Genomic Region Operation Kit for Flexible Processing of Deep Sequencing Data

Kristian Ovaska, Lauri Lyly,
Biswajyoti Sahu, Olli A. Jänne, and
Sampsa Hautaniemi

Abstract—Computational analysis of data produced in deep sequencing (DS) experiments is challenging due to large data volumes and requirements for flexible analysis approaches. Here, we present a mathematical formalism based on set algebra for frequently performed operations in DS data analysis to facilitate translation of biomedical research questions to language amenable for computational analysis. With the help of this formalism, we implemented the Genomic Region Operation Kit (GROK), which supports various DS-related operations such as preprocessing, filtering, file conversion, and sample comparison. GROK provides high-level interfaces for R, Python, Lua, and command line, as well as an extension C++ API. It supports major genomic file formats and allows storing custom genomic regions in efficient data structures such as red-black trees and SQL databases. To demonstrate the utility of GROK, we have characterized the roles of two major transcription factors (TFs) in prostate cancer using data from 10 DS experiments. GROK is freely available with a user guide from http://csbi.ltdk.helsinki.fi/grok/.

Index Terms—Bioinformatics, deep sequencing, genomic data analysis, region set algebra, software

1 Introduction

Biomedical research benefits from the use of genome-scale measurement technologies that allow rapid quantization of genetic variation, transcriptional or protein activity, binding of transcription factors (TFs), and epigenetic modifications in a high resolution. In particular, deep sequencing (DS) is a general technology in which target molecules, such as DNA and RNA, are fragmented, sequenced and the resulting short reads are then processed computationally [1]. The short reads can be used in various applications, such as *de novo* sequence assemblies, resequencing of individuals, transcriptional and epigenetic profiling, and determination of DNA-protein interactions, depending on the experimental setup. The number of short reads in a DS experiment can be hundreds of millions, which poses challenges in managing and analyzing DS data. Indeed, the lack of flexible and efficient bioinformatics analysis is recognized as a major bottleneck in DS applications [2].

Bioinformatics analysis of DS data is typically implemented by multiple connected steps that form computational workflows. Common steps include sequence alignment, quality control, peak detection, variant calling, differential expression determination and database annotation [3], [4]. These steps may be implemented using various software packages that often do not have standardized interfaces. To connect these steps together, there is a need for general-purpose tools that act as glue, or adaptors, between various analysis software. These "glue tools" implement functionality such as preprocessing, filtering, file conversion, and comparing and combining replicate samples. A useful abstraction for such glue tools is the *genomic region*: an annotated chromosomal interval that may represent sequencing reads, genomic variants, exon coordinates, and copy numbers. Importantly, many cases of genomic processing can be expressed in terms of genomic regions.

Existing glue tools for processing genomic regions include interactive and scripting-based solutions. While interactive applications, such as Galaxy Browser [5] and Savant [6] may be relatively easy to learn, scripting-based approaches, such as BEDTools [7], Pybedtools [8], BEDOPS [9], and Tabix [10], have better scalability to complex and large experimental setups due to their extensive automation support. Even though the BEDTools family supports command-line and Python interfaces, there is no comparable solution for the popular R language. In addition, the existing tools are composed of fragmented sets of utilities where each operation is implemented as a special case, which may hinder their use and extension. Accordingly, there is a need for a library that systematically implements genomic region operations derived from an underlying mathematical framework.

We introduce here the Genomic Region Operation Kit (GROK), which is a software toolkit and framework for general purpose processing of genomic region data. The software has two target audiences: data analysts who process experimental DS data, and method developers who implement novel algorithms for DS analysis. For practical data analysis tasks, GROK provides flexible tools to process large-scale data sets using R, Python, Lua, or a command-line interface (CLI). For method developers, GROK provides a clean and extensible C++ API that enables access to genomic file format I/O and efficient data structures, such as an SQLite database, which facilitates method development.

Design of genomic analysis tools is often driven by practical needs. However, it is also important to develop theoretical frameworks that allow systematic development and use of mathematical operations applied to genomic data. To have a solid formal foundation for GROK and to identify all relevant region operations, we formalized the key concepts used by the library in mathematical form with a region algebra. This formalism fully specifies the genomic region abstraction. The algebra developed herein can be seen as a facilitator that provides a vocabulary to translate biomedical research questions into computational language. The formalism allows answering these questions with computational tools, such as the freely available GROK software. To demonstrate the utility of GROK and the region algebra, we analyze androgen receptor (AR) binding sites in a prostate cancer cell line using 10 DS experiments.

2 Region Algebra

2.1 Sequences and Regions

A *sequence* represents a nucleotide polymer such as a reference chromosome or a sequencing read. We denote the set of all sequences Q and an element of this set $q \in Q$. A sequence is composed of a concatenation of symbols from the DNA alphabet {A, C, G, T} that represents covalently bound nucleotides in the 5′ to 3′ direction. Given a sequence q with symbols $a_0 \ldots a_{n-1}$, the length of the sequence $|q| = n$ is the number of nucleotides in the polymer. Sequences have an identity independent from the string of nucleotide elements: it is possible for two sequences $q_1 \neq q_2$ to have identical nucleotide composition. This formulation allows to consider unmapped short reads from the same genomic locus as distinct entities.

A key concept in our formalism is the *region*, which represents a segment (interval) of a specific sequence. For reference

- *K. Ovaska, L. Lyly, and S. Hautaniemi are with the Research Programs Unit, Genome-Scale Biology and Institute of Biomedicine, Biochemistry and Developmental Biology, Biomedicum Helsinki, University of Helsinki, Room B524a, PO Box 63 (Haartmaninkatu 8), Helsinki 00014, Finland. E-mail: sampsa.hautaniemi@helsinki.fi.*
- *B. Sahu is with the Institute of Biomedicine, Physiology, Biomedicum Helsinki, University of Helsinki, PO Box 63 (Haartmaninkatu 8), Helsinki 00014, Finland.*
- *O.A. Jänne is with the Institute of Biomedicine, Physiology, Biomedicum, Helsinki, University of Helsinki, PO Box 63 (Haartmaninkatu 8), and the Department of Clinical Chemistry, Helsinki University Central Hospital, Helsinki 00014, Finland.*

Manuscript received 28 June 2012; revised 12 Nov. 2012; accepted 14 Dec. 2012; published online 28 Dec. 2012.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2012-06-0158.
Digital Object Identifier no. 10.1109/TCBB.2012.170.

