RESEARCH

3DGB: A Low-Latency, Big Database System and Browser for Storage, Querying and Visualization of 3D Genomic Data

Alexander Butyaev[1†], Ruslan Mavlyutov[2†], Mathieu Blanchette[1], Philippe Cudré-Mauroux[2] and Jérôme Waldispühl[1*]

[*]Correspondence: jeromew@cs.mcgill.ca
[1]School of Computer Science, McGill University, 3480 University Street, H3A 0E9 Montreal, Canada
Full list of author information is available at the end of the article
[†]Equal contributor

Abstract

Recent releases of genome 3D structures have the potential to transform our understanding of genomes. Nonetheless, the storage technology and visualization tools need to evolve to enable users fast and easy access to these data. We introduce simultaneously a database system to store and query 3D genomic data (3DBG), and a genome browser to visualize and explore 3D genome structures (3DGB). We benchmark 3DBG against state-of-the-art systems, and demonstrate that it is faster than previous solutions, and importantly gracefully scales with the size of data. The 3D genome browser is available at `http://3dgb.cs.mcgill.ca/`.

Keywords: Genome Browser; 3D structure; Database

Rationale

Biological motivation

The release of the first draft of the human genome [1, 2] announced the beginning of a data era in genomics. Gathering information does no longer appear as a major bottleneck in molecular biology studies. However, by contrast, storing and mining the massive amounts of data generated by genomics studies becomes increasingly difficult.

The development of efficient computing infrastructures to store and retrieve genomic information is thus an essential part of the discovery pipeline in genomic research. The UCSC genome browser [3, 4] and the Ensembl genome browser [5, 6] were among the first systems specifically developed to address this issue and opened the access of sequencing data to the whole scientific community.

Since then, the diversity and size of the genomic data continued to grow, but the technology used to store the data did not drastically evolve. Recent endeavours such as the 1000 Genomes Project [7] or the 10k Genomes Project [8] illustrate the exponential growth of sequencing data generated by new genomic projects. Yet, such projects continue to rely on the same types of storage paradigms developed more than 15 years ago.

In addition to the sheer size of the datasets, the complexity and the nature of the data themselves are also changing. In particular, the primary structure of genomes does no longer appear to contain all the information required to decipher our genetic code. Instead, recent studies suggest that the three-dimensional structure of genomes is essential to understand some regulatory mechanisms [9].

Three-dimensional structures and Hi-C data sets of Human [10, 11] or Yeast genomes [12] are now available. Viewers have been developed to visualize complete genome structures [13] and Hi-C data annotations have been integrated in classical genome browsers [14]. However, to date, there is no scalable solution to query simultaneously primary and tertiary genome structures. Moreover, unlike classical human genome browsers, these viewers are currently only available as OS-specific standalone applications that are not embedded into Web browsers.

Database technology

There has been substantial related work on storing and querying three-dimensional data. Existing approaches can be broadly categorized along three axes:

- Index-based vs. cluster-based. Most existing work builds a 3D index over the raw data, but does not attempt to physically reorganize the raw data (index-based) so that co-queried objects are stored near each other (cluster-based).
- Adaptive vs. non-adaptive. Adaptive systems adjust storage structures based on the query size and / or the data. They generally require less tuning and can typically handle a broader range of data sets than non-adaptive systems.
- On-line vs. off-line. On-line systems change their storage representation as data arrives, whereas off-line systems assume that the indexed data does not change frequently and must recompute their storage layout from scratch as data arrives.

The "classic" database structure for indexing objects along multiple dimensions is the R-Tree [15]. Unlike 3DBG, R-Trees do not *per se* cluster data and are optimized for accessing arbitrary spatial objects, rather than large amounts of data organized along 3D trajectories. Of course, it is possible to attempt to physically co-locate (cluster) objects in the same R-Tree rectangle together on disk. Even so, as R-Trees consider nested bounding rectangles to index the objects, it is very likely that if there is much data within a small area, there will be large overlaps in these bounding rectangles, resulting in many I/Os to answer any query.

There have been many optimizations to R-Trees for spatio-temporal data, including TB-Trees [16] and SEB-Trees [17]. TB-Trees are optimized R-Tree indices with special support for temporal predicates. They also do not deal well with very long 3D trajectories that tend to have very large bounding rectangles, and can include a high number of I/Os per lookup. SEB-Trees segment space and time, but are not specifically designed for indexing trajectories. Research on TB-trees and SEB-trees does not explicitly discuss how to cluster data, and both are non-adaptive (i.e., they do not reorganize previously added pages as new data arrives.)

To address the concern with very large 3D meshes or trajectories, several systems have proposed segmenting the trajectories to reduce the sizes of bounding boxes and group portions of trajectories that are near each other in space together on disk. Rasetic *et al.* [18] propose splitting trajectories into a number of sub-trajectories, and then indexing those segments in an R-Tree. They propose a formal model for the number of I/Os needed to evaluate a query, and use a dynamic programming algorithm to minimize the I/O for an average query size. 3DBG also includes an algorithm for optimally splitting 3D genomic meshes and their associated metadata, but in addition physically clusters those segments rather than just indexing them.

SETI [19] also advocates a segmentation-based approach like 3DBG. It segments incoming 3D meshes/trajectories into sub-trajectories, groups them into a collection of "spatial partitions", and then runs queries over just the spatial partitions that are most relevant to a given query. The principal differences between 3DBG and SETI are that: 1) the SETI paper does not describe how the size or geometry of partitions are selected, or whether it changes as inserts occur, which is a key contribution of 3DBG, and 2) SETI does not discuss metadata storage and clustering, read-optimized operations, or scalability features.

PIST [20] focuses on indexing individual points rather than 3D meshes. PIST is similar in spirit to 3DBG in that it attempts to optimally partition a collection of points into a variable-sized grid according to the density of the data and query size using a quad-tree like data structure. Unlike 3DBG, PIST is off-line (i.e., it does not adapt to new data being added dynamically).

A number of other systems, such as STRIPES [21], use a dual transformed space to index meshes or trajectories. While such indices are very compelling when indexing the future positions of moving objects, they are known to be suboptimal for answering historical or ad-hoc queries [22].

Spatial clustering has been extensively studied [23, 24, 25]. These approaches focus on generic methods to extract cluster information from large collections of ad hoc data points. Our clustering problem is more specific, since we deal with series of points ordered along 3D genes, and more importantly on the (potentially large) metadata associated to the 3D models.

Contribution

In this paper, we introduce a complete efficient and scalable database system to query genomes in space. The system includes two components: (i) a database 3DBG to store and query the 3D genomic data, and (ii) a web browser 3DGB to visualize and navigate 3D genome structures. As far as we are aware, 3DBG is the first database system that is online, adaptive, and cluster-based. We designed 3DBG to optimize the speed of searching and accessing genomic annotations from their 3D spatial coordinates in genome structures. We also develop a lightweight 3D genome viewer 3DGB that is fully embedded in Web browsers and accessible to any web user who wishes to browse and query 3D genome structures.

Our system aims to foster the discovery of spatial relationships between genomic elements and accelerate the large-scale analysis of space-dependent regulatory mechanisms. Here, we map data from the 1000 genomes project [7] and experimental Chip-Sequencing data [4] onto most recent 3D models of the Human genome [10, 11], and use 3DBG to mine these data. We benchmark 3DBG against state-of-the-art systems, and demonstrate than our database system is faster than previous solutions, and more importantly that it scales better with the size of data. We also illustrate the usefulness of our system and use our 3D genome Web browser to explore the 3D neighbourhood of the retinoblastoma gene (RB1) and identify potentially interesting genetic relationships between retinoblastoma and sleep disorders.

Our system is freely available at `https://github.com/mavlyutovrus/3d_genome_browser` and a sample deployment of our 3D genome explorer is accessible at `http://3dgb.cs.mcgill.ca/`.

