Dynamite: A flexible code generating language for dynamic programming methods used in sequence comaprison.

Ewan Birney and Richard Durbin

The Sanger Centre, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SA, UK.
{birney,rd}@sanger.ac.uk

Abstract

We have developed a code generating language, called Dynamite, specialised for the production and subsequent manipulation of complex dynamic programming methods for biological sequence comparison. From a relatively simple text definition file Dynamite will produce a variety of implementations of a dynamic programming method, including database searches and linear space alignments. The speed of the generated code is comparable to hand written code, and the additional flexibility has proved invaluable in designing and testing new algorithms. An innovation is a flexible labelling system, which can be used to annotate the original sequences with biological information. We illustrate the Dynamite syntax and flexibility by showing definitions for dynamic programming routines (i) to align two protein sequences under the assumption that they are both poly-topic transmembrane proteins, with the simultaneous assignment of transmembrane helices and (ii) to align protein information to genomic DNA, allowing for introns and sequencing error.

Introduction

Dynamic programming methods have a successful and varied use in computational biology. The most sensitive techniques for comparison of biological sequences, at the DNA or protein levels, rely on dynamic programming (DP), including standard pairwise comparisons (Needleman and Wunsch 1971; Smith and Waterman 1984) and motif modelling such as profiles (Gribskov *et al.* 1987; Bork and Gibson 1996). More recently the use of dynamic programming has been broadened to sequence to structure comparisons (Bowie *et al.* 1991), gene prediction (Snyder and Stormo 1994; Solovyev *et al.* 1995), frameshift tolerant alignment (Birney *et al.* 1996) and other applications. The techniques of hidden Markov models provide a formalised probabilistic framework for many of these DP methods (Krogh *et al.* 1994; Eddy 1996).

In all the cases listed above, the DP code has been written specifically for each problem. We discovered in our own work that the number of DP routines which we wanted to develop was making their successive implementations time consuming. To overcome this we have developed a code generating language, called Dynamite, which implements a standard set of DP routines in C from an abstract definition of an underlying state machine given in a relatively simple text file.

Dynamite is designed for problems where one is comparing two objects, the **query** and the **target** (figure 1), that both have a repetitive sequence-like structure, i.e. each can be considered as a series of units (e.g. for proteins each unit would be a residue). The comparison is made in a dynamic programming **matrix** of **cells**, each corresponding to a unit of the query being compared to a unit of the target. Within a cell, there can be an arbitrary number of **states**, each of which holds a numerical value that is calculated using the DP recurrence relations from state values in previous cells, combined with information from the two units defining the current cell. More specifically, each recurrence allows for a number of **transitions** (called **sources** in the Dynamite definitions) from specific states in previous cells at specific unit offsets. The value for the current state is taken as the maximum of the values of these transitions.

For example, the following recurrence relations define the Smith-Waterman algorithm (Smith Waterman 1984).

$$M(i,j) = \max\{M(i-1,j-1), I(i-1,j-1), D(i-1,j-1)\} + Match(i,j)$$
$$D(i,j) = \max\{M(i-1,j) + gapopen, D(i-1,j) + gapextension\}$$
$$I(i,j) = \max\{M(i,j-1) + gapopen, D(i,j-1) + gapextension\}$$

The corresponding Dynamite definition is given in figure 2. Although this example corresponds to a standard algorithm, for which there are many implementations, it is easy to use the same approach to generate the code for more complicated algorithms, with additional states and transitions representing alternative interpretations of the units, as we will show later in this paper.

Searls and Murphy (Searls and Murphy 1995; Searls 1996) have previously described a similar dynamic programming architecture based on Finite State Automata for aligning biological sequences. We introduce a number of new features which extend their model and improve the power and practical utility of these methods. First, we have

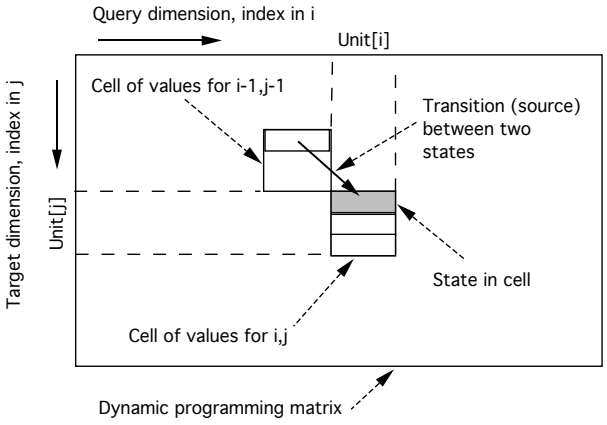

Figure 1 A diagram showing the general datastructures common to all Dynamite DP routines