1545-5963/13/$31.00 © 2013 IEEE Published by the IEEE CS, CI, and EMB Societies & the ACM

TABLE 1
Region Operation Definitions

Definition	IP	Description
$\mathrm{FREQ}_L(l, h, \overline{\mathrm{A}}_n)$	no	Locations that are present in at least l (≥ 1) and at most h $(\leq n)$ region sets.
$\mathrm{A}_1 \cup_L \ldots \cup_L \mathrm{A}_n$	no	Locations that are present in at least one of the sets A_i: equal to $\mathrm{FREQ}_L(1, n, \overline{\mathrm{A}}_n)$.
$\mathrm{A}_1 \cap_L \ldots \cap_L \mathrm{A}_n$	no	Locations that are present in all A_i: $\mathrm{FREQ}_L(n, n, \overline{\mathrm{A}}_n)$.
$\mathrm{A} \setminus_L \mathrm{B}$	no	Locations that are present in A but not in B.
$\mathrm{MERGE}(\mathrm{A}, k \geq 0)$	no	Merge all regions that overlap or whose gap is at most k nt.
$\mathrm{FLIP}(A)$	no	Switch strand from forward to reverse and vice versa: new strand $h_D(r') = -h_D(r)$.
$\mathrm{EXPAND}(\mathrm{A}, k_e, k_s)$	no	Extend all regions $k_e \in \mathbb{Z}$ nt from the end and $k_s \in \mathbb{Z}$ nt from the start. For -1 strand regions, extending from the end affects the leftmost position $h_L(r)$. Negative values shrink regions.
$\mathrm{SHIFT}(\mathrm{A}, k)$	no	Move all regions $k \in \mathbb{Z}$ nt in sense direction. For positive k, $+1$ strand regions are moved to the right and -1 strand regions to the left; for negative k, the move will be the other way around. Regions outside sequence boundaries are trimmed.
$\mathrm{FREQ}(L, H, \overline{\mathrm{A}}_n)$	yes	Regions are present in at least L (≥ 1) and at most H $(\leq n)$ region sets.
$\mathrm{A}_1 \cup \ldots \cup \mathrm{A}_n$	yes	Regions that are present in at least one of the sets A_i: equal to $\mathrm{FREQ}(1, n, \overline{\mathrm{A}}_n)$.
$\mathrm{A}_1 \cap \ldots \cap \mathrm{A}_n$	yes	Regions that are present in all A_i: $\mathrm{FREQ}(n, n, \overline{\mathrm{A}}_n)$.
$\mathrm{A} \setminus \mathrm{B}$	yes	Regions that are present in A but not in B.
$\mathrm{OVERLAP}(\mathrm{A}, \mathrm{B})$	yes	Regions in A that share nucleotide locations with any region in B.
$\mathrm{FILTER}(\mathrm{A}, f)$	yes	General filter that takes an accept function $f : \mathrm{R} \rightarrow \{0, 1\}$ as argument. Produces sequences in A for which f produces 1. Examples include filtering by region length and strand.

IP indicates whether the operation is identity-preserving. A, A_i, *and* B *denote region sets* $\subset \mathrm{R}$. $\overline{\mathrm{A}}_n$ *denotes a vector of* n *regions sets* $\mathrm{A}_1, \ldots, \mathrm{A}_n$. *All operations produce a set of regions.*

chromosome sequences, regions represent genomic intervals such as gene, exon, or single nucleotide polymorphism (SNP) genomic coordinates. Regions can also represent custom entities such as aligned short reads and TF binding sites located by ChIP-seq (chromatin immunoprecipitation followed by massive parallel sequencing) peak detection algorithms [11].

Regions have attributes that define which nucleotides are covered by the region $r \in R$ in the sequence $h_Q(r)$. The leftmost position $h_L(r) \in \{0, 1, \ldots, |h_Q(r)| - 1\}$ is the first position in the sequence $h_Q(r)$ covered by the region when reading from the 5′ end. Length $h_N(r) \geq 0$ denotes how many nucleotides are covered in total; a zero-length region is allowed. Strand $h_D(r) \in \{-1, 0, 1\}$ indicates whether the region is located in the forward (1) or reverse (-1) strand, or whether the strand is not relevant (0). An identity tag $h_I(r) \in \mathbb{N}$, which is an integer with no semantic meaning, is used in conjunction with other attributes to define an equivalence relation for regions. These five attributes of a region form an *identity tuple* $(h_Q(r), h_L(r), h_N(r), h_D(r), h_I(r))$ that defines the equivalence relation for regions: Two regions are equivalent if and only if their identity tuples are identical.

A region covers the nucleotides in the half-open interval $[h_L(r), h_L(r) + h_N(r))$. Strandedness $h_D(r)$ affects how this interval is interpreted: For forward strand regions, the region is read rightward from $h_L(r)$, while for reverse strand regions, the region is read leftward from $h_L(r) + h_N(r) - 1$. For "none" strand regions with $h_D(r) = 0$, the reading direction is not relevant. We use the term "leftmost" position for $h_L(r)$ instead of "start" position to avoid ambiguity with reverse strand coordinates.

