Information Sciences 000 (2016) 1–20

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Indexing Next-Generation Sequencing data

Vahid Jalili*, Matteo Matteucci, Marco Masseroli, Stefano Ceri

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) Politecnico di Milano, Piazza Leonardo da Vinci, 32, Milan, Italy

ARTICLE INFO

Article history:
Received 16 February 2016
Revised 11 July 2016
Accepted 26 August 2016
Available online xxx

Keywords:
Genomic computing
Domain-specific data indexing
Region-based operations and calculus
Data integration

ABSTRACT

Next-Generation Sequencing (NGS), also known as high-throughput sequencing, has opened the possibility of a comprehensive characterization of the genomic and epigenomic landscapes, giving answers to fundamental questions for biological and clinical research, e.g., how DNA-protein interactions and chromatin structure affect gene activity, how cancer develops, how much complex diseases such as diabetes or cancer depend on personal (epi)genomic traits, opening the road to personalized and precision medicine.

In this context, our research has focused on *sense-making*, e.g., discovering how heterogeneous DNA regions concur to determine particular biological processes or phenotypes. Towards such discovery, characteristic operations to be performed on region data regard identifying co-occurrences of regions, from different biological tests and/or of distinct semantic types, possibly within a certain distance from each others and/or from DNA regions with known structural or functional properties.

In this paper, we present Di3, a 1D Interval Inverted Index, acting as a multi-resolution single-dimension data structure for interval-based data queries. Di3 is defined at data access layer, independent from data layer, business logic layer, and presentation layer; this design makes Di3 adaptable to any underlying persistence technology based on key-value pairs, spanning from classical B+ tree to LevelDB and Apache HBase, and makes Di3 suitable for different business logic and presentation layer scenarios.

We demonstrate the effectiveness of Di3 as a general purpose genomic region manipulation tool, with a console-level interface, and as a software component used within MuSERA, a tool for comparative analysis of region data replicates from NGS ChIP-seq and DNase-seq tests.

1. Introduction

Next-Generation Sequencing (NGS) is a family of technologies for precisely, quickly and cheaply reading the DNA or RNA of biological samples [49,50], producing huge amounts of data. Large-scale sequencing projects are spreading and very numerous genomic features, produced by processing NGS raw data, are collected by research centers, often organized through world-wide consortia, e.g., ENCODE [17], TCGA [58], 1000 Genomes Project [1], Roadmap Epigenomics [47], and others.

The availability of NGS data has opened the possibility of a comprehensive characterization of genomic and epigenomic landscapes. Answers to fundamental questions for biological and clinical research are hidden in these data, e.g., how DNA-protein interactions and chromatin structure affect gene activity, how cancer develops, how much complex diseases such as

* Corresponding author.
E-mail address: vahid.jalili@polimi.it (V. Jalili).

http://dx.doi.org/10.1016/j.ins.2016.08.085
0020-0255/

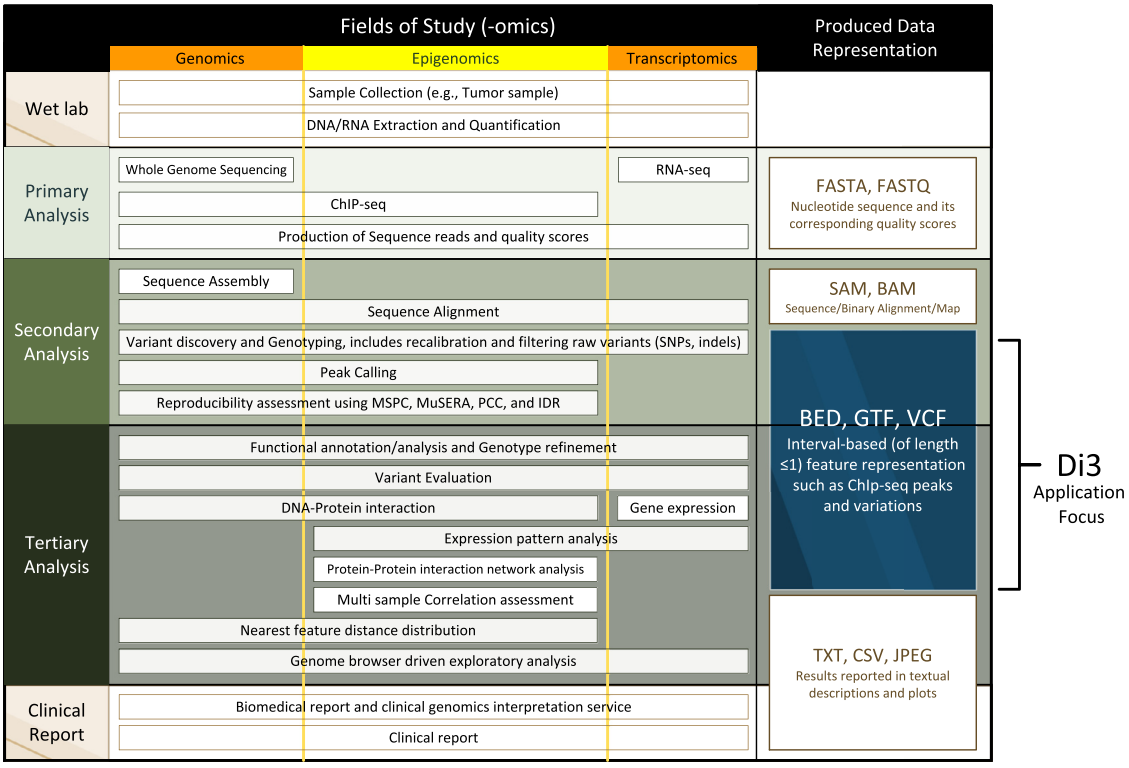

Fig. 1. Di3 application focus.

diabetes or cancer depend on personal (epi)genomic traits. Personalized and precision medicine based on genomic information is becoming a reality; the potential for data querying, analysis and sharing may be considered as the biggest and most compelling big data problem of mankind.

NGS technologies [9,48] allow collecting genome-wide genomic and epigenomic features, including DNA mutations or variations (DNA-seq), transcriptome profiles (RNA-seq), DNA methylations (BS-seq), DNA-protein interactions and chromatin characterizations (ChIP-seq and DNase-seq) [12,42]. The processing of raw data (i.e., NGS reads) produced by these technologies returns lists of regions of cellular DNA, characterized by some common properties; such regions, often referred as *peaks* (of NGS reads), are defined through their linear genomic coordinates and they are usually associated with several attribute values, including a statistical significance score, i.e., a *p*-value [44,62].

Fig. 1 summarizes the various kinds of data which are produced by NGS technologies. Data analysis is partitioned into three phases; *primary analysis* focuses on the production of sequence reads; *secondary analysis* focuses on alignment of raw data (short reads) to reference genomes and on feature calling. *Tertiary analysis* is responsible of sense-making, e.g., discovering how heterogeneous features/regions synergically concur to determine particular biological processes or phenotypes. Fig. 1 shows that the various analysis phases apply to genomics, epigenomics and transgenomics, and also shows the most relevant types of data representations used at each phase.

Our main research focus is on tertiary analysis; we recently proposed a new holistic approach to genomic data modeling and querying[1] that takes advantage of cloud-based computing to manage heterogeneous data produced by NGS technologies. In [37], we introduced the novel *GenoMetric Query Language* (GMQL), built on an abstract model for genomic data; we sketched out its main operations and demonstrated its usefulness, expressive power and flexibility through multiple different examples of biological interest (including finding ChIP-seq peaks in promoter regions, finding distal bindings in transcription regulatory regions, associating transcriptomics and epigenomics, and finding somatic mutations in exons).

We also developed methods for secondary data analysis, with a focus on data integration. Indeed, NGS experimental protocols recommend the production of at least two replicates for each sequenced sample, in order to reduce the number of false-positive calls and “rescue” (i.e., call) regions with low significance score which would probably be discarded in a single sample evaluation, but that are supported by a sufficiently strong evidence when combined across multiple replicate samples. To perform such task, and assess the reproducibility in high-throughput experiments, we recently proposed a novel

[1] http://www.bioinformatics.deib.polimi.it/genomic_computing/

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

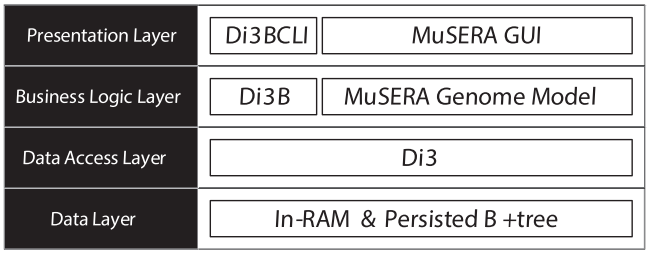

Fig. 2. Two applications' design of Di3.

method [27], named Multiple Sample Peak Calling (MSPC),[2] which has been then implemented in MuSERA [28],[3] an efficient tool for locally combining evidence on replicates and for interactive graphical evaluations of results in the genomic context (see Fig. 1 for MSPC and MuSERA application focus).

1.1. Our contribution

Both GMQL and MuSERA are grounded on the efficient execution of operations for the composition and comparison of (epi)genomic regions, and their associated attributes. Region-based operations to be performed towards these goals include the identification of co-occurrences or accumulations of regions, possibly from different biological tests and/or of distinct semantic types, within the same area of the DNA, sometimes within a certain distance from each others and/or from DNA regions with known structural or functional properties (e.g., describing particular DNA sequence motifs, genes involved in certain biochemical pathways, or regulatory regions of gene transcription activity). Nowadays, such complex operations are only partially supported by existing tools, e.g., BEDTools [43], BEDOPS [40], and GROK [41]; these tools typically support only algebraic operations based on the genomic coordinates of the regions within a single data sample or a pair of samples at the time, requiring the use of scripts to perform complex operations on multiple data samples.

To cope efficiently with complex region calculus, we have developed the 1D Interval Inverted Index (Di3), a multi-resolution single-dimension indexing framework (see Fig. 1). Di3 is defined at the data access layer, and it is independent from data layer, business logic layer, and presentation layer (see Fig. 2). This design decision has two significant advantages; firstly, being independent from the data layer, Di3 is adaptable to any key-value pair persistence technology. These may range from Apache Cassandra, LevelDB, Kyoto Cabinet, and Berkeley DB for persisted large scale data (NoSQL databases are surveyed in [26] and [56]), to simple in-memory key-value collections implemented by most of modern programming languages (e.g., "Dictionary" in C# and "map" in C++).[4] Secondly, Di3 design can support different business logic and presentation layer scenarios, which are complemented by user-defined functions (UDF) provided via behavioral design patterns such as *strategy pattern* [57]. To best of our knowledge, Di3 is the first comprehensive indexing framework for interval-based NGS data.

In this paper, we describe two contexts of Di3 application:

- General-purpose, self-contained use of Di3 from the Di3 Bioinformatics (Di3B) command-line interface (Di3BCLI) that provides console-level accessibility to Di3B operations (see Fig. 2). This is also used for performance evaluation.
- Use of Di3 as a component within the MuSERA tool [28], showing how Di3 adapts to the general requirements of secondary and tertiary data analysis (see Fig. 2).