Description of the database

3D genome database

We have implemented a fully-functional database based on YARN [26], currently one of the most promising Big Data processing framework available. 3DBG takes advantage of the lower-level distributed filesystem of YARN (HDFS) to store the data chunks over large clusters of commodity machines. Our system is based on three main components:

- a **3D client**, which is pre-loaded with low resolution 3D models of the genome considered. The client allows the user to navigate through the 3D genomic space. It dynamically retrieves hi-resolution 3D data and genomic metadata from the rest of the system as the user moves through the 3D space and makes queries about certain 3D regions.
- a **sparse, adaptive 3D index**, which dictates how genomic metadata associated to contiguous regions in the 3D space are co-located in the distributed filesystem. The 3D index translates the 3D query posed by the user into a series of data chunks that have to be retrieved from the distributed filesystem.
- **immutable data chunks** that compactly store genomic data and metadata in the distributed filesystem.

Figure 1 gives an overview of our database. We implemented our own indices and ancillary data structures to optimize all operations, and bypass the Hadoop NameNode whenever possible to reduce the end-to-end latency of the queries. We do not rely on higher-level Hadoop data structures such as those offered by HBase [27] or Impala [28], since these higher-level structures negatively impact the performance of online queries. Along similar lines, we do not directly use large-scale batch-processing features *a la* MapReduce, since they would introduce unreasonably high latencies in our context, but could take advantage of such functionalities for offline operations such as batch-updates or complex analytics.

A detailed description of each component of our database, as well as an explanation on our query insertion and query execution techniques, is available as supplementary material. The full codebase of our current implementation is available online `https://github.com/mavlyutovrus/3d_genome_browser`.

Data and Web-Services

We describe the data stored in our database and the syntax of web queries to access them. Additional instructions for javascript users can be found at `http://3dgb.cs.mcgill.ca/scripting/general_functions_to_access_3DGB_data_JSON.js`.

3D structures

Currently, three complete models of human 3D genome structures are stored in our database. We retrieve these data from [13, 11] and describe them in Table 1. It is worth noting that [11] provides individual structures for each chromosome, but no global relative arrangement of all chromosomes. For this reason, we provide independently each chromosome structure.

The structures are interpolated with a finite number of points. This number of points depends of the resolution of the model and therefore varies from one model to another.

The volume encompassing the genome structure is segmented in 3D cubic cells. Spatial queries use the coordinates of a cube (starting and ending positions on the x, y, and z axis) as an input, and return the coordinates of the interpolation points modelling the DNA chains contained within this cell. In particular, it allows us to identify the ranges of DNA subsequences within this volume.

The syntax of a query is `http://1kgenome.exascale.info/<mode>?xstart=<x1>&xend=<x2>&ystart=<y1>&yend=<y2>&zstart=<z1>&zend=<z2>`, where `<mode>` should be replaced by `js_test` to query the model from [13], or `3d` to query the model from [11]. `<x1>` to `<z2>` indicate the spatial coordinate of the cell. Queries to the structure issued from [11] should also include the chromosome number and the type of the cell (normal or leukemia). In that case, an example of a full query could be: `http://1kgenome.exascale.info/3d?chr=19&m=normal&xstart=1&xend=2&zstart=1&zend=2&ystart=1&yend=2`. The output is represented as an array of arrays, which represent contiguous chains within the volume.

Nucleotide sequences

We use GRCh38 assembly of the human genome from the UCSC genome browser as our reference human sequence [3, 4]. Nucleotide sequences can be accessed from their chromosomic location. The syntax of a query is `http://1kgenome.exascale.info/range?start=<start>&end=<end>&chrid=<chr>`, where `<start>` and `<end>` are the first and last index of the subsequence of interest, and `<chr>` is the chromosome number (N.B.: X and Y chromosomes are identified using letters X and Y instead of numbers).

Single Nucleotide Polymorphism

We store the Single Nucleotide Polymorphism (SNP) data from the 1000 Genomes Project [7]. Web users can retrieve SNPs data within a specific range of a chromosome with the following query `http://1kgenome.exascale.info/js_snp?chr=<chr>&start=<start>&end=<end>`, where `<start>` and `<end>` are the first and last index of the subsequence of interest, and `<chr>` is the chromosome number.

A query returns an array of arrays showing information for each individual SNP found within this interval. This information is represented as a 4-tuple including the SNP position, the SNP ID and the two alleles.

Experimental ChIP-Sequencing data

We recorded experimental ChIP-Sequencing (Chip-Seq) data from the ENCODE project [29] stored in the UCSC genome browser [3, 4]. These data help us to identify transcription factors binding sites (TFBS).

ChIP-Seq data can be retrieved with a query to `http://1kgenome.exascale.info/chipseq?chr=<chr>&start=<start>&end=<end>&celline=<cellid>`, where `<start>` and `<end>` are the first and last index of the subsequence of interest, `<chr>` is the chromosome number, and `<cellid>` is the cell line from which we obtained the experimental data. It is worth noting that in practice, Chip-Seq data may not always be available for all 3D structures models and cell types.

The output of such query is an array of 7-tuples that contain basic information on the Chip-Seq data. A 7-tuple stores the chromosome number, starting and ending index of the Chip-Seq peak, the transcription factor name, a normalized value (ranging from 1 to 1000) indicating the magnitude of the binding, the cell lines with similar TFBS, and a list of SNPs occurring in this binding site.

Determining single nucleotide 3D coordinates
A key feature of a system for querying genomes in space is its capacity to directly access the 3D coordinates of any nucleotide. However, 3D genome structures are often modelled with (sparse) discrete sets of points corresponding to enzyme cut sites. In that case, it is useful to directly access the closest cut site (in each strand direction) of a model.

This information is accessible with a web query to `http://1kgenome.exascale.info/chr_pos?chrid=<chr>&bp=<index>&m=<mode>`, where `<chr>` is the chromosome number, and `<index>` the sequence index of the nucleotide. The variable `<mode>` should be set at "normal" to query the GM06990 cell data or "leukemia" to query the leukemia cell data (N.B.: these key words are subject to change for more precise acronyms with the addition of new cell types). This argument can be simply ignored if the user wishes to query the K562 data. The query returns an array of triplets indicating the 3D coordinate of the closest interpolation points.

Benchmark of the database system

Experimental Setting

To evaluate the performance of our system, we used sequence read alignments of chromosome 11 available from the 1000 Genomes project [7]. This data consists of short (around 100 bases) DNA sequence reads, mapped onto the Human reference genome. We used approximatelly 1.5 billions of records, which constitute 250GB of raw data.

All data have been stored in a cluster (Hadoop version 2.2.0) of 10 machines. Worker nodes were commodity machines with Quad-Core Intel i7-2600 CPUs @ 3.40GHz, 8GB of DDR3-1600 RAM, 500GB Serial ATA HDD, running Ubuntu 12.04.2 LTS. The index node was similar, but with 16GB RAM. The replication factor was set to 3.

The main metric we take into account is response time (latency). As a matter of fact, execution time depends on the amount of records to be returned. In our context, we considered simple, uniform, and fixed-size cube queries returning from 100 to 1000 records.

Benchmarking against the PostGIS Database

The performance of storage systems can be characterized by their speed to access the data (i.e., by the average time needed to execute a query) and the influence of the size of the output on the time required for returning a response. In this section, we evaluate the performance of 3DBG compared to the PostGIS database [30] to store and query 3D neighborhoods of a genome.

We uploaded in the database a data set of approximately 1.5 GB (gigabytes) that contains the 3D coordinates of reference points of a simulated model of the

human genome [13]. All these positions were indexed in the database using the spatial index (The description of PostGIS's spatial index can be found at `http://revenant.ca/www/postgis/workshop/indexing.html`). Then, we measured the speed of reaching the data through the Java application using the PostGIS JDBC driver, and the influence of the size of the output (results of query) on the processing time.

