Contents lists available at ScienceDirect

Journal of Biotechnology

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

The SeqAn C++ template library for efficient sequence analysis: A resource for programmers[☆]

Knut Reinert[a,*], Temesgen Hailemariam Dadi[a], Marcel Ehrhardt[a], Hannes Hauswedell[a], Svenja Mehringer[a], René Rahn[a], Jongkyu Kim[b], Christopher Pockrandt[b], Jörg Winkler[b], Enrico Siragusa[c], Gianvito Urgese[d], David Weese[e]

[a] Algorithmic Bioinformatics, Institute for Bioinformatics, FU Berlin, Takustrasse 9, 14195 Berlin, Germany
[b] Efficient Algorithms for -Omics Data, Max Planck Institute for Molecular Genetics, Ihnestrasse 62-73, 14195 Berlin, Germany
[c] IBM Watson Research, Yorktown Heights, NY, USA
[d] Department of Control and Computer Engineering, Politecnico di Torino, Italy
[e] SAP Innovation Center, Potsdam, Germany

ARTICLE INFO

Keywords:
NGS analysis
Software libraries
C++
Data structures

ABSTRACT

Background: The use of novel algorithmic techniques is pivotal to many important problems in life science. For example the sequencing of the human genome (Venter et al., 2001) would not have been possible without advanced assembly algorithms and the development of practical BWT based read mappers have been instrumental for NGS analysis. However, owing to the high speed of technological progress and the urgent need for bioinformatics tools, there was a widening gap between state-of-the-art algorithmic techniques and the actual algorithmic components of tools that are in widespread use. We previously addressed this by introducing the SeqAn library of efficient data types and algorithms in 2008 (Döring et al., 2008).
Results: The SeqAn library has matured considerably since its first publication 9 years ago. In this article we review its status as an established resource for programmers in the field of sequence analysis and its contributions to many analysis tools.
Conclusions: We anticipate that SeqAn will continue to be a valuable resource, especially since it started to actively support various hardware acceleration techniques in a systematic manner.

1. Introduction

The analysis of biological sequences is at the core of computational biology. Advances in biotechnology have driven the development. of many successful algorithms (e.g., Myers' bit-vector search algorithm Myers, 1999, BLAST Altschul et al., 1990) and data structures (e.g., suffix arrays Abouelhoda et al., 2002, *q*-gram based string indices, FM-indices Ferragina and Manzini, 2000) over the last 20 years. The assemblies of large eukaryotic genomes like *Drosophila melanogaster* (Adams et al., 2000), human (Venter et al., 2001), and mouse (Mural et al., 2002) are prime examples where algorithm research was successfully applied to advance biological knowledge. However, with entire genomes at hand, large scale analysis algorithms that require considerable computing resources are becoming increasingly important and so do their implementations as efficient tools. Although these tools use slightly different algorithms, nearly all of them require some basic algorithmic components, like string indices, string searches, or alignments.

It is non-trivial to program efficient implementations of these components, especially if vectorization and multi-threading becomes more and more mandatory. This leads in practice often to the use of suboptimal data types and *ad-hoc* algorithms or the analysis is conducted by stringing together standalone tools. Both approaches may be suitable at times, but it would clearly be much more desirable to use an integrated library of state-of-the-art components that use modern hardware. Those components can be combined in various ways, either to develop new applications or to compare alternative implementations. Also, relying on a software library cuts down development time and ensures correctness and compatibility.

We previously addressed this by introducing the SeqAn library of efficient data types and algorithms in 2008 (Döring et al., 2008). Since then, SeqAn has matured considerably and is currently being supported

[☆] SeqAn is supported in the CIBI as part of the de.NBI network.
[*] Corresponding author.
E-mail address: knut.reinert@fu-berlin.de (K. Reinert).

http://dx.doi.org/10.1016/j.jbiotec.2017.07.017
Received 26 February 2017; Received in revised form 17 July 2017; Accepted 19 July 2017
Available online 06 September 2017

by the de.NBI network for bioinformatics infrastructure as part of the CIBI (Center for Integrative BIoinformatics). In this article we review its status as an established resource for programmers in the field of sequence analysis and its contributions to many analysis tools.

2. Material and methods

The SeqAn library was first published in 2008 and designed with certain goals in mind, namely to promote (1) *high performance* of the provided algorithmic components, (2) *simplicity* and usability, (3) *generality* of data types and core algorithms, (4) the definition of special *refinements* of generic classes of algorithms, (5) the *extensibility* of the library and easy *integration* with other libraries.

2.1. Design and content of SeqAn

The SeqAn library currently consists of around 175,000 lines of code which are split into 47 modules of varying sizes. SeqAn was implemented in C++, a multi-paradigm high level programming language, and is at the time of writing this article available in version 2.3. The key concepts that are used in SeqAn are algorithm-oriented programming (Musser and Stepanov, 1994), *generic* programming using *templates* and *template meta-programming*, and *partial template specialization* as a form of polymorphism. We also make use of *tag dispatching* to select the most efficient algorithm for a certain task at compile time. The reasoning and implementation details are covered in Döring et al. (2008), and have remained largely unchanged for SeqAn 2.x.