introduced a new class of states called **special states**, which sit outside the cell structure of the matrix, and allow complex boundary conditions. Second we support user defined data types, in addition to the standard DNA and Protein sequence types, allowing us to use hidden Markov models of protein domains, Profiles, and other information such as gene splicing predictions in query or target objects. Third we have added the ability to label the units on the basis of the transitions used in the optimal alignment. This allows Dynamite to classify each sequence into biologically meaningful regions at the same time as finding the alignment between them. This classification process is often more important than the alignment itself.

Overview of data types used in Dynamite

Dynamite defines a data structure to hold the query and target objects, along with any additional resource which the programmer requires and a structure for the actual dynamic programming values. The memory structure for these matrix values will change depending on the implementation of the DP method, and can be accessed directly by debugging routines. The query dimension has the index **i** associated with it, the target dimension the index **j**.

Each of the query and target objects can either have user defined types or logical types, which support standard biological types. The logical types provided, with the allowed methods to retrieve data from them are give in table below. Dynamite can handle the IO itself for these types, which means in particular that it can generate efficient database searching routines.

Type	**Methods**
PROTEIN	AMINOACID(obj,pos)
DNA	BASE(obj,pos)
CODING	BASE(obj,pos) CODON(obj,pos)
GENOMIC	BASE(obj,pos) CODON(obj,pos) 5'SS(obj,pos) 3'SS(obj,pos)
COMPMAT	AAMATCH(obj,a,b)
CODONTABLE	AACODON(obj,c)

Dynamite produces two basic types of alignment: a) a path alignment, b) a label alignment. The path alignment is simply the list of [i,j,state] triples (each triple defining one number in the matrix) which produced the final score. The label alignment can be produced when each transition is provided with both a query and target label. The label alignment is a series of coupled regions, each having a start and end point in its appropriate index and a label. For example, it can be used to label a transmembrane segments in a protein or exons in DNA sequence.

The Dynamite state machine definition

A Dynamite file (see figure 3 for an example) has a header and footer regions, defined by %{ %} and a central definition region, where any number of Dynamite definitions can occur. A Dynamite state machine definition, indicated by the lines **matrix** <matrix-name> ... **endmatrix**, contains the following information:

a) A query data structure, indicated by the keyword **query**, with a name, type and method to determine its length.

b) A target data structure, indicated by the keyword **target**, with a name, type and method to determine its length.

c) Any number of resource data structures, indicated by the keyword **resource**, with a name and type.

d) A series of state definitions, defined in the lines between the keywords **state** <state-name> .... **endstate.** Special states are indicated **!special**.

Each state must have one or more of **sources**, each source corresponding to the origin for one transition to this state, indicated by the lines **source** <source-state-name> .... **endsource**. Each source block must have an offset in i (query dimension) given by **offi**="<number>" and an offset in j (target dimension) given by **offj**="<number>".

Each source block must have a single calc line, indicated by the keyword **calc**="..." with the update calculation which can involve any combination of the indexes i,j, and the query, target or resource data structures, using C syntax.

```
%{
#include "dyna.h"

%}


#
# ProteinSW: Dynamite implementation of
#            Smith-Waterman algorithm
#

matrix ProteinSW
query    type="PROTEIN"  name="query"
target   type="PROTEIN"  name="target"
resource type="COMPMAT"  name="comp"
resource type="int"    name="gap"
resource type="int"    name="ext"
globaldefaultscore NEGI
state MATCH offi="1" offj="1"
  calc="AAMATCH(comp_matrix, \
                AMINOACID(query,i),\
                AMINOACID(target,j))"
  source MATCH calc="0" endsource
  source INSERT calc="0" endsource
  source DELETE calc="0" endsource
  source START calc="0" endsource
  query_label  SEQUENCE
  target_label SEQUENCE
endstate
state INSERT offi="0" offj="1"
  source MATCH calc="gap" endsource
  source INSERT calc="ext" endsource
  query_label  INSERT
  target_label SEQUENCE
endstate
state DELETE offi="1" offj="0"
  source MATCH calc="gap" endsource
  source DELETE calc="ext" endsource
  query_label  SEQUENCE
  target_label INSERT
endstate
state START SPECIAL_I defscore="0" !start
  query_label  START
  target_label START
endstate
state END SPECIAL_I !end
  source MATCH ALLSTATES
    calc="0"
    endsource
  query_label  END
  target_label END
endstate
endmatrix
#
# End Dynamite definition
#
```