In our experiments, we queried for all different reference points available in the model, and called the database to get all points that were stored in the cube centered around the current reference point, with a constant edge size (100, 200, 300 and 400 points). Our results are shown in Figure 3 and Figure 4.

Figure 3 shows the speed of accessing the data. Here, PosGIS has on average a query execution time well above 300 ms, and thus well above the time to gracefully retrieve and visualize data dynamically for online 3D browsing. By contrast, when we run the same experiment with 3DBG, the access time is clearly below this threshold. This observation demonstrates that 3DBG performs satisfactorily to visualize the 3D space at high resolution, while the latency of standard solutions such as PosGIS is too high, even for relatively small datasets.

Next, for each edge size (size of the neighborhood delimited by the cube query), we plot in Figure 4 the relation between the processing time and the size of the neighborhood that we wish to explore. Here again, we observe that PostGIS yields unsatisfactory latencies, which rapidly grow as we retrieve more data.

Benchmarking against the Hbase Database

To compare the performance of our back-end solution with Hbase [27], we installed an Hbase cluster (version 0.96.1) on our experimental infrastructure. We also split all data according to our index for HBase, but used a standard HBase database rather than our own chunk storage. The caching for the Hbase cluster was switched off to ensure valid results. Figure 5 shows the results. As can be observed, the execution times of our system is much lower than those of HBase. 3DBG is even several times faster than HBase for relatively small queries (left of the graph), thus ensuring a smooth navigation from the client side. Overall, both systems scale gracefully, thanks to the indexing and clustering offered by our adaptive 3D index.

Description of the 3D genome browser

3D genome Web visualization interface

Web users can access and visualize data stored in our servers via a GUI accessible at `http://3dgb.cs.mcgill.ca/`. Our client, based on dynamic Javascript mostly allows the user to navigate through the 3D structure in real-time, fetching genomic data as well as high-resolution 3D meshes representing the DNA backbone from the server. It runs on most common web browsers (Firefox and Chrome). This contrasts with previous viewers that were implemented as standalone applications for specific operating systems. The source code of our browser is freely available at `https://github.com/mavlyutovrus/3d_genome_browser`.

Before starting to explore 3D genome structures, the users must select a model. The front page of 3DGB allows users to select which model they wish to use. Currently, three complete 3D data sets have been implemented in the database. The

first is a simulation of the complete diploid human genome by T. M. Asbury *et al.* [13], while the second and third ones are recent reconstruction of individual chromosomes by T. Trieu and J. Cheng [11] for normal B-cells (GM06990) and acute lymphoblastic leukemia cells respectively. New models will be added to the database as they appear in the literature.

Once a model is selected, users access a search engine that enables them to directly request specific genomic locations (i.e. chromosome number and position), target genes, or arbitrary spatial coordinates. Queries re-direct the users to a 3D structure viewer pointing at the desired location. From there, they can explore and navigate the genome structure in real-time. The web client downloads all genomic and structural data in the neighbourhood of the query location. More data are dynamically loaded when the user travels in the 3D space. This allows a smooth exploration of the 3D genome structure on any computing device. A screenshot of the 3D genome browser is presented in Figure 2.

Web tools

The viewer implements multiple features allowing its users to access and visualize Human genome data stored in the database. At the core of 3DGB resides our ability to define and query a 3D neighbourhood, and thus to identify potential spatial relationship between genomic elements. In our viewer, a cursor of points at the centre of a box representing a neighbourhood to be explored. This neighbourhood is represented by the red box in Figure 2. The size of the box is adjustable (by scrolling up or down), allowing the users to tune the range of spatial relationships. Once a volume has been selected (directly from a query or following an exploration of the genome structure), the user can retrieve and download the list of all SNPs located within that box, or use hyperlinks to directly access detailed information stored on the NCBI databases [31] for each individual SNP. In addition, it is also possible to access the list of all genes present in the query cell.

Alternatively, the users can switch to a linear mode. In that case, the neighbourhood of the query position is defined as a sequence interval. It is equivalent to the viewing frame used in classical (i.e. one dimensional) genome browsers. This mode also allows the users to retrieve all SNPs present in this 1D neighbourhood.

The third mode enables the users to highlight transcription factor binding sites in the viewer. TFBSs are represented as coloured regions of the DNA chain. The colour indicates the intensity of the Chip-Seq experiment (green for low to red for high). The user can access detailed information about the Chip-Seq data by clicking on the TFBS region, or access UCSC genome browser records through an hyperlink.

Finally, we implemented a distance calculation tool that enables the users to automatically determine the physical distance between two points in space. We intentionally did not used physical units, but instead rely on the model coordinates. Indeed, the determination of physical distances require to interpret experimental data and make approximations which are often subject of discussions. By contrast, we believe that arbitrary units allows the users to estimate relative differences and leave them the freedom to interpret the experimental data used to obtain the 3D model.

Visualization of custom genotyping data

An important feature of our viewer is to enable users to map their private genotyping data onto reference 3D architectures, and allow them to visualize the data within our browser. This functionality is intended to provide users with tools to identify geometrical dependencies in custom genotyping data sets. The query interface allows users to upload a local file containing genotyping data. In order to prevent any formatting issues, we implemented a program to validate and convert most standard genotyping data file.

Once uploaded, the users can browse and query the 3D genome as described above. In addition to the reference data stored in our database, the users can now access simultaneously the reference SNPs collected from [32] together with those stored in the local file. To prevent any privacy issues, user data are stored locally and not transmitted to our server. A similar solution has been adopted by the UCSC genome browser [33].

Exploring of the 3D neighbourhood of a gene

Methodology and experimental settings

We illustrate the usefulness of 3DGB with an exploration of the 3D neighbourhood of the Retinoblastoma 1 gene (RB1) – a tumor suppressor gene that has been associated with many types of cancer. This experiment does not necessarily intend to provide new insights into RB1 regulatory mechanisms, but aims to demonstrate what type of novel information can be obtained with the use of 3DGB.

We started our investigation by exploring a 3D neighbourhood centred on the promoter of the RB1 gene in the 3D structure of chromosome 13 in normal B-cells (GM06990) [11]. We retrieved the list of SNPs found in the promoter region of RB1, and in other DNA strands that are not in the immediate sequence neighbourhood of RB1 promoter.

Distribution of SNPs in the 3D neighbourhood of RB1

In addition to the promoter region, we found 3 other strands in the spatial vicinity of RB1: S_1 (44762907, 45379923), S_2 (57184305, 57531250), and S_3 (58747059, 59087738). These strand are located in a radius $R = 0.2$ of RB1 transcription start site, which corresponds approximately to 88 Å.

A total of 1199 SNPs were identified in this 3D neighbourhood, for which we retrieve their associated phenotype from [31]. A complete list of these SNPs with associated phenotypes is available in the supplementary data. As expected we identified many SNPs related to various types of cancer. However, another interesting finding has been to detect the occurrence of one SNP (rs10492604) related to sleep disorders in the strand S_3. Importantly, we found only 2 SNPs related to sleep disorders in the whole chromosome. Moreover, with a distance of 230 Å from the beginning of RB1 gene ($R = 0.53$), this other SNP (rs10492507) is also in the vicinity of RB1 gene.

Previous studies have identified that children with hereditary retinoblastoma have also an increased risk of developing trilateral retinoblastoma [34]. Trilateral retinoblastoma is the combination of retinoblastoma (usually bilateral) and pineoblastoma (a tumour in the brain's pineal gland). The pineal gland secretes

multiple hormones (including melatonin) that are implicated in the regulation of sleep patterns in seasonal and circadian rhythms [35].

