An Example in Kleisli: Codon Usage Extraction Made Easy

Jiren Wang and Limsoon Wong
BioInformatics Centre & Kent Ridge Digital Labs
21 Heng Mui Keng Terrace, Singapore 119613
Email: {jrwang,limsoon}@krdl.org.sg

19 March 1999

Codon usage information was useful to many molecular biologists in designing appropriate degenerate oligonucleotides and in optimizing expression of genes. It could also be used in improving the sensitivity of alignment tools in detection of short coding regions. We had two objectives in this paper. First, we wanted to build a system that could extract coding sequences of a specified organism from public DNA sequence databases and could compute their codon usage. Second, we wanted to demonstrate how the general database integration system called Kleisli could help build such a system and other sophisticated bioinformatics applications easily. We achieved these two objectives by showing that short and clear programs could be written in Kleisli, using its high-level query language CPL, to build such a system. The codon usage information of rice was produced as an example.

Introduction

The recent explosion of genomic information, as gleaned from the Human Genome Project and other similar efforts, had been fueled by engineering and technological advances. However, as the amount of information grows the bioinformatics challenge became one of managing and making sense of the data. Many bioinformatics problems (1) required access to data sources that were high in volume, highly heterogeneous and complex, constantly evolving, and geographically dispersed; (2) required solutions that involved multiple carefully sequenced steps; and (3) required information to be passed smoothly between the steps.

As observed by Baker and Brass [1], many existing biology data retrieval systems[2, 3, etc.] were not fully up to the demand of flexible and painless bioinformatics data integration. These systems relied on low-level direct manipulation by biologists. The archetypal example was the Entrez system[2]. Here a biologist used a keyword to extract summary records, then clicked on each record to view its contents or to perform operations. This worked well for simple actions. However, as the number of actions or records increased, such direct manipulation quickly became a repetitive drudgery. Also, when the questions became more complex and involved many databanks, assembly of the data needed exceeded the skill and patience of most biologists. Merely providing a library package that interfaced to a lot of databases and analysis softwares was also not useful if it required long-winded and tedious programming to make use of and/or adding to the package, as demonstrated [4, etc.] by the difficulties with CORBA[5].

The advanced integration technology called Kleisli[6, 7] went further in integrating and querying complex data sources. In particular, Kleisli provided the high-level query language CPL [8]. CPL offered a nice data model and many high-level operators to express complex queries and tranformations on these biology databases and analysis softwares in a manner that was extremely straightforward and without overly taxing a user's programming skill.

There was some degree of belief that Kleisli had reduced the difficulty of integrating biology data [9, 10, 1, etc.]

In this paper, we would like to demonstrate Kleisli on a problem in bioinformatics that belongs to the "Frequently Asked Questions" category: The extraction and assembly of exons from large DNA sequence databases and the derivation of a codon usage table from them. Codon usage information should help in designing appropriate degenerate oligonucleotides and should also be useful for experiments designed to optimize overexpression of selected genes [12]. It could also be used in improving the sensitivity of alignment tools in detection of short coding regions [13]. There were several problems in extracting this information from a large public sequence database such as GenBank [11]. Firstly, such a database had a fairly complicated structure. Secondly, feature annotations needed for correct identification of exons were buried within such a complicated structure. Thirdly, available public resources such as Entrez [2] did not provide a convenient means for extracting these buried feature annotations.