There are two semantic interpretations for regions that we denote by *relation* and *interval* interpretations. In the former, a region is analogous to a relation in a database that has its own unique identity, such as a specific gene. If the attributes of the region are modified, for example, by splitting the region, the identity is lost. By contrast, in the interval interpretation, the region represents a genomic interval whose important characteristics are the nucleotides covered, but not the particular identity of the region itself. For instance, the region can be split into two adjacent regions covering the same genomic area with no loss of information. Such regions may represent DNA-protein interaction sites, GC-rich genomic areas, or other custom markers.

TABLE 2
Illustration of Nonidentity Preserving Region Operations

Region set	Operation
`====  ====  ====     ==`	A
`  ======      ====     ==`	B
`==========  ======   ====`	$\mathrm{A} \cup_L \mathrm{B} = \mathrm{B} \cup_L \mathrm{A}$
`  ==  ==      ==`	$\mathrm{A} \cap_L \mathrm{B} = \mathrm{B} \cap_L \mathrm{A}$
`==      ==  ==       ==`	$\mathrm{A} \setminus_L \mathrm{B}$
`    ==          ==     ==`	$\mathrm{B} \setminus_L \mathrm{A}$
`    ==    ==    =====  ==`	$\{r\} \setminus_L \mathrm{A}$
`================     ==`	$\mathrm{MERGE}(\mathrm{A}, 2)$
`=======================`	$\mathrm{MERGE}(\mathrm{A}, 5)$
`===== ===== =====    ===`	$\mathrm{EXPAND}(\mathrm{A}, 1, 0)$
`  ==    ==    ==`	$\mathrm{EXPAND}(\mathrm{A}, 0, -2)$
`   ====  ====  ====     =`	$\mathrm{SHIFT}(\mathrm{A}, 3)$

Region r *spans the whole sequence.*

2.2 Region Operations

Related regions are often aggregated and processed together in *regions sets*. For example, all gene coordinates in the human genome form a region set, as do all aligned short reads for a particular biological sample. Operations that filter, combine, or modify region sets are common in sequencing applications. Such operations are mathematically of the form $f : \mathcal{P}(R) \times \cdots \times \mathcal{P}(R) \rightarrow \mathcal{P}(R)$, where $\mathcal{P}(R)$ denotes the power set of regions R. In other words, these operations take one or more region sets as arguments and produce a single region set as result. Algebraic functions with this form are the core operations of the formalism. Operations can be divided into two main categories based on whether they maintain the identities of individual argument regions. Identity-preserving (IP) operations are filters that produce a subset of their input regions as result. They are most naturally used with the relation interpretation of regions. Nonidentity-preserving operations may split, combine, move, and expand regions and are used in the interval interpretation. Table 1 lists all region operations and Tables 2 and 3 illustrate their usage.

The most fundamental region set operations are the classical set operations: union, intersection, and difference. These are defined in two flavors to accommodate both relation and interval interpretations. Relation operations $\cup$, $\cap$, and $\setminus$ are defined using region identity tuples and correspond directly to

TABLE 3
Illustration of IP Region Operations

Region set	Operation
`====  ====  ====     ==`	A
`  ======      ====     ==`	B

`====  ====  ====     ==`
`  ======      ====     ==`
	$\mathrm{A} \cup \mathrm{B} = \mathrm{B} \cup \mathrm{A}$

`      ====  ====`	$\mathrm{A} \cap \{a_2, a_3\} = \{a_2, a_3\} \cap \mathrm{A}$
`====                 ==`	$\mathrm{A} \setminus \{\mathrm{a}_2, \mathrm{a}_3\}$
`====  ====  ====`	$\mathrm{OVERLAP}(\mathrm{A}, \mathrm{B})$
`  ======      ====`	$\mathrm{OVERLAP}(\mathrm{B}, \mathrm{A})$
`  ======`	$\mathrm{LENGTH}(\mathrm{B}, 6, \infty)$

Regions a_2 *and* a_3 *denote the second and third regions of* A*, respectively. The function* $LENGTH(R, L, H)$ *is defined using* $FILTER$ *so that the accept function produces those regions in R whose length is at least L and at most H.*

TABLE 4
AFs

Function	Formula	Description
count	$A(S) = \|S\|$	Count the number of score values.
first	$A(S) = S_0$	Use the first defined value.
last	$A(S) = S_{n-1}$	Use the last defined value.
min	$A(S) = \min_{s \in S}(s)$	Minimum of values.
max	$A(S) = \max_{s \in S}(s)$	Maximum of values.
product	$A(S) = \Pi_{s \in S}(s)$	Product of values.
sum	$A(S) = \sum_{s \in S}(s)$	Sum of values.
zero	$A(S) = 0$	Set score to zero.

Here, S denotes a vector $\mathrm{I\!R}^n$ of n existing score values. Its length n is denoted by $|S|$.

TABLE 5
Core Region Attributes in GROK

Attribute	Type	Description
seqid	String	Identifier that specifies the reference sequence. Example: chr1.
strand	Integer	Forward (1), reverse (-1) or not relevant (0).
left	Integer (64-bit)	The leftmost position of the genomic interval (≥ 0).
length	Integer (64-bit)	Length of the genomic interval (≥ 0).
tag	Integer (64-bit)	Identity tag with no semantic interpretation.

the classical set-theoretical definition. Interval operations $\cup_L$, $\cap_L$, and $\setminus_L$ discard region identity and operate at the level of individual genomic locations (denoted by L in operation names). Conceptually, the L variants first split all regions into one-nucleotide regions with unit length and then compute set operations for these. In essence, each nucleotide location of the genome is considered individually. The union and intersection operations are generalized into the FREQ and $\mathrm{FREQ_L}$ operations that take lower and upper frequency boundaries as parameter. It allows to define k-out-of-n operations such as "select genomic regions present in three out of four samples."

Other operations allow filtering and modifying regions. The OVERLAP operation is a filter that compares whether regions in two region sets share common nucleotide positions: For instance, whether gene coordinates overlap with transcription factor binding sites. It is related to the interval intersection operation $\cap_L$ in the sense that when overlapping regions are present, their interval intersection is nonempty. However, OVERLAP is IP and thus can be used in contexts where the relation interpretation is used. The operations FLIP, EXPAND, and SHIFT allow manipulating region attributes. MERGE combines adjacent regions (as defined by a gap parameter) into one.