Figure 2 The Dynamite defintion for the Smith Waterman algorithm

Each state information can have a source independent calc line, and give defaults for either offi, offj or the labels which will be applied to all sources of that state

The dynamite compiler parses the data structure and cross checks the information provided in it. For example, every source which refers to a state is cross referenced to the state, and a "stateless" source triggers a compiler error. Although the calc lines are parsed by the dynamite compiler, the dynamite compiler can only check the usage of logical types which it knows about: complete type-checking is provided later by the C compiler.

Each state machine definition produces code for the following routines:

a) Functions for creating and destroying the primary data structure.
b) A function to execute the dynamic programming calculations to fill the matrix.
c) A path alignment trace-back on a calculated matrix.
d) Debugging routines to access and query positions in the matrix.
e) A linear space score implementation.
f) A linear space score implementation also returning start/end point for local methods.
g) A divide and conquer function for linear space path alignment.
h) A converter from a path alignment to a label alignment.

If the target type is one of the Sequence logical types, a database searching routine is also generated which iterates over an appropriate sequence database. This routine has no restriction on the size of the sequence database entries.

Special states and boundary conditions

There are two mechanisms for initialising a matrix and handling boundary conditions: special states and boundary sources.

Any state can be described as “special”. Rather than these states being repeated in every cell, only one copy of each state is held for each strip of matrix at a position j in the target (this means that the target and query dimensions are not completely symmetric). Source definitions to and from special states to normal states are considered to happen for every i position at that j position, and the i,j index for the calc line is taken as being the recipient or source cell. Source definitions between special states are allowed as long as there is an offset in j. Every Dynamite matrix must have two special states, given any name, but one with the identifier **!start**, and one with the identifier **!end**.

The special state mechanism was developed to handle local and wrapped alignments. Local alignments generally will have a source line from the special !start state to all states, and the !end special state will have a sources from all the normal states. Wrapped alignments allow multiple matches of one object to the other object. In the Dynamite definition, only the query object can be matched multiple times. In wrapped alignments additional special states are added, which act as both the source and recipient of the main cell entry/exit points. By including specific transitions between special states, complex wrapped alignments can be developed. For example, one can model the common occurrence of specific linker regions (for example, high glycine content) between copies of a protein domain using a wrapped method.

```
%{
#include "dyna.h"
#include "tmpara.h"
%}

matrix TMrix
query    type="PROTEIN"       name="query"
target   type="PROTEIN"       name="target"
resource type="TMparaScore *" name="tms"
globaldefaultscore NEGI
state  INTERNAL
  source INTERNAL offi="1" offj="1"
    calc="tms->cyt[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[INTMATCH]"
    endsource
  source INTERNAL offi="1" offj="0"
    calc="tms->cyt_ins[AMINOACID(query,i)]\
     + tms->trans[INTMATCH]"
    target_label INSERT
    endsource
  source INTERNAL offi="0" offj="1"
    calc="tms->cyt_ins[AMINOACID(target,j)]\
     + tms->trans[INTMATCH]"
    query_label INSERT
    endsource
  source HY_OUT2INT offi="1" offj="1"
    calc="tms->cyt[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[HY2INT]"
    endsource
  source START offi="1" offj="1"
    calc="tms->trans[START2INT]" !top !left
    endsource
  query_label INTERNAL
  target_label INTERNAL
  endstate
state  EXTERNAL
  source EXTERNAL offi="1" offj="1"
    calc="tms->ext[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[EXTMATCH]"
    endsource
  source EXTERNAL offi="1" offj="0"
    calc="tms->ext_ins[AMINOACID(query,i)]\
     +tms->trans[EXTINSERT]"
    target_label INSERT
    endsource
  source EXTERNAL offi="0" offj="1"
    calc="tms->ext_ins[AMINOACID(target,j)]\
     +tms->trans[EXTINSERT]"
    query_label INSERT
    endsource
  source HY_INT2OUT offi="1" offj="1"
    calc="tms->ext[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[HY2EXT]"
    endsource
  source START offi="1" offj="1"
    calc="tms->trans[START2EXT]" !top !left
    endsource
  query_label EXTERNAL
  target_label EXTERNAL
  endstate
state  HY_INT2OUT
  source HY_INT2OUT offi="1" offj="1"
    calc="tms->hy[AMINOACID(query,i)]
     [AMINOACID(target,j)]+tms->trans[HYMATCH]"
    endsource
  source HY_INT2OUT offi="1" offj="0"
    calc="tms->hy_insert[AMINOACID(query,i)]\
     +tms->trans[HYINSERT]"
    target_label INSERT
    endsource
  source HY_INT2OUT offi="0" offj="1"
    calc="tms->hy_insert[AMINOACID(target,j)]\
     +tms->trans[HYINSERT]"
    query_label INSERT
    endsource
  source INTERNAL offi="1" offj="1"
    calc="tms->hy[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[INT2HY]"
    endsource
  query_label  HYDROPHOBIC
  target_label HYDROPHOBIC
  endstate
state  HY_OUT2INT
  source HY_OUT2INT offi="1" offj="1"
    calc="tms->hy[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[HYMATCH]"
    endsource
  source HY_OUT2INT offi="1" offj="0"
    calc="tms->hy_insert[AMINOACID(query,i)]\
     +tms->trans[HYINSERT]"
    target_label INSERT
    endsource
  source HY_OUT2INT offi="0" offj="1"
    calc="tms->hy_insert[AMINOACID(target,j)]\
     +tms->trans[HYINSERT]"
    query_label INSERT
    endsource
  source EXTERNAL offi="1" offj="1"
    calc="tms->hy[AMINOACID(query,i)]\
     [AMINOACID(target,j)]+tms->trans[EXT2HY]"
    endsource
  query_label  HYDROPHOBIC
  target_label HYDROPHOBIC
  endstate
state START defscore="0" SPECIAL_I !start
  endstate
state END offj="0" !end SPECIAL_I
  source EXTERNAL ALLSTATES
    calc="tms->trans[EXT2END]" !bottom !right
    endsource
  source INTERNAL ALLSTATES
    calc="tms->trans[INT2END]" !bottom !right
    endsource
  query_label END
  target_label END
endstate
endmatrix
```