Although our finding does not implies any causation, it suggests possible interesting genetic relationships between retinoblastoma and sleep disorders. The scripts used in this experiment are available at `http://3dgb.cs.mcgill.ca/scripting.html`.

Conclusions

We presented 3DBG, a novel storage paradigm and database system to store and query genomic data in a 3D space, and developed a lightweight 3D genome browser to visualize and navigate these data from any internet browser.

We compared 3DBG to existing systems, and demonstrated that our technology enables us to significantly lower the latency of spatial queries. Importantly, we also showed that our system scales gracefully when handling more data. This technology aims to develop the infrastructure needed to mine big data sets generated by new large-scale genomic studies, and to prepare the next-generation of genome browsers.

Although this paper focuses on the technical description of the database system and the evaluation of its performances, we designed 3DBG to permit complex queries in the 3D space. In particular, we also aim to use our system to extract spatial-relationship between genomic elements in genome-scale studies, for example using efficient batch-oriented operations *a la* MapReduce on top of our data chunks (implementing such features is easy, as our whole system is based on the lower-levels of the Hadoop/Yarn stack). An example of such queries could be to retrieve all pairs of enhancers/promoters that are co-localized in the the 3D genome structure.

Finally, even though we did not specifically tailored 3DBG to optimize the storage space, 3DBG is already at least as efficient as existing systems. Further versions of the database will integrate compression techniques in order to reduce the space requirements and further reduce query latencies.

Acknowledgments

This work was supported in part by a Genome Canada and Genome Québec grant (Bioinformatics and Computational Biology competition) and a Canadian Institutes of Health Research grant CIHR BOP-130836 to JW and MB, and by a Natural Sciences and Engineering Research Council of Canada Discovery grant NSERC RGPIN 386596-10 to JW.

Competing interests

The author(s) declare that they have no competing interests.

Contributions

AB and RM implemented the 3D genome browser and the database system. All authors participated to the design of the system and the writing of the manuscript.

Author details

[1]School of Computer Science, McGill University, 3480 University Street, H3A 0E9 Montreal, Canada. [2]eXascale InfoLab, University of Fribourg, Bd de Pérolles 90, 1700 Fribourg, Switzerland.

References

1. Lander, E.S., Linton, L.M., Birren, B., Nusbaum, C., Zody, M.C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., Funke, R., Gage, D., Harris, K., Heaford, A., Howland, J., Kann, L., Lehoczky, J., LeVine, R., McEwan, P., McKernan, K., Meldrim, J., Mesirov, J.P., Miranda, C., Morris, W., Naylor, J., Raymond, C., Rosetti, M., Santos, R., Sheridan, A., Sougnez, C., Stange-Thomann, N., Stojanovic, N., Subramanian, A., Wyman, D., Rogers, J., Sulston, J., Ainscough, R., Beck, S., Bentley, D., Burton, J., Clee, C., Carter, N., Coulson, A., Deadman, R., Deloukas, P., Dunham, A., Dunham, I., Durbin, R., French, L., Grafham, D., Gregory, S., Hubbard, T., Humphray, S., Hunt, A., Jones, M., Lloyd, C., McMurray, A., Matthews, L., Mercer, S., Milne, S., Mullikin, J.C., Mungall, A., Plumb, R., Ross, M., Shownkeen, R., Sims, S., Waterston, R.H., Wilson, R.K., Hillier, L.W., McPherson, J.D., Marra, M.A., Mardis, E.R., Fulton, L.A., Chinwalla, A.T., Pepin, K.H., Gish, W.R., Chissoe, S.L., Wendl, M.C., Delehaunty, K.D., Miner, T.L., Delehaunty, A., Kramer, J.B., Cook, L.L., Fulton, R.S., Johnson, D.L., Minx, P.J., Clifton, S.W., Hawkins, T., Branscomb, E., Predki, P., Richardson, P., Wenning, S., Slezak, T., Doggett, N., Cheng, J.F., Olsen, A., Lucas, S., Elkin, C., Uberbacher, E., Frazier, M., Gibbs, R.A., Muzny, D.M., Scherer, S.E., Bouck, J.B., Sodergren, E.J., Worley, K.C., Rives, C.M., Gorrell, J.H., Metzker, M.L., Naylor, S.L., Kucherlapati, R.S., Nelson, D.L., Weinstock, G.M., Sakaki, Y., Fujiyama, A., Hattori, M., Yada, T., Toyoda, A., Itoh, T., Kawagoe, C., Watanabe, H., Totoki, Y., Taylor, T., Weissenbach, J., Heilig, R., Saurin, W., Artiguenave, F., Brottier, P., Bruls, T., Pelletier, E., Robert, C., Wincker, P., Smith, D.R., Doucette-Stamm, L., Rubenfield, M., Weinstock, K., Lee, H.M., Dubois, J., Rosenthal, A., Platzer, M., Nyakatura, G., Taudien, S., Rump, A., Yang, H., Yu, J., Wang, J., Huang, G., Gu, J., Hood, L., Rowen, L., Madan, A., Qin, S., Davis, R.W., Federspiel, N.A., Abola, A.P., Proctor, M.J., Myers, R.M., Schmutz, J., Dickson, M., Grimwood, J., Cox, D.R., Olson, M.V., Kaul, R., Raymond, C., Shimizu, N., Kawasaki, K., Minoshima, S., Evans, G.A., Athanasiou, M., Schultz, R., Roe, B.A., Chen, F., Pan, H., Ramser, J., Lehrach, H., Reinhardt, R., McCombie, W.R., de la Bastide, M., Dedhia, N., Blöcker, H., Hornischer, K., Nordsiek, G., Agarwala, R., Aravind, L., Bailey, J.A., Bateman, A., Batzoglou, S., Birney, E., Bork, P., Brown, D.G., Burge, C.B., Cerutti, L., Chen, H.C., Church, D., Clamp, M., Copley, R.R., Doerks, T., Eddy, S.R., Eichler, E.E., Furey, T.S., Galagan, J., Gilbert, J.G., Harmon, C., Hayashizaki, Y., Haussler, D., Hermjakob, H., Hokamp, K., Jang, W., Johnson, L.S., Jones, T.A., Kasif, S., Kaspryzk, A., Kennedy, S., Kent, W.J., Kitts, P., Koonin, E.V., Korf, I., Kulp, D., Lancet, D., Lowe, T.M., McLysaght, A., Mikkelsen, T., Moran, J.V., Mulder, N., Pollara, V.J., Ponting, C.P., Schuler, G., Schultz, J., Slater, G., Smit, A.F., Stupka, E., Szustakowski, J., Thierry-Mieg, D., Thierry-Mieg, J., Wagner, L., Wallis, J., Wheeler, R., Williams, A., Wolf, Y.I., Wolfe, K.H., Yang, S.P., Yeh, R.F., Collins, F., Guyer, M.S., Peterson, J., Felsenfeld, A., Wetterstrand, K.A., Patrinos, A., Morgan, M.J., de Jong, P., Catanese, J.J., Osoegawa, K., Shizuya, H., Choi, S., Chen, Y.J., Szustakowki, J., International Human Genome Sequencing Consortium: Initial sequencing and analysis of the human genome. Nature **409**(6822), 860–921 (2001)
2. Venter, J.C., Adams, M.D., Myers, E.W., Li, P.W., Mural, R.J., Sutton, G.G., Smith, H.O., Yandell, M., Evans, C.A., Holt, R.A., Gocayne, J.D., Amanatides, P., Ballew, R.M., Huson, D.H., Wortman, J.R., Zhang, Q., Kodira, C.D., Zheng, X.H., Chen, L., Skupski, M., Subramanian, G., Thomas, P.D., Zhang, J., Gabor Miklos, G.L., Nelson, C., Broder, S., Clark, A.G., Nadeau, J., McKusick, V.A., Zinder, N., Levine, A.J., Roberts, R.J., Simon, M., Slayman, C., Hunkapiller, M., Bolanos, R., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., Florea, L., Halpern, A., Hannenhalli, S., Kravitz, S., Levy, S., Mobarry, C., Reinert, K., Remington, K., Abu-Threideh, J., Beasley, E., Biddick, K., Bonazzi, V., Brandon, R., Cargill, M., Chandramouliswaran, I., Charlab, R., Chaturvedi, K., Deng, Z., Di Francesco, V., Dunn, P., Eilbeck, K., Evangelista, C., Gabrielian, A.E., Gan, W., Ge, W., Gong, F., Gu, Z., Guan, P., Heiman, T.J., Higgins, M.E., Ji, R.R., Ke, Z., Ketchum, K.A., Lai, Z., Lei, Y., Li, Z., Li, J., Liang, Y., Lin, X., Lu, F., Merkulov, G.V., Milshina, N., Moore, H.M., Naik, A.K., Narayan, V.A., Neelam, B., Nusskern, D., Rusch, D.B., Salzberg, S., Shao, W., Shue, B., Sun, J., Wang, Z., Wang, A., Wang, X., Wang, J., Wei, M., Wides, R., Xiao, C., Yan, C., Yao, A., Ye, J., Zhan, M., Zhang, W., Zhang, H., Zhao, Q., Zheng, L., Zhong, F., Zhong, W., Zhu, S., Zhao, S., Gilbert, D., Baumhueter, S., Spier, G., Carter, C., Cravchik, A., Woodage, T., Ali, F., An, H., Awe, A., Baldwin, D., Baden, H., Barnstead, M., Barrow, I., Beeson, K., Busam, D., Carver, A., Center, A., Cheng, M.L., Curry, L., Danaher, S., Davenport, L., Desilets, R., Dietz, S., Dodson, K., Doup, L., Ferriera, S., Garg, N., Gluecksmann, A., Hart, B., Haynes, J., Haynes, C., Heiner, C., Hladun, S., Hostin, D., Houck, J., Howland, T., Ibegwam, C., Johnson, J., Kalush, F., Kline, L., Koduru, S., Love, A., Mann, F., May, D., McCawley, S., McIntosh, T., McMullen, I., Moy, M., Moy, L., Murphy, B., Nelson, K., Pfannkoch, C., Pratts, E., Puri, V., Qureshi, H., Reardon, M., Rodriguez, R., Rogers, Y.H., Romblad, D., Ruhfel, B., Scott, R., Sitter, C., Smallwood, M., Stewart, E., Strong, R., Suh, E., Thomas, R., Tint, N.N., Tse, S., Vech, C., Wang, G., Wetter, J., Williams, S., Williams, M., Windsor, S., Winn-Deen, E., Wolfe, K., Zaveri, J., Zaveri, K., Abril, J.F., Guigó, R., Campbell, M.J., Sjolander, K.V., Karlak, B., Kejariwal, A., Mi, H., Lazareva, B., Hatton, T., Narechania, A., Diemer, K., Muruganujan, A., Guo, N., Sato, S., Bafna, V., Istrail, S., Lippert, R., Schwartz, R., Walenz, B., Yooseph, S., Allen, D., Basu, A., Baxendale, J., Blick, L., Caminha, M., Carnes-Stine, J., Caulk, P., Chiang, Y.H., Coyne, M., Dahlke, C., Mays, A., Dombroski, M., Donnelly, M., Ely, D., Esparham, S., Fosler, C., Gire, H., Glanowski, S., Glasser, K., Glodek, A., Gorokhov, M., Graham, K., Gropman, B., Harris, M., Heil, J., Henderson, S., Hoover, J., Jennings, D., Jordan, C., Jordan, J., Kasha, J., Kagan, L., Kraft, C., Levitsky, A., Lewis, M., Liu, X., Lopez, J., Ma, D., Majoros, W., McDaniel, J., Murphy, S., Newman, M., Nguyen, T., Nguyen, N., Nodell, M., Pan, S., Peck, J., Peterson, M., Rowe, W., Sanders, R., Scott, J., Simpson, M., Smith, T., Sprague, A., Stockwell, T., Turner, R., Venter, E., Wang, M., Wen, M., Wu, D., Wu, M., Xia, A., Zandieh, A., Zhu, X.: The sequence of the human genome. Science **291**(5507), 1304–51 (2001)
3. Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., Zahler, A.M., Haussler, D.: The human genome browser at ucsc. Genome Res **12**(6), 996–1006 (2002)
4. Karolchik, D., Barber, G.P., Casper, J., Clawson, H., Cline, M.S., Diekhans, M., Dreszer, T.R., Fujita, P.A., Guruvadoo, L., Haeussler, M., Harte, R.A., Heitner, S., Hinrichs, A.S., Learned, K., Lee, B.T., Li, C.H., Raney, B.J., Rhead, B., Rosenbloom, K.R., Sloan, C.A., Speir, M.L., Zweig, A.S., Haussler, D., Kuhn, R.M., Kent,

