LASSAP, a LArge Scale Sequence compArison Package

Eric Glémet and Jean-Jacques Codani

Abstract

Motivation: This paper presents LASSAP, a new software package for sequence comparison. LASSAP is a programmable, high-performance system designed to raise current limitations of sequence comparison programs in order to fit the needs of large-scale analysis. LASSAP provides an API (Application Programming Interface) allowing the integration of any generic pairwise-based algorithm.

Results: Whatever pairwise algorithm is used in LASSAP, it shares with all other algorithms numerous enhancements such as: (i) intra- and inter-databank comparisons; (ii) computational requests (selections and computations are achieved on the fly); (iii) frame translations on queries and databanks; (iv) structured results allowing easy and powerful post-analysis; (v) performance improvements by parallelization and the driving of specialized hardware. LASSAP currently implements all major sequence comparison algorithms (Fasta, Blast, Smith/Waterman), and other string matching and pattern matching algorithms. LASSAP is both an integrated software for end-users and a framework allowing the integration and the combination of new algorithms. LASSAP is used in different projects such as the building of PRODOM, the exhaustive comparison of yeast sequences, and the subfragments matching problem of TREMBL.

Availability: Binaries for Unix and documentations are available at the following address: http://www-rocq.inria.fr/rgene

Contact: E-mail: Eric.Glemet@inria.fr, Jean-Jacques.Codani@inria.fr

Introduction

Large-scale sequencing projects and daily worldwide production of sequences lead to a considerable amount of data. This data flow tends to become incompatible with the existing tools for sequence comparison.

Let us consider the three major programs covering the basic needs: Blast (Altschul et al., 1990), Fasta (Pearson and Lipman, 1988) and dynamic programming—best fit/profile-search (Smith and Waterman, 1981; Gribskov et al., 1990). Each of them has a specific behavior and often one has to deal with all the others at the same time in order to obtain better results. Each of them takes as input sequence data and scoring matrices stored in different formats, and produces results in different formats. This implies many conversions among the different formats and many duplications of databanks plus several filtering programs to analyze each kind of result. Consequently, it is not easy to compare results coming from different algorithms and, more generally, analysis of results is difficult.

Moreover, current programs allow one to compare only one sequence against a databank. This is not practical for needs such as sequencing centers and databank analysis that require intra- or inter-bank comparisons. Furthermore, if one wants to compare one or many sequences against a subset of a databank, the subset will be obtained using ad hoc query systems like SRS (Etzold and Argos, 1993) or ACNUC (Gouy et al., 1985), or using DataBases Management Systems (DBMS). It leads to a three-step approach: extracting data, reformatting it for the chosen algorithm, then computing. Selection and computation on the fly would be better.

Last, intensive comparisons become more and more frequent. Programs should run on various architectures (shared memory or distributed memory) composed of tens or more processors to accomplish tasks in a reasonable time.

In order to raise the above limitations, we have designed and implemented a software package called LASSAP, which stands for LArge Scale Sequence compArison Package.

LASSAP is made of a kernel and a set of algorithms. The kernel provides a simple way to add any generic pairwise-based algorithm through an API (Application Programming Interface). Thus, LASSAP is a framework allowing the integration of new algorithms (section on API). The kernel provides numerous services shared by all algorithms. As a result, LASSAP is an integrated software for end-users (section on Kernel modules). LASSAP currently implements all major sequence comparison algorithms: Fasta, Blast, Smith/Waterman (sw) and other string matching and pattern matching algorithms (section on Implemented algorithms). LASSAP is already involved in different projects (see Results).
Concepts

Foundations

A study of the overall process of sequence comparison leads us to the following conclusion: whatever algorithm is used, the process can be split into four independent treatments:

1. **Input management**: this includes command line parsing, scoring matrices and databank handling which can also be decomposed into three states: loading data, selecting subsets, and translating in frames (in the case of nucleic sequences).

2. **Computation**: as a first approximation, a computation between two sequences is formed from a pairwise sequence comparison algorithm, which includes the initialization of parameters of the algorithm, the algorithm itself, and the appropriate post-treatments.

3. **Control-flow**: this controls the global computation (sequence against bank, bank against bank, etc.) by looping on all pairwise comparisons induced by data.

4. **Output management**: this includes filtering and storing of results. One has to note that every kind of algorithm computing alignments can produce the same kind (same structure in a C sense) of results.