Figure 3 The Dynamite defintion for the algorithm Tmrix, which compares two proteins under the assumption that they are both intergal membrane proteins

In addition to this basic mechanism to initialise and terminate DP matrices, certain sources can be described as boundary sources which are restricted to be used only on the left, right, top or bottom of the matrix, as specified by the modifiers **!left !right** etc. These modifiers can only occur to or from special states, and are used to force global or semi-local methods.

Dynamite compiler implementation

The Dynamite compiler (dyc) reads in a dynamite file (by convention .dy file) and produces two C files; a header file (.h) which contains a data structure definition and function prototypes which manipulate it and a function file (.c) file which provides the actual function code. Mapping between

```
Tmrix output (pre-alpha)

Sequence 1:cbm.pep
Sequence 2:cbp.pep

Score (in bits over random models) 521.17


                    ++++++++++++++++++++++ ++++++  +++++  ++
CB21_MAIZE    1     MAASTMAISSTAMAGTPIKVGSF-GEGRIT--MRKTV--GK
CB23_LYCES    1     MA-S-MAA--TASSTTVVKATPFLGQTKNANPLRDVVAMGS
                    ++ + +++  ++++++++++++++++++++++++++++++

                    ++++++++++++++++++++++++++++++++++++++++
CB21_MAIZE    37    PKVAASGSPWYGPDRVKYLGPFSGEPPSYLTGEFPGDYGWD
CB23_LYCES    38    ARFTMSNDLWYGPDRVKYLGPFSAQTPSYLNGEFPGDYGWD
                    ++++++++++++++++++++++++++++++++++++++++

                    +++++++++++++++++++++++++++++++++------ -
CB21_MAIZE    78    TAGLSADPETFAKNRELEVIHSRWAMLGALGCVFPELLS-R
CB23_LYCES    79    TAGLSADPEAFAKNRALEVIHGRWAMLGALGCIFPEVLEKW
                    +++++++++++++++++++++++++++++++++--------

                    -----------------------------============
CB21_MAIZE    118   NGVKFGEAVWFKAGSQIFSEGGLDYLGNPSLIHAQSILAIW
CB23_LYCES    120   VKVDFKEPVWFKAGSQIFSDGGLDYLGNPNLVHAQSILAVL
                    -----------------------------============

                    ================++++ ++++++++++++ +++++++
CB21_MAIZE    159   ACQVVLMGAVEGYRIAGGP-LGEVVDPLYPGGS-FDPLGLA
CB23_LYCES    161   GFQVVLMGLVEGFRINGLPGVGEGND-LYPGGQYFDPLGLA
                    ================++++++++++ ++++++++++++++

                    +++++++++++++++==================--------
CB21_MAIZE    198   DDPEAFAELKVKELKNGRLAMFSMFGFFVQAIVTGKGPLEN
CB23_LYCES    201   DDPTTFAELKVKEIKNGRLAMFSMFGFFVQAIVTGKGPLEN
                    +++++++++++++++==================--------

                    ------------------------
CB21_MAIZE    239   LADHIADPVNNNAWAYATNFVPGN
CB23_LYCES    242   LLDHLDNPVANNAWVYATKFVPGA
                    ------------------------
```