2.3 Score and Aggregate Functions (AFs)

In addition to processing interval data, many sequencing applications require quantitative processing in which numeric values are attached to genomic locations. Examples include short read coverage counts, Phred quality values for base calling as well as p-values and fold enrichments for ChIP-seq peak detection and RNA-seq results. These numeric values can be associated to regions using *score functions* of the form $s : \mathrm{R} \rightarrow \mathrm{I\!R}$ that assign a unique real value to each region. There may be multiple score functions defined for a region set; for example, replicate samples produce individual coverage functions.

While score functions can be defined for any region set, they are most natural in a context where each genomic location obtains a unique score value, such as short read coverage. In our formalism, this is accomplished by defining a score value over a region set that is a *partition*, i.e., whose regions cover all locations of their associated sequences exactly once. Then, each region defines one piece of the piecewise-defined function from genomic locations to real values. Here, the interval interpretation of regions is used.

Many region set operations create new regions by modifying identities of existing regions, which raises the need to synthesize scores for the new regions. Since scores are defined based on region identity, score values could be "lost" when the identity changes. To avoid this, synthesis of new scores is done using AFs that map a vector of old scores into a unique new score. These AFs are conceptually similar to SQL AFs. AFs are used in conjunction with nonidentity preserving region set operations to ensure that score information is preserved, or to summarize a collection of scores into a single composite value (e.g., maximum). AFs are listed in Table 4.

3 GROK Software Design

GROK is an extensible software library for practical processing of DS and other genomic data sets. The design of the library is based on the concepts elaborated in the region algebra. A design goal of GROK is to implement elementary operations with generic interfaces that are used as building blocks for complex operations. For example, computing short read coverage is a special case of a union operation that assigns region score annotations based on short reads counts.

GROK architecture is based on two core abstractions: A *region model* and a *region store* interface. Regions represent the genomic data for processing, and region stores provide programmatic access to collections of regions. Complex operations are implemented using these abstractions. The region model is derived from the region tuples in the region algebra, and supplemented with additional annotations. Similarly, region stores are the software analogue of region sets.

3.1 DNA Region Model

In the software context, a region is an annotated DNA interval that corresponds to one record in files such as BED, GFF, and BAM [12]. The region model is not specific to any single file format, but rather abstracts features found in commonly used genomic formats. Regions can be read from files but also created from custom analysis algorithms. Regardless of their origin, regions use the same data model and can be compared to each other and processed in similar manner. File format parsers implemented in GROK transform regions into a standard format. However, regions from different origins (such as BED and BAM files) may support different sets of annotations.

Regions are represented as sets of attribute values, i.e., key-value pairs. Keys are always strings and values may be strings, integers or floating point values. Attributes are divided into core attributes, which are available for all regions regardless of their origin, and an arbitrary number of optional *annotations*. Core attributes are listed in Table 5 and directly correspond to the elements of region identity tuples in the formalism.

Optional annotation attributes provide context specific information to regions, such as file format specific fields. Example annotations include region identifiers (e.g., column four in BED format), numeric scores (e.g., column five in BED format), and nucleotide sequence strings for short reads. A region may have any number of annotations as long as their keys are unique. Regions from the same origin (e.g., from the same file format) share the same set of available annotations. Annotation values do not affect region identity, since they are not part of the identity tuple. That is, two regions with the same core attributes but different annotations are still considered to be identical.

3.2 Region Store Interface

GROK provides access to region collections through a region store interface. This is an abstract interface that has multiple ready-made implementations in GROK; in addition, users can define new implementations using C++. The region store interface defines methods for adding, removing, querying, and iterating over

TABLE 6
Region Store Methods

Method	Description
`add`	Add a region to the database, or write (append) a region to a file.
`iter`	Provide access to each region in the collection one at a time. Order depends on the region store: regions may be sorted or unsorted.
`get`	Fetch a specific region from the collection using region identity. Produce an empty result if the region is not present.
`remove`	Remove a region from the database using region identity.
`overlap`	Iterate over those regions that overlap with a given query region.
`set_annotation`	Replace annotation values of a region present in the collection.
`set_score`	Set numeric score values for a region store that represents a real-valued function over the genome.

regions. All methods are listed in Table 6. Region store implementations generally do not support all methods, but rather a subset of them. Region stores provide facilities for file I/O, intermediate storage for multipass algorithms, and region filtering and transformation operations. In combination with the DNA region model, the interface provides uniform access to these diverse use cases. Implementation details, such as the actual file formats or data structures used, are largely hidden from the user.

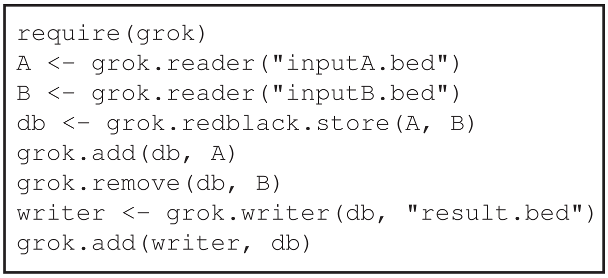
The major categories of region store implementations are file readers, file writers, and region databases. Readers support only the iteration method (`iter`) and they iterate over each region defined in an input file. Similarly, writers support only the writing (appending) method (`add`): A region "added" to the region collection is written to an output file. Region databases are in-memory or on-disk stores that support adding, removing, and querying methods. They are used for multipass algorithms that need to keep intermediate results in storage, and they provide index structures for efficient processing. Databases use the file format independent region model to store regions efficiently and in homogeneous format regardless of region origin. Databases can store arbitrary region annotations in addition to core attributes.