W.J.: The ucsc genome browser database: 2014 update. Nucleic Acids Res **42**(1), 764–70 (2014)
5. Hubbard, T., Barker, D., Birney, E., Cameron, G., Chen, Y., Clark, L., Cox, T., Cuff, J., Curwen, V., Down, T., Durbin, R., Eyras, E., Gilbert, J., Hammond, M., Huminiecki, L., Kasprzyk, A., Lehvaslaiho, H., Lijnzaad, P., Melsopp, C., Mongin, E., Pettett, R., Pocock, M., Potter, S., Rust, A., Schmidt, E., Searle, S., Slater, G., Smith, J., Spooner, W., Stabenau, A., Stalker, J., Stupka, E., Ureta-Vidal, A., Vastrik, I., Clamp, M.: The ensembl genome database project. Nucleic Acids Res **30**(1), 38–41 (2002)
6. Flicek, P., Amode, M.R., Barrell, D., Beal, K., Billis, K., Brent, S., Carvalho-Silva, D., Clapham, P., Coates, G., Fitzgerald, S., Gil, L., Girón, C.G., Gordon, L., Hourlier, T., Hunt, S., Johnson, N., Juettemann, T., Kähäri, A.K., Keenan, S., Kulesha, E., Martin, F.J., Maurel, T., McLaren, W.M., Murphy, D.N., Nag, R., Overduin, B., Pignatelli, M., Pritchard, B., Pritchard, E., Riat, H.S., Ruffier, M., Sheppard, D., Taylor, K., Thormann, A., Trevanion, S.J., Vullo, A., Wilder, S.P., Wilson, M., Zadissa, A., Aken, B.L., Birney, E., Cunningham, F., Harrow, J., Herrero, J., Hubbard, T.J.P., Kinsella, R., Muffato, M., Parker, A., Spudich, G., Yates, A., Zerbino, D.R., Searle, S.M.J.: Ensembl 2014. Nucleic Acids Res **42**(1), 749–55 (2014)
7. 1000 Genomes Project Consortium, Abecasis, G.R., Altshuler, D., Auton, A., Brooks, L.D., Durbin, R.M., Gibbs, R.A., Hurles, M.E., McVean, G.A.: A map of human genome variation from population-scale sequencing. Nature **467**(7319), 1061–73 (2010)
8. Genome 10K Community of Scientists: Genome 10k: a proposal to obtain whole-genome sequence for 10,000 vertebrate species. J Hered **100**(6), 659–74 (2009)
9. Mercer, T.R., Edwards, S.L., Clark, M.B., Neph, S.J., Wang, H., Stergachis, A.B., John, S., Sandstrom, R., Li, G., Sandhu, K.S., Ruan, Y., Nielsen, L.K., Mattick, J.S., Stamatoyannopoulos, J.A.: Dnase i-hypersensitive exons colocalize with promoters and distal regulatory elements. Nat Genet **45**(8), 852–9 (2013)
10. Lieberman-Aiden, E., van Berkum, N.L., Williams, L., Imakaev, M., Ragoczy, T., Telling, A., Amit, I., Lajoie, B.R., Sabo, P.J., Dorschner, M.O., Sandstrom, R., Bernstein, B., Bender, M.A., Groudine, M., Gnirke, A., Stamatoyannopoulos, J., Mirny, L.A., Lander, E.S., Dekker, J.: Comprehensive mapping of long-range interactions reveals folding principles of the human genome. Science **326**(5950), 289–93 (2009)
11. Trieu, T., Cheng, J.: Large-scale reconstruction of 3d structures of human chromosomes from chromosomal contact data. Nucleic Acids Res **42**(7), 52 (2014)
12. Duan, Z., Andronescu, M., Schutz, K., McIlwain, S., Kim, Y.J., Lee, C., Shendure, J., Fields, S., Blau, C.A., Noble, W.S.: A three-dimensional model of the yeast genome. Nature **465**(7296), 363–7 (2010)
13. Asbury, T.M., Mitman, M., Tang, J., Zheng, W.J.: Genome3d: a viewer-model framework for integrating and visualizing multi-scale epigenomic information within a three-dimensional genome. BMC Bioinformatics **11**, 444 (2010)
14. Zhou, X., Lowdon, R.F., Li, D., Lawson, H.A., Madden, P.A.F., Costello, J.F., Wang, T.: Exploring long-range genome interactions using the washu epigenome browser. Nat Methods **10**(5), 375–6 (2013)
15. Guttman, A.: R-Trees: a Dynamic Index Structure for Spatial Searching. In: SIGMOD (1984)
16. Pfoser, D., Jensen, C.S., Theodoridis, Y.: Novel Approaches to the Indexing of Moving Object Trajectories. In: VLDB (2000)
17. Song, Z., Roussopoulos, N.: SEB-tree: An Approach to Index Continuously Moving Objects. In: MDM (2003)
18. Rasetic, S., Sander, J., Elding, J., Nascimento, M.A.: A Trajectory Splitting Model for Efficient Spatio-Temporal Indexing. In: VLDB (2005)
19. Prasad, V., Adam, C., Everspaugh, C., Patel, J.M.: Indexing Large Trajectory Data Sets With SETI. In: CIDR (2003)
20. Botea, V., Mallett, D., Nascimento, M.A., Sander, J.: PIST: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data. GeoInformatica **12**(2), 143–168 (2008)
21. Patel, J.M., Chen, Y., Chakka, V.P.: STRIPES: An Efficient Index for Predicted Trajectories. In: SIGMOD (2004)
22. Porkaew, K., Lazaridis, I., Mehrotra, S.: Querying mobile objects in spatio-temporal databases. In: International Symposium on Advances in Spatial and Temporal Databases (2001)
23. Ankerst, M., Breunig, M.M., Kriegel, H.-P., Sander, J.: OPTICS: Ordering Points to Identify the Clustering Structure. In: SIGMOD (1999)
24. Wang, W., Yang, J., Muntz, R.R.: STING: A Statistical Information Grid Approach to Spatial Data Mining. In: VLDB (1997)
25. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An Effcient Data Clustering Method for Very Large Databases. In: SIGMOD (1996)
26. Apache Hadoop NextGen MapReduce. `http://hadoop.apache.org/`
27. HBase. `http://hbase.apache.org/`
28. Impala. `http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html`
29. ENCODE Project Consortium: The encode (encyclopedia of dna elements) project. Science **306**(5696), 636–40 (2004)
30. PostGIS. `http://postgis.net/`
31. Sayers, E.W., Barrett, T., Benson, D.A., Bolton, E., Bryant, S.H., Canese, K., Chetvernin, V., Church, D.M., DiCuccio, M., Federhen, S., Feolo, M., Fingerman, I.M., Geer, L.Y., Helmberg, W., Kapustin, Y., Landsman, D., Lipman, D.J., Lu, Z., Madden, T.L., Madej, T., Maglott, D.R., Marchler-Bauer, A., Miller, V., Mizrachi, I., Ostell, J., Panchenko, A., Phan, L., Pruitt, K.D., Schuler, G.D., Sequeira, E., Sherry, S.T., Shumway, M., Sirotkin, K., Slotta, D., Souvorov, A., Starchenko, G., Tatusova, T.A., Wagner, L., Wang, Y., Wilbur, W.J., Yaschenko, E., Ye, J.: Database resources of the national center for biotechnology information. Nucleic Acids Res **39**(Database issue), 38–51 (2011)
32. Rustici, G., Kolesnikov, N., Brandizi, M., Burdett, T., Dylag, M., Emam, I., Farne, A., Hastings, E., Ison, J., Keays, M., Kurbatova, N., Malone, J., Mani, R., Mupo, A., Pedro Pereira, R., Pilicheva, E., Rung, J., Sharma, A., Tang, Y.A., Ternent, T., Tikhonov, A., Welter, D., Williams, E., Brazma, A., Parkinson, H., Sarkans, U.: Arrayexpress update–trends in database growth and links to data analysis tools. Nucleic Acids Res **41**(Database issue), 987–90 (2013)

