sequential from the start position to the end position. The node type "locus" and node type "variation" have a one-to-many relationship since a particular locus can have one or more variations.

In order to check the correctness of the graph, we checked to see if the the entire VCF Record could be reconstructed without loss of any information by querying the graph. The following Cypher query generated the VCF Record in Figure 8 :

```
MATCH(p:POS{name: '10695'})-[:AT]-(c:CHROM{: '20'})
RETURN c.name AS #CHROM, p.name AS POS, rec.id AS
ID, r.name AS REF, alt.name AS ALT, rec.qual AS QUAL,
rec.filter AS FILTER, rec.info AS INFO rec.format AS FOR-
MAT, rec.sample AS NA0001
```

B. Performance

1) Load time: Existing tools load the VCF file before any analysis can be done. Our graph model goes through an additional step of converting the VCF to Cypher queries before being converted to a graph. It was, therefore, expected that our graph model would take a longer time to load into the environment where it would be ready to be queried.

Our graph model took 32 hours to load a VCF file from the 1000 Genomes Project containing 39.7 million variants to the Neo4j Graph Database, as compared to 28 hours by GEMINI [4] which is a relational database, as shown in Figure 9.

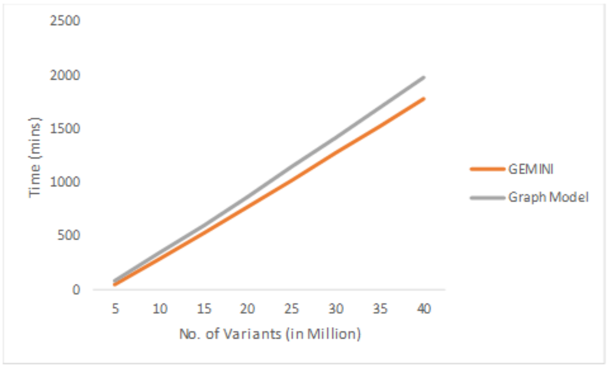


Figure 9. The graph showing Data Insertion Time as the number of variants increase.

The advantage with the graph model is that load needs to be done only once. After the graph model is loaded to the graph database, only the nodes containing the variations need to be modified, instead of loading new VCF files and all the records.

2) Browsing: It takes 2ms to access a data node of our graph model in Neo4j. This makes browsing time a lot faster since the access time is constant for any number of nodes present. In contrast, a text file is parsed till the locus is reached. Depending on whether the locus is at the beginning of the file or at the end, it can take hours to browse the data. This is shown in Figure 10.

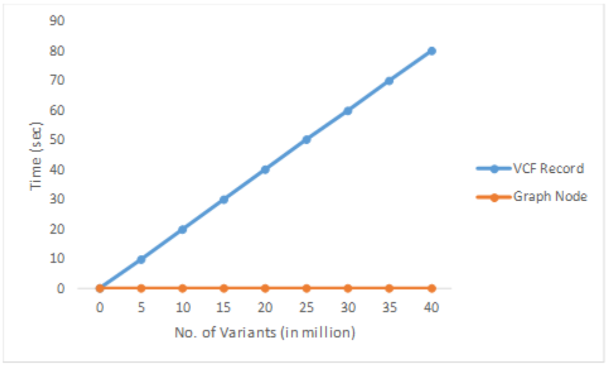


Figure 10. The graph showing the time it takes to browse a VCF record and the nodes in the Graph Model.

VIII. Conclusion and future direction

All underlying approaches in Genome Analysis make use of VCF files. Due to the iterative nature of analysis, the VCF files need to be parsed several times to extract the desired information. In order to improve the query and processing times, we introduced a graph model as an alternative data structure. The data from VCF was converted to nodes and edges in our graph. The native graph database, Neo4j was used to construct our graph. This reduced our query time significantly as it takes a constant of 2ms to reach any node, no matter how many nodes are present.

In future, we will move the graph model to a high performance in memory environment, which will make the model more efficient. It will make it faster to analyse the data, since the time to load data each time for analysis will be reduced. New variations could be added to the model, without inserting the graph model to the graph database again. The use of in-memory and HPC frameworks will make it easy to parse variations which will in turn make analysis faster, less expensive, and more accurate.

References

[1] E. M. Coonrod, R. L. Margraf, A. Russell, K. V. Voelkerding, and M. G. Reese, “Clinical analysis of genome next-generation sequencing data using the Omicia platform,” Expert review of molecular diagnostics, vol. 13, no. 6, pp. 529–540, 2013.

[2] S. Salatino, and V. Ramraj, “BrowseVCF: a web-based application and workflow to quickly prioritize disease-causative variants in VCF files,” Briefings in bioinformatics, vol. 18, no. 5, pp. 774–779, 2016.

[3] P. Danecek, A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, and R. Durbin, “The variant call format and VCFtools,” Bioinformatics, vol. 27, no. 15, pp. 2156–2158, 2011.