It is the reason why LASSAP has been designed in a modular way.

Overview

LASSAP is composed of modules. Each module is in charge of the above treatments (modules 1–4, see Figure 1). The independence of all treatments is the key-point of LASSAP.

An algorithm in LASSAP interacts with the kernel (modules 1, 3, 4), and any enhancements of these modules benefit the algorithm. The kernel provides numerous services such as: intra- and inter-databank comparisons, selections on databanks, frame translations and performance improvements (comparisons can be computed in parallel).

From a programmer point of view, module 2 is user programmable through an API. Consequently, LASSAP allows one to choose an algorithm as a parameter of the sequence comparison process. The chosen method is a parameter of the shell command line (\(-M\) flag).

For example, the following command line (Figure 2) searches for sw similarities (\(-M\ sw\)) between my sequence (myseq) and the entries of primates which contain the word promoter in the annotation part and the pattern TATA in the sequence part ((\(-h\ promoter\) & \(-s\ TATA\)): this is a LASSAP selection). The 50 best scores are reported (\(-keep\ 50\)) and full alignments are displayed (\(-a\) flag). The matrix used is the Blast matrix NUC.4.4.

Note that the same command line can run Fasta or any other algorithm by changing only the method name (here sw). Parameters of a method are specified with the \(-P\) flag.

LASSAP can perform more complex computation with the help of services provided by modules 1, 3 and 4. For example, the following command line (Figure 3) runs the Blast algorithm, with parameter S set to 50 (\(-P\ \{ -S\ 50\}) between two databanks. The first databank is the result of the translation (on the whole sequences) on the three positive frames (\(-f\ top\) of the set of sequences from the primate
LASSAP sequence comparison package

% lspcalc
-Μ sw
-μν NUC.4.4
-db /embl/pri {(-h promoter) & (-a TATA)}
-db myseq
-keep 50
-a

Fig. 2. A LASSAP Smith/Waterman scanning on a subset of a nucleic databank.

section of EMBL (/embl/pri). The second one is the result of the SRS query [sprot-def:(adenylate & cyclase)] (sequences from SWISSPROT containing adenylate and cyclase in the field ‘definitions’ of an entry). The matrix used is BLOSUM62. Results are stored in structured format in the file res (-o res). The computation runs on eight workstations (-pvm flag for message passing implementation).

These examples show some capabilities of LASSAP, which can imply a quite complicated command line, parsed by the command line handler. A Java graphical interface will simplify interaction.

Kernel modules
This section details which services LASSAP provides to any pairwise algorithm.

Databanks and scoring matrices
With the help of the integration of some C functions of readseq program (Gilbert, 1993), LASSAP can read well-known sequence formats (Genbank, EMBL, etc.). Nevertheless, we need to take into account the following. (i) Some algorithms, such as blast, require more information on databanks for statistical reasons (size of databanks, etc.). This a priori information needs to be known before computation. (ii) For performance considerations, especially for parallelism, we need to access sequences efficiently. A simple index mechanism is necessary. The Blast format obeys these rules, but it has the drawbacks of not being human readable (consequently grep incompatible), and to be not symmetrical between proteic and nucleic sequences. The Fasta format is human readable, but not indexed. The LASSAP format is a mix of the two formats: it is indexed and readable. It looks like the Fasta format (Fasta can operate '/.lspcalc
-Μ blast -P [-S 50]
-mp BLOSUM62
-db -f top /embl/pri
-ds [sprot-def:(adenylate & cyclase)]
-o res
-pvm 8

Fig. 3. A LASSAP Blast all vs. all computation

on a LASSAP sequence file), e.g. has a header line beginning with >, but implements a sequence as a single Unix line. This allows grep searches and efficient memory management of databanks by using Unix mmap system call. An entry in LASSAP format looks like:

>KC2B_YEAST P43639 #DE CASEIN KINASE II ...
MSEQFEVDYSTRGSDDEDSGAYDEWIPSFC ... ILTK

The bank21sp program, part of the LASSAP package, translates well-known formats into this format. Command line options allow one to specify which fields (AC, ID, DE, etc.) must be kept in the header line. This information can be reused by the LASSAP selection mechanism (see the next subsection). This format is recommended for efficiency, for both parallelism and selections. Finally, a LASSAP databank can be a virtual set of databanks, allowing several databanks, physically distinct, to be treated as one.

