BMC Bioinformatics

BioMed Central

Methodology article

tacg – a *grep* for DNA

Harry J Mangalam

Address: tacg Informatics, 1 Whistler Ct, Irvine, CA, 92612, USA

E-mail: hjm@tacgi.com

Published: 6 March 2002

BMC Bioinformatics 2002, **3**:8

Received: 14 November 2001

Accepted: 6 March 2002

This article is available from: http://www.biomedcentral.com/1471-2105/3/8

Abstract

Background: Pattern matching is the core of bioinformatics; it is used in database searching, restriction enzyme mapping, and finding open reading frames. It is done repeatedly over increasingly long sequences, thus codes must be efficient and insensitive to sequence length. Such patterns of interest include simple motifs with IUPAC degeneracies, regular expressions, patterns allowing mismatches, and probability matrices.

Results: I describe a small application which allows searching for all the above pattern types individually, which further allows these atomic motifs to be assembled into logical rules for more sophisticated analysis.

Conclusion: tacg is small, portable, faster and more capable than most alternatives, relatively easy to modify, and freely available in source code.

Background

String searching is a well-developed field of computer science, with highly optimized algorithms implemented in a variety of applications to efficiently search large collections of text [12,15]. Many of these algorithms are intuitive; some of them are beautiful, the best of them are quite extraordinarily efficient [1,3,6]. Many of these algorithms have been applied to the field of searching genomic sequences and associated databases, either for exact [2,17,18] or probablistic [11,4] matches, but it is striking that many useful algorithms in combinatorial pattern matching have not made their way into tools designed explicitly for bioinformaticists.

Most of the tools that have been developed for pattern matching in molecular biology have been prototyped on Unix systems but there has been a perplexing deficit of free bioinformatics tools (comparable to the GNU text utilities [9]) for this platform. Recently, however, the EMBOSS suite of about 100 tools [20] has approached the commercial GCG application suite for breadth of utility. There are other good tools (ie. SEALS [24]) that can be coerced into providing searching capability although often at the cost of non-standard interfaces, large installation or maintenance overhead, or that provide considerable utility but at the cost of building the application yourself, such as the Bio* toolkits [5].

A useful utility would be a *grep*[13] for DNA, a command line program requiring a short learning curve and few resources, but providing sophisticated search abilities as well as being easily extensible. tacg was developed to provide this functionality. For example, tacg -pName, pattern, err# < input_file provides a table of all the sites in the sequence(s) contained in *input_file* where the pattern named *Name* matched with *err#* mismatches. It is relatively compact; ~10,000 lines of ANSI C code with another 20,000 lines of included library code, mostly from James Knight's Seqio package [16].

While tacg provides a number of capabilities, it is particularly good at searching for multiple matches in DNA. It was originally designed for restriction enzyme mapping and can search for short degenerate patterns in degenerate DNA as well as searching with errors (fuzzy searching). It can generate lists of fragments (if the patterns are assumed to cleave the DNA) and produce character-based outputs, such as GCG-like ladder maps, gel simulations, full linear maps with optional co-translation in 1, 3, or 6 frames. It has support for some Amplified Fragment Length Polymorphism (AFLP [23]) analysis, and much of its internal data can be exported in tabular format (including GNUPLOT[25]) for external plotting or analysis. It can also produce circular summary maps (in Adobe Postscript or Portable Document Format) decorated with ORFs from any frame. Most standard sequence formats are supported transparently via *Seqio* so reformatting sequence files before use is not required; it can also process multi-sequence files and databases. Much of this functionality is available via the Web form and matching CGI program which accompany the application (a list of such public sites, along with the source code and other information, is maintained at the project website tacg central [http://tacg.sourceforge.net]).

Implementation

Most of the fast searching algorithms (BLAST [2], FASTA[17], DNA Strider [18]) use variants of a discrete finite-state automaton (DFA) which use a *shift-add* algorithm (well described in [12]) to incrementally calculate an index of the next N-tuple characters in the input string. This index is used to point into to a large array of identically precalculated subpattern values that if non-null, indicate that this tuple is a complete or partial hit worth examining in more detail. The core algorithm is very efficient and the precalculation allows null indices to be discarded rapidly.

tacg uses this approach, with a DFA based on a tuple of 6, indexing into a lookup table of size 4096 (6^4). Populating this table with the results of a recent REBASE [21] file of 225 patterns yields 1523 non-null elements meaning that only 37% of the indices need to to be evaluated further. Of those, 802 elements indicate that the hexamer includes the entire hit, a special case for restriction enzymes, which have small recognition sites. Especially for searching for short sites, the search can proceed very rapidly. There are complications to be considered – the REBASE file mentioned has only 225 patterns to match, but populates 1523 elements; the difference in number is that many of the patterns include degeneracies which must be expanded to all possible characters.