1.2. Outline

This paper is organized as follows. Section 2 presents the state of the art. Section 3 is dedicated to the Di3 method, further sub-structured in its model, data structure, and supported operations. Section 4 reports experiments which assess the effectiveness of Di3 with big data sets and compare to BEDOPS and BEDTools, which are currently used in the state of the art for region-based operations. In Section 5, we demonstrate how Di3 is used within MuSERA, a recently developed tool for multiple sample peak calling.

2. Related work

The sequence data, mainly produced by the pipelines of primary and secondary analysis, are relatively well-studied; efficient algorithms and techniques for inference and querying on such data are developed. Some examples are iBLAST [13],

[2] http://www.bioinformatics.deib.polimi.it/MSPC/
[3] http://www.bioinformatics.deib.polimi.it/MuSERA/
[4] In this manuscript, we use an implementation of Di3 which relies on classical B+ tree and runs both in-memory and persistent.

and an indexed version of BLAST [4], or a vectorization-based method [59], which benefits SIMD (single instruction, multiple data) operations that are widely used features of modern CPUs, or [31] and [39], all with the objective of improving performance and accuracy of sequencing matching. However, genomics interval-based data, mainly produced by the pipelines of secondary and tertiary analysis, has recently drawn attention. In general, the cornerstone of interval-based data manipulation in genomics is interval intersection; 'given a query interval, find all reference intervals that overlap it' [7,22]. Interval intersection is fundamental for retrieval on annotation data and integration of diverse biological information using a reference genome [3]. Queries to retrieve features overlapping specific regions are frequently used when examining genomic features (e.g., on a genome browser). Such queries might be executed by linearly scanning of entire file, that might be reasonable for few retrievals. However, reading the entire file is an inefficient strategy in long run. A trivial solution would be adopting database systems [30,54]; however, the limited functionality and poor performance practices [3] of generic database system, in addition to the complexity of setting up and designing schemas of the database for an average end user, discourage adoption of such systems. Therefore, a wide range of research is focused on efficient algorithms for interval intersection, and many applications implement it. In general, interval intersection algorithms are of three types, described as follows.

i. Tree-based, e.g., BITS [34] (binary tree), and Segtor [45] (segment tree). Previous work for Di3 can be traced to classical search trees, such as interval-tree [14], segment-tree [5], range-tree [6], or Fenwick tree [18]; these are optimal solutions, each for particular interval-based retrieval, and some are used in common bioinformatics tools as underlying data structure. For instance, UCSC Genome Browser, BEDTools, and SAMTOOLS [36] use R-trees [25]; so that intervals of a sample are partitioned into hierarchical 'bins', and query intervals are compared with 'bins' to narrow the search for interval intersection to a focused portion of the genome. The algorithm is inefficient, as it necessitates a linear-sweep on all the intervals in each candidate bin for those overlapping the reference. Additionally, such algorithms are poor candidates for parallelism, because non-uniformly distributed intervals (which are common for ChIP-seq, RNA-seq, and exome sequencing) unbalance bin loads; consequently, some bins take considerably longer time to be processed than others. Additionally, individually such data structures do not provide a comprehensive solution for tertiary analysis challenges. For instance, retrieval queries such as 'find all the intervals (i.e., regions) intersecting with a given interval' can be determined in $O(\log_2 n)$, with n the number of searched intervals, using interval-tree; while queries such as 'find the n-th closest interval' require re-mastering the very same data structure, which becomes inefficient. Moreover, some of such data structures are mainly designed as in-memory data structures; and when made persisted, they cause a severe overhead in terms of Input/Output operations.
ii. Plane-sweep-based on pre-sorted samples, e.g., FJoint [46] which concurrently scans pre-sorted query sample and reference intervals to find overlaps. Some tools use plane-sweep-based algorithms on pre-sorted samples; for instance, recent versions of BEDTools and BEDOPS use plane-sweep as complement to tree data structure. Theoretically, plane-sweep-based algorithms are optimal; except for parallelization considering 'sweep invariant' challenge [38]: 'given a sweep line and its associated split, all intersections and event points are known between the sweep line and the beginning of the split'. Plane-sweep algorithms are parallelized by partitioning the input statically or dynamically, such that each partition can be swept without violating 'sweep invariant' [38]. However, parallelization degree of this method is limited to the number of partitions, and execution load of partitions is prone to non-uniformly distributed data.
iii. Index-based, e.g., NC-lists [3]. Efficient retrieval requires a dedicated data structure, such as an indexed flat file; for instance BAM [36], or Tabix [35] that adapts BAM indexing techniques for generic tab-delimited files, convert a file of sequential access to its equivalent random access file. Such data structures are commonly built during preprocessing steps, and are persisted for further references. Building such data structures may take a considerably long time; however, as the processing time is significantly reduced, the pre-processing and processing balance is encouraging. The observation led to the conception of BAM format. A common method is Nested Containment lists (NC-lists) [3]; an extension to NC-lists is [63]) which is mainly designed for: 'given a query interval, find all reference intervals that overlap it'. Commonly, algorithms and tools available in the bioinformatics community are for pairwise intersection of genomic intervals. Extension to beyond pairwise is commonly provided by custom scripts and algorithms, but they suffer scalability and orthogonality, which makes it difficult to perform a comparative analysis on a collection of homogeneous samples. Identifying intersecting intervals of multiple samples is an important aspect to study a variety of biological characteristics. To address the challenge, a 'slice-then-sweep' algorithm [33] for N-Way interval intersection was developed.

Temporal databases and spatial data structures provide solutions for interval manipulation, from efficient storage and retrieval to manifold operations on top of that. Generally, in the temporal databases the challenge is formalized as object/tuple versioning, e.g., [51,55], a big data spatio-temporal storage applied to geographical informations; in such methods, objects have different versions at various timespans, and can be valid or invalid at a given period, see also [16]. Other works have shown temporal indexing schemas which handle the different notions of time, as well as non-temporal attributes, by using classical B+ tree [23] or by exploiting the built-in functionalities of relational database systems [53]. Such systems hold the concept of time; hence, they render *past* (*history*), *now*, and *future*. Consequently, the storage technology is commonly 'append-only', and removing an object is generally defined as invalidating the object at a given timespan. Additionally, temporal operations commonly target recent events; hence, events prior to a certain time point are usually archived to reduce the amount of information to be processed for regular queries.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

A common prerequisite in temporal database design is that the update, insertion, and deletion operations target *now* and the history is intact (e.g., John's paid salary at last year does not change); however, few works exist concerning bi-temporal data, where updating *past* is possible through an additional dimension [29]. Compared to temporal databases, genomic databases have a fundamental difference: they do not hold any synonym for temporal model concerning *past*, *now*, and *future* concepts, as all events on the genomics domain are of equal importance from the location perspective; manipulation functions can target any position on the genome, and thus storage, indexing, and operations should be targeted at this aim.

Similarly to what we propose here, several works have proposed the embedding of region query processing functions within libraries that can be integrated within programs [10,41]. In particular, GROK [41] presents a rather elegant mathematical formalism based on set algebra; its authors propose a genomic region abstraction (that may represents reads, genomic variants, mutations, and so on), and then define a set of region operations, delivered as the *Genomic Region Operation Kit* (GROK) library. In comparison, GROK supports lower-level abstractions than Di3 and some low-level operations (e.g., flipping regions) that Di3 does not directly support, but they must be embedded into C++ programming language code. Furthermore, high-level declarative operations, such as `COVER` and `MAP` (see Section 3.1), can be encoded in GROK, but they must be invoked from line editors or C++ programs.

BEDTools [43] and BEDOPS [40] are customarily used by biologists for processing region data in BED format; they can be used from within software environments for bioinformatics (e.g., *BioPerl*, *BioPython*, *R* or *Bioconductor*), and support algebraic operations based on the genomic coordinates of regions, but only within a single or a pair of data samples at the time, requiring the use of scripts to evaluate multiple data samples. A thorough comparison of Di3 with BEDTools and BEDOPS is presented in Section 4.

Alternative approaches exist to efficiently process intervals in the Apache Hadoop ecosystem [8,15] by implementing algorithms which partition the operands in order to speed up their evaluation. In [11] the authors propose an algorithm based on data binning; recently, Afrati et al. [2] further analyze binning-based algorithms in order to assess their computation bounds. For Apache Spark [61], similar problems have been addressed by projects such as GeoSpark [52,60]. Besides cloud-based systems, scientific databases can also be used to support genomic computing, including *Vertica*[5] (used by the Broad Institute and NY Genome Center), and *SciDB*[6] (further enhanced by *Paradigm4*[7], a company whose products include genomics adds-on to SciDB and access to NGS data from TCGA and 1000 Genomes Project).

Several organizations are considering genomics at a global level. For instance, *Global Alliance for Genomics and Health* (GA4GH)[8] is a large consortium of over 200 research institutions with the goal of supporting voluntary and secure sharing of genomic and clinical data; their work on data interoperability is producing a conversion technology for the sharing of data on DNA sequences and genomic variations [20]. Also *Google* recently provided an API to store, process, explore, and share DNA sequence reads, alignments and variant calls, using Google's cloud infrastructure.[9]

3. Di3 design

Di3 is a general-purpose indexing framework which provides fast access to intervals; therefore, it applies to several domains, including genomics (any genomic region is a linear interval defined by the genomic coordinates of the region ends). Di3 main strengths are its ability to adapt to domain needs, thanks to the native support for user-defined functions (UDFs), and its portability to several implementation technologies, thanks to its high-level, layered design that abstracts from implementation details.

Di3 models homogeneous and heterogeneous intervals on a domain; related events are collected in sets, which collectively constitute a *sample*. For instance, a genomic data sample may contain regions (i.e., intervals) of DNA-protein interactions occurring on a genome under an experimental setup; each interval can be associated with values, e.g., a significance score.

In general, let $\mathbb{S} = \{S_1, \ldots S_j, \ldots S_J\}$ denote the available samples, where each sample is a set of intervals $S_j = \{I_1^j, \ldots I_i^j, \ldots I_{|S_j|}^j\}$; each interval $I = [\underline{I}, \bar{I})$ is included within its lower (left) and upper (right) bounds (ends). Di3 organizes intervals by means of *snapshots* (see Fig. 3); each snapshot corresponds to a point on the domain, and is associated with all the intervals overlapping that point. More precisely, each snapshot B_b is a key-value pair element, where the key (e_b) is the coordinate of the snapshot on the domain, and the value (λ_b) is a set of pointers to descriptive metadata of all the intervals overlapping the coordinate e_b.