GROK implements over 10 region stores for various purposes. For I/O, GROK provides readers and writers supporting major genomic file formats (BAM/SAM [12], BED, BedGraph, CSV, FASTQ [13], GFF, VCF [14], and Wiggle). For databases, there are three general-purpose implementations and one specialized for processing score functions. General-purpose databases include in-memory hash and redblack stores that support efficient membership queries using hash table and redblack tree indices [15], respectively, and an SQLite store that allows processing regions in an on-disk or in-memory SQL database. The redblack store is intended for data sets that fit into memory and should be iterated in genomic order. The hash store uses a different indexing strategy that does not order regions. The SQLite store can be used for large data sets with its on-disk option; in addition, it provides an efficient implementation of overlap queries (the OVERLAP operation). A variant of the redblack store, a redblack partition store, is used for manipulating score functions using the `set_score` method.

Region stores can be flexibly combined into ad hoc workflows that compute composite operations using several instances of elementary region stores. A simple workflow would consist of a BAM reader combined to a BED writer, i.e., regions produced by the BAM iterator are written (added) to the BED region store. This implements BAM to BED file conversion and also illustrates how this common use case can be implemented directly using the core facilities of GROK instead of a special case. As a more complex example, the set difference operation $A \backslash B$ is implemented by as follows:

1. Constructing region readers for A and B.
2. Initializing an empty region database (e.g., a redblack store) for temporary storage.
3. Adding all regions from A to the database.
4. Removing all regions from B from the database.
5. Writing database contents into output file.

For convenience, this composite operation and other similar operations are available in the scripting language APIs.



```
require(grok)
A <- grok.reader("inputA.bed")
B <- grok.reader("inputB.bed")
db <- grok.redblack.store(A, B)
grok.add(db, A)
grok.remove(db, B)
writer <- grok.writer(db, "result.bed")
grok.add(writer, db)
```

Fig. 1. Example of using the R API of GROK. The code computes the set difference $A \setminus B$ using relation interpretation of regions. Inputs are read from two BED files. An in-memory redblack database is initialized; the store supports the union of annotation attributes of A and B as specified in the constructor `grok.redblack.store`. Results of the set difference are written to a BED file. For convenience, the set difference operation is available as `grok.diff` in the GROK API.

3.3 Multilanguage API

Region store implementations and other GROK facilities are accessible through a uniform interface from the scripting languages R, Python, and Lua. The scripting interfaces are generated using SWIG [16] to ensure consistent access for different languages. In addition, a CLI provides a subset of GROK functionality for shell scripting. The CLI includes commands for file conversion and set operations. Finally, a C++ API is available for extending GROK and for integrating GROK into other C++ applications such as genomic analysis programs. An illustration of implementing the $A \backslash B$ set operation using the R API of GROK is shown in Fig. 1, this function is also available as a predefined high-level function in the GROK API. Full details of the APIs, as well as additional code examples, are in the GROK User Guide.

4 Benchmark Studies

We assessed the performances of GROK, BEDTools 2.16.2, and BEDOPS 1.2.5 using a set of six benchmarks listed in Table 7. Each benchmark was implemented using GROK, BEDTools, and BEDOPS, and the results were checked for consistency, excluding minor differences such as BED annotation columns or row order (except in the Sort benchmark). The benchmarks measure both execution (wall clock) time and memory usage (maximum resident set size), as reported by the GNU *time* utility. To minimize the effect of I/O performance on timing results, a memory file system was used. The benchmarks were executed three times and the best measurement for each method was selected. For GROK, the Lua bindings were used. Benchmarks were executed on a 64-bit Xubuntu 12.04 machine with an Intel i5-2520M 2.5-GHz CPU, 8-GB RAM and a solid disk drive.

The main input for the benchmarks was a BAM file (wgEncodeBroadHistoneHepg2CtcfStdAlnRep1.bam) having 2.4 million short reads and taking 101 MB of size from the ENCODE project [17]. The BAM file was converted to a sorted BED file that was used for all benchmarks except BAM to BED. A secondary input was a 14-MB BED file containing annotations of

TABLE 7
Performance Benchmark

Case	GROK time	BEDTools time	BEDOPS time	GROK memory	BEDTools memory	BEDOPS memory
BAM to BED	5.58 s	5.01 s	–	16 MB	2 MB	–
Intersection	11.3 s	6.95 s	1.75 s	1251 MB	213 MB	1 MB
Sort	6.33 s	7.90 s	2.62 s	543 MB	1130 MB	156 MB
Merge	4.49 s	4.69 s	1.66 s	2 MB	2 MB	1 MB
Overlap	3.10 s	6.89 s	2.00 s	90 MB	213 MB	1 MB
Complement	7.96 s	11.2 s	1.68 s	533 MB	1118 MB	1 MB

For each benchmark case, execution times and peak memory usages of GROK, BEDTools, and BEDOPS are shown. Benchmarks that process BAM files were omitted for BEDOPS as BEDOPS only supports the BED format.

human gene and exon coordinates, totaling 423,234 records. We denote these three files EBAM (ENCODE BAM file), EBED (converted EBAM file), and GENES (gene coordinates).

The BAM to BED benchmark measures basic file read and write routines. For GROK, time and memory complexities are $O(n_A)$ (where n_A is the size of the EBAM input) and $O(1)$, respectively. The Intersection benchmarks executes the "EBED $\cap_L$ GENES" operation, where EBED and GENES denote the region sets corresponding to these two inputs. The time complexity of the GROK implementation is $O(n_B \log n_B + n_G \log n_G)$, where n_B and n_G denote sizes of the EBED and GENES inputs, respectively, and the memory complexity is $O(n_B + n_G)$. The Sort benchmark sorts a BED file and has both time and memory complexity of $O(n_B \log n_B)$ in GROK. The Merge benchmark merges regions in a pre-sorted BED file and has time complexity of $O(n_B)$ and memory complexity of $O(1)$. The Overlap benchmark computes the `OVERLAP(EBED, GENES)` operation, which has time complexity of $O(n_B + n_G \log n_G)$ and memory complexity of $O(n_G)$. Finally, the Complement benchmark computes "G $\setminus_L$ EBED," where G represents the whole genome. Its time and memory complexities are both $O(n_B \log n_B)$. Benchmark source codes are distributed along with GROK.