Kleisli possessed all necessary attributes to address this problem. After we showed how Kleisli could be used to extract and derive codon usage of DNA sequences from Entrez in a simple way, we also showed how it was used to handle two related queries. One was to generate a probable DNA sequence given a protein sequence from an organism. The other was the automatic assembly of the underlying exons given a protein sequence (or its Entrez uid) of an organism. The over 100 bioinformatics functions provided by Kleisli made the solutions to these problems trivial. In other words, unlike other commercial systems which were designed to solve one or two specific problems, **Kleisli was a general platform that had no inherent limitation on the variety of bioinformatics and integration problems it could handle, nor on sequence length, nor on database size, nor on data structure complexity.**

Materials and Method

Data Sources. We used the GenBank section of Entrez[2] at Washington DC as our source of genomic sequences. In particular we used rice DNA sequences for illustration. This data source had a fairly complicated format. A slightly truncated example, of the rice gene for bZIP (Entrez uid 4115745), was selected for illustration below.

```
LOCUS       AB021736     5651 bp    DNA             PLN       08-JAN-1999
DEFINITION  Oryza sativa gene for bZIP protein, complete cds.
ACCESSION   AB021736
NID         g4115745
KEYWORDS    bZIP protein.
SOURCE      Oryza sativa (sub_species:Indica, isolate:IR36) DNA.
  ORGANISM  Oryza sativa
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta; ...
REFERENCE   1  (sites)
  AUTHORS   Nakase,M., Matsuura,A., Okabe,S., Matsuda,T. and Adachi,T.
  TITLE     Gene structure and DNA-binding specificity of the ...
  JOURNAL   Unpublished (1999)
REFERENCE   2  (bases 1 to 5651)
  AUTHORS   Nakase,M.
  TITLE     Direct Submission
  JOURNAL   Submitted (22-DEC-1998) to the DDBJ/EMBL/GenBank ....
FEATURES             Location/Qualifiers
     source          1..5651
                     /organism="Oryza sativa"
                     ...
     CDS             join(1653..2061,3074..3167,3702..3897,4704..4779,
                     4868..4993,5197..5561)
```

```
                      ...
                      /translation="MERVFSVEEIS..."
BASE COUNT     1515 a   1163 c   1290 g   1683 t
ORIGIN
        1 ggagggagga aagtaagccc agattcacaa aaatgtggac acgagtcatt atcagatgaa
       61 aggaataaat ...
```

As could be seen, the "CDS" feature annotations needed for correct identification of exons were buried within a nested subfield of a GenBank report. However, Entrez did not directly provide a convenient means for extracting these buried feature annotations. So it was necessary to employ more sophisticated data manipulation tools.

Data Manipulation Tools. We used the high-level query language, CPL, of the Kleisli system [6, 7]. The industrial-strength version of Kleisli that we developed in Singapore had over 100 functions for manipulating over 50 bioinformatics data sources. A list of some sources that Kleisli interfaced to could be found at `http://adenine.krdl.org.sg:8080/ publications/drivers.html`. One of these functions was selected for illustration below. Detail of this and other functions could be found in [14].

The `na-get-seqfeat-by-uid` function, when applied to a unique identifier in Entrez such as uid 4115745 of our rice gene, would return the GenBank report associated with that identifier. The output corresponding to our example rice gene looked as given below.

```
{(#uid: 4115745, #accession: "AB021736",
  #title: "Oryza sativa gene for bZIP protein, complete cds.",
  #organism: "Oryza sativa", #taxon: 4530,
  #lineage: ["Eukaryota", "Viridiplantae", "Streptophyta", ...],
  #seq: "GGAGGGAGGAAAGTAAGCCCAGATTCACAAAAATGTGGACACGAGTCATT...",
  #feature: {(#name: "source", #continuous: true,
              #position: [(#accn: "AB021736", #start: 0, #end: 5650, #negative: false)],
              #anno: [(#anno_name: "organism", #descr: "Oryza sativa"), ...])
             (#name: "CDS", #continuous: true,
              #position: [(#accn: "AB021736", #start: 1652, #end: 2060, #negative: false),
                          (#accn: "AB021736", #start: 3073, #end: 3166, #negative: false),
                          (#accn: "AB021736", #start: 3701, #end: 3896, #negative: false),
                          (#accn: "AB021736", #start: 4703, #end: 4778, #negative: false),
                          (#accn: "AB021736", #start: 4867, #end: 4992, #negative: false),
                          (#accn: "AB021736", #start: 5196, #end: 5560, #negative: false)],
              #anno: [ ..., (#anno_name: "translation", #descr: "MERVFSVEEISDPFWVPPPPPQSAA...")])})}
```