3.1. Di3 operations

Fig. 4 summarizes the design of Di3 from a high-level perspective. It starts from the presentation and business logics of the two applications that are discussed in this paper, then it enumerates Di3 abstract uses at a semantic level, then

[5] https://www.vertica.com/
[6] http://www.scidb.org/
[7] http://www.paradigm4.com/
[8] http://genomicsandhealth.org/
[9] https://cloud.google.com/genomics/

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

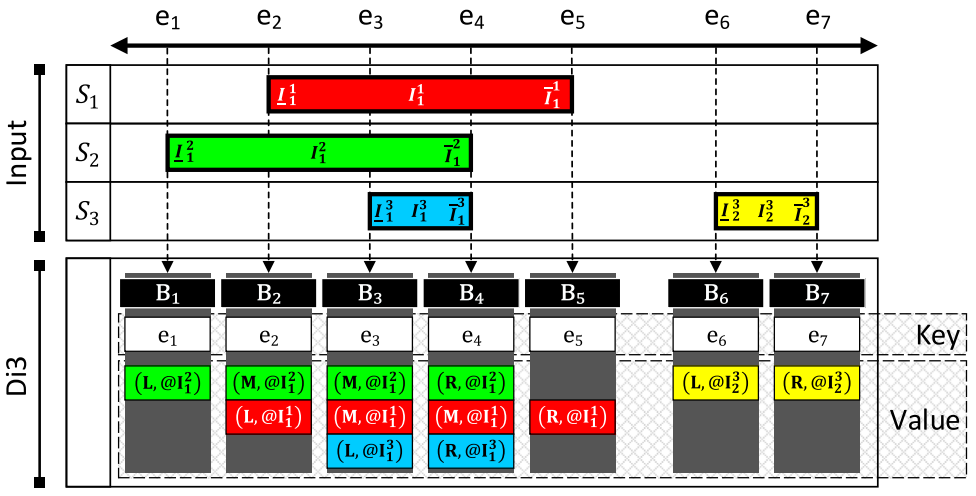

Fig. 3. Di3 data structure constructed for four input regions.

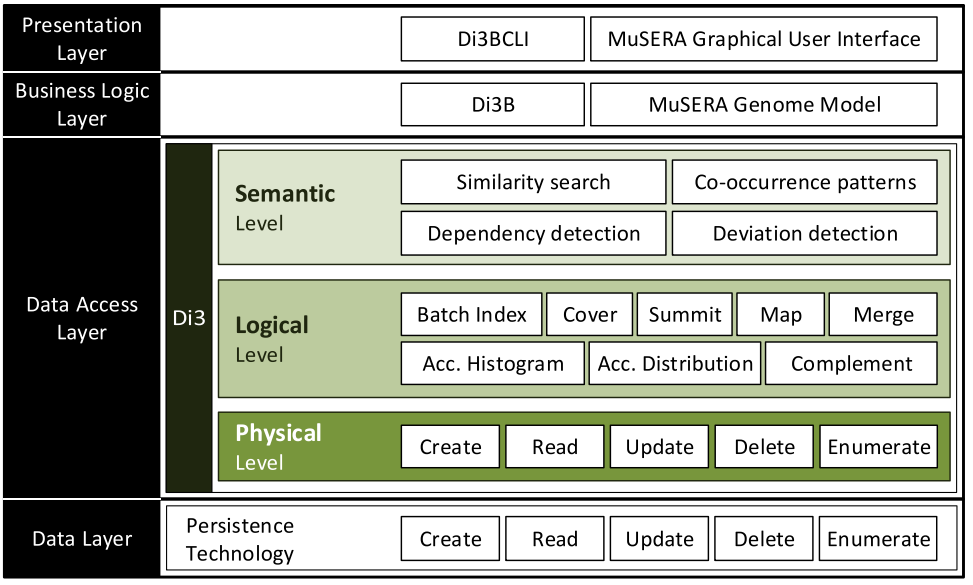

Fig. 4. The layers of Di3 and supported abstractions and operations at each level.

shows the supported logical operations that make such abstractions possible, and then shows that the physical operations are matched to a layer providing the persistence technology. Next, each Di3 layer is discussed (bottom-up.)

The physical level functions bridge the Di3 data model to the data layer, using some key-value pair persistence technology. Operations at the physical layer include *Create*, *Read*, *Update*, *Deleted* (CRUD), and *Enumerate*. These operations create and manipulate the snapshots and organize them in a key-value pair persistence technology, by translating input intervals to snapshots and retrieving intervals from snapshots. They are internal to Di3 and accordingly do not incorporate UDFs.

Logical level functions leverage on physical level operations, and provide the essential elements for region calculus. These functions cover classical region calculus that benefits from the information of a single snapshot, e.g., 'given a point on the domain, find intervals overlapping with it' (similar to queries on segment-tree), or that leverages on information provided by a set of consecutive snapshots, e.g., 'given an interval, find all intervals overlapping with it' (similar to queries on interval-tree). Logical level functions leverage on snapshots to optimally retrieve co-occurrences of intervals, or co-occurrence histograms and distributions; some of them define the Di3 public application programming interface (API), whereas other functions can be user-defined.

Upon physical level operations and logical level functions, Di3 builds semantic level functions. The goal of these functions is to facilitate both high-level reasoning on data that include coordinate-attribute criteria, and UDFs creation for extensibility to application requirements. These functions are: *similarity search*, which finds samples that best match the criteria defined in a query; *co-occurrence patterns*, which searches for density-based co-occurrence patterns; *dependency detection*, which

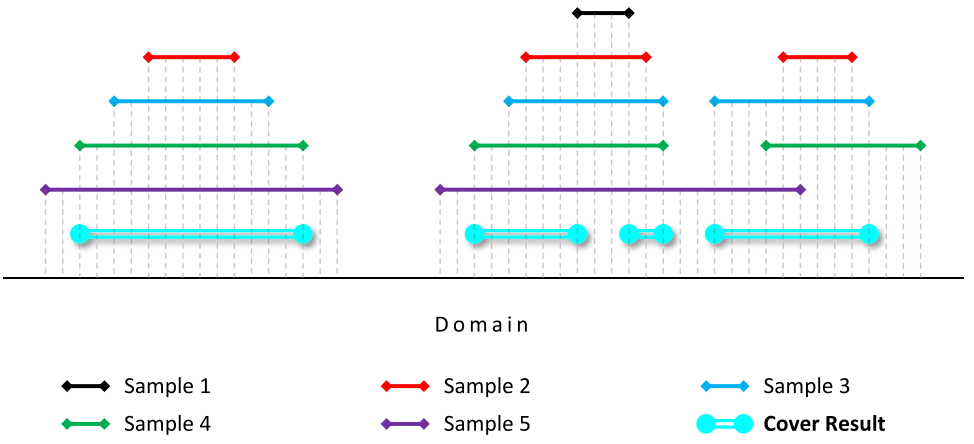

Fig. 5. An example of COVER function with minAcc=2 and maxAcc=4 on 5 samples.

Algorithm 1 Cover algorithm.

```
1: procedure COVER(minAcc, maxAcc, UDF)
2:     determined_intervals ← ∅
3:     accumulation ← 0
4:     B_marked ← null
5:     for each snapshot B in first resolution of Di3 do
6:         accumulation ← accumulation at B
7:         if minAcc ≤ accumulation ≤ maxAcc then
8:             B_marked ← B
9:             determined_intervals ← determined_intervals ∪ intervals of B
10:        else
11:            if B_makred ≠ null AND (accumulation > maxAcc OR accumulation < minAcc) then
12:                determined_intervals ← determined_intervals ∪ intervals of B
13:                UDF(B_makred, B, determined_intervals)
14:                B_marked ← null
15:                determined_intervals ← ∅
```

determines the positions on the domain where query regions co-occur; and *deviation detection*, which finds positions on the domain where the regions of a given set do not commonly co-occur, based on the information stored in the Di3 model.

In this paper we focus on the logical layer, being this the *de-facto* Di3 API, whereas the physical layer operations are strictly related to the specific persistence technology used for the Di3 application scenario, and the semantic layer functions are application dependent (some of them are used by MuSERA and are described in Section 5). Concerning the logical layer, we have the following operations natively supported by Di3:

Cover. The COVER function applies to snapshots and computes a single sample from intervals constituting the snapshots, by taking into account interval intersections and a UDF. Each resulting interval I is the contiguous intersection of at least minAcc (minimum accumulation) and at most maxAcc (maximum accumulation) of intervals. See Fig. 5 as an example of the COVER function, and Algorithm 1 for COVER pseudocode. For each resulting interval I, the COVER function determines contributing intervals, which then are passed to the UDF for further evaluation. A UDF may assign to a resulting region I any user-defined value type, spanning from 'list of all contributing intervals' or 'cardinality of contributing intervals', to analytical evaluations such as 'custom aggregation of significance values'. In COVER, the use of UDFs is thus supported, being count the default aggregation function.

Summit. The SUMMIT function is a variation of the COVER function; similarly it takes minAcc, maxAcc and a UDF, and reports the local intersections of COVER. In other words, SUMMIT returns regions that start from a position where the number of intersecting regions is between minAcc and maxAcc and does not increase afterwards, and stop at a position where either the number of intersecting regions decreases, or it violates the maxAcc parameter (see Fig. 6). Similar to COVER, SUMMIT determines contributing intervals and passes them to the UDF, which can aggregate any property of intervals and return any user-defined aggregated value type.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

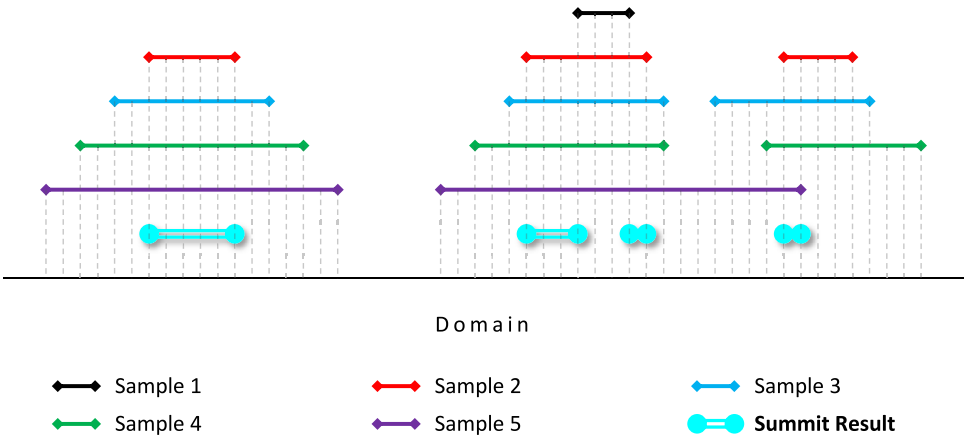

Fig. 6. An example of `SUMMIT` function with `minAcc`=2 and `maxAcc`=4 on 5 samples.

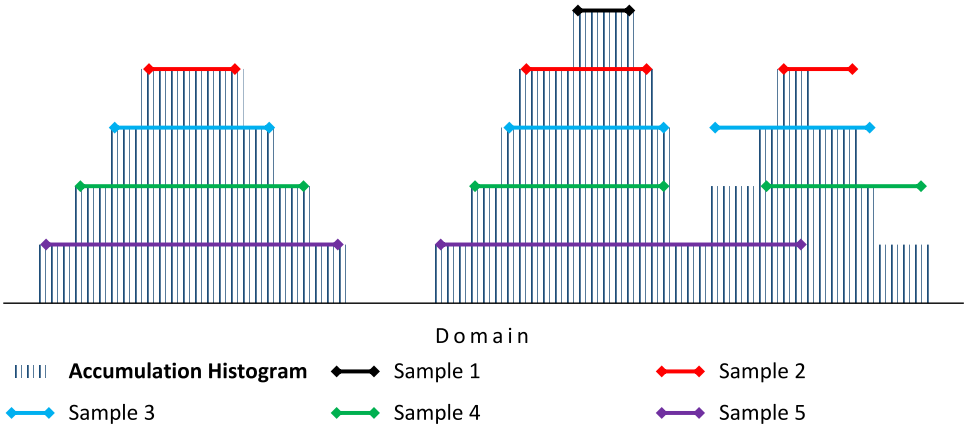

Fig. 7. An example of *Accumulation Histogram* function on 5 samples.