Longer sites are pre-scanned for the most informative hexamer which is used as the target, and the extension is evaluated base-by-base, as tacg matches degeneracies appropriately in the input sequence. While it sounds egregiously slow, it works reasonably well in practice.

In terms of speed, tacg processes sequence at about 1 MB/s when searching for single patterns and about 1/4 that speed when searching for the 225 patterns in the REBASE file (all tests were done on a 200 MHz Intel Pentium Pro (256 K cache) PC running Linux). As a comparison, in processing the *E. coli* genome (4.6 MB) with 6-cutters, the EMBOSS RESTRICT program used more than 100 MB of memory and ran more than 3 minutes before I terminated it. tacg used 13 MB of RAM and finished in 6 s. On smaller sequences that RESTRICT could process, tacg was ~50 times faster than RESTRICT, and about 20–35 times faster than similar routines in the commercial GCG package (Map, MapPlot, MapSort, PlasmidMap, FindPatterns for example).

Even tacg's speed is slow relative to BLAST or FASTA, or indeed when compared to general purpose routines like the grep/egrep/agrep family [27,13], but tacg examines both strands, is case-insensitive, considers more parameters, and reallocates memory to retain internal data until the end of the run, rather than treating the input line by line and printing the results immediately to STDOUT as the grep family does. Since tacg was initially coded to support multiple pattern matching, optimizations such as the Boyer-Moore 'skip-to-end' [6] approach have been ignored, resulting in tacg's relatively sluggish performance on single patterns, but good performance on large numbers of patterns.

Most data structure sizes are dynamically allocated in tacg, which allow it to process patterns and sequence limited only by system memory or Operating System quirks. It is regularly tested with the *E. coli* genome as input and has been used to process several chromosome assemblies.

Other seach modes

tacg also provides regular expression (regex) searching for nucleic acids, using a POSIX 1003.2-compatible regex library [10] for searching. These regex functions are usually included with the Operating System libraries, but does allow you to replace those regex functions with those from a higher performance regex library if desired. It also provide nucleic acid-specific translations for regex's and takes care of escaping the required characters for the user (ie. *gy(tt|gc)nc{2,3}m* is translated to *g[ct]\(tt\|gc\).c\{2,3\}[ca]* before it is passed to the regex function. This may encourage regex infantilism, but I found it useful.

Profiles or matrices are probabilistic summaries of alignments of sequences that have been very useful in searching for other related sequences, tacg also allows searching

using TRANSFAC matrices [26], a simple matrix format that is appropriate for DNA binding motifs but which does not allow insertions or deletions to be considered. Because of the format's inherent simplicity, tacg can process the multiple matrices quite rapidly and simultaneously. For example, it can scan for 23 of the yeast entries from TRANSFAC over 140 KB in less than 20 seconds and can simultaneously search for all entries in the 1998 TRANSFAC (269 entries) over the same sequence in about 3 minutes. In all the searching that tacg performs, both strands are evaluated; in some output formats, non-palindromic matches are indicated as to strand matched.

tacg has a variety of other mundane but useful functions, including subsequence extraction, cloning functions, sequence renumbering, finding silent sites, and others.

Proximity and Rules-based searching

A particularly compelling utility of tacg is demonstrated when the short patterns it finds can be evaluated with formal logic in a sliding window of sequence to find higher level structures. An example of this that EMBOSS also supports is the *marscan* program, which searches for 2 short patterns (one 8 bp with no errors, the other 16 bp with 1 error) within 200 bases of each other, typically flanking one or more genes). This constitutes a Matrix Attachment Region, an area that is implicated in conferring a higher order regulation of transcription onto a region of DNA [14]. *marscan* is specifically dedicated to finding this combination of sites, but tacg provides a general framework for searching for combinations of multiple sites (not just two); tacg is also much faster. In fact, tacg provides 2 sets of proximity analyses; one which allows extremely precise specification for 2 motifs at a time (*proximity* option) and one which is more general and allows you to specify arbitrarily complex association rules (*rules* option). These options work on all types of patterns that tacg supports, although you cannot currently mix *types* of patterns – you cannot set up a rule that references both a regex and a matrix for example.