For the scoring matrices part, LASSAP reads Blast matrix files.

Complex queries
LASSAP’s input is made of one or two databanks. In the latter case, the second databank is considered as the query. A ‘sequence against bank’ scanning is the case where the second databank holds one sequence. When only one databank is specified, the databank is computed against itself. LASSAP allows ‘complex queries’ in the following sense: a databank (or query) can be a whole databank or a subset of a databank, obtained through a selection mechanism. On both of them, frame (or phase) translations can be applied.

Bank against bank. It is possible to compare a databank against itself or against another databank. This feature is integrated: it avoids launching a shell-script running multiple ‘sequence against bank’ programs and avoids dealing with numerous result files. Moreover, it allows an efficient parallel implementation (cf. the section on performance). Above all, this feature combined with structured results (see below) is the only way to perform complex post-analysis.

This is useful for many problems such as building specific databanks [e.g. PRODOM (Sonnhammer and Kahn, 1994)], search for families, databank redundancy studies and analysis of daily sequences produced by high-throughput sequencing centers.

Selections. It is also very useful to select a subset of a databank. Two kinds of selections are available: built-in (a_) or external (b_).

a_ A selection can be provided by LASSAP itself. A LASSAP selection is composed of atomic selections combined with logical operators (AND, OR, etc.). An atomic selection can be:
This allows, for a given algorithm, all possibilities among the chunk. For example, in the 'sequence against bank (P control-flow splits the iteration space into chunks. One processor is in charge of the sequential computation of a chunk. For example, in the 'sequence against bank (P sequences)' case, with N processors, the size of the chunk can be P/N. The shape and the size of the chunks depend on the complexity of the algorithm, the input data, the number of processors and the architecture of the parallel machine. LASSAP implements different load balancing strategies. The most used strategy is a dynamic one, which implements a self-scheduling algorithm. This work benefits from our former work in the physical mapping of the human genome conducted by Genethon laboratories (Bellanné-Chantelot et al., 1992; Lacroix and Codani, 1992).

For a given pairwise algorithm, this module provides automatic spreading of the computation on shared memory and message passing architectures. Different implementations exist, including servers, networks of workstations and parallel supercomputers. Speed-ups depend of course on many factors: speed of the algorithm, size of the problem, number of processors, etc. Most of the time (databank scanning, large-scale comparisons, etc.) near perfect speed-ups have been observed with up to tens of processors. This is true for nearly all algorithms of LASSAP. Details on that topic can be found in Codani and Glemet (1995), related works in Miller et al. (1991), Deshpande et al. (1991) and Julich (1995), and a complete review in Yap et al. (1996).

An optimized implementation of sw has been devised by A.Wozniak for LASSAP (Wozniak, 1997). Performance reaches nearly 200 million matrix cells per second on a 12 processor UltraSparc Sun server.

Specialized hardware. Thanks to BIOCCELERATOR, API, LASSAP can drive a BIOCCELERATOR board. This feature is transparent from the command line. This allows complex queries with dynamic programming without performance penalty.

Lastly, LASSAP can handle both cases. For example, several BIOCCELERATOR boards can be driven at the same time with the MIMD implementation.

Structured results

Alignments in LASSAP can be displayed in ASCII form, in a format close to the usual ones (Blast, Fasta, etc.). They can also be stored as a structured binary file, and then post-analyzed.

As already mentioned in the section on kernel modules, there are few differences between results among different sequence comparison algorithms. The way LASSAP treats an alignment is as follows: the two sequence identifiers, a score, the beginning and end matching positions, the frame number, a statistical significance (probability, expect value, Z-score, etc.). Extra information, such as gap positions, can be kept—during computation or after—if needed.

The advantages of structured results are that multiple post-analyses on results can be carried out without a new run. For example, one can perform: various simple post-analyses (sort...
LASSAP sequence comparison package

by scores, by probabilities, by lengths of alignments, etc.); complex post-analysis—bank against bank, connected components, cluster of related sequences, etc.; a multiple alignment with a first pass based on all pairwise comparisons (e.g., clustal, pileup); analysis of results coming from different algorithms.

It is, therefore, possible to retrieve alignments with constraints. As an example, one can extract alignments such as: their percentage of identities is in the range \([x, y]\), their length is below \(l\) and they contain a common PROSITE pattern. Note that this kind of request applies to most comparison algorithms and it avoids writing \textit{ad hoc} output parsers, such as BLA (Tatusov and Koonin, 1994).