Algorithm 2 Map algorithm.

```
1: procedure MAP(I_reference, UDF)
2:     determined_intervals ← ∅
3:     for each snapshot B in first resolution of Di3 between I_reference and Ī_reference do
4:         determined_intervals ← determined_intervals ∪ intervals of B
5:         UDF(determined_intervals)
6:     determined_intervals ← ∅
```

Accumulation histogram/distribution. The functions `Accumulation histogram` and `Accumulation distribution` compute the genome-wide accumulation of intervals, and respectively report a histogram or distribution of the calculated information (see Fig. 7).

Nearest neighbor. Given a reference interval I, the `Nearest neighbor` function determines the nearest neighbor indexed interval, that is either an overlapping interval, or the closest up-stream or down-stream non-overlapping interval.

Map. Given a reference interval I, the `MAP` function determines all the intervals indexed in Di3 that overlap with I (similar to classical interval-tree operation). In addition to a reference, the function takes a UDF and passes the determined intervals to the UDF (see Fig. 8 as an example of the `MAP` function, and Algorithm 2 for `MAP` pseudocode). The output of the UDF is then reported back as an attribute of I. For instance, a UDF may take all the intervals overlapping I and return their cardinality.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

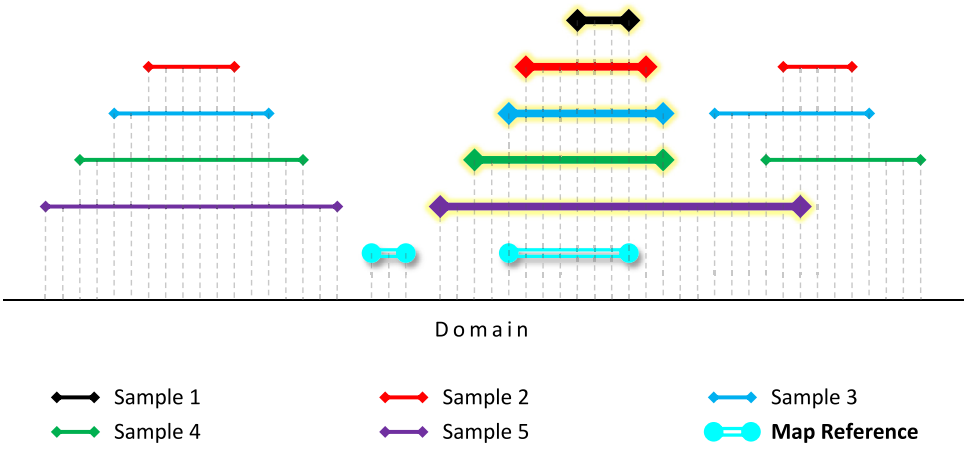

Fig. 8. An example of MAP function on 5 samples.

3.2. Di3 internals

The design of Di3 is motivated by giving the best possible support to logic operations; in particular, Di3 supports `MAP` and `Nearest neighbor` operations in *logarithmic time* in the number of regions, while it supports `COVER`, `SUMMIT` and `Accumulation histogram` operations in `linear time` (indeed, second resolution indexing supports sub-linear time search, as it is later explained). These aspects motivate the superior performance of Di3 over the state-of-the-art, as discussed in Section 4.2.

Di3 adopts a *single-dimension* paradigm, and targets the interval's coordinates; however, it also adopts a *double-resolution* paradigm. The *first resolution* organizes intervals based on their coordinates through snapshots and provides logarithmic access on the coordinates; the *second resolution* builds groups of snapshots based on their coordinates and computes suitable aggregation functions for each group, using all the attributes (and not just the coordinates). These groups act as secondary key for improved access based on specific attributes. In the following sections we describe each resolution in details.

3.2.1. First resolution

Snapshots summarize information about the coordinates where a *variation* occurs in intervals, i.e., when an interval starts or ends; a snapshot at e_b exists *iff* at least one interval introduced such variation. Hence, a finite set of snapshots can be used to index a finite set of intervals on both discrete and contiguous domains. Note that multiple intervals with the same starts or ends are possible.

Each snapshot holds a list of the IDs of all the intervals overlapping with its position; having a pointer to each interval overlapping a point on the domain is advantageous mainly for queries that target specific or relative positions. For instance, 'given an interval, find all intervals overlapping with it' requires $O(\log_2 n)$ to find the snapshot which has a pointer to all intersecting intervals; or 'given an interval, find all its neighbors at 200 base-pair (unit of the genomic domain) distance' requires $O(\log_2 n)$ to find the snapshot at overlapping position, plus few additional siblings of the determined snapshot, and then to union the intervals represented by the snapshots (which asymptotically is still $O(\log_2 n)$).[10]

This organization has been shown already in Fig. 3; the upper part of the figure describes 4 input intervals; the lower part of the figure describes the Di3 index, where each snapshot B_b includes a key-value pair, the key is e_b, and the value is the list of intervals which have an intersection with B_b. Precisely, B_b can be at the left (L), at the right (R), or in the middle (M) of an interval; the type of positioning (i.e., L, R or M) is included in each entry of the interval list, which also contains a pointer to descriptive metadata for the interval. For instance, the third snapshot, B_3, intersects with three intervals, being in the middle of the S_1 and S_2 intervals and at the left of the S_3 interval.

The first resolution index is created using *batch indexing*, a method which includes input intervals one after the other; such method is preferred over *bulk indexing* or *range indexing*, which require the sorting of intervals prior to index creation. These methods are related to the physical layer being implemented through a B+ tree and they are related to classical methods for B+ tree data insertion [24], [21]. To perform batch indexing, we contrasted two methods, respectively called *single-pass indexing* and *double-pass indexing*. The former one considers each interval in input and precisely updates the data structure in a single-pass; the latter one at the first pass orders the snapshots of new intervals with respect to all existing intervals, and at the second-pass updates all the snapshots with the information about the list of the intersecting intervals. Fig. 9 explains step-by-step the single-pass indexing of the four intervals of Fig. 3.

[10] Computational complexities are based on a B+ tree data structure for the implementation of the physical layer operations.

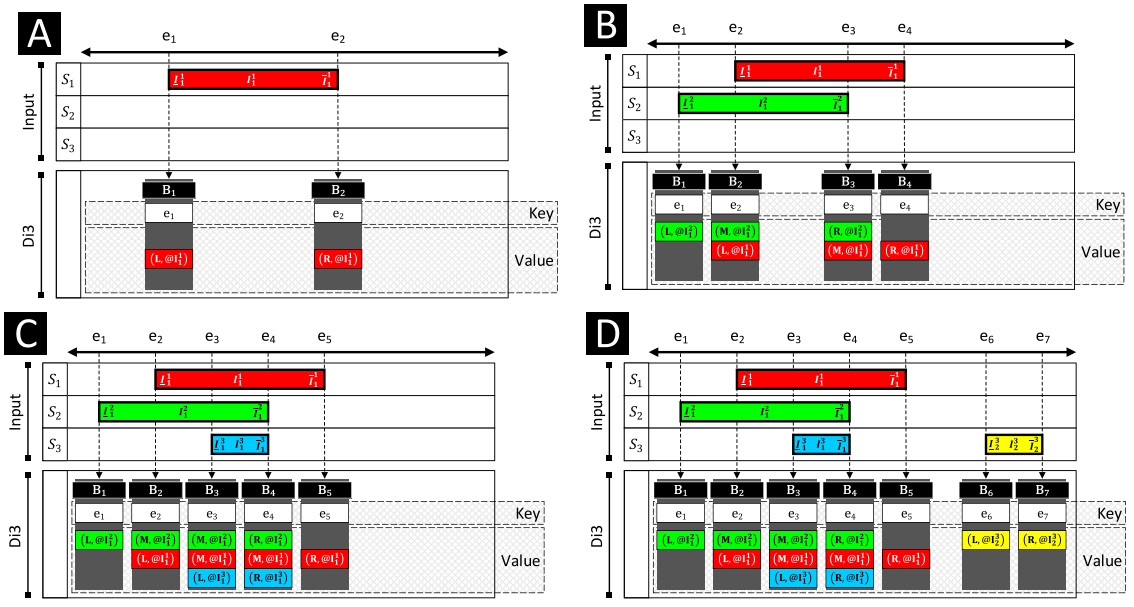

Fig. 9. Step-by step construction of the Di3 index status represented in Fig. 3, as result of inserting four regions.

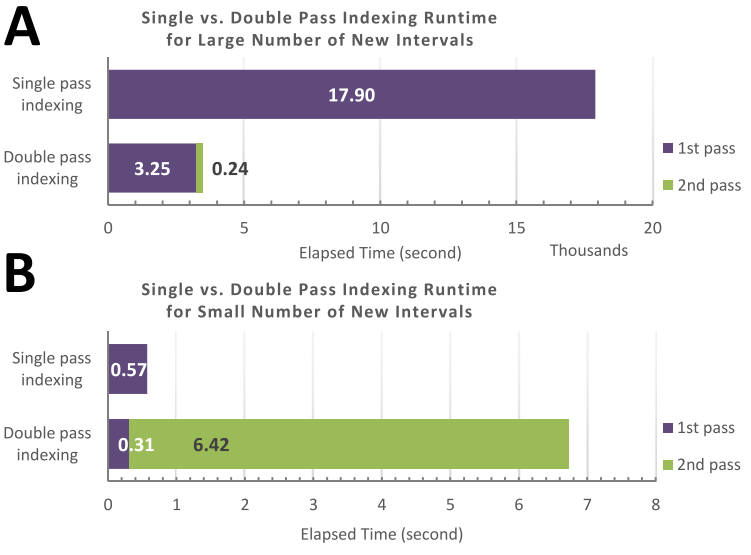

Fig. 10. A comparison between single-pass and double-pass indexing in two scenarios: insertion of (A) large number vs. (B) small number of new intervals.

The two methods are compared in Fig. 10, which clearly shows that no method dominates over the other one. In general, single-pass indexing is superior with a small number of new intervals, while double-pass indexing is superior for a large number of new intervals. Based on such analysis, the initial indexing of intervals in Di3 is performed by using the double-pass method, while updating the index is performed by single-pass procedure.

3.2.2. Second resolution

The second resolution is built from the first resolution by grouping a set of snapshots and aggregating the relative information. The elements of the second resolution of Di3 are collections of consecutive snapshots grouped together, called *blocks*. Each block is defined as a key-value pair; the *key* is defined by a *grouping* function, while the *value* is defined by an *aggregation* function. More precisely:

In *Grouping*, a user-defined function groups a set of adjacent snapshots; depending on the semantic of the UDF, such groups reveal various aspects. For instance, in the genomic domain with intervals representing a biological activity, a group of snapshots is a region on a genome where at least one biological activity is observed. The grouping function defines the

Table 1
Di3BCLI commands.