[4] U. Paila, B. A. Chapman, R. Kirchner, and A. R. Quinlan, “GEMINI: integrative exploration of genetic variation and genome annotations,” PLoS computational biology, vol. 9, no. 7, pp. e1003153, 2013.

[5] W. Wang, W. Hu, F. Hou, P. Hu, and Z. Wei, “SNVer-GUI: a desktop tool for variant analysis of next-generation sequencing data,” Journal of medical genetics, vol. 49, no. 12, pp. 753–755, 2012.

[6] M. Ou, R. Ma, J. Cheung, K. Lo, P. Yee, T. Luo, T. L. Chan, C. H. Au, A. Kwong, R. Luo, and T. W. Lam, “Database. bio: a web application for interpreting human variations,” Bioinformatics, vol. 31, no. 24, pp. 4035–4037, 2015.

[7] I. H. Lee, K. Lee, M. Hsing, Y. Choe, J. H. Park, S. H. Kim, J. M. Bohn, M. B. Neu, K. B. Hwang, R. C. Green, I. S. Kohane, and S. W. Kong, “Prioritizing DiseaseâĂŘLinked Variants, Genes, and Pathways with an Interactive WholeâĂŘGenome Analysis Pipeline,” Human mutation, vol. 35, no. 5, pp. 537–547, 2014.

[8] E. Paradis, T. Gosselin, J. Goudet, T. Jombart, and K. Schliep, “Linking genomics and population genetics with R,” Molecular ecology resources, vol. 17, no. 1, pp. 54–66, 2017.

[9] P. Cingolani, A. Platts, L. L. Wang, M. Coon, T. Nguyen, L. Wang, S. J. Land, X. Lu, and D. M. Ruden, “A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3,” Fly, vol. 6, no. 2, pp. 80–92, 2012.

[10] K. Wang, M. Li, and H. Hakonarson, “ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data,” Nucleic acids research, vol. 38, no. 16, pp. e164-e164, 2010.

[11] W. McLaren, B. Pritchard, D. Rios, Y. Chen, P. Flicek, and F. Cunningham, “Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor,” Bioinformatics, vol. 26, no. 16, pp. 2069–2070, 2010.

[12] B. S. Pedersen, R. M. Layer, and A. R. Quinlan, “Vcfanno: fast, flexible annotation of genetic variants,” Genome biology, vol. 17, no. 1, pp. 118, 2016.

[13] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M.A. DePristo,“ The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data,” Genome research, vol. 20, no. 9, pp. 1297-1303, 2010.

[14] S. N. Hart, P. Duffy, D. J. Quest, A. Hossain, M. A. Meiners, and J. P. Kocher, “VCF-Miner: GUI-based application for mining variants and annotations stored in VCF files,” Briefings in bioinformatics, vol. 17, no. 2, pp. 346-351, 2015.

[15] R. McClatchey, A. Branson, A. Anjum, P. Bloodsworth, I. Habib, K. Munir, J. Shamdasani, K. Soomro and neuGRID Consortium, “Providing traceability for neuroimaging analyses,” International journal of medical informatics, vol. 82, no. 9, pp.882-894, 2013.

[16] F. van Lingen, C. Steenberg, M. Thomas, A. Anjum, T. Azim, F. Khan, H. Newman, A. Ali, J. Bunn and I. Legrand, “The Clarens Web service framework for distributed scientific analysis in grid projects,” IEEE 2005 International Conference on Parallel Processing Workshops (ICPPW’05) pp. 45-52, 2005.

[17] F. Van Lingen, M. Thomas, T. Azim, I. Chitnis, A. Anjum, D. Bourilkov, M. Kulkarni, C. Steenberg, R.J. Cavanaugh, J. Bunn and J. Ukin, “Grid enabled analysis: architecture, prototype and status,” 2005.

[18] S.L. Kiani, A. Anjum, M. Knappmeyer, N. Bessis and N. Antonopoulos, “Federated broker system for pervasive context provisioning,” Journal of Systems and Software, vol. 86. No. 4, pp. 1107-1123, 2013.

[19] K. Hasham, A.D. Peris, A. Anjum, D. Evans, S. Gowdy, J.M. Hernandez, E. Huedo, D. Hufnagel, F. van Lingen, R. McClatchey and S. Metson, “CMS workflow execution using intelligent job scheduling and data access strategies,” IEEE Transactions on Nuclear Science, vol. 58, No. 3, pp. 1221-1232, 2011.

[20] R. McClatchey, I. Habib, A. Anjum, K. Munir, A. Branson, P. Bloodsworth, S.L. Kiani and neuGRID Consortium, “Intelligent grid enabled services for neuroimaging analysis,” Neurocomputing, vol. 122, pp. 88-99, 2013.

[21] I. Habib, A. Anjum, R. Mcclatchey and O. Rana, “Adapting scientific workflow structures using multi-objective optimization strategies,” ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 8, no. 1, pp. 4, 2013.