Results of the benchmark analyses are shown in Table 7. GROK and BEDTools perform at comparable levels for speed and memory efficiency. In this benchmark, BEDOPS is the fastest and least memory consuming method, which was expected due to performance-optimized implementation of its operations [9]. The optimized performance of BEDOPS, however, entails stronger assumptions for the input than GROK and BEDTools, in particular the requirement for presorting the input BED files.

5 Case Study

Cancer is a disease of uncontrolled cell proliferation where expression levels for a number of genes are dysregulated to promote cell processes driving cancer progression. Gene expression is mainly controlled at the transcriptional level by TFs that bind to their *cis*-elements, i.e., TF-specific DNA sequences. In prostate cancer, a key TF driving prostate cancer progression is the androgen receptor (AR), which regulates cell growth and differentiation in prostate and other tissues [18]. High AR expression in a tumor correlates with poor patient survival, and AR is a key therapeutical target in prostate cancer [19], [20]. AR signal transduction is initiated by the binding of the cognate ligand to the AR, which results in translocation of the AR to the nucleus. Inside the nucleus, the AR binds to androgen response elements to form an active TF complex with coregulatory proteins. AR-regulated genes include several genes that have been shown to be involved in prostate cancer development and progression, such as the prostate-specific antigen gene. Even though AR signaling plays a central role in prostate cancer progression, the exact roles of AR and AR-responsive genes in prostate cancer are still poorly understood.

Here, we demonstrate the utility of GROK in an experimental setting where we analyzed genome-wide binding sites of AR and its collaborating pioneer TF, the forkhead protein FoxA1 [21] in LNCaP-1F5 prostate cancer cell line using ChIP-seq data. FoxA1 acts as a pioneer by remodeling chromatin structure in order for AR to bind. AR expressing LNCaP cells are derived from a lymph node metastasis of a prostate cancer patient, and the LNCaP-1F5 variant is engineered to express the glucocorticoid receptor, another steroid receptor.

Using ChIP-seq data from our recent study [22], we searched for chromosomal locations corresponding to the pioneering role of FoxA1. ChIP-seq was performed on AR and FoxA1 in the LNCaP-1F5 cells (denoted *parental*) and in an experiment in which FoxA1 was depleted using siRNA (denoted *siFoxA1*), totaling four experimental settings. Two biological replicate samples were acquired for each experimental setting. In addition, for siRNA depleted FoxA1 ChIP, two technical replicates were done, which corresponds to a total of $2 + 2 + 2 + 4 = 10$ replicates. Binding sites of AR and FoxA1 were determined by MACS [23], which provided 10 region sets that were used as input for GROK. The fold enrichment metric produced by MACS was used as the region score function.

In FoxA1 pioneered locations, binding by FoxA1 is a prerequisite to enable AR binding. In these locations, both AR and FoxA1 binding is present in parental cells, and AR binding is

TABLE 8
Algebraic Formulation of ChIP-Seq Case Study

Region set definition	Description
pAR$_i$	Parental cells, AR ChIP-seq replicates ($i \in \{1,2\}$)
pFoxA1$_i$	Parental cells, FoxA1 ChIP-seq replicates ($i \in \{1,2\}$)
sAR$_i$	siFoxA1 cells, AR ChIP-seq replicates ($i \in \{1,2\}$)
sFoxA1$_i$	siFoxA1 cells, FoxA1 ChIP-seq replicates ($i \in \{1,2,3,4\}$)
pAR = pAR$_1$ $\cap_L$ pAR$_2$	Combined parental AR peaks
pFoxA1 = pFoxA1$_1$ $\cap_L$ pFoxA1$_2$	Combined parental FoxA1 peaks
sAR = pAR$_1$ $\cap_L$ pAR$_2$	Combined siFoxA1 AR peaks
sFoxA1 = FREQ$_L$(3, 4, sFoxA1$_1$, sFoxA1$_2$, sFoxA1$_3$, sFoxA1$_4$)	Combined siFoxA1 FoxA1 peaks
PIONEER = (pAR $\cap_{\mathrm{L}}$ pFOX) $\setminus_{\mathrm{L}}$ (sAR $\cup_{\mathrm{L}}$ sFOX)	FoxA1 pioneer locations

All operations use the L interval variants of algebra operations. Biological replicates are combined with intersection, except for the FoxA1 ChIP-seq in siFoxA1 cells, which uses a three-out-of-four operation.

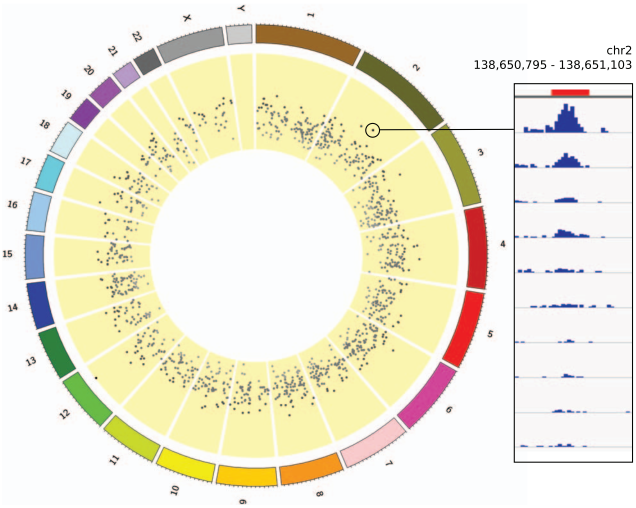


Fig. 2. ChIP-seq case study results visualized using Circos [27]. Each dot represents a genomic region that corresponds to the pioneering role of FoxA1. Distance from the inner circle indicates the composite logarithmic fold enrichment score using the `min` AF, with low values being near the center. The linear range of fold enrichments is 6 to 125. Highlighted is a region in chromosome 2 that produced one of the 1,258 result regions by combining information from all 10 samples. In the right, short read coverages of samples are visualized in the same order as in Table 8 so that replicate samples are next to each other. The red region indicates the pioneer peak regions.