Command	Description
Index	Takes the name of a file and indexes all its regions.
BatchIndex	Takes the names of a set of files specified using wildcard characters.
2Pass	Runs second-pass of indexing in double-pass indexing mode.
2RI	Indexes second resolution.
Cover	Executes `COVER` function and exports results.
Summit	Executes `SUMMIT` function and exports results.
Map	Executes `MAP` function and exports results.
Merge	Executes `MERGE` function and exports results.
Complement	Executes `COMPLEMENT` function and exports results.
AccHis	Determines *Accumulation histogram* and exports results.
AccDis	Determines *Accumulation distribution* and exports results.
GetIM	Reports current setting for indexing mode.
SetIM	Sets indexing mode to the specified one.
GetDP	Reports current setting for degree of parallelization.
SetDP	Sets degree of parallelization to the specified one.

block *key*, as an interval on first resolution with start and stop being respectively the minimum and maximum coordinates of the snapshots in the block.

With *Aggregation*, the information of the snapshots within each block are then aggregated, thereby providing summary statistics on the specific attribute(s). Storing custom aggregations for each block reduces snapshot access demands when a particular aggregation is commonly used. The aggregation function defines the block *value*, which is generic in type and size, for instance, 'count' of intervals represented by grouped snapshots, or standard deviation and median of interval lengths.

Di3 already provides built-in functions for default grouping and aggregation; the default grouping function groups consecutive snapshots that point to consecutive overlapping intervals, and the built-in aggregation function stores the maximum accumulation of each group. These functions improve operations such as `COVER`, `SUMMIT` and `Accumulation histogram`, discussed in Section 3.1, which depend on accumulation of intervals. Indeed in a `COVER` query, when the max accumulation of a block is lower than the min accumulation `accMin` of the query, or the min accumulation of a block is higher than the max accumulation `accMax` of the query, the entire group of intervals does not have an interval that satisfies the query and can be fully skipped. Moreover, disjoint groups can be processed in parallel with no need for synchronization mechanisms.

4. Experiments

We start with an evaluation of Di3 enacted from a user interface, and then we present a comparison with BEDTools and BEDOPS, the most popular tools for genomic region calculus.

4.1. Di3 evaluation

We customized Di3 to the genomic domain by building the Di3B component at the business logic layer that initializes several independent Di3 instances, one for each DNA chromosome and strand; the Di3BCLI client at presentation layer (see Fig. 2) provides user interaction through a set of commands. These include: primitives for initializing the indexes, primitives for the operations of Section 3.1, and primitives for setting indexing modes and parallelism.

In general, the Di3BCLI commands (listed in Table 1) have standard command argument structure, where the number of arguments varies between different commands. Having executed a command, its runtime is reported on console, and also saved in a user-defined log file. The *Index* and *BatchIndex* primitives take respectively single sample or a collection of samples (specified using wildcard characters) as argument and, based on the indexing mode, index the intervals in single-pass or double-pass mode. Under double-pass indexing mode, the command *2Pass* (which takes no arguments) executes the second-pass of the indexing. The second resolution of Di3 is created/updated by the *2RI* command (that takes no arguments). The *Cover* and *Summit* commands take *minAcc*, *maxAcc*, *aggregate*, and *output* arguments, execute the functions with the parameters and export results to the output file. The *Map* command takes *reference*, *aggregate*, and *output* arguments, executes the function and exports results in the output file. The *Merge*, *Complement*, *AccHis*, and *AccDis* commands take *output* argument, execute the function and report results to the output file. The *GetIM* reports current setting for indexing mode, and *SetIM* takes a *mode* argument which is either *single* or *multi*, and sets indexing mode accordingly. Finally, *GetDP* reports current setting for degree of parallelization, and *SetDP* takes two numbers as *chr-level* and *Di3-level* degree-of-parallelization and updates the execution environment accordingly.

We benchmarked Di3 by exploiting two levels of parallelism: chromosome-level parallelism (i.e., executing operations on multiple chromosomes concurrently) and Di3-level (i.e., each chromosome is further divided into multiple sections or *bins*, and multiple threads process the resulting sections). In all the experiments of this section, we used an Amazon EC2 machine running Microsoft Windows Server 2012 with an Intel® Xeon® E5-2670 v2 CPU, 320 GFLOPS, and 122 GB RAM. We used

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

Table 2
Datasets used for Di3 benchmarking.

Label	Sample count	Region count	Dataset size (GB)	Type
A1	500	28,392,674	1.35	Narrow peaks
A2	1000	59,980,303	3.17	Narrow peaks
A3	1500	94,997,460	4.87	Narrow peaks
A4	2000	143,563,549	6.98	Narrow peaks
A5	2970	177,903,976	10.77	Narrow and broad peaks

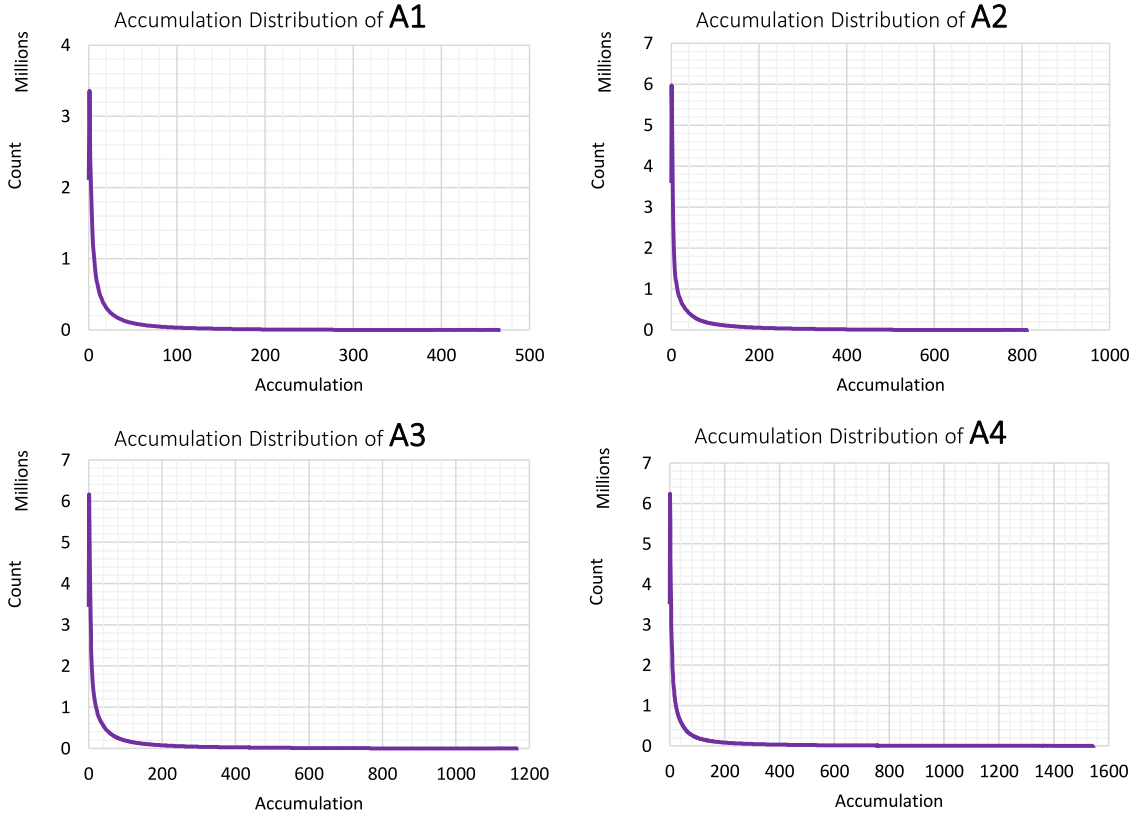

Fig. 11. Interval/region accumulation distribution in datasets A1, A2, A3, and A4.

five datasets of ENCODE narrow and broad peak samples, as described in Table 2; the datasets vary in size, but are similar in interval accumulation distribution. Fig. 11 shows the accumulation distribution of datasets A1, A2, A3, and A4; statistics for the A5 dataset follows similar distribution.

4.1.1. Benchmark of MAP

First, we assessed the performance of `MAP`, an operation which directly operates upon coordinates. We considered as reference a genomic region sample from the ENCODE repository, which includes 196,180 regions (8.9 MB in size); the `MAP` was performed over the five datasets in Table 2. Fig. 12 shows excellent scalability with respect to growth in data size. Note that the reference sample is also an ENCODE narrow peak sample; hence, its intervals are mostly co-localized with the indexed intervals. Therefore, a very big percentage of indexed data overlaps with the reference intervals.

4.1.2. Benchmark of accumulation operations

Second, we assessed Di3 performance for `COVER`, `SUMMIT`, `Accumulation histogram` and `Accumulation distribution` operations. The function `SUMMIT` is a variation of `COVER` and, similarly, the function `Accumulation distribution` is a variation of `Accumulation histogram`; therefore, their performance is at the same scale (see Fig. 13). Both the Accumulation histogram and distribution functions scan all snapshots; hence, their performance can be evaluated as the maximum time required for a full scan of Di3, which is linear to dataset size. Likewise, the functions `COVER` and `SUMMIT` require linear scan of snapshots for regions of specific accumulation of intervals. However, the second resolution index prunes a percentage of linear scan based on `minAcc` and `maxAcc` parameters, and on their overlap with the accumulation distribution of the data.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

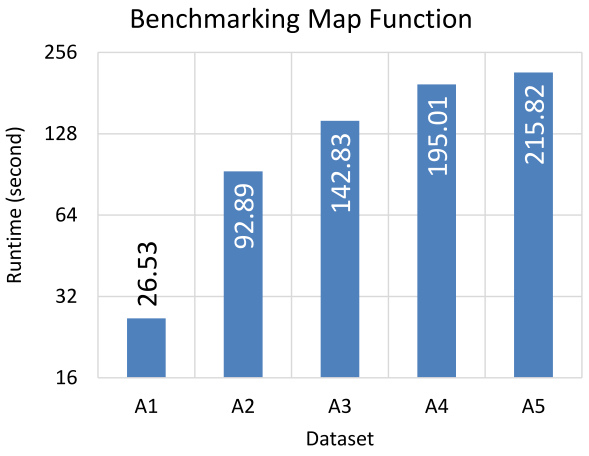

Fig. 12. Benchmarking MAP function of Di3.

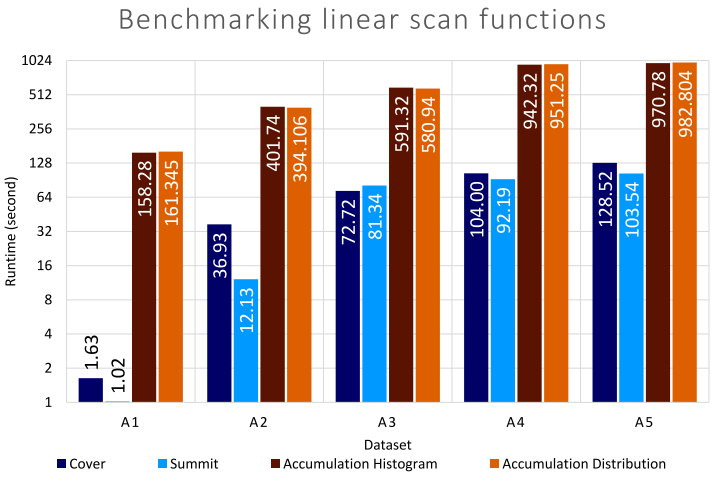

Fig. 13. Benchmarking linear scan functions of Di3.