Figure 4 An example output of the Tmrix algorithm. The location of the peptide sequence is indicated by + for intracellular, = for transmembrane and - for extracellular

.dy files and .c/.h files can be provided as a suffix rule in UNIX makefiles, allowing the seamless use of the .dy file as "source code".

The C code produced by Dynamite has been designed with the following guidelines: (i) it passes strict ANSI C checks, and does not produce memory leaks or array access violations as tested by Purify™ (Pure Software Inc) (ii) it has been tested on multiple UNIX platforms (Sun, SGI, Digital, Linux), (iii) it uses explicit parameters for inter function communication rather than static buffers, (iv) the function bodies are readable C (formatted and commented) and the header file has function documentation. The definitions and functions to manipulate the logical data types and the underlying memory and alignment structures common to all DP routines produced by Dynamite are provided as a library as part of the distribution.

The run time speed of the C code produced by Dynamite is just a little slower hand written code: for example, when used to produce a standard protein HMM comparison to a protein sequence it is 15%-20% (depending on the platform) slower than the corresponding programs in the HMMER package (Eddy 1996). There is room for improvement in the dyc compiler, such as pre-processing certain common sub-parts of calc lines, however this is not implemented. A more promising route is perhaps to provide a compiler to specialised or parallel hardware. To this end we are writing a port to the BIOXL/G family of products from Compugen, and we are interested in any other hardware or parallel implementation.

Examples

Figure 2 is the Dynamite file for the Smith-Waterman DP algorithm. Notice the two special states START and END. A more interesting example is shown in Figure 3, which gives a Dynamite file for the algorithm Tmrix. Tmrix compares two transmembrane protein sequences to each other with a concurrent transmembrane topology prediction (Hugh Salter, E.B and R.D., work in progress). Figure 4 is an example output. The algorithm allows for different substitution matrices and different insertion frequencies for the different regions held in the data structure “TMparaScore *”, which is a resource for this matrix. This

way we can model directly specific properties of different regions of transmembrane proteins; in particular there is a bias towards positively charged residues in the internal domains with a similar increase of negatively charged residues in external domains, and the substitution patterns transmembrane helices are different from globular domains. With these biological insights providing the basis of the construction of this algorithm, and by labelling the transitions correctly, Dynamite can produce predictions of the transmembrane topology of these proteins simultaneously with finding their alignment. Performing both tasks together significantly enhances the performance of each taken separately.

A more complex example, GeneWise21:93, is shown schematically in Figure 5. GeneWise21:93 matches a hidden Markov model of a protein domain to genomic DNA. Figure 6 is an example output. It finds the best alignment considering both the protein information and gene prediction information, and is one instance of a family of algorithms, the GeneWise family, designed for this purpose (E.B., Mor Amitai and R.D., manuscript in preparation). Note that the short match after the second intron in Figure 6 is unlikely to be found using a simple six frame translation comparison. In this case the standard protein HMM method had to be recast to (i) match codon triplets rather than amino acids (ii) allow for sequencing errors of insertions or deletions of bases, which would make for positions in the HMM being represented by more than (insertion) or less than (deletion) 3 bases in the DNA (iii) allow for introns of up to 5,000 or more bases with characteristic sequence features defining their start and end points. In particular we wished to combine a probabilistic model of gene intron/exon structure with the probabilistic model of a protein domain to allow all the available information about a gene to be combined consistently. There are many different features known to be important in gene structure, for example the 5' and 3' splice sites of the introns, the poly pyrimidine tract (a region of C or Ts often seen before the 3' splice site), the branch point (a very weak consensus upstream of the poly-pyrimidine tract), all of which can be modelled by specific Dynamite states and transitions. Again, if this is done, as well as potentially improving the search one would be identifying the specified features in the sequence.