Note that positions of features were given in the original GenBank report using "biologist" numbering (which started from 1). But positions were given in Kleisli using "computer scientist" numbering (which started from 0). The "computer scientist" numbering was used for two reasons: (i) it made interfacing to the large body of existing software libraries more convenient and less error prone, and (ii) it was trivial to transform them back into "biologist" number during final display time of results if desired.

The output of this function were no longer in the "free text" form of the original GenBank reports. Rather it had been put into a form where its underlying structure became explicit. For example, records would be formatted as $(\#l_1 : e_1, ..., \#l_n : e_n)$, sets as $\{e_1, ..., e_n\}$, and lists as $[e_1, ..., e_n]$. Data in such as format could be arbitrarily nested. Thus any of the e_i's above could itself be a record, set, list, or other complex objects. Data that conformed to this format could be directly manipulated, composed, or taken apart by using CPL in Kleisli in a simple manner.

An important manipulation construct of CPL was the comprehension construct, which had the following typical form: $\{e(x) \mid \backslash x \leftarrow R, C(x)\}$. It built the set of $e(x)$ where each x came from the collection R and satisfied the condition $C(x)$. Here the collection R was assumed to be a set if the $\leftarrow$ was `<-`, or a list if the $\leftarrow$ was `<---`. Also, if the {- and }-brackets were replaced by [- and]-brackets, the result would be a list. Another important manipulation construct was the field projection construct, which had the following form: $e.\#l$. It extracted the value at the field labelled $\#l$ in the record e. Thus, for example, to obtain the start positions of all exons on the positive strand of our example rice DNA sequence, we merely needed to execute the following CPL program:

```
{ p.#start
| \t <- na-get-seqfeat-by-uid (4115745),
  \f <- t.#feature, f.#name = "CDS",
  \p <--- f.#position, not (p.#negative) };
```

Codon Usage Extraction Procedure. Our procedure to extract and assemble the exons of rice genes, and to derive the underlying codon usage, consisted of four steps. The first step was to obtain Entrez records of rice DNA sequences whose coding regions were explicitly annotated. In other words, there had to be some "CDS" feature annotations in their GenBank reports. The Entrez specification to download rice DNA sequences having coding regions that were explicitly annotated was "rice[Organism] AND cds[Feature key]." The CPL function `na-get-seqfeat-general` would remotely query Entrez for GenBank reports satisfying a given specification. Thus to find out which rice sequences had annotated coding regions, we just applied `na-get-seqfeat-general` to the specification "rice[Organism] AND cds[Feature key]" and wrote the records obtained to a file `seqfeats`. A database-style index `accn2seq` was also created on the `#accession` field of the file `seqfeats`, so that we had fast access to a sequence given its accession number. This step was done using the CPL program below.

```
writefile na-get-seqfeat-general "rice[Organism]+AND+cds[Feature+key]" to "seqfeats" using stdout;
readfile seqfeats from "seqfeats" using stdin;
writefile "seqfeats" to "#accession" using setindex-create;
setindex-access (#name: "accn2seq", #file: "seqfeats", #key: "#accession");
```

Next, we needed functions to implement the extraction of an exon given its position information. There were two functions, `get-+ve-strand` for extracting from the positive strand and `get--ve-strand` for extracting from the negative strand. One of them was implemented as shown below.

```
primitive get-+ve-strand == \p =>
  { string-span (S.#seq, p.#start, p.#end)
  | \S <- process <#key: p.#accn> using accn2seq };
```

The input to `get-+ve-strand` is a position record `p`, specifying how an exon was derived. The `#accn` field of `p` gave the accession number of the sequence of this exon. We fetched the sequence `s` from our index `accn2seq`. Then we used the `#start` and `#end` fields of `p` to extract from `s` the specific sequence segment constituting the exon. The `get--ve-strand` function was similar, except we needed to take care of reverse-complementation.