The less effective pruning is expected with parameters set around the peak of accumulation distribution; the COVER and SUMMIT functions are expected to be faster than the Accumulation histogram and Accumulation distribution with such choice of parameters. We executed COVER and SUMMIT functions with minAcc and maxAcc at the peak of accumulation distribution (e.g., minAcc=80 and maxAcc=100 for A2); Fig. 13 confirms that even with the peak of the parameter values the functions perform faster than full scan.

4.1.3. Effect of accumulation distribution

Data distribution may have a strong effect on the performance of an index. For instance, non-uniformly distributed data may accumulate a big load of information on some keys, while other keys may have lighter load; this is suboptimal because some keys are very expensive to process, while others are cheap. This affects also parallel execution, as some threads are busy for a very long time, while others are set free very early. The first step to avoid such draw backs is at design level, by making correct decisions, based on the nature of the data, for the key and value of the index.

In this section, we evaluate Di3 design and performance for data with different accumulation distribution. For this we simulated 10 datasets, each containing 500 samples and 200,000 regions in each sample. Datasets differ in the percentage of intersecting intervals (in the extreme case, all samples contribute to intersection; therefore, the accumulation of the intersection region equals to the number of samples). A dataset with no intersection is labeled '0%', while a dataset with all intervals intersecting is labeled '100%'. We executed Di3 functions on all simulated datasets. Fig. 14 shows the results. Minor variations in function runtime throughout the datasets are due to the randomly generated positions and overlapping on intervals in the datasets. The figure highlights that Di3's performance is independent from the accumulation distribution of the input datasets.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

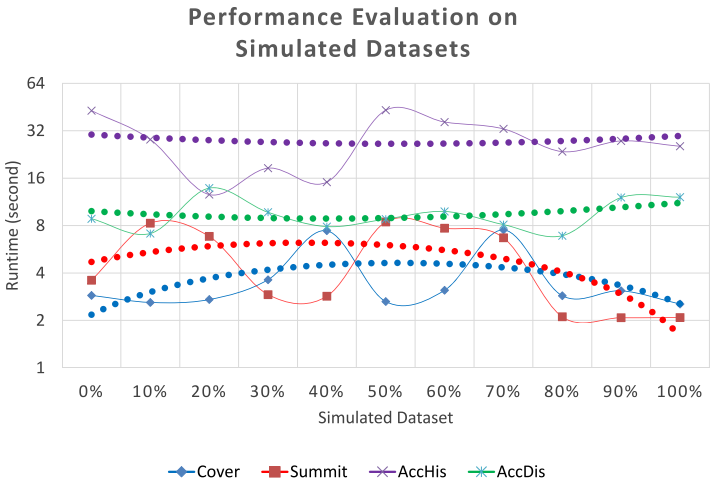

Fig. 14. Function performance on simulated data. The dotted lines are the polynomial trend line of order 2 of each function.

Table 3
Datasets used for Di3 benchmarking versus BEDTools and BEDOPS.

Label	Sample count	Region count	Dataset size (MB)
B1	90	1,407,493	97.4
B2	180	4,649,767	322.0

4.2. Comparison with state of the art tools

In this section we compare the performance of Di3 with the one of two tools commonly used in genomic region processing, namely, BEDTools [43] and BEDOPS [40]. We ran the tools and Di3 on a standard laptop (Intel® Core™ i3 2.10 GHz and 8 GB RAM). We ran BEDTools and BEDOPS under Linux, and Di3 under Microsoft Windows® 10 operating system. We prepared Python and shell scripts for the batch execution of BEDTools and BEDOPS, and executed Di3 in-memory.

Among the possible operations that are available from the Di3BCLI, BEDTools and BEDOPS implement the `MAP` operator, i.e., given a reference sample, find input intervals overlapping with the reference regions. Therefore, we compared the `MAP` operator of Di3 against the '*bedtools map*' from BEDTools and the '*bedmap*' from BEDOPS. We considered two typical *usage scenarios* in genomic region processing: (*i*) personal repository, and (*ii*) on-the-fly processing.

4.2.1. Personal repository

This is a common scenario for bioinformaticians, where a personal repository of in-house data, as output of the execution of a NGS data processing pipeline, and/or data obtained from publicly available repositories is persistent on their machine. Such data are stored for further processing or to be used for comparative evaluation and cross-referencing. The repository is a collection of properly organized files, indexed and persistent in Di3. We used two datasets of ENCODE narrow peak samples as in Table 3, and a reference sample including 196,180 regions (8.9 MB in size).

The data were pre-processed, i.e., filtered, and regions in samples were sorted for BEDTools and BEDOPS, and indexed for Di3. Hence, benchmark started from pre-processed data, and for comparison considered only execution time. As Fig. 15 shows, Di3 performs significantly faster than BEDTools and BEDOPS.

4.2.2. On-the-fly processing

Processing data on-the-fly is a bioinformaticians' daily-based scenario, where a relatively small dataset is obtained from the execution of a NGS data processing pipeline and may not be archived for further evaluation. We benchmarked the in-memory version of Di3 versus BEDTools and BEDOPS on `MAP` operation using one reference, including 196,180 regions (8.9 MB in size), and three datasets of ENCODE narrow peak samples, described as in Table 4, as target.

The data were not pre-processed: data were not sorted for BEDTools and BEDOPS, and not indexed for Di3. Hence, the execution time incorporated pre-processing time in all cases. Fig. 16 shows that Di3 performs faster than BEDTools and BEDOPS on the three datasets. This highlights that Di3 is also an agile back-end data structure for on-the-fly processing, even by incorporating the indexing time within the processing time.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

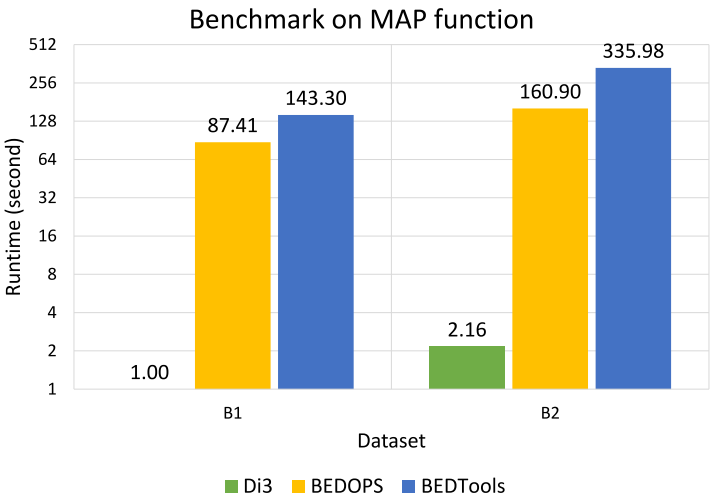

Fig. 15. Benchmark: Personal repository scenario.

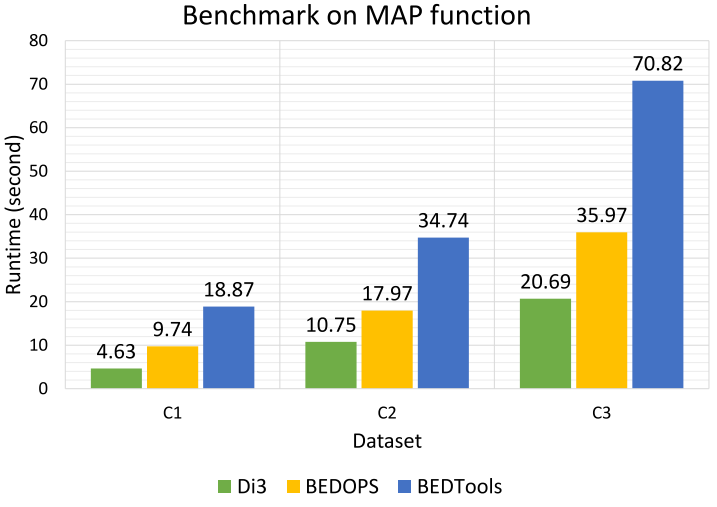

Fig. 16. Benchmark: On-the-fly processing scenario.

Table 4
Datasets used for Di3 benchmarking versus BEDTools and BEDOPS for on-the-fly processing.

Label	Sample count	Region count	Dataset size (MB)
C1	12	89,623	6.17
C2	22	258,406	17.80
C3	45	456,385	31.50

5. Use of Di3 as a component within MuSERA

The analysis of NGS ChIP-seq samples outputs a number of enriched regions (ERs) for NGS reads, each indicating a protein-DNA interaction or a specific chromatin modification. ERs (or 'peaks') are called when the enrichment p-value is below a user-defined threshold. ERs with a p-value close to that of the background signal are either a background signal which is slightly enriched due to some biological or technical bias in the experiment, or indeed are biologically important regions with an enrichment less significant than expected. The NGS protocol is subject to noise; to avoid a large number of false-positive ERs, commonly used thresholds are often very stringent, yielding many false-negatives. However, the guidelines of the ENCODE project recommend repeating a NGS experiment at least twice [32] on replicate samples, where the information contained in replicates is expected to be largely overlapping. We recently proposed a novel method [27] that, by leveraging

on available replicates, differentiates between sub-thresholded ERs and truly non-ERs, using a rigorous combination of the significance of individual ERs that overlap in replicate samples.

The results of the method should be further assessed using common procedures, such as visualization on a genome browser, functional analysis, or nearest-neighbor search. To this end, we recently proposed Multiple Sample Enriched Region Assessment (MuSERA) [28], a novel graphical tool that leverages on Di3 (On-the-Fly scenario) to implement (*i*) comparative analysis of replicate samples using the method originally proposed in [27], (*ii*) ER functional analysis, (*iii*) region nearest neighbor search, (*iv*) correlation assessment, and (*v*) an integrated user-friendly genome browser.

MuSERA builds the business logic layer and presentation layer above Di3 (recall Figs. 2 and 4). It encapsulates peak regions as intervals for Di3, and provides particular comparative analysis driven operations (e.g., methods for combining p-values, and classification of peaks as true sub-thresholded ERs or artifacts) via UDF. Additionally, it uses an in-memory implementation of B+ tree to implement the physical layer (Fig. 4).

5.1. Comparative analysis

MuSERA combines statistical evidence of an ER (i.e., its p-value) with the co-localized evidence from replicates (it uses only the strongest evidence from each replicate if more than one evidence - i.e., ER - from a replicate is co-localized), and confirms/discards ERs based on the comparison between their combined evidence and a user-defined threshold. For this functionality, MuSERA benefits from Di3 `MAP` function with a UDF for comparative analysis, explained in [27]; the procedure is as follows:

i. MuSERA provides graphical means to select and load replicates.
ii. MuSERA indexes replicates in Di3 on-the-fly.
iii. Through a graphical user interface (GUI), MuSERA gets user-defined parameters for a UDF for the comparative analysis.
iv. MuSERA executes Di3 `MAP` with reference sample (the replicate against which ERs are comparatively evaluated) and a UDF that:
   (a) In each replicate sample, choses the interval with the lowest p-value from the set of intervals overlapping with a reference interval.
   (b) If the number of overlapping intervals satisfies a user-defined minimum threshold, then combines the p-values of the overlapping intervals and the reference interval using the Fisher's method [19].
   (c) If the combined stringency satisfies a user-defined threshold, then stores the intervals as confirmed, or as discarded otherwise.