33. Haeussler, M., Raney, B.J., Hinrichs, A.S., Clawson, H., Zweig, A.S., Karolchik, D., Casper, J., Speir, M.L., Haussler, D., Kent, W.J.: Navigating protected genomics data with ucsc genome browser in a box. Bioinformatics (2014)
34. de Jong, M.C., Kors, W.A., de Graaf, P., Castelijns, J.A., Kivelä, T., Moll, A.C.: Trilateral retinoblastoma: a systematic review and meta-analysis. Lancet Oncol **15**(10), 1157–67 (2014)
35. Macchi, M.M., Bruce, J.N.: Human pineal physiology and functional significance of melatonin. Front Neuroendocrinol **25**(3-4), 177–95 (2004)
36. Cudre-Mauroux, P., Wu, E., Madden, S.: Trajstore: An adaptive storage system for very large trajectory data sets. In: ICDE, pp. 109–120. IEEE, ??? (2010)
37. Pagel, B.-U., Six, H.-W., Toben, H., Widmayer, P.: Towards an Analysis of Range Query Performance in Spatial Data Structures. In: PODS (1993)
38. Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems: The Complete Book, 2nd edn. Prentice Hall Press, Upper Saddle River, NJ, USA (2008)
39. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., Zdonik, S.: C-store: A column-oriented dbms. In: Proceedings of the 31st International Conference on Very Large Data Bases. VLDB '05, pp. 553–564. VLDB Endowment, ??? (2005). http://dl.acm.org/citation.cfm?id=1083592.1083658

Additional Files

Methods

Adaptive 3D Index

3DBG was devised to efficiently handle extremely large amounts of genomic data associated to fine-grained 3D models. Classical spatial indices such as R-Trees are inappropriate to our context, since they consider too many overlapping bounding boxes (and hence, too many distinct reads) when indexing complex 3D models and their associated metadata (see the database description section). Instead, we adapt our own, state-of-the-art TrajStore [36] spatial index to efficiently index arbitrary large amounts of genomic data and metadata along three dimensions.

Our index has the three following desirable properties: it is i) sparse (i.e., it does not grow linearly as the indexed data grows), ii) non-overlapping (yielding limited index look-ups event for 3D range queries on complex meshes) and iii) adaptive (i.e., it continuously adjusts to the local densities of the indexed data). Formally, our basic index structure is an adaptive 3D octree (i.e., a 3D multi-level grid, that can also be represented as a tree data structure in which each internal node has exactly eight children), where each cell (sub-cube) in the octree corresponds to a distinct entry in the chunk index (see below).