Dynamite was central to the development of the GeneWise algorithms. First many of the methods are inherently complex, with 93 transitions occurring in GeneWise21:93 (the numbering refers to the number of states:number of transitions), involving over 300 array access positions per cell. Maintaining such a method by hand, in particular the more complicated linear space methods would become quite intensive. Second the flexibility of Dynamite allowed us to experiment with a variety of methods without a heavy overhead of writing new functions and without worrying about keeping every implementation in sync. Third we did not have to commit to one best method but can both provide a variety of methods and we can continue to try to incorporate new features in a consistent and robust manner. For example, in addition to the 21:93 model we have designed a smaller 4:21 model (4 states, 21 transitions) for faster database searching. Given a match to GeneWise4:21 it would be possible to run GeneWise21:93 to check the detailed alignment and assignment of gene structure features.

Again the labelling process of Dynamite provided an intuitive way of converting the combined homology and gene prediction process to simply a gene prediction parse with the confidence that the additional homology information had enforced the correct gene prediction. The more accurate GeneWise methods, such as the one shown in figure 5 required quite complex boundary conditions. The homology information does not necessarily span the entire gene; however, in such cases the gene prediction apparatus in a GeneWise algorithm will find gene features which it can distinguish from random DNA. The correct algorithm therefore requires a general gene predictor flanking each side of the homology segment to prevent a spurious matching of the homology based information solely on the basis of the gene prediction information. This type of boundary condition requires a series of special states to model the gene structure in flanking regions (the special states are omitted from Figure 5 for clarity). We have also experimented with wrapped alignments for cases where there are multiple copies of a single domain in a protein: in this case the linker sequence models a general gene prediction method that does not lose frame information even across introns.

Discussion

Dynamite provides a flexible dynamic programming definition, both with respect to the types of dynamic programming matrices produced and the types of input data used. The run-time speed is comparable to hand written code, and the ability to be provided immediately with a full set of implementations, including linear space and database searching methods, allows rapid prototyping and testing of new algorithms. The above two examples illustrate how useful the flexibility is in designing and implementing new algorithms. All these methods can use the established probabilistic frameworks of HMMs or probabilistic Finite State Machines to allow rigorous Bayesian deductions to be inferred from the algorithms. The additional flexibility which Dynamite provides allows us to model the biological phenomena in an intuitive manner, by using labels to provide separate interpretations of a biological sequence, while the probabilistic framework greatly simplifies the problem of parameterisation from annotated datasets. In some cases, it may also be feasible to find optimal parameters from unannotated datasets using posterior maximum likelihood techniques