5.2. Genome browser

MuSERA plots a given ER and all the neighbor ERs from replicates, as well as user-uploaded known genomic features (e.g., genes or promoter regions) at a given distance (e.g., within 100k base-pair). This function leverages again on the `MAP` (i.e., 'given an interval, find intervals overlapping with it') and the `Nearest neighbor` search functions of Di3. The determined intervals are then plotted using a dynamic plot that allows zoom in and out, and pan, which together with optimal retrieval from Di3 and plot features provides a high-end genome browser (see Fig. 17 Panel A).

5.3. Functional analysis

Assigning an ER to the closest up-/down-stream genomic feature (e.g., gene Transcription Start Site (TSS), or Coding Sequence (CDS)) is a very useful and common task to help the biological interpretation of NGS results. MuSERA implements it optimally using Di3; it leverages on relative ordering of intervals accessible via a consecutive set of snapshots, and on the `Nearest neighbor` search, to calculate the *ER-to-feature conservation rate* (which is determined as the number of ERs intersecting with genomic annotations) and the *ER-to-feature distance distribution* between ERs and the closest up-/down-stream features (see Fig. 17 Panel B).

5.4. Nearest neighbor

MuSERA computes the ER nearest neighbor distance distribution with functional annotations, leveraging on `Nearest neighbor` search function of Di3. The distribution facilitates biological assessment of ERs; for instance, it helps evaluating that comparatively confirmed weak ERs are relatively closer to functional annotations than comparatively evaluated true non-ERs (see Fig. 17 Panel C).

5.5. Correlation assessment

MuSERA determines correlations, both at region-level and base-pair-level, between replicates. They are respectively computed as the ratio between the number of overlapping regions (region-level correlation), or genomic bases (base-pair-level

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

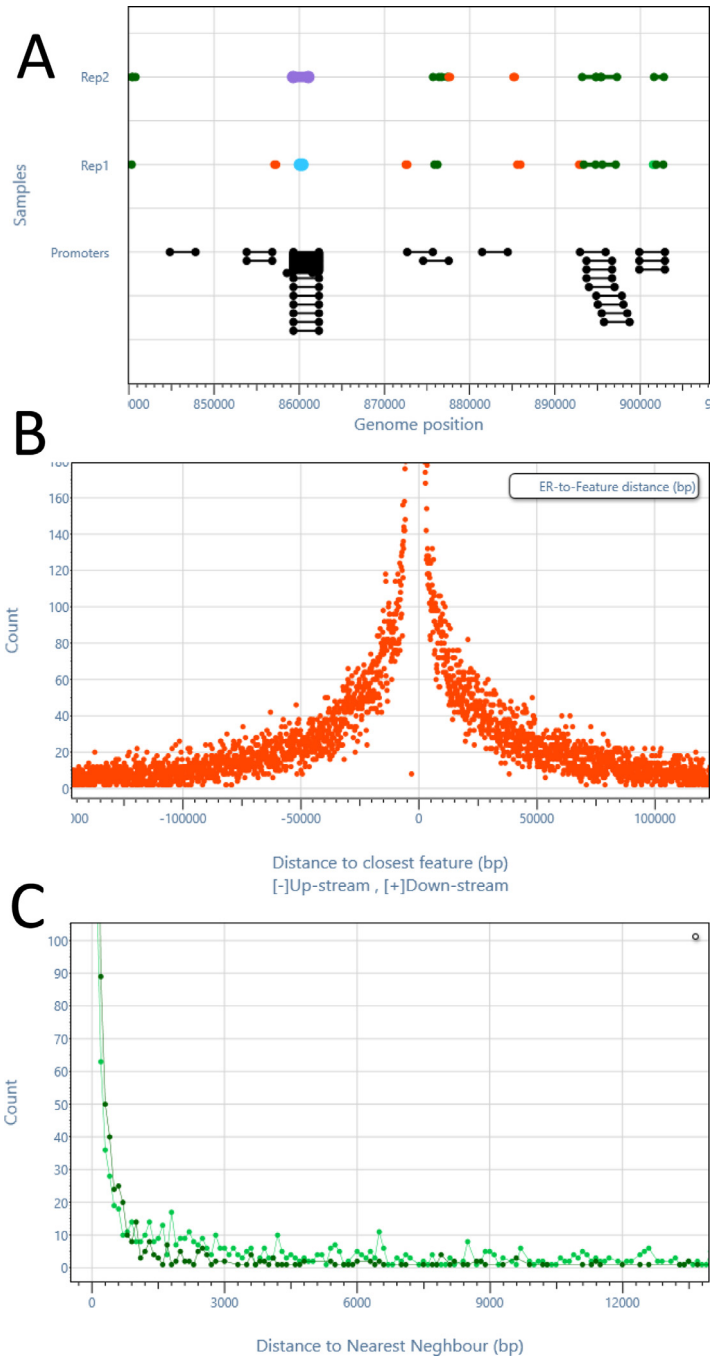

Fig. 17. MuSERA at work. (A) ERs on the genome browser together with known genomic feature annotations (i.e., promoter regions); (B) Up-stream and down-stream distances of a given ER to the closest genomic features; (C) Distances of a given ER to the nearest neighbors. Distances are measured in base pairs (bp).

correlation), and the total number of regions, or genomic bases considered, producing the respective Jaccard similarity coefficients. Base-pair-level correlation is more stringent and is to be preferred when the position of the ERs is known with certainty, while region-level correlation is instead more permissive, as it scores the overlap of entire regions rather than quantifying the magnitude of this overlap; this correlation measure is then to be preferred in presence of heterogeneous or noisy data sets. MuSERA estimates both correlations leveraging on the `SUMMIT` function of Di3, that calculates density-based co-occurrences.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

6. Conclusion

We proposed Di3 (1D Interval Inverted Index), a multi-resolution single-dimension data structure for interval-based data queries. We presented Di3's method by illustrating its model, data structure, and supported operations, and then we demonstrated its effectiveness, both as a stand-alone tool and in comparison with the state-of-the-art systems BEDOPS and BEDTools. Finally, we showed how Di3 was used as a software component within MuSERA, a tool for multiple sample peak calling. We proved that Di3 is a flexible, high-performance component, whose use as a self-standing system is much superior to current state-of-the-art.

Previously, we implemented GMQL, the innovative region-based genomic data querying system, by using cloud computing and specifically the Apache Spark and Flink engines within the Apache Hadoop framework; as future work, we plan to support an alternative implementation of the domain-specific operations of GMQL by using Di3, so as to compare cloud-based computing with specialized indexing on very large datasets.

Acknowledgments

We acknowledge the essential contributions of Fernando Palluzzi, who has adviced us about the biological use of Di3, and of Roger Knapp, who provided a B+ tree implementation. Research was supported by the Data-Driven Genomic Computing (GenData 2020) PRIN project (2013-2016), funded by the Italian Ministry of the University and Research, and by a grant from Amazon Web Services.

Author contributions

Concept: Jalili, Matteucci, Ceri. Software Design: Jalili, Matteucci. Software Implementation: Jalili. Data acquisition: Jalili. Data analysis and interpretation: Jalili, Matteucci. Manuscript drafting: Jalili, Matteucci, Masseroli, Ceri. Critical revision: Ceri.

References

[1] 1000_Genomes_Project_Consortium, et al., A map of human genome variation from population-scale sequencing, Nature 467 (7319) (2010) 1061–1073.
[2] F.N. Afrati, S. Dolev, S. Sharma, J.D. Ullman, Bounds for overlapping interval join on MapReduce, in: EDBT/ICDT workshops, 2015, pp. 3–6.
[3] A.V. Alekseyenko, C.J. Lee, Nested Containment List (NCList): a new algorithm for accelerating interval query of genome alignment and interval databases, Bioinformatics 23 (11) (2007) 1386–1393.
[4] S.F. Altschul, T.L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucleic Acids Res. 25 (17) (1997) 3389–3402.
[5] J.L. Bentley, Solutions to Klee's rectangle problems, Technical Report, Carnegie-Mellon Univ., Pittsburgh, PA, 1977.
[6] J.L. Bentley, Decomposable searching problems, Inf. Process. Lett. 8 (5) (1979) 244–251.
[7] E. Birney, D. Andrews, M. Cáccamo, Y. Chen, L. Clarke, G. Coates, T. Cox, F. Cunningham, V. Curwen, T. Cutts, et al., Ensembl 2006, Nucleic Acids Res. 34 (Suppl. 1) (2006) D556–D561.
[8] J.B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, S. Brandt, SciHadoop: array-based query processing in Hadoop, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, 66, ACM, 2011, pp. 1–11.
[9] J.G. Caporaso, C.L. Lauber, W.A. Walters, D. Berg-Lyons, J. Huntley, N. Fierer, S.M. Owens, J. Betley, L. Fraser, M. Bauer, et al., Ultra-high-throughput microbial community analysis on the illumina HiSeq and MiSeq platforms, ISME J. 6 (8) (2012) 1621–1624.
[10] M. Cereda, M. Sironi, M. Cavalleri, U. Pozzoli, GeCo++: a C++ library for genomic features computation and annotation in the presence of variants, Bioinformatics 27 (9) (2011) 1313–1315.
[11] B. Chawda, H. Gupta, S. Negi, T.A. Faruquie, L.V. Subramaniam, M.K. Mohania, Processing interval joins on Map-Reduce, in: EDBT, 2014, pp. 463–474.
[12] P.N. Cockerill, Structure and function of active chromatin and DNase I hypersensitive sites, FEBS J. 278 (13) (2011) 2182–2210.
[13] G. Cooper, M. Raymer, T. Doom, D. Krane, N. Futamura, Indexing genomic databases, in: Proceedings of Fourth IEEE Symposium on Bioinformatics and Bioengineering, BIBE 2004., IEEE, 2004, pp. 587–591.
[14] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Section 14.3: Interval Trees, third ed., MIT press Cambridge, 2009, pp. 348–354.
[15] A. Eldawy, M.F. Mokbel, A demonstration of spatialhadoop: an efficient MapReduce framework for spatial data, Proc. VLDB Endowment 6 (12) (2013) 1230–1233.
[16] R. Elmasri, G.T. Wuu, Y.-J. Kim, The time index: an access structure for temporal data, in: Proceedings of the 16th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1990, pp. 1–12.
[17] ENCODE_Project_Consortium, et al., An integrated encyclopedia of DNA elements in the human genome, Nature 489 (7414) (2012) 57–74.
[18] P.M. Fenwick, A new data structure for cumulative frequency tables, Software 24 (3) (1994) 327–336.
[19] R.A. Fisher, Statistical Methods for Research Workers, Genesis Publishing Pvt Ltd, 1925.
[20] GA4GH_Data_Working_Group, GA4GH API, 2015. URL http://ga4gh.org/#/documentation
[21] T.M. Ghanem, R. Shah, M.F. Mokbel, W.G. Aref, J.S. Vitter, Bulk operations for space-partitioning trees, in: Proceedings of 20th International Conference on Data Engineering, 2004., IEEE, 2004, pp. 29–40.
[22] B. Giardine, L. Elnitski, C. Riemer, I. Makalowska, S. Schwartz, W. Miller, R.C. Hardison, GALA, a database for genomic sequence alignments and annotations, Genome Res. 13 (4) (2003) 732–741.
[23] C.H. Goh, H. Lu, B.-C. Ooi, K.-L. Tan, Indexing temporal data using existing B+-trees, Data Knowl. Eng. 18 (2) (1996) 147–165.
[24] G. Graefe, Sorting and indexing with partitioned B-trees., in: CIDR, 3, 2003, pp. 5–8.
[25] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: ACM SIGMOD International Conference on Management of Data, vol. 14, ACM, 1984, pp. 47–57.
[26] J. Han, E. Haihong, G. Le, J. Du, Survey on NoSQL database, in: 6th International Conference on Pervasive Computing and Applications (ICPCA), 2011, IEEE, 2011, pp. 363–366.
[27] V. Jalili, M. Matteucci, M. Masseroli, M.J. Morelli, Using combined evidence from replicates to evaluate ChIP-seq peaks, Bioinformatics 31 (17) (2015) 2761–2769.
[28] V. Jalili, M. Matteucci, M.J. Morelli, M. Masseroli, MuSERA: multiple sample enriched region assessment, Briefings Bioinf. (2016) 1–15.
[29] M. Kaufmann, P.M. Fischer, N. May, C. Ge, A.K. Goel, D. Kossmann, Bi-temporal timeline index: a data structure for processing queries on bi-temporal data, in: 31st IEEE International Conference on Data Engineering (ICDE), 2015., IEEE, 2015, pp. 471–482.
[30] W.J. Kent, C.W. Sugnet, T.S. Furey, K.M. Roskin, T.H. Pringle, A.M. Zahler, D. Haussler, The human genome browser at UCSC, Genome Res. 12 (6) (2002) 996–1006.
[31] T. Kowalski, S. Grabowski, S. Deorowicz, Indexing arbitrary-length k-mers in sequencing reads, PloS One 10 (7) (2015) e0133198.