Our octree index is always updated to yield the minimal query execution cost given a set of queries $q \in W$ on the current indexed data points. In our Big Data context where potentially large amounts of data are scanned to answer the queries, the query execution cost is dominated by the cost of *reading large chunks of data* from the distributed filesystem holding the bulk of the data. State-of-the-art distributed filesystems (e.g., like the very popular HDFS[1]) organize data into large contiguous blocks (typically 64 MB) to amortize the cost of locating and reading data from the distributed infrastructure. Our goal in the following is to take advantage of those large blocks in order to regroup as much relevant data as possible in such chunks. Hence, we attempt to intelligently *co-locate spatially adjacent data* from the 3D space into the same 1D chunks on disk, in order to *minimize* the number of data chunks accessed through the filesystem to answer a given spatial query. We hence express the cost of a query as follows:

$$Cost(q) \sim \#Chunks\ accessed.$$

If the query only targets a small, homogeneous cubic cell in the octree $cell_w^3$ with a density $\mathcal{D}$ of data points per square unit and an average of $\mathcal{C}$ chunks of data and metadata stored per indexed point, the query answering cost becomes:

$$Cost_{cell}(q) \sim \left\lceil (cell_w^3)\ \mathcal{C}\ \mathcal{D} \right\rceil$$

We note that the above function is highly non-linear—a cell with no point costs 0 (in practice, keeping track of empty cells can be done without disk access) whereas a cell containing a one-byte metadata associated to a single point costs 1, corresponding to the cost of locating, accessing and reading a large data chunk (several megabytes) in the distributed file system used to store the actual data. We see from this expression how important it is to reorganize and co-locate spatially adjacent data on disk, in order to avoid accessing many chunks containing only little relevant data.

To cover a larger homogeneous region of volume $volume$, more cells are needed. The cost associated with accessing this region given a random spatial query of size $q_x \times q_y \times q_z$ for a given cell size is thus [37]:

$$Cost_{area}(q) \sim \sum_{cell} P(q \cap cell) \left\lceil (cell_w^3)\ \mathcal{C}\ \mathcal{D} \right\rceil$$

where $P(q \cap cell)$ is the probability that the random query intersects the cell $cell$. This probability depends on the spatial extents of both the query and the cell. Clearly, the query intersects a cell when its center falls within the boundaries of the cell. It also intersects the cell when its center falls just outside of the cell (e.g., if its center is up to a distance of $\{q_x, q_y, q_z\}/2$ of the corresponding edge of the cell [18]). For a volume $volume$ and by neglecting border effects that happen at the edges of the region (outside of which queries would not be allowed), the probability of a random query q intersecting a given cell is thus:

$$P(q \cap cell) \sim \frac{(cell_w + q_x)(cell_w + q_y)(cell_w + q_z)}{volume}.$$

By substituting this probability in the cost expression, we obtain the final cost expression for a query:

$$Cost_{area}(q) = \sum_{cell} \frac{(cell_w + q_x)(cell_w + q_y)(cell_w + q_z)}{volume} \left\lceil (cell_w^3)\ \mathcal{C}\ \mathcal{D} \right\rceil.$$

Our 3D spatial index dynamically splits cells into 8 sub-cells or merge 8 sub-cells into one super-cell in order to yield the minimal cost, for a (training or observed) sample workload W defined by a series of queries q_i and for a given state of the database:

$$octree_{opt} \leftarrow \underset{octree}{\operatorname{argmin}} \sum_{q_i \in W} Cost_{octree}(q_i).$$

[1] `http://hadoop.apache.org/docs/stable1/hdfs_design.html`

We describe the exact technique through which we build and maintain our optimal octree index below (Database Insertions and Update section).
Using the octree and a low-resolution 3D model of the genes, the 3D index translates the 3D query issued by the client into a series of chunk ids that hold the targeted data and that have to be retrieved from the distributed filesystem. Query execution is described in greater detail below the Query execution section.

Immutable Data Chunks
Finally, the data itself is stored compactly on disk in series of immutable data chunks. The size of each data chunk is fixed, and is dictated by the underlying distributed filesystem used[2]. Each data chunk corresponds to one and exactly one cell in the 3D index. Each data chunk contains data or metadata pertaining to the gene portions contained in its corresponding 3D cell in the 3D index. Each chunk typically regroups data pertaining to several nucleotides (from one or several genes) that are adjacent in the 3D space.
Data is laid-out in the chunk using a standard slotted-page mechanism [38]: the page starts with an index pointing to different subregions (slots) in the chunk (e.g., by base index, where each entry points to the exact location where the information associated to each nucleotide is stored). Each chunk may either contain one type or several types of data/metadata depending on the exact application. Our architecture is agnostic to the exact contents of the slots, which may store structured or unstructured data of varying size. In our experimental evaluation (see main manuscript), we for instance store all information currently available from the 1'000 genomes project[3] as well as higher-resolution 3D models of the genes. Other applications could for instance store arbitrary semi-structured data (e.g., serialized in XML or RDF/OWL) associated to the nucleotides.
The chunks in 3DBG are immutable and cannot be updated once written. Only append-only operations (e.g., when adding some data at the end of chunk if space permits) are allowed. This allows us to dense-pack data into the chunks (potentially even compressing the payload) in order to optimize scan operations on the chunks. When data needs to be modified in the chunks or when new information needs to be inserted, a batch-oriented operation is launched in order to completely swipe and rewrite all affected chunks. Such operations are very expensive in our system and should only be carried out whenever necessary. In case the application requires frequent punctual updates, a write-optimized store should be implemented in addition to our read-optimized chunks (this is today a common solution, promulgated by the latest generations of columnar database systems like C-store [39]).

Data Insertions & Updates
As mentioned above, data is typically inserted or updated in a batch-oriented manner in our system. Inserts and updates are expensive in 3DBG—since they often require to update the 3D octree index and rewrite the data chunks, though their cost is amortized when considered in bulk.
Data insertion proceeds as follows:

1. first, the data to be inserted is associated to the corresponding 3D points p in the low-resolution 3D model. For each point $p_i \in P$ in the model, the system computes the total amount of data [in MB] associated to this point by summing up the corresponding data already stored in the chunks to the newly inserted data. A weight w_p corresponding to that amount of data stored is hence derived for each point affected by the update.
2. for each cell $cell$ in the octree, the system updates the cost associated to the cell by summing the weights w_p of all points p appearing in the cell. The resulting cost for a given query q is:

$$Cost_{cell}(q) = \frac{(cell_w + q_x)(cell_w + q_y)(cell_w + q_z)}{volume} \left\lceil \frac{\sum_{p \in cell} w_p}{ChunkSize} \right\rceil .$$

   The total cost follows by summing on all queries as described previously.
3. a similar cost is recomputed for the cell considered, by splitting this time the cell into 8 subcells (hence, by considering finer cells in the octree for this region) and by summing the individual costs of the eight sub-cells potentially considered. The configuration (i.e., either the whole cell or the 8 sub-cells) yielding the smaller cost is chosen.
4. points 2. and 3. are repeated until the octree yielding the minimal query cost is found, by iteratively splitting some of the cells or sub-cells in eight.
5. finally, once the optimal octree index is found, the overall 3D index is updated and the affected chunks are written onto disk to consider the newly inserted data as well as the new organization of the index.

Updates and deletes are handled in a similar manner, but may also require 8 sub-cells to be merged into one parent cell when the weight of some of the points diminishes (e.g., when data is deleted or replaced by more compact data). The index constructed in this way is optimal, in the sense that the expected cost of answering the workload is iteratively minimized by splitting and merging cells. This is of paramount importance in our context, where query resolution is directly dependent on the reorganization of the data into the filesystem chunks, dictated by the organization of the cells in the index. Big cells are inadequate for smaller queries, as they contain many segments and data points that do not intersect with the user query. Smaller cells, on the other hand, require more filesystem accesses to retrieve a given spatial region. The optimal grid cell is hence challenging to determine, as it depends both on the exact spatial extents of the 3D objects, the volume of the associated data, and the query load.