The *proximity* analysis allows you to specify the exact relationship between 2 motifs – whether one must be 5' of the other, whether they must be separated by greater than or less than a distance, or within a range of sequence. The *rules*-based analysis allows the use of logical AND (&), OR (|), XOR(^), and negation in a sliding window of sequence to specify a rule for evaluating whether that window meets the rule criteria. You can specify the rule from a file or on the command line (in the latter case, the pattern is written to a log for later incorporation into a rules file). These rules can be arbitrarily complex and can be broken over multiple lines for ease of composition or comprehension. Unlike the *proximity* analysis, the *rules* analysis does not allow specification of how the sub-motifs are oriented or their exact spacing. An example of such a rule is:

RuleName,((LabA:m:M&LabB:m:M)I(LabC:m:M&LabD:m:M))^(LabE:m:M), window

where **RuleName** is the label for the whole rule, and window is the sliding window over which the rule is evaluated. The logic is enforced by parentheses; otherwise it is evaluated left to right. The core symbol is **LabX:m:M**, where the **LabX** is the name of the individual sub-motif, obtained from a file of such motifs, **m** is the minimum times the sub-motif *must be* present, and **M** is the Maximum times it *can be* present. Such rules can provide quite sophisticated searching with very simple primitives.

Discussion

tacg has been under punctuated evolution for several years, with the result that the oldest features have been been fairly well-debugged and it has benefitted greatly from user feedback. It has been installed in numerous institutions around the world (typically as Web interfaces) and even incorporated into commercial software suites.

However, tacg is weak in searching for patterns with degeneracies. Embarrassingly, it generates all degenerate patterns corresponding to the core sequence then churns through all the combinations to eliminate the duplicates. The agrep algorithm [27], provides a very efficient algorithm for searching for degenerate matches (including deletions and insertions) and can be extended for searching for multiple patterns [19]. This approach is about 10 times faster than the EMBOSS *fuzznuc* program, more memory efficient, and is currently being coded into tacg. Additionally, while tacg includes a simple Open Reading Frame-finder and produces statistics on the ORFs it finds (molecular weight, amino acid frequency, pI), it does not support the same range of searching ability for protein sequences, an oversight that is also being corrected. There are a variety of other standard analyses that it does not support, but which it generates the internal data for (or easily could) – dinucleotide frequencies, hexamer frequency analyses of the type described by Van Helden [22], and especially more graphical output to synthesize and present the data it generates.

tacg is a directed pattern matching tool; it can only find explicit patterns that the user defines. However, it cannot find easily find patterns based on general matching rules, such as 'find a possible stem-loop structure with a stem of >8 bp and a loop of <20 bases' or 'find any direct repeats of >17 bases separated by 6–13 bases'. While it is possible to define arbitrarily complex strings with regular expressions (which tacg supports), it may be easier for a naive user to use another application that explicitly supports ge-

neric, rules-based matching. An example of such a program is Overbeek's *scan_for_matches*[7], a fast (compiled C), compact application that supports such rules for single (albeit complex) patterns in both DNA and protein, although the specification of the rules and decoding the output requires some dedication.

Availability

Since the source code is made available with the package, obvious errors in coding may be caught and corrected; it is obvious to anyone perusing the code that the author trained as a biologist, not as a programmer. Perhaps most importantly, tacg provides a good scaffold for others to add their own analyses and an example function is included to show how to do this. It uses the *Seqio* routines to parse most sequence formats and provide malloc'ed sequence in a simple **char** * array on which you can immediately start your own manipulations and calculations. tacg is available via dual licensing, either under the Free Software Foundation's General Public License [8] or a proprietary license, if desired.