absent in FoxA1 depleted cells. Naturally, FoxA1 binding is also expected to be absent in siFoxA1 cells. This hypothesis is encoded using the region algebra in Table 8, demonstrating the usefulness of a succinct notation in formalizing complex experimental settings. Using the algebraic expression as the starting point, the high-level functions of the R API of GROK were used to implement the analysis in computational form. We derived scores for pioneer regions using the `min` AF so that the score of final result locations is the minimum of fold enrichments in parental cell biological replicates. The minimum function provides a conservative estimate of AR and FoxA1 binding enrichment over IgG negative control.

GROK analysis resulted in 1,258 nonoverlapping FoxA1 pioneer regions spanning 401,005 bases. These regions together with their scores are visualized in Fig. 2. To characterize these regions in comparison to all AR binding sites in parental cells (pAR), we matched known DNA sequence motifs [24] against the sequences contained within the regions. Motif matches within these two region sets were compared to random DNA regions with the same length distribution. As expected, AR and Fox family motifs are enriched in both region sets (see Supplementary Data, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2012.170). A difference between the pioneer and pAR regions is the occurrence of the MYC/MAX E-box motif. It is enriched in the pAR set compared to random DNA (fold enrichment, $\mathrm{FE} = 1.3$), but depleted in the pioneer set ($\mathrm{FE} = 0.58$). This motif, with its 5′-CACGTG-3′ sequence, occurs with high frequency close to gene promoters in the genome [25] and is associated with several TFs other than MYC, such as ChREBP, SREBP, and HIF-1 [26]. While interpretation of differential binding for the E-box motif is complicated due to the multitude of its roles, one possible explanation is that FoxA1 pioneer regions have reduced dependency for E-box binding TFs due to the pioneering actions of FoxA1. In this hypothetical model, E-box binding TFs and FoxA1 could represent alternative pathways to AR-mediated transcription.

To validate the correctness of the GROK implementation, we implemented a modified version of the analysis using BEDTools 2.16.2 [7]. In Table 8, we replaced the three-out-of-four operation in sFoxA1 with intersection ($\cap_L$) and used the `first` AF instead of `min`. These modifications were necessary because these functionality are not implemented in BEDTools. The modified analyses yielded identical results in GROK and BEDTools.

6 Discussion

DS data analysis requires flexible, efficient, and well-documented software to perform operations that facilitate answering research questions based on DS data. Here, we have developed a formalism that consists of operations often performed in DS data analysis, and implemented them as an extensible toolkit called GROK. It is useful in a variety of environments due to flexible use of computational resources and the support for multiple scripting languages.

Applications for GROK in data analysis include diverse tasks in preprocessing and analysis such as filtering, sorting, merging, file conversion, and comparing biological samples. For combining and comparing samples, GROK provides a comprehensive set of region operations such as union, intersection, difference, and overlap queries. GROK supports both small and large data sets by providing alternative storage options with different performance tradeoffs. Small data sets are efficiently stored in memory, whereas an on-disk SQL database allows working on data sets whose intermediate results do not fit in memory. The latter is also useful in environments with constrained memory, such as cloud computing. Support for multiple scripting languages allows using the toolkit in heterogeneous environments: R and Python are popular in the bioinformatics community, and Lua is an elegant scripting language with low overhead. GROK supports major genomic file formats and provides a C++ extension API for adding support for new formats, such as the BigWig/BigBed [28] or Goby [29] formats.

Compared to existing tools, novel features of GROK include a multilanguage API, customizable functionality for processing numeric region scores, an SQLite region database supporting arbitrary annotations and a modular architecture that allows implementing custom complex operations in a high-level language. An example of the flexibility of the API is the built-in FREQ function, which computes a k-out-of-n filtering operation: It is implemented directly in a scripting language using the elementary methods exposed from the C++ API. To accommodate the need to take advantage of biological replicates in sequencing experiments, GROK set operations can easily be applied to an arbitrary number of samples.

The utility of both GROK and the region algebra was demonstrated in a prostate cancer case study in which we investigated the pioneering role of the TF FoxA1 in AR-mediated gene regulation. With this case study, we showed how a biomedical hypothesis can be encoded in algebraic form using the region algebra and the resulting formulas were translated into an executable GROK script. The case study demonstrates that GROK can facilitate answering biomedical research questions and establish experimentally testable predictions. GROK forms a part of an analysis workflow that is composed of upstream analysis, such as sequence alignment and peak detection, and downstream analysis, such as motif matching. This highlights the importance of flexibility in the design of GROK as it can be used in various experimental settings. GROK is freely available with a comprehensive user manual.

Acknowledgements

The authors would like to thank Marko Laakso and Elena Czeizler for discussions on the region algebra. This work was supported by the Academy of Finland (125826, Center of Excellence in Cancer Genetics Research), the Sigrid Jusélius Foundation, the Finnish Cancer Organizations, Biocentrum Helsinki, ERASysBio+ EU ERA-NET Plus scheme in FP7 (project SYNERGY), and the Helsinki Biomedical Graduate Program (KO).

REFERENCES

[1] M. Metzker, "Sequencing Technologies—The Next Generation," *Nature Rev. Genetics,* vol. 11, no. 1, pp. 31-46, 2009.

[2] J. McPherson, "Next-Generation Gap," *Nature Methods,* vol. 6, pp. S2-S5, 2009.

[3] S. Pepke, B. Wold, and A. Mortazavi, "Computation for ChIP-Seq and RNA-Seq Studies," *Nature Methods,* vol. 6, pp. S22-S32, 2009.