[2] typically, 32 MB, 64 MB, or 128 MB
[3] `http://www.1000genomes.org/`

Query Execution

Finally, we describe the end-to-end query resolution strategy we devised for 3DBG using the various structures and techniques explained above. The client first issues some query by navigating through the 3D genomic space or by selecting some area of interest in that space. The exact query is sent to our back-end system, which processes it as follows:

1. first, the 3D range query issued by the user (e.g., $q \equiv metadata\ A \in [x_1, x_2][y_1, y_2][z_1, z_2]$) is compared with the 3D octree; all octree cells that intersect the query are selected.
2. for each selected cell $cell$ in the octree, the system determines the corresponding list of chunks $\{C_1, \ldots, C_N\}$ holding the actual data in the filesystem
3. for each chunk C considered, the system determines whether or not it holds the type of data requested by the client (metadata of type A in our example query); in case the chunk is relevant to the query, the system issues a subquery to the distributed filesystem to fetch the chunk and select the appropriate data contained in the chunk
4. each subquery is processed in parallel in the distributed filesystem, which reads the targeted chunk, extracts the relevant pieces of data from the payload of the chunk, and sends the results directly back to the client
5. finally, the client gathers all pieces of data retrieved from the distributed filesystem and displays the desired information to the end-user (e.g, all metadata of type A in the selected region).

The whole process has been optimized to allow for interactive queries even with very large amounts of related metadata. Points 1. to 3. are straightforward, given that our index is sparse, non-overlapping, and given that it keeps relevant metadata about each data chunk. Point 4. is executed in parallel in a *scale-out* fashion. It is definitely the most expensive part of our query resolution but can be handled efficiently by adding new machines to the back-end as data or query traffic grows (our back-end is in that sense *scalable* and *elastic*). Each read operation from the distributed filesystem is amortized as much as possible in 3DBG since i) chunks co-locate spatially contiguous data and metadata and since ii) chunks are optimized for reads.

Cell type	Organism	Type	Scale	Number of fragments	Reference
K562	human	simulated	whole genome	1	[13]
B-cell GM06990	human	real	individual chromosomes	13	[11]
B-cell leukemia	human	real	individual chromosomes	13	[11]

Table 1 Origin and description of the 3D models stored in 3DBG.

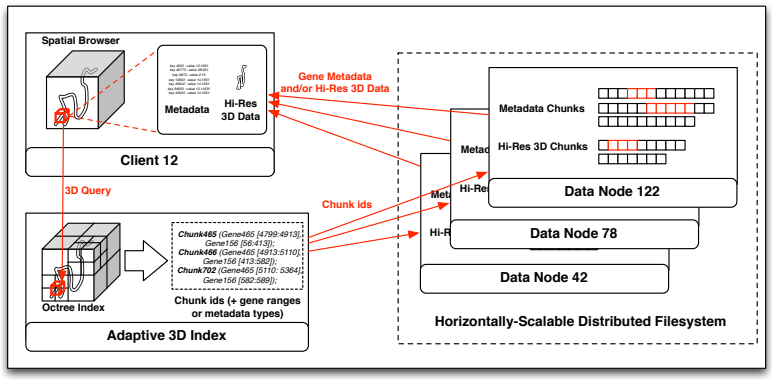

Figure 1 The overall architecture of 3DBG

Figure 2 A Sample screenshot of our 3D Web browser

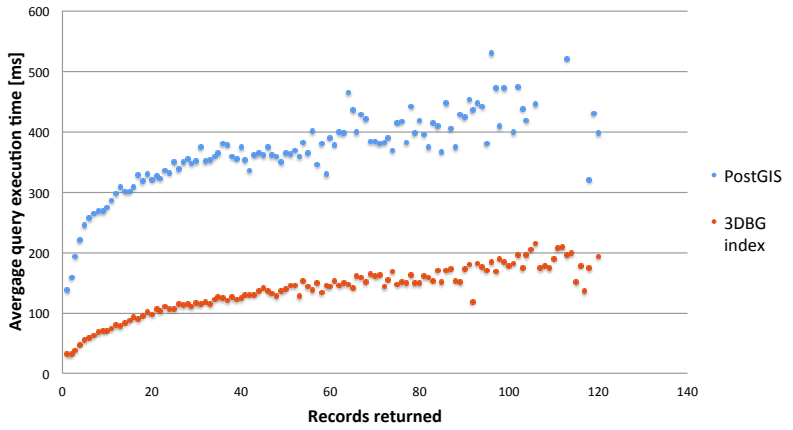

Figure 3 Comparison of latencies for reaching the data through 3DBG and PostGIS. The x-axis shows the number of record returned and the y-axis the latency in ms.

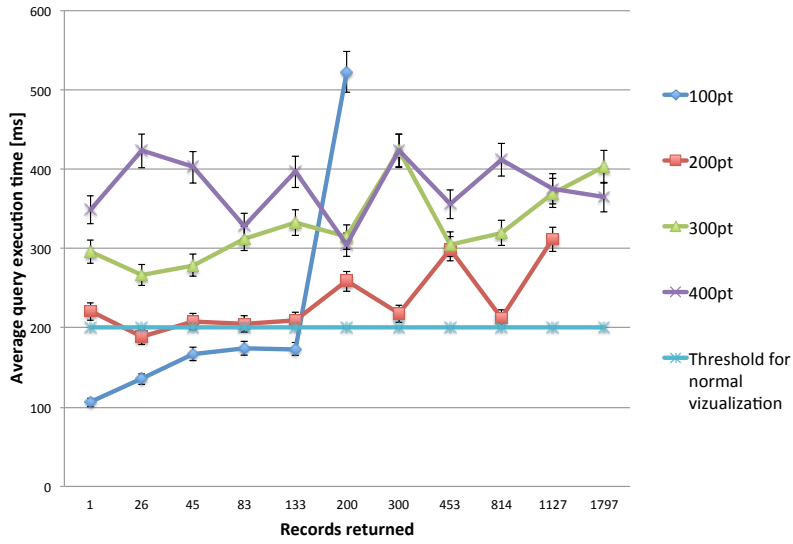

Figure 4 Comparison of latencies (y-axis) with the size of the cube query (100 points to 400 points depending of the curve considered) and number of records retrieved (x-axis).

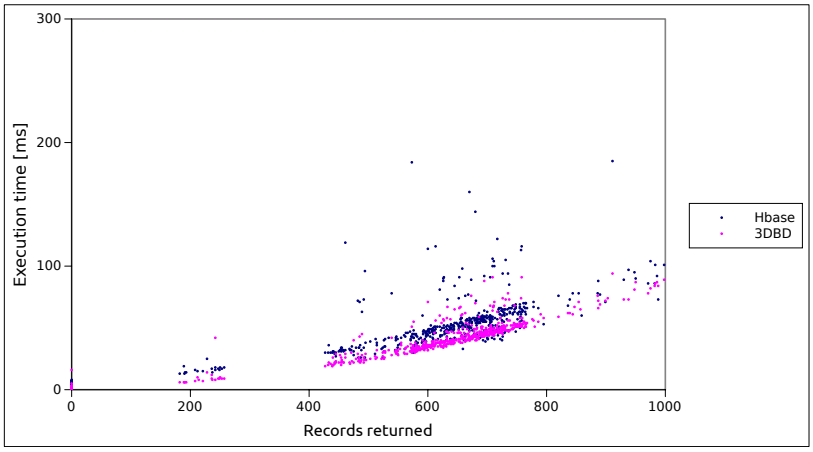

Figure 5 Comparison of query latencies between 3DBG and HBase. The x-axis shows the number of record returned and the y-axis the latency in ms.