We could now proceed to the third step, which was to assemble our gene sequences. For each record `x` in `seqfeats`, we iterated through each feature `f` in `x`'s feature table to look for a feature that corresponded to coding regions; these were those feature whose name was `CDS`. We had to make sure that the feature was complete, hence the test `f.#continuous`. To ensure that the coding region was proper, we check to see if its annotations `a` contained a protein translation and if that translation begun with methionine (`M`). If all these tests passed, then we knew that `f.#position` was a list of position records identifying all exons of the gene identified by `x`. We set `p` to iterate over these position records. Then depending on whether the exon was on the positive or negative strand, we invoked `get-+ve-strand` or `get--ve-strand` to get the corresponding sequence `z` of the exon. These `z`'s were then "imploded" or concatenated to recover the sequence of the gene. The sequences of genes, their unique identifiers, and their translations were all written to the file `genes` for subsequent processing.

```
writefile
```

```
  [ (#uid: x.#uid, #translation: a.#descr, #gene: string-implode (g))
  | \x <- seqfeats,
    \f <- x.#feature, f.#continuous, f.#name = "CDS",
    \a <--- f.#anno, a.#anno_name = "translation", a.#descr string-islike "M%",
    \g == [ z | \p <--- f.#position.list-head,
                \z <- if p.#negative then get--ve-strand (p) else get-+ve-strand (p)] ]
to "genes" using stdout;
readfile genes from "genes" using stdin;
```

Theoretically, we could produce the codon usage table by applying the `codon-usage` function of Kleisli to sequences in `genes` like this:

```
process [ x.#gene | \x <--- genes ] using codon-usage;
```

Duplicate Elimination. Since some genes might be sequenced many times from different strains or by different laboratories, the sequences of these genes in GenBank might repeat several times. Thus a codon-usage table as produced above might not be accurate. Consequently, it would be desirable if duplicates that were identical up to small differences were also eliminated before the condon-usage table was produced.

Given the information available in file `genes`, we actually had two alternate levels at which duplicate elimination could be conducted: the coding DNA or the translated protein. The latter alternative was possible because in our case the translated proteins extracted from Entrez were in the correct translation frames and were in one-to-one correspondence to the coding DNAs. As the protein sequences were three times shorter than the coding DNA sequences and came from a larger alphabet, performing duplicate elimination on them would be many times more efficient than on the coding DNAs. So we chose this more favourable alternative and used the general S-Hash sequence indexing technology [15] in Kleisli to accomplish the task.

First, a S-Hash sequence index `genes-index` was created on protein translations in `genes`. This `genes-index` allowed us to efficiently compare any sequence `g` against all sequences in `genes` and to extract those that were similar to `g`.

```
writefile {(#uid: x.#uid, #seq: x.#translation) | \x <--- genes } to "genes-index" using seqindex-out;
seqindex-scanseq (#name: "genes-index", #index: "genes-index", #level: 1);
```

Next, we defined a function `get-similar-genes` that returned all sequences in `genes` that were similar (modulo 10% mismatches) to an input sequence `g`. Then a sequence `g` was a "representative" of a group of sequences in `genes` that were similar to it if its Entrez uid was smallest amongst those of the group. This idea was captured by these definitions:

```
primitive get-similar-genes == \g => let \d == 0.1 * string-length (g.#translation) in
  process <#doit:(#restrict:false, #pattern:g.#translation, #edit:d, #overlap:30, #favourshort:true)>
  using genes-index;

primitive representative == \g => { } = { x | \x <- get-similar-genes (g), x.#uid < g.#uid };
```

