BIOINFORMATICS APPLICATIONS NOTE Vol. 27 no. 9 2011, pages 1313–1315
doi:10.1093/bioinformatics/btr123

Genome analysis

Advance Access publication March 12, 2011

GeCo++: a C++ library for genomic features computation and annotation in the presence of variants

Matteo Cereda[1,2], Manuela Sironi[1], Matteo Cavalleri[3] and Uberto Pozzoli[1,*]

[1]Bioinformatics Lab, Scientific Institute I.R.C.C.S. 'E. Medea', Via Don L. Monza, 23852 Bosisio Parini (LC), Italy, [2]Department of Theoretical Physics, University of Turin, Via P. Giuria 1 -10125, Torino and [3]Bioingineering Lab, Scientific Institute I.R.C.C.S. 'E. Medea', Via Don L. Monza, 23852 Bosisio Parini (LC), Italy

Associate Editor: Martin Bishop

ABSTRACT

Summary: We propose a C++ class library developed to the purpose of making the implementation of sequence analysis algorithms easier and faster when genomic annotations and variations need to be considered. The library provides a class hierarchy to seamlessly bind together annotations of genomic elements to sequences and to algorithm results; it allows to evaluate the effect of mutations/variations in terms of both element position shifts and of algorithm results, limiting recalculation to the minimum. Particular care has been posed to keep memory and time overhead into acceptable limits.

Availability and Implementation: A complete tutorial as well as a detailed doxygen generated documentation and source code is freely available at http://bioinformatics.emedea.it/geco, under the GPL license. The library was written in standard ISO C++, and does not depend on external libraries.

Contact: uberto.pozzoli@bp.lnf.it

Received on December 15, 2010; revised on February 15, 2011; accepted on March 1, 2011

1 MOTIVATION

Sequence analysis algorithms play a crucial role in the identification of genomic functional elements. Several algorithms have been and are being developed to identify putative binding sites for a great number of functional elements involved in many cellular processes. Algorithms are usually implemented through 'sequence-driven' programs: a sequence is provided upon which calculations are made and results are returned with a reference to sequence positions. In the great majority of cases the results need to be interpreted in the light of some functional element annotated elsewhere (transcripts, genes, intron/exon boundaries, etc.) or their significance can be improved by comparison with some other features such as sequence conservation among different species or chromatin accessibility. These are time-consuming tasks that are often implemented using specialized packages based on interpreted languages like, for example BioPerl (Stajich *et al*., 2002) and Bioconductor (Gentleman *et al*., 2004). The same holds true when sequence variations are involved in the analysis. Sequence-driven programs need multiple runs to make calculations on multiple input sequences: this often lead to redundant recalculations and data I/O operations. Again the comparison of the results between 'varied' and 'unvaried' sequences is left to the user. Furthermore, variations can change the positions of genomic elements of interest making the interpretation of the results even more difficult. Despite a number of efficient and feature-rich libraries and tools have been developed for sequence-driven algorithm implementation, the above-mentioned tasks are usually left to the user. The complexity of these tasks is a severe limitation to an extensive application of algorithms implemented in this way. Several useful bioinformatics workflow tools like Taverna (Oinn *et al*., 2004), Gaggle (Shannon *et al*., 2006) and Galaxy (Blankenberg *et al*., 2010; Goecks *et al*., 2010), have been developed to automatize, standardize and speed up analyses and visualizations integrating different programs and data sources. These workflow management tools can be used to partially get around the above-mentioned limitations, but still they do not resolve the inefficiency of the sequence-driven software model, particularly in the frequent case of repeated runs on similar sequences. Moreover, there is an additional cost in terms of data formats conversion, I/O operations and script interpretation. On the other hand, most successful algorithms are employed to analyse entire genomes: results are collected in databases along with functional annotations and experimental data like in the UCSC Genome browser (Fujita *et al*., 2010) and in Ensembl (Hubbard *et al*., 2009). Genome browsers exploit these databases to give a useful, general and extensive picture of a genomic region features. Nevertheless, for any quantitative analysis, users have to query the underlying databases to fetch the data they need: a comparable situation to the one described above with the additional limitation that pre-computed algorithm results (often obtained under fixed default parameters) cannot be used to evaluate the effect of variations.

We developed GeCo++ (Genomic computation C++ library), a C++ class library to the purpose of making easier and faster the efficient implementation of algorithms for sequence analysis when functional annotations and genomic variations need to be considered. The library is not intended as a substitute for more specialized libraries; instead it frees the programmers from the burden of keeping track of genomic annotations and variations, giving them the opportunity to use the libraries they prefer for specific fields like, for example Bio++ (Dutheil *et al*., 2006) and seqAn (Döring *et al*., 2008) for general sequence analysis or libSequence (Thornton, 2003) for population genetics.

2 IMPLEMENTATION

The library has been developed starting from the idea to represent and manage the numeric results of computational algorithms

*To whom correspondence should be addressed.

keeping them tied to annotations of genomic elements (transcripts, binding sites, conserved regions, transposable elements, etc.), to their sequences and to genomic variations. It provides the definition of a genomic element model that tightly integrates information about genomic ranges, positions, genomic variations and computed/retrieved numerical features. The core idea of the model is to refer genomic ranges and positions to a reference sequence (e.g. a chromosome assembly) and to add a set of variations (substitutions, insertions and deletions) to represent actual elements (e.g. mutated ones, individual haplotypes, and so on). Based on this model the library provides the following capabilities:

- memory efficient representation of alignments in terms of reference/variations: this can save a lot of memory especially when relatively few variations are involved;
- automatic tracking of position changes introduced by insertions/deletions;
- easy mapping of corresponding positions between different sequences; and
- automatic recalculation of numerical features only where variations make it necessary thus avoiding unnecessary recalculations in unvaried regions.