**Implemented algorithms**

Algorithms currently implemented are:

- **Blast** (Altschul et al., 1990). LASSAP blast uses the same automaton (DFA) and the same extension procedure for nucleic and proteinic sequences. Moreover, nucleic sequences are not compressed, so ambiguities are handled directly. Thus, in nucleic mode, results differ slightly from Blast 1.4 and runs are sometimes slower.
- **Fasta** (Pearson and Lipman, 1988). The version is 2.0 compatible.
- **Dynamic programming with local—sw** (Smith and Waterman, 1981)—and global—nw (Needleman and Wunsch, 1970)—similarity searches. Statistical measurements (Z-score, probabilities), using Monte-Carlo techniques, can be reported (Comet et al., 1997).
- **String matching** with errors (Hunt and Szymanski, 1977; Landau and Vishkin, 1986) or without errors (Boyer and Moore, 1977). These algorithms offer other kinds of results than the usual sequence comparison ones. For example, the Landau–Vishkin algorithm can find subfragments with a given percentage of errors, which can be insertions, deletions or substitutions. The percentage can be a function of the length of the smallest sequence.
- **Pattern matching** with errors (Baeza-Yates and Gonnet, 1989; Wu and Manber, 1991) or without errors (‘a la regexp’) which allows PROSITE pattern searching.

This list is not exhaustive. Other algorithms, which combine the algorithms above, are implemented (cf. the paragraph on combining algorithms). The way an algorithm is plugged into LASSAP is described in the next section.

**API**

The API is the fundamental bridge between LASSAP internal structures and the algorithms to be linked with. This section highlights the principles of the API. Two examples are given: the first one shows how it is convenient to add a simple algorithm; the second highlights the capabilities of refining algorithms by combining existing ones.

**Architecture**

As shown in Figure 1, LASSAP is built around a kernel which is composed of three modules: input management, control flow and output management. Some of the services previously evoked (loading databanks, selections, etc.) are implemented as C functions. The kernel, through the command line module, loads various input data (databanks, etc.) and determines which algorithm should run. The name \(A\) of the algorithm is specified by the \(-M A\) flag.

An algorithm is a set of functions and data structures, called \(\phi\). The programmer defines \(\phi\) and tells the kernel that \(\phi\) is named \(A\) (a global array \(<A, 0>\) is kept to assume the correspondence). The definition of \(\phi\) must obey some rules of the API, which is described in the next section.

**Integration of a pairwise algorithm**

In order to integrate a new algorithm, one needs to connect it to specific data structures and functions to LASSAP API, i.e.:

- Specify the input parameters (name, initial value, etc.) which will be parsed by the command line handler (-P flag).
- Define variables local to the algorithm ( automata, hash-tables, buffers, etc.). These variables can be referenced through a pointer to a unique structure (env variable described below) provided by LASSAP. This ensures privacy for the shared memory implementation.
- Define functions called by the control-flow module. Six functions are necessary: \(A\_check\_args\) post-processes input parameters, \(A\_init\_comp\) initializes the computation, \(A\_init\_query\) performs some analysis on the query (building automata, etc.), \(A\_pairwise\_compute\) implements the pairwise algorithm itself, \(A\_end\_query\) and \(A\_end\_comp\) deallocate buffers if any. These functions are called user functions.

To illustrate how user functions are called, let us focus on the main loop handled by the control-flow module in the case of a bank (QueryBank) against another bank (TargetBank). Each user function called (env->F) is surrounded by its equivalent LASSAP function which manages pre- and/or post-user treatments. For example, \texttt{Lsp\_end\_query} stores results or sends them in the message passing parallel implementation.

```c
foreach i in QueryBank {
    Lsp\_init\_query (env, i);
    env->init\_query (env);
    foreach j in TargetBank {
```
E. Gémets and J.-J. Codani

Each user function takes as argument a pointer to a LASSAP environment (env), which is dynamically created by the command line handler. It holds various data (databanks, scoring matrices, etc.), pointers to the user functions, and pointers to variables local to the algorithm.