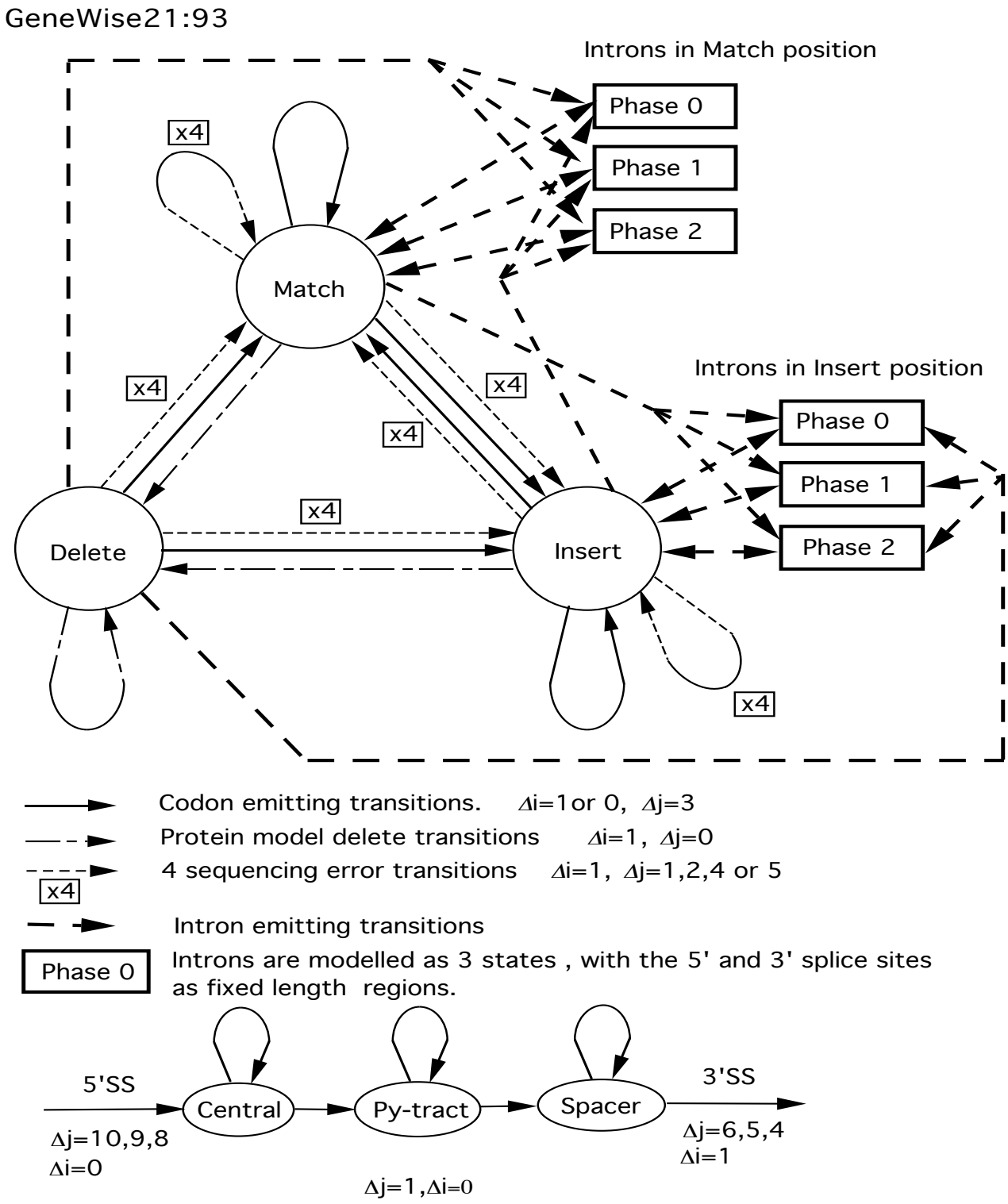

Figure 5 A schematic diagram of the GeneWise 21:93 algorithm, which compares a protein HMM to genomic sequence allowing for introns and frameshifts. The states are represented as circles, with transitions (sources) between them as arrows; this is similar to the notation used by Searls and Murphy (1995).

```
Protein  HMM: RRM
DNA sequence: HSHNRNPA

Score in bits 141.42

Output notation

<X ---- X>       intron of phase X
+                 predicted poly-pyrimidine tract
Central Intron    Central region of introns residues omitted


RRM          1     KLFVGNLPPDTTEEELRELFSQFGEIESIK                                VMRDKPETGKSRGFGFVT
HSHNRNPA     1     KLFIGGLSFETTDESLRSHFEQWGTLTDCV        Central Intron          VMRD-PNTKRSRGFGFVT
             1404  actaggtatgaaggacaactgctgacagtgGTAAGATT 1500 <-> 1780 TTCTcAGgaag caaactagtgtga
                   atttggtgtaccaagtggataaggctcagt<0------                ++++ 0>ttga cacagcggtgttc
                   gcctaggctaattgcggcttgagagcgctg                              agat accgctgctgtca


RRM          49    FESEEDAEKAIEALNGKVLGGRPLRVKAAQKK                                    EERQ
HSHNRNPA     48    YATVEEVDAAMNARPHKVDGRVVEPKRAVSRE        Central Intron              DSQR
             1840  tgagggggggaagaccagggagggcaaggtagGTGAGTGG 1942 <-> 2067 TTTTTCTCCTattAGgtca
                   acctaatacctacgcaataggttacagctcga<0------              ++++++++++   0>acag
                   tctggggtatgtagacggtaatgaagatccaa                              ttaa
```

Figure 6 The alignment of a Hidden Markov Model of the RNA Recognition motif to the genomic sequence of hnRNPA1.

We believe that the more biological information that we can use in designing these finite state machine models, the better we will be able to generate correct alignments, and, perhaps more importantly, the better we will be able to automatically annotate the individual sequences with biological information.