At last, we could produce a codon usage table with good accuracy by considering only members of `genes` that were representatives of their groups—duplicates that had over 90% identity in their translated proteins had been eliminated from consideration.

```
primitive codon-usage-table == process [ x.#gene | \x <--- genes, representative (x)] using codon-usage;
```

Web Interfaces. A convenient web interface to the automatic codon usage extraction procedure described above had been put up at `http://adenine.krdl.org.sg:8080/demos/biokleisli/codon-usage`. This interface required only one input: the name of an organism whose codon usage table was desired. This interface also supported an additional parameter that specified a threshold on sequence identities for the elimination of highly similar sequences.

A convenient web interface to perform back translation of protein sequence into DNA sequence had also been put up at `http://adenine.krdl.org.sg:8080/demos/biokleisli/backtrans`. The user merely needed to supply his protein sequence and to select a codon usage table we had previously extracted. The DNA sequence corresponding to a probable back translation would be returned.

Results and Discussion

We produced codon usage tables for many organisms, including: *Arabidopsis*, barley, beans, *brassica*, conifers, corn, *glycine*, *lepidoptera*, pisum, rice, sorghum, wheat, monocot and dicot RNA viruses, monocot and dicot DNA viruses, etc. The codon usage tables of an organism `X` could be accessed at `http://adenine.krdl.org.sg:8080/limsoon/kozak/X/all`. Thus, beans' codon usage table could be accessed at `http://adenine.krdl.org.sg:8080/limsoon/kozak/beans/all`. The codon usage table produced for rice was selected for illustration below.

aa	codon	%	codon	%	codon	%	codon	%	codon	%	codon	%
A	GCA	17.52	GCC	33.53	GCG	24.72	GCT	24.23				
C	TGC	70.04	TGT	29.96								
D	GAC	56.74	GAT	43.26								
E	GAA	33.99	GAG	66.01								
F	TTC	63.15	TTT	36.85								
G	GGA	20.40	GGC	37.93	GGG	19.95	GGT	21.72				
H	CAC	56.87	CAT	43.13								
I	ATA	18.50	ATC	48.53	ATT	32.96						
K	AAA	29.33	AAG	70.67								
L	CTA	7.25	CTC	28.39	CTG	22.36	CTT	17.87	TTA	10.10	TTG	14.03
M	ATG	100.00										
N	AAC	61.90	AAT	38.01								
P	CCA	23.23	CCC	24.78	CCG	27.71	CCT	24.28				
Q	CAA	37.94	CAG	62.06								
R	AGA	16.27	AGG	24.05	CGA	8.46	CGC	23.75	CGG	14.40	CGT	13.07
S	AGC	21.50	AGT	11.48	TCA	13.99	TCC	23.09	TCG	13.76	TCT	16.18
T	ACA	20.55	ACC	36.34	ACG	19.31	ACT	23.80				
V	GTA	11.21	GTC	30.96	GTG	33.55	GTT	24.28				
W	TGG	100.00										
Y	TAC	62.37	TAT	37.63								
*	TAA	33.64	TAG	27.44	TGA	38.92						

Applying the extraction procedure as described in the previous section, we retrieved 845 rice nucleotide records from Entrez containing explicitly annotated CDS features. From these 845 rice genes, the codon usage table above was produced. The entire process from the downloading of GenBank reports to the displaying of the table took less than 10 minutes. This codon usage table produced by the `codon-usage` function of Kleisli was in a different form than that of [16]. The codon usage tables of [16] gave the overall frequency of a triplet amongst all possible triplets. Ours gave the relative frequency of a triplet amongst all triplets that coded for the same amino acid. Note that Kleisli also had a function `codon-usage3` that could produce codon tables in a form like those of [16]. What was remarkable was that we could produce these tables from scratch using a mere 25 lines of CPL programming.

Comments on Kleisli

To better appreciate the power of the Kleisli system in solving bioinformatics integration problems, some additional queries and comparisons were made.