Example

We implement here a test for databank redundancy with the following pairwise algorithm which determines if a sequence is included in another one. This 'scholar' version uses the C library function (strstr).

```c
nrdb_pairwise_compute (LassapEnv * env) {
    Seq * Q = Lsp_get_seq (env, QUERY);
    Seq * T = Lsp_get_seq (env, DATABASE);
    char * q = Q->seq, * t = T->seq;
    int lq = Q->len, lt = T->len;
    char * hq = Q->header, /* Pointers to */
            * ht = T->header; /* annotations */
    if (lq<lt) swap (q, t);
    if (strstr (t, q) ! =NULL)
        Lsp_save_res(env);
}
```

This function uses some basic LASSAP functions to retrieve pairs of sequences (Lsp_get_seq), and tells LASSAP to keep results (Lsp_save_res).

This example is very simple, but it is nearly all you have to describe to plug it into LASSAP. All other functions, not defined, are not called. This algorithm, pairwise described, is called by the kernel for any pair of sequences to be compared and then benefits of LASSAP modules. Indeed, the following command line:

```bash
% lspcalc -M nrdb
    -db trembl {(-h_YEAST)} -dd
```

events all pairs (i, j) of yeast sequences of TREMBL strictly included in others (i < j, -dd flag).

Note that this piece of code references annotations of sequences (hq, ht variables). This can be used for taking in account annotations. This is the way the redundancy of TREMBL (Bairoch and Apweiler, 1996) is assumed (cf. Results).

Combining algorithms

Algorithms implemented in LASSAP can share a common alignment structure. Thus, algorithms can then be considered as 'basic bricks' which can be combined to devise new algorithms. As an example, LASSAP provides a method mixing Blast and Smith/Waterman algorithms. Indeed, Blast can lead to imperfect results when overlapping HSPs (high segment pairs) highlight an obvious gap miss (Figure 4).

The following method, called blsw (Blast-Smith/Waterman), does the following on two sequences: the first time, it calls Blast. If no match occurs, no alignment is reported. Otherwise, a best local alignment is performed (Smith/Waterman, sw). Then all Blast HSPs included in the sw alignment are discarded, the other ones are kept, reflecting distant similarities.

```c
blsw_pairwise_compute (LassapEnv * env) {
    blast_pairwise_compute (env);
    if (get_cur_res (env) !=NULL) {
        sw_pairwise_compute (env);
        merge_blast_inclusion (env);
    }
}
```

This example can of course be discussed and refined, but this strategy introduces how LASSAP can combine Blast selectivity and quickness with sw sensitivity.

Results

LASSAP is used for:

Building of PRODOM (Sonnhammer and Kahn, 1994). SWISS-PROT (50,000 entries) is compared against itself (all vs. all) using the blast algorithm. It can run on a network of workstations or any parallel supercomputer. On 30 processors, this computation takes about 24 h. Exhaustive comparison of yeast sequences (6000 entries). An all vs. all Smith/Waterman run is done and when the
Smith/Waterman score is greater than a given threshold, a Z-score is computed (Comet et al., 1997). This is achieved in LASSAP by driving a BIOCCELERATOR board and a 32 nodes IBM SP2 parallel machine.

Subfragment matching of the 122 000 entries of TREMBL (Bairoch and Apweiler, 1996). Here, again, an exhaustive comparison of TREMBL is done with the Boyer–Moore algorithm. This can be done in a few hours on a single workstation.

For all these projects, many post-analyses are performed which typically build sets of related sequences.

Conclusion

LASSAP is an integrated framework which offers extensibility, speed and accuracy for sequence comparison. In addition to basic databank scanning needs, it is targeted to: scientific projects which require intensive comparisons (clustering of databanks, etc.); sequencing centers with high throughput; centers in charge of the production of international databanks (redundancy, etc.); centers of services willing to provide high-end performances.

Mid-term developments of LASSAP include multiple alignment, structured data handling (OO-DBMS or RDBMS) and a graphical interface.

An experimental LASSAP server will be set up on an IBM SP2 parallel machine at CNUSC (Montpellier), in collaboration with the INFOBIOGEN center (http://www.infobiogen.fr).

Acknowledgements

Thanks to W.R. Pearson, D. Kahn and A. Bairoch for very helpful discussions, and to R. Baeza-Yates, J.L. Risler, P. Dessen, G. Vaysseix and J.M. Hulot for their useful comments. This research is funded by GREG under contract 1410454 and Ministry of Research under contract 9513054.

References


Received on June 18, 1996; revised on September 18, 1996; accepted on October 15, 1996.