Homologous sequence detection is now the primary method used to infer function when a new gene is sequenced, either in a targetted investigation, or as part of a large scale sequencing project. During the last few years, there has also been increased use of homology information to assist in gene prediction methods, sequencing error detection and secondary structure assignment. The basis of these methods is that different selective pressures act in regions of different biological function. The labelling mechanism of Dynamite codifies this common practice, by combining pairwise comparison with the partitioning of each sequence into biologically meaningful regions. The two examples illustrate this. The Tmrix method predicts transmembrane helix placement and topology in two sequences by relying on previously observed features of integral membrane protein regions to drive the prediction process. The GeneWise21:93 method derives a gene structure prediction on a single piece of DNA which is consistent with protein similarity to another sequence or a HMM of a protein domain. We hope to use the labelling mechanism of Dynamite as a general way of "marking-up" sequences into biologically interesting regions in a way which can be easily automated and implemented.

There are a variety of improvements we plan to incorporate into Dynamite in the future. First we want to support sequencing traces (as produced by ABI automated sequencing machines) as a logical type in Dynamite. This would allow alignment of single sequencing reads, such as ESTs, to genomic sequence, extending the alignment into the poorer quality part of the read, where however there is still sufficient information to confirm an exon assignment, for example. Second, we hope to make the calculation process used in the dynamic programming pass through the DP matrix more flexible. In particular, the ability to add transition scores (in fact probably log-add them because scores correspond typically to log probabilities) rather than take their max, would allow use of the forward-backward algorithm used in HMM training (Rabiner 1989). This would also allow derivation of posterior probabilities on sequence features, such as exons, TM helices etc., rather than relying on the optimal path correctly representing every feature. Third, it should be relatively easy for Dynamite to produce the necessary wrapper code to bundle each DP matrix as a valid object type into scripting languages such as Perl, Python or Tcl. This would combine the speed of compiled C code written specifically for each DP method with the flexibility of scripting languages to develop new programs on demand.

Acknowledgements

This work was started in Professor I. Campbell's group (OCMS, Oxford University). E.B. is funded by a Wellcome Trust Prize Studentship. R.D. is supported by the Wellcome Trust. We would like to thank Ian Holmes, Sean Eddy, Mor Amitai and Toby Gibson for their interest, support and discussions during the development of Dynamite. A stable URL for Dynamite documentation, help and information is
http://www.sanger.ac.uk/~birney/dynamite/

References

Birney, E.; Thompson, J.D; and Gibson, T.J. 1996. PairWise and SearchWise: finding the optimal alignment in a simultaneous comparison of a protein profile against all DNA translation frames *Nucleic Acids Research* 24: 2730-2739

Bork P.; and Gibson T.J. 1996 Applying motif and profile searches. *Methods in Enzymology* 266: 162-184

Bowie, J.U.; Luthy, R. and Eisenberg, D. 1991 A method to Identify Protein Sequences That Fold into a Known Three-Dimensional Structure *Science* 253: 164-170

Eddy, S. 1996 Hidden Markov Models. *Current Opinion in Structual Biology*. 6: 361-365

Gribskov, M.; McLachlan, A.D.; and Eisenberg, D. 1987 Profile analysis: Detection of distantly related proteins. *Proc Natl. Acad. Sci USA* 84: 4355-4358

Krogh, A.; Brown, M.; Mian, I.S.; Sjolander, K.; and Haussler D. 1994. Hidden Markov Models in computational biology: Applications to protein modeling. *Journal Molecular Biollogy* 235: 1501-1531

Krogh, A. 1995. Hidden Markov models for labeled sequences. *Proc 12th International Conference on Pattern Recognition* (Jerusalem, Oct. 1994).

Needlman, S.B. and Wunsch, C.D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins *Journal Molecular Biology.* 48: 443-453

Rabiner, L.R. 1989 A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition *Proceedings of the IEEE* 77: 257-285

Searls, D.B. 1996 Sequence alignment through pictures *Trends in Genetics* 12: 35-37

Searls, D.B. and Murphy, K.P. 1995 Automata-Theoretic Models of Mutation an d Alignment *Intelligent Systems for Molecular Biology* 3: 341-350

Smith, T.F. and Waterman, M.S. 1981. *Advanced Applied. Mathematics*. 2: 482-489

Snyder, E.E. and Stormo, G.D. 1994. Identification of coding regions in genomic DNA sequences: an application of dynamic programming and neural networks. *Nucleic Acids Research* 21: 607-613

Solovyev, V.V.; Salamov, A.A. and Lawrence, C.B. 1995 Identification of human gene structure using linear discriminant functions and dynamic programming *Intelligent Systems for Molecular Biology* 5: 367