Some Related Queries. The CPL programs developed so far could be used to handle some related bioinformatics questions. We chose two for illustration below. The simplicity of their implementation was remarkable.

Assume that a rice protein `P` was given. We would like to know a probable underlying coding DNA sequence. One reasonable method to generate a probable underlying coding DNA sequence for `P` was to make use of the rice codon usage table `codon-usage-table` produced earlier. For every amino acid in `P`, a corresponding DNA triplet was randomly selected according to the probabilities described in the codon usage table. The function `codon-gen2` provided in Kleisli could be used for this purpose, as shown below.

```
process (#code: codon-usage-table, #seq: P) using codon-gen2;
```

Assume that an Entrez uid `U` to a rice protein was given. We would like to know its actual underlying coding DNA sequence if this information was available. A reasonable method to obtain this information was to access the Entrez web page for protein query to bring up the GenPept report for `U`. Then clicked on its DNA link button to view the GenBank report of the DNA sequence from which this protein was derived. If the DNA sequence was a genomic DNA sequence, then we would have to assemble the exons ourselves using the "CDS" feature annotation of this GenBank report. This process could be automated using Kleisli. Kleisli had a function `aa-get-na-uid`, that given an Entrez uid of a protein, returned the uid of the GenBank report of its underlying DNA sequence. Thus we could modify our earlier exon assembly program to answer this query as follows:

```
{ string-implode (g)
| \u <- aa-get-na-uid (U),
  \s <- aa-get-seq-by-uid (U),
  \x <- na-get-seqfeat-by-uid (u),
  \f <- x.#feature, f.#continuous, f.#name = "CDS",
  \a <--- f.#anno, n.#anno_name = "translation", a.#descr = s.#sequence,
  \g == [ z | \p <--- f.#position.list-head,
             \z <- if p.#negative then get--ve-strand (p) else get-+ve-strand (p)] };
```

Comparison With Other Query Systems. Several other approaches had been available to explore information in heterogeneous biological databases and to address the problems of multiple database query and data integration. We singled out several representative approaches for comparison.

One of the more successful approaches for integrating biological databases was by connecting heterogeneous databases via hypertext links on the Web and providing comprehensive indexing systems for query. Such link-based approach was adopted by SRS [3] and LinkDB[17]. These systems were convenient to use for simple operations. However, they had no facilities for flexible transformation of data from heterogeneous sources. As a consequence, they required a significant amount of manual work for data integration.

The TAMBIS project [18] aimed at providing transparent access to multiple databases and analysis tools using a graphic user interface. It employed a knowledge-driven user interface for query formulation. The knowledge-driven graphic user interface was implemented on top of an old version of our Kleisli system for the actual execution of database query and data exchange. The TAMBIS query system was user friendly and gave a simple way for non-programmers to specify queries. However, the kind of queries that could be expressed were rather restricted by the designs of the TAMBIS query templates.

The OPM*QS [19] used the common data model approach to accomplish the task of multiple database query and data integration. The OPM*QS provided graphic interfaces, an SQL-like structured query language, an object data model, and sophisticated OPM data management tools. However, OPM*QS lacked a simple data exchange format and relied on a dictionary of schema mapping to map individual database schema into the common OPM global schema. As a result, OPM*QS required a lot more effort and planning to add new sources than Kleisli.

In contrast, Kleisli was indifferent to database schema and did not require a common global schema. Kleisli was designed around a nested relational data model that allowed arbitrary nesting of types such as sets, bags, lists, records, and variants. Kleisli provided a self-describing data exchange format to convert various data types into the Kleisli data model. The Kleisli nested relational data model was a generalization of the conventional relational data model, as well as other data models embedded in flat files and analysis software frequently used in bioinformatics. Therefore, conventional relational data and other biological data could be readily converted into the Kleisli internal data model. Furthermore, Kleisli was equipped with an advanced "type inference system" that could deduce the structure of input and output data directly from the structure of a query. The powerful data model, the self-describing data exchange format, and the advanced type inference system made the use of a global schema unnecessary in Kleisli. Kleisli could readily handle the "on demand" query of databases. In addition, the high-level CPL query language, which provided rich expressions for pattern matching, string manipulation, and other facilities, greatly eases the tasks of data manipulation and integration. The Kleisli system did require users to understand the CPL query language. To simplify usage, we experimented with a graphic user interface and a bioinformatics query wizard for routine queries [20]. A more robust graphic interface is now under way.