[32] S.G. Landt, G.K. Marinov, A. Kundaje, P. Kheradpour, F. Pauli, S. Batzoglou, B.E. Bernstein, P. Bickel, J.B. Brown, P. Cayting, et al., ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia, Genome Res. 22 (9) (2012) 1813–1831.
[33] R.M. Layer, A.R. Quinlan, A parallel algorithm for N-way interval set intersection, in: Proceedings of the IEEE, IEEE, 2015, pp. 1–10.
[34] R.M. Layer, K. Skadron, G. Robins, I.M. Hall, A.R. Quinlan, Binary interval search: a scalable algorithm for counting interval intersections, Bioinformatics 29 (1) (2013) 1–7.
[35] H. Li, Tabix: fast retrieval of sequence features from generic TAB-delimited files, Bioinformatics 27 (5) (2011) 718–719.
[36] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, et al., The sequence alignment/map format and SAMtools, Bioinformatics 25 (16) (2009) 2078–2079.
[37] M. Masseroli, P. Pinoli, F. Venco, A. Kaitoua, V. Jalili, F. Palluzzi, H. Muller, S. Ceri, GenoMetric Query Language: a novel approach to large-scale genomic data management, Bioinformatics 31 (12) (2015) 1881–1888, doi:10.1093/bioinformatics/btv048.
[38] M. McKenney, T. McGuire, A parallel plane sweep algorithm for multi-core systems, in: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2009, pp. 392–395.
[39] D. Mrozek, B. Socha, S. Kozielski, B. Maysiak-Mrozek, An efficient and flexible scanning of databases of protein secondary structures with the segment index and multithreaded alignment, J. Intell. Inf. Syst. 46 (1) (2016) 213–233.
[40] S. Neph, M.S. Kuehn, A.P. Reynolds, E. Haugen, R.E. Thurman, A.K. Johnson, E. Rynes, M.T. Maurano, J. Vierstra, S. Thomas, et al., BEDOPS: high-performance genomic feature operations, Bioinformatics 28 (14) (2012) 1919–1920.
[41] K. Ovaska, L. Lyly, B. Sahu, O.A. Janne, S. Hautaniemi, Genomic region operation kit for flexible processing of deep sequencing data, IEEE/ACM Trans. Comput. Biol. Bioinf. 10 (1) (2013) 200–206.
[42] P.J. Park, ChIP–seq: advantages and challenges of a maturing technology, Nat. Rev. Genet. 10 (10) (2009) 669–680.
[43] A.R. Quinlan, I.M. Hall, BEDTools: a flexible suite of utilities for comparing genomic features, Bioinformatics 26 (6) (2010) 841–842.
[44] N.U. Rashid, P.G. Giresi, J.G. Ibrahim, W. Sun, J.D. Lieb, ZINBA integrates local covariates with DNA-seq data to identify broad and narrow regions of enrichment, even within amplified genomic regions, Genome Biol. 12 (7) (2011) R67.
[45] G. Renaud, P. Neves, E.L. Folador, C.G. Ferreira, F. Passetti, Segtor: rapid annotation of genomic coordinates and single nucleotide variations using segment trees, PloS One 6 (11) (2011) e26715.
[46] J.E. Richardson, Fjoin: simple and efficient computation of feature overlaps, J. Comput. Biol. 13 (8) (2006) 1457–1464.
[47] C.E. Romanoski, C.K. Glass, H.G. Stunnenberg, L. Wilson, G. Almouzni, Epigenomics: roadmap for regulation, Nature 518 (7539) (2015) 314–316.
[48] S.C. Schuster, Next-generation sequencing transforms today's biology, Nature 200 (8) (2007) 16–18.
[49] S.C. Schuster, Next-generation sequencing transforms today's biology, Nature Meth. 5 (1) (2008) 16–18.
[50] J. Shendure, H. Ji, Next-generation DNA sequencing, Nature Biotechnol. 26 (10) (2008) 1135–1145.
[51] R. Snodgrass, I. Ahn, A taxonomy of time databases, in: ACM SIGMOD Record, vol. 14, ACM, 1985, pp. 236–246.
[52] R. Sriharsha, Magellan, 2015. URL http://spark-packages.org/package/harsha2010/magellan
[53] B. Stantic, R. Topor, J. Terry, A. Sattar, Advanced indexing technique for temporal data, Comput. Sci. Inf. Syst. 7 (4) (2010) 679–703.
[54] L.D. Stein, C. Mungall, S. Shu, M. Caudy, M. Mangone, A. Day, E. Nickerson, J.E. Stajich, T.W. Harris, A. Arva, et al., The generic genome browser: a building block for a model organism system database, Genome Res. 12 (10) (2002) 1599–1610.
[55] H. Tan, W. Luo, L.M. Ni, Clost: a hadoop-based storage system for big spatio-temporal data analytics, in: Proceedings of the 21st ACM international conference on Information and Knowledge Management, ACM, 2012, pp. 2139–2143.
[56] B.G. Tudorica, C. Bucur, A comparison between several NoSQL databases with comments and notes, in: 10th RoEduNet International Conference (RoEduNet), 2011, IEEE, 2011, pp. 1–5.
[57] J. Vlissides, R. Helm, R. Johnson, E. Gamma, Design patterns: elements of reusable object-oriented software, Reading: Addison-Wesley 49 (120) (1995) 11.
[58] J.N. Weinstein, E.A. Collisson, G.B. Mills, K.R.M. Shaw, B.A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J.M. Stuart, C.G.A.R. Network, et al., The Cancer Genome Atlas pan-cancer analysis project, Nat. Genet. 45 (10) (2013) 1113–1120.
[59] T.D. Wu, Bitpacking techniques for indexing genomes: I. Hash tables, Algorithms Mol. Biol. 11 (1) (2016) 1–13, doi:10.1186/s13015-016-0069-5.
[60] J. Yu, J. Wu, M. Sarwat, Geospark: A cluster computing framework for processing large-scale spatial data, in: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2015, p. 70.
[61] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2012, pp. 15–28.
[62] Y. Zhang, T. Liu, C.A. Meyer, J. Eeckhoute, D.S. Johnson, B.E. Bernstein, C. Nusbaum, R.M. Myers, M. Brown, W. Li, et al., Model-based analysis of ChIP-seq (MACS), Genome Biol. 9 (9) (2008) R137.
[63] M. Zytnicki, Y. Luo, H. Quesneville, Efficient comparison of sets of intervals with nc-lists, Bioinformatics 29 (7) (2013) 933–939.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085

Vahid Jalili was awarded his Ph.D. degree on Information Technology at the Politecnico di Milano, Italy, in 2016. His research on tertiary analysis of Next Generation Sequencing data is focused on systematic solutions for analytical and computational challenges. His research interest spans topics in computational biology, cognitive science, and artificial intelligence; such as information retrieval and data mining in Genomics, cognitive inhibition and visual perception, and application of artificial intelligence to games.

Matteo Matteucci is Associate Professor at the Dipartimento di Elettronica Informazione e Bioingegneria of Politecnico di Milano, Italy. In 1999 he got a Laurea degree in Computer Engineering at Politecnico di Milano, in 2002 he got a Master of Science in Knowledge Discovery and Data Mining at Carnegie Mellon University (Pittsburgh, PA), and in 2003 a Ph.D. in Computer Engineering and Automation at Politecnico di Milano (Milan, Italy). His main research topics are pattern recognition, machine learning, machine perception, robotics, computer vision and signal processing. He has co-authored more than 150 scientific international publications and he has been involved in national and international funded research projects.

Marco Masseroli is Associate Professor at the Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano, Italy, and lecturer of Bioinformatics and BioMedical Informatics. His research activity is on the application of information technology to the medical and biological sciences in several Italian and international research centers. He has also been Visiting Professor at the Departamento de Anatoma Patolgica, Facultad de Medicina of the Universidad de Granada - Spain, and Visiting Faculty at the Cognitive Science Branch of the National Library of Medicine, National Institute of Health, Bethesda - US. His research interests are in the area of bioinformatics and biomedical informatics, focused on distributed Internet technologies, biomolecular databases, controlled biomedical terminologies and bio-ontologies to effectively retrieve, manage, analyze, and semantically integrate genomic information with patient clinical and high-throughout genomic data. He is the author of more than 180 scientific articles, which have appeared in international journals, books and conference proceedings.

Stefano Ceri is Professor at the Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano, Italy. He was Visiting Professor at the Computer Science Department of Stanford University (1983–1990), Chairman of the Computer Science Section of DEI (1992–2004), Director of Alta Scuola Politecnica (ASP) of Politecnico di Milano and Politecnico di Torino (2010–2013). In 2008 he has been awarded an advanced ERC Grant on Search Computing (2008–2013). He is co-founder (2001) of WebRatio (http://www.webratio.com/). His research work has been generally concerned with extending database technology to incorporate new features: distribution, object-orientation, rules, streaming data, crowd-based and genomic computing. He is currently leading the PRIN project GenData 2020, focused on building query and data analysis systems for genomic data as produced by fast DNA sequencing technology. He is the recipient of the ACM-SIGMOD "Edward T. Codd Innovation Award" (2013), and an ACM Fellow and member of the Academia Europaea.

Please cite this article as: V. Jalili et al., Indexing Next-Generation Sequencing data, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.08.085