References

1. Aho AV, Corasick MJ: **Fast pattern matching: an aid to bibliographic search.** *Communications of ACM* 1975, **18(6):**333-334
2. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: **Basic local alignment search tool.** *J Mol Biol* 1990, **215(3):**403-10
3. Baeza-Yates R, Gonnet GH: **A new approach to text searching.** *Communications of the ACM.* 1992, **35(10):**75-82
4. Baldi P, Chauvin Y: **Hidden Markov Models of the G-protein-coupled receptor family.** *J Comput Biol* 1994, **1(4):**311-36
5. Birney E, Brenner SE, Dagdigian C, Daike A, Lapp H: *The Open Bioinformatics Foundation.* 2001 [http://bioperl.org, http://biojava.org, http://biopython.org, http://biocorba.org, http://bioxml.org, http://biodas.org]
6. Boyer RS, Moore JS: **A fast string searching algorithm.** *Communications of the ACM.,* 1977, **20:**762-772
7. Dsouza M, Larsen M, Overbeek R: **Searching for patterns in genomic data.** *Trends Genet* 1997, **13(12):**497-498 [http://www-unix.mcs.anl.gov/compbio/PatScan/HTML/patscan.html]
8. Free Software Foundation: *The GNU general public license.* 1989 [http://www.gnu.org/copyleft/gpl.html]
9. Free Software Foundation: *GNU textutils – a set of text utilities.* 2000 [http://www.fsf.org/manual/textutils-2.0/textutils.html]
10. Jeffrey Friedl EF: *Mastering regular expressions : powerful techniques for Perl and other tools. A Nutshell handbook.* 1997
11. M Gribskov: **Profile analysis.** *Methods Mol Biol* 1994, **25:**247-66
12. Gusfield Dan: *Algorithms on strings, trees, and sequences : computer science and computational biology.* 1997505-523
13. Haertel M: *GNU grep-2.0. Internet Archive: Usenet archive comp. sources.reviewed* 1993, **3:**
14. Hall G Jr, Allen GC, Loer DS, Thompson WF, Spiker S: **Nuclear scaffolds and scaffold-attachment regions in higher plants.** *Proc Natl Acad Sci USA* 1991, **88(20):**9320-4
15. Hirschberg Dan, Myers Gene: *Combinatorial pattern matching : 7th Annual Symposium, CPM 96, Laguna Beach, California: proceedings. Lecture notes in computer science ; 1075. Springer-Verlag, Berlin ; New York, 1996. Symposium on Combinatorial Pattern Matching (7th : 1996 : Laguna Beach, Calif.) Dan Hirschberg, Gene Myers (eds.). Includes bibliographical references and index.* 1996
16. Knight J: *SEQIO – a package for sequence file i/o.* 1996 [http://www.cs.ucdavis.edu/gusfield/seqio.html]
17. Lipman DJ, Pearson WR: **Rapid and sensitive protein similarity searches.** *Science* 1985, **227(4693):**1435-41
18. Marck C, Strider DNA: **a C program for the fast analysis of DNA and protein sequences on the Apple Macintosh family of computers.** *Nucleic Acids Res* 1988, **16(5):**1829-36
19. Muth R, Manber U: **Approximate multiple string search.** *In Combinatorial Pattern Matching '96* 199675-86
20. Rice P, Bleasby A, Williams G, Carver T, Ison J, Mullan L, Morgan H, Awan W, Ranasinghe R, Beazley C, Martin D: *EMBOSS – The European Molecular Biology Open Software Suite.* 2002 [http://www.hgmp.mrc.ac.uk/Software/EMBOSS/]
21. Roberts RJ, Macelis D: **REBASE – restriction enzymes and methylases.** *Nucleic Acids* 2001, **29(1):**268-269 [http://rebase.neb.com/rebase/rebase.html]
22. van Helden J, Andre B, Collado-Vides J: **Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies.** *J Mol Biol* 1998, **281(5):**827-42
23. Vos P, Hogers R, Bleeker M, Reijans M, van de Lee T, Homes M, Frijters A, Pot J, Peleman J, Kuiper M, *et al*: **AFLP: a new technique for DNA fingerprinting.** *Nucleic Acids Res* 1995, **23(21):**4407-14
24. Walker DR, Koonin EV: **SEALS: a system for easy analysis of lots of sequences.** *Proc Int Conf Intell Syst Mol Biol* 1997, **5:**333-9 [Using Smart Source Parsing]
25. Williams T, Kelley C, Lang R, Kotz D, Campbell J, Elber G, Woo A: **gnuplot – a command-driven interactive function plotting program.** 2002 [http://www.gnuplot.info/]
26. Wingender E, Chen X, Fricke E, Geffers R, Hehl R, Liebich I, Krull M, Matys V, Michael H, Ohnhauser R, Pruss M, Schacherer F, Thiele S, Urbach S: **The TRANSFAC system on gene expression regulation.** *Nucleic Acids Res* 2001, **29(1):**281-3
27. Wu S, Manber U: **agrep – a fast approximate pattern-matching tool.** *In Usenix Winter 1992 Technical Conference, San Francisco, CA,* 1992153-162