Availability

Kleisli runs on SUN Solaris platform, preferrably with 64 MB memory or more. It is available from KrisTech Inc. in California under the name KRIS. KRIS stands for both "Kleisli-Related Integration System" and "Kent Ridge Integration System" to distinguish this industrial-strength version from the early prototype developed by one of us (Wong) several years ago at the University of Pennsylvania.

References

[1] Baker PG and Brass A (1998) Recent development in biological sequence databases. *Current Opinion in Biotechnology*, **9**, 54–58.

[2] Schuler GD, et. al. (1996) Entrez: Molecular biology database and retrieval system. *Methods in Enzymology*, **266**, 141–162.

[3] Etzold T and Argos P (1996) SRS: Information retrieval system for molecular biology data banks. *Methods Enzymol.*, **266**, 114–128.

[4] Selletin J and Mitschang B (1998) Data-intensive intra- & internet applications—Experiences using Java and CORBA in the World Wide Web. *Proceedings of 14th IEEE International Conference on Data Engineering*, 302–311.

[5] Siegel J (1997) *CORBA: Fundamentals and Programming.* Wiley, New York.

[6] Davidson SB, et. al. (1996) BioKleisli: A digital library for biomedical researchers. *International Journal of Digital Libraries*, **1**, 36–53.

[7] Buneman P, et. al. (1995) A data transformation system for biological data sources. *Proceedings of 21st International Conference on Very Large Data Bases*, 158–169.

[8] Buneman P, et. al. (1994) Comprehension syntax. *SIGMOD Record*, **23**, 87–96.

[9] Benton D (1996) Bioinformatics — principles and potential of a new multidisciplinary tool. *Trends in Biotechnology*, **14**, 261–272.

[10] Karp PD (1996) Database links are a foundation for interoperability. *Trends in Biotechnology*, **14**, 273–279.

[11] Burks C, et. al. (1992) GenBank. *Nucleic Acids Research*, **20** Supplement, 2065–2069.

[12] Gasch A, et. al. (1992) Gene isolation with the polymerase chain reaction. *Methods in Arabidopsis Research*, 342–356.

[13] States DJ and Gish W (1994) Combined use of sequence similarity and codon bias for coding region identification. *Journal of Computational Biology*, **1**, 39–50.

[14] Wong L (1998) *The Collection Programming Language Reference Manual.* Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613.

[15] Pang HH, et. al. (1999) S-Hash: An indexing scheme for approximate subsequence matching in large sequence databases. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[16] Nakamura Y, et. al. (1999) Codon usage tabulated from the international DNA databases; its status 1999. *Nucleic Acids Research*, **27**, 292.

[17] Fujibuchi W, et. al. (1998) DBGET/LinkDB: An integrated database retrieval system. *Proceedings of Pacific Symposium on Biocomputing'98*, 683–694.

[18] Baker PG, et. al. (1998) TAMBIS—transparent access to multiple bioinformatics information sources. *Proceedings of 6th International Conference on Intelligent Systems for Molecular Biology*, 25–34.

[19] Chen IMA and Markowitz VM (1995) An overview of the object-protocol model (OPM) and OPM data management tools. *Information Systems*, **20**, 393–418.

[20] Tan WC, et. al. (1998) A graphical interface to genome multidatabases. *Journal of Database Management*, **9**, 24–32.