[4] R. Nielsen, J. Paul, A. Albrechtsen, and Y. Song, "Genotype and SNP Calling from Next-Generation Sequencing Data," *Nature Rev. Genetics,* vol. 12, no. 6, pp. 443-451, 2011.

[5] J. Goecks et al., "Galaxy: A Comprehensive Approach for Supporting Accessible, Reproducible, and Transparent Computational Research in the Life Sciences," *Genome Biology,* vol. 11, no. 8, p. R86, 2010.

[6] M. Fiume, V. Williams, A. Brook, and M. Brudno, "Savant: Genome Browser for High-Throughput Sequencing Data," *Bioinformatics,* vol. 26, no. 16, pp. 1938-1944, 2010.

[7] A. Quinlan and I. Hall, "BEDTools: A Flexible Suite of Utilities for Comparing Genomic Features," *Bioinformatics,* vol. 26, no. 6, pp. 841-842, 2010.

[8] R. Dale, B. Pedersen, and A. Quinlan, "Pybedtools: A Flexible Python Library for Manipulating Genomic Datasets and Annotations," *Bioinformatics,* vol. 27, pp. 3423-3424, 2011.

[9] S. Neph et al., "BEDOPS: High-Performance Genomic Feature Operations," *Bioinformatics,* vol. 28, no. 14, pp. 1919-1920, 2012.

[10] H. Li, "Tabix: Fast Retrieval of Sequence Features from Generic TAB-Delimited Files," *Bioinformatics,* vol. 27, no. 5, pp. 718-719, 2011.

[11] E. Wilbanks and M. Facciotti, "Evaluation of Algorithm Performance in ChIP-Seq Peak Detection," *PLoS ONE,* vol. 5, no. 7, p. e11471, 2010.

[12] H. Li et al., "The Sequence Alignment/Map Format and SAMtools," *Bioinformatics,* vol. 25, no. 16, pp. 2078-2079, 2009.

[13] P. Cock, C. Fields, N. Goto, M. Heuer, and P. Rice, "The Sanger FASTQ File Format for Sequences with Quality Scores, and the Solexa/Illumina FASTQ Variants," *Nucleic Acids Research,* vol. 38, no. 6, pp. 1767-1771, 2010.

[14] P. Danecek et al., "The Variant Call Format and VCFtools," *Bioinformatics,* vol. 27, no. 15, pp. 2156-2158, 2011.

[15] T. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* MIT Press, 2001.

[16] D. Beazley et al., "SWIG: An Easy to Use Tool for Integrating Scripting Languages with C and C++," *Proc. Fourth USENIX Tcl/Tk Workshop,* pp. 129-139, 1996.

[17] The ENCODE Project Consortium, "An Integrated Encyclopedia of DNA Elements in the Human Genome," *Nature,* vol. 489, no. 7414, pp. 57-74, 2012.

[18] C. Heinlein and C. Chang, "Androgen Receptor in Prostate Cancer," *Endocrine Rev.,* vol. 25, no. 2, pp. 276-308, 2004.

[19] T. Visakorpi, E. Hyytinen, P. Koivisto, M. Tanner, R. Keinänen, C. Palmberg, A. Palotie, T. Tammela, J. Isola, and O. Kallioniemi, "In Vivo Amplification of the Androgen Receptor Gene and Progression of Human Prostate Cancer," *Nature Genetics,* vol. 9, no. 4, pp. 401-406, 1995.

[20] C. Chen, D. Welsbie, C. Tran, S. Baek, R. Chen, R. Vessella, M. Rosenfeld, and C. Sawyers, "Molecular Determinants of Resistance to Antiandrogen Therapy," *Nature Medicine,* vol. 10, pp. 33-39, 2004.

[21] G. Bernardo and R. Keri, "FOXA1: A Transcription Factor with Parallel Functions in Development and Cancer," *Bioscience Reports,* vol. 32, no. 2, pp. 113-130, 2012.

[22] B. Sahu et al., "Dual Role of Foxa1 in Androgen Receptor Binding to Chromatin, Androgen Signalling and Prostate Cancer," *EMBO J.,* vol. 30, no. 19, pp. 3962-3976, 2011.

[23] Y. Zhang et al., "Model-Based Analysis of ChIP-Seq (MACS)," *Genome Biology,* vol. 9, no. 9, p. R137, 2008.

[24] J. Bryne, E. Valen, M. Tang, T. Marstrand, O. Winther, I. Da Piedade, A. Krogh, B. Lenhard, and A. Sandelin, "JASPAR, the Open Access Database of Transcription Factor-Binding Profiles: New Content and Tools in the 2008 Update," *Nucleic Acids Research,* vol. 36, no. suppl 1, pp. D102-D106, 2008.

[25] X. Xie, J. Lu, E. Kulbokas, T. Golub, V. Mootha, K. Lindblad-Toh, E. Lander, and M. Kellis, "Systematic Discovery of Regulatory Motifs in Human Promoters and 3′ UTRs by Comparison of Several Mammals," *Nature,* vol. 434, no. 7031, pp. 338-345, 2005.

[26] C. Dang, "MYC on the Path to Cancer," *Cell,* vol. 149, no. 1, pp. 22-35, 2012.

[27] M. Krzywinski, J. Schein, İ. Birol, J. Connors, R. Gascoyne, D. Horsman, S. Jones, and M. Marra, "Circos: An Information Aesthetic for Comparative Genomics," *Genome Research,* vol. 19, no. 9, pp. 1639-1645, 2009.

[28] W. Kent, A. Zweig, G. Barber, A. Hinrichs, and D. Karolchik, "BigWig and BigBed: Enabling Browsing of Large Distributed Datasets," *Bioinformatics,* vol. 26, no. 17, pp. 2204-2207, 2010.

[29] F. Campagne et al., Goby Framework, http://goby.campagnelab.org/, 2013.