A genomic element is defined as an interval of a given reference sequence in a given strand. Positions can be referred to the reference sequence (reference positions, unsigned values) or to the Element (element positions, signed values relative to the element start along its strand). Element sites are defined as particular element positions (e.g. transcription start sites, splice sites or protein binding sites) while a connection represents a directed relation between two sites (e.g. introns, exons). Positional features are defined as properties that vary along an element. While no assumption is made on the biological meaning of sites, connections and features, the model is general enough to represent the majority of real-world genomic elements as well as their features. The GeCo++ library defines the class gElement as an implementation of this model: it allows users to instantiate objects representing genomic element which can contain sequence as well as sites, connections and features information. Element positions can be converted to reference ones and back, positions can be mapped between elements. The most important characteristic of gElement objects is that they can be instantiated as sub-intervals of another one considering the strand and the presence of genomic variations (relative to the reference). Sequence, sites, connections and features are inherited by the new object consistently with its interval, strand and variations avoiding redundant recalculation and retrieval at unaffected positions. Users can easily write algorithms to retrieve sequences or calculate features by deriving new classes from a hierarchy of retriever objects (gRetriever). This requires one single virtual member function implementation. Retrievers are then used by gElement objects that, in this way, are independent from specific algorithms for retrieval and calculation. To hold sequences, positions, connections, variations and features we defined another class, called gArray: a general purpose template array class. It provides tracking of undefined/invalid elements [not available (NA)] and memory efficient array subsetting. NA tracking is obtained through an optimized bits array class (gBitsArray). The gMatrix, gString and gSequence classes have been derived from gArray to manage matrices, character strings and DNA sequences, respectively. Given the level of abstraction and the inherent complexity, we considered an object oriented software model to be the most appropriate. The choice of ISO C++ guaranteed speed, portability and, most importantly for users, the access to a great number of other efficient and specialized computational biology libraries.

3 EXAMPLES

The tutorial available at http://bioinformatics.emedea.it/geco contains a detailed description of the library usage and features. Many specific code examples are reported to illustrate the capabilities of the library as well as their usage. A fully commented example is also reported for an application that takes as input a refSeq ID and the name of a file containing haplotype information and calculates haplotype specific sequence features. Furthermore, we provide comparison between this application and an equivalent R/bioconductor script.

4 DISCUSSION

New cost-effective high-throughput sequencing and array techniques are now able to generate huge amounts of information on DNA, RNA as well as protein–DNA and protein–RNA interactions. Systems biology approaches can integrate these information at a genomic level describing and studying complex regulatory networks. Prediction and interpretation of the functional meaning of individual genomic variations could be inferred by studying the modifications and rewiring events they produce in these networks. Despite this being considered as one of the most promising challenges (Gonzalez-Angulo *et al*., 2010; Peng *et al*., 2009), a gap remains to be filled between sequence analysis algorithms, genomic annotations and variations; the lack of tools integrating these three levels of analysis is going to become more evident as high-throughput techniques on one side and system biology on the other start to converge to translational research applications. Beside providing C++ programmers with features usually present in interpreted languages, our library introduces a genomic element model that tightly and time/space efficiently integrates computed features, position annotations and variations, allowing for feature rich ‘in memory’ representations of what usually is provided by genome browsers. The most important difference is that in our case this representation is able to dynamically keep track of genomic variations. To our knowledge this is an approach that has not been implemented in any other package yet, irrespectively by the language used.

As a first attempt to fill this gap, the GeCo++ library makes the development of complex and efficient applications straightforward, not bounded to specific data sources or computational algorithms and that can easily evaluate the effect of sequence variations on genomic functional elements.

Conflict of Interest: none declared.

REFERENCES

Blankenberg,D. *et al*. (2010) Galaxy: a web-based genome analysis tool for experimentalists. *Curr. Protoc. Mol. Biol*., **Chapter 19**, Unit 19.10.1-21.

Döring,A. *et al*. (2008) SeqAn an efficient, generic C++ library for sequence analysis. *BMC Bioinformatics*, **9**, 11.

Dutheil,J. *et al*. (2006) Bio++: a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics. *BMC Bioinformatics*, **7**, 188.

Fujita,P.A. *et al.* (2010) The UCSC Genome Browser database: update 2011. *Nucleic Acids Res*., **39**, D876–882.

Gentleman,R.C. *et al*. (2004) Bioconductor: open software development for computational biology and Bioinformatics. *Genome Biol.*, **5**, R80.

Goecks,J. *et al*. (2010) Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*., **11**, R86.

Gonzalez-Angulo,A.M. *et al*. (2010) Future of personalized medicine in oncology: a systems biology approach. *J. Clin. Oncol*., **28**, 2777–2783.

Hubbard,T.J.P. *et al.* (2009) Ensembl 2009. *Nucleic Acids Res*., **37**, D690–D697.

Oinn,T. *et al.* (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, **20**, 3045–3054.

Peng,X. *et al*. (2009) Virus-host interactions: from systems biology to translational research. *Curr. Opin. Microbiol*., **12**, 432–438.

Shannon,P.T. *et al*. (2006) The Gaggle: an open-source software system for integrating bioinformatics software and data sources. *BMC Bioinformatics*, **7**, 176.

Stajich *et al*. (2002) The Bioperl toolkit: Perl modules for the life sciences. *Genome Res.*, **12**, 1611–1168.

Thornton,K. (2003) Libsequence: a C++ class library for evolutionary genetic analysis. *Bioinformatics*, **19**, 2325–2327.