The core of the SeqAn project is the SeqAn library, but under the umbrella of the project there are also support utilities including a custom documentation system modeled on doxygen (van Heesch, 2008), a custom test system and a selection of applications that rely heavily on the library (more on these later). For practical reasons, we will only focus on a subset of the modules and give the reader an overview of the extensive content of SeqAn.

align, align_*, score. The align modules offer all functionality to perform the many different types of pairwise alignments that are useful in sequence analysis. For example they provide two data structures to efficiently incorporate gaps into the underlying sequence: `ArrayGaps` and `AnchorGaps`. While `ArrayGaps` are efficient for linear access patterns, `AnchorGaps` have a better runtime complexity for random access patterns, because they can employ binary search for the lookup. In Listing 1 we show how to compute a standard pairwise global alignment using `ArrayGaps` as the gap structure of choice.

Our sequence library also implements many adaptions of the original dynamic programming algorithms among the default pairwise alignments. For example, we have implementations for split break point calculation (Emde et al., 2012a,b; Holtgrewe et al., 2015a,b), seed extensions using x-drop with affine gap costs (Hauswedell et al., 2014a,b), banded chain alignment, a new gap model called Dynamic Gap Selector (Urgese et al., 2014) and many more.

In addition to the sequential one-to-one interface we also implemented a vectorized many-to-many pairwise alignment interface, which uses extended instruction sets of modern CPU architectures to speed-up the computation of many pairwise sequence alignments (see Section 3.1).

arg_parse. SeqAn offers a generic argument parser and option handler that supports convenient access to the command line parameters. The *arg_parse* module makes it easy to define, restrict and retrieve any command line input while remaining flexible for individual needs. A well formatted help page is automatically generated, providing a clear and common interface to all SeqAn based applications.

Since SeqAn version 2.0.0, the argument parser is able to export interface descriptor files in an XML based format, that allow seamless integration of all applications into workflow systems like KNIME (Berthold et al., 2007) or Galaxy (Afgan et al., 2016) (see Section 2.2). Recently, we also introduced a mechanism to inform our users and developers whenever the library or one of our registered applications can be updated to a newer version (see Section 2.2).

blast. The newly introduced blast module offers e-value statistics based on official NCBI source code (Camacho et al., 2009), as well as support for the most commonly used blast output formats, including tab-separated output and full report.

index. SeqAn also provides an indexing module offering numerous string indices for arbitrary alphabets such as (enhanced) suffix arrays, FM indices, lazy suffix trees or q-gram indices. This includes fast and practical implementations of unidirectional and bidirectional FM indices, also the first and currently only openly available implementation of a constant-time bidirectional FM index (Pockrandt et al., 2017). For more details, see the results section. Listing 2 demonstrates an offline-search with the FM-Index in SeqAn.

journaled_string_tree. The journaled string tree is a data structure suitable for streaming over a set of referentially compressed sequences and is applied in the field of compressive genomics (Marschall et al., 2016). It works generically for all algorithms, whose state depends on a sequence context, i.e. a finite number of contiguous characters (Rahn et al., 2014).

modifier. In addition to the many container types that SeqAn provides, it also offers so called modifiers which are what is now commonly referred to as a view in C++ terminology – a light-weight data structure that behaves like a container, but does not store the data. Instead it performs a transformation "on-the-fly" during access. An intuitive biological example is the reverse complement modifier that appears like the reverse complement of a DNA sequence without copying the actual string data.

stream. The stream module is the basis for all other I/O modules. It provides a stream abstraction on top of STL stream buffers with built-in support for transparent (de-)compression with GZIP and BZIP2 (Gailly and Adler, 2003; Seward, 1996) if the corresponding libraries are present on the system. Consequently, all I/O done through SeqAn interfaces has automatic support for reading/writing compressed versions of the corresponding files. This module has been completely rewritten for SeqAn 2 and now profits from simpler interfaces, better performance and improved error handling via exceptions.

seq_io, gff_io, bam_io, rna_io, vcf_io. Sequence I/O is part of most bioinformatics applications and SeqAn supports many popular formats like fasta, fastq (Cock et al., 2010) and genbank, GFF, VCF SAM, BAM and certain RNA formats. The bam_io module contains a full implementation of the SAM and BAM formats, independent of SAMTOOLS and HTSLIB (Li et al., 2009). Listing 3 demonstrates a simple bam-to-sam converter implemented in SeqAn. The listing reads in a BAM file and displays its content on the standard output. A prominent change in SeqAn 2 is the addition of a highly parallel decompressor for BGZF so that our performance in reading and parsing BAM files is significantly higher than in any other implementation. Comparisons with other implementations are available in the results Section 3.3. In SeqAn 2 SAM and BAM files can also be treated as regular sequence input files. This is useful in case the original sequence files are no longer available or if BAM is preferred as storage for its compression. The VCF module contains functionality to read and write files in VCF format (Danecek et al., 2011). We also plan to add BCF format in the near future.

Files for RNA may contain structural information of a sequence or alignment. This module supports I/O for common RNA structure formats, as there are Connect, Dot-bracket, Vienna, Bpseq, and extended Bpseq for sequences and Stockholm for alignments. We created the extension of Bpseq to offer a format that contains multiple base pair probabilities of sequence interactions, which can be gained from experiments or statistical methods.

sequence, sequence_journaled. This module contains the wide range of generic containers, including *std::vector*-like strings, fixed-size arrays, external strings and bit-compressed strings. External strings behave just like regular containers but use the hard disk as storage and keep only a small fraction of the sequence in memory. This is e.g. useful if the