BIOINFORMATICS

Vol. 17 no. 3 2001
Pages 226–236

Basic Gene Grammars and DNA-ChartParser for language processing of Escherichia coli *promoter DNA sequences*

Siu-wai Leung, Chris Mellish and Dave Robertson

Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, UK

Received on March 13, 2000; revised on October 13, 2000; accepted on November 21, 2000

ABSTRACT

Motivation: The field of 'DNA linguistics' has emerged from pioneering work in computational linguistics and molecular biology. Most formal grammars in this field are expressed using Definite Clause Grammars but these have computational limitations which must be overcome. The present study provides a new DNA parsing system, comprising a logic grammar formalism called Basic Gene Grammars and a bidirectional chart parser DNA-ChartParser.

Results: The use of Basic Gene Grammars is demonstrated in representing many formulations of the knowledge of *Escherichia coli* promoters, including knowledge acquired from human experts, consensus sequences, statistics (weight matrices), symbolic learning, and neural network learning. The DNA-ChartParser provides bidirectional parsing facilities for BGGs in handling overlapping categories, gap categories, approximate pattern matching, and constraints. Basic Gene Grammars and the DNA-ChartParser allowed different sources of knowledge for recognizing *E.coli* promoters to be combined to achieve better accuracy as assessed by parsing these DNA sequences in real-world data sets.

Availability: DNA-ChartParser runs under SICStus Prolog. It and a few examples of Basic Gene Grammars are available at the URL: http://www.dai.ed.ac.uk/~siu/DNA

Contact: {siu,chrism,dr}@dai.ed.ac.uk

INTRODUCTION

The genetic encoding in DNA has often been described using formal systems with origins in computational linguistics (Searls, 1989; Gribskov, 1992; Searls, 1997; Ji, 1999). Formal DNA linguistics research has used finite-state automata, stochastic grammars based on hidden Markov models (Durbin *et al.*, 1998), and grammars based on computational logic (Searls and Noordewier, 1991; Searls, 1989, 1992, 1993, 1997). The logic grammar approach to DNA language analysis involved mainly representing DNA structures in Definite Clause Grammars (DCG) (Pereira and Warren, 1980). Advanced programming features in DCGs such as parameter-passing, procedural attachment, and arbitrary Prolog code embedding (Clocksin and Mellish, 1994) were employed to represent special structural features of DNA. DCGs and Prolog are also useful in modeling gene regulation (Collado-Vides, 1992, 1996; Rosenblueth *et al.*, 1996). The cumulative experience of many research groups in using these sorts of systems suggests that no single one of them is capable of solving the parsing problem on its own. The more specialised formal systems (such as finite-state automata) have difficulty in representing the varied forms of knowledge which appear necessary in obtaining accurate solutions. The more powerful representational systems (such as DCGs which compile to a general purpose programming language, Prolog) raise problems of appropriate knowledge representation and integration. In this paper we describe a system of equivalent representational power to DCGs which overcomes some of these problems.

DCGs, although in theory allowing different styles of processing, are in practice used in a top-down left-to-right manner. When used in this way, they have difficulty in handling the special features of DNA processing such as overlapping syntactic categories. More sophisticated natural language processing techniques are needed for linguistic studies of DNA. The present paper describes *Basic Gene Grammars*, an attempt to simplify the representation of DNA sequence knowledge, and a bidirectional chart parser called the *DNA-ChartParser* to support the development of Basic Gene Grammars. The use of Basic Gene Grammars and the DNA-ChartParser will be demonstrated in representing the knowledge of *Escherichia coli* promoter sequences, which are probably the most studied and cited sequences in molecular biology. Many kinds of formulations of such DNA sequence patterns acquired from human and machine learning can be represented in Basic Gene Grammars and processed by the DNA-ChartParser.

```
lhs_cat ---> rhs_cat_1, ..., rhs_cat_n.
lhs_cat : constraint_formulae --->
             rhs_cat_1, ..., rhs_cat_n.
```

Fig. 1. The syntax of grammar rules.

BASIC GENE GRAMMARS

The development of Basic Gene Grammars was inspired by work using Definite Clause Grammars in representing DNA sequences (Searls, 1989) and further driven by the need to represent some special features of *E.coli* promoter DNA sequences. The term 'grammar(s)' in the following sections refers to Basic Gene Grammar(s) unless otherwise stated.

Grammar rule syntax

Figure 1 shows the syntax of basic grammar rules. A grammar rule consists of a single left-hand-side (LHS) category, an arrow symbol, and one or more right-hand-side (RHS) categories (`rhs_cat_1,..., rhs_cat_n.`). The LHS category and RHS categories are separated by an arrow symbol (e.g. `--->`). One or more optional formulae representing constraints may be added in between the LHS category and the arrow symbol. The operator '`:`' separates the LHS category from the constraint formulae. The constraint formulae are extra conditions for the LHS category to be established.

Syntactic categories

In a grammar rule, the LHS category is a label for the sequence comprising all the RHS categories. Each syntactic category describes a structural and/or functional region of the sequence. Similar to the convention in Prolog (Clocksin and Mellish, 1994), the name of a constant starts with a lower case character. A variable begins with an upper case character.

Basic lexicon. A basic lexicon comprises lexical rules, in which base categories are on the LHS and terminals are on the RHS. The basic lexicon was built in accordance with the International Union of Biochemistry (IUB) Standard Nucleotide Codes. We use `b(Code)`, where `Code` is a nucleotide code, to specify a base category. For instance, `b(a)` specifies base `a`. Lexical rules for base categories other than `b(a)`, `b(c)`, `b(g)`, and `b(t)` are optional. The optional lexical rules can be added or removed for different parsing requirements.

Ordinary and variable categories. An ordinary category is a simple constant symbol (e.g. `regionA`) or a functor with feature arguments (e.g. `regionB(x,y,z)`). The number of feature arguments should be kept minimal for clarity.

If a syntactic category itself or its feature arguments are variables, then we call it a variable category. A variable category can be used to represent an unknown category or the multiple occurrence of the same unknown category in a single grammar rule. The values of variables with an identical name but in separate grammar rules may differ. To represent a variable category of bases, we may use `b(X)` where `X` is a variable. To represent a general variable category, we may use `V`. Examples of grammar rules with variable categories are given below.

```
regionA ---> b(X), b(t), b(X).
regionB ---> V, regionA, V.
tandem_repeat ---> Y, Y.
```

Suppose that there are only four lexical rules for bases, i.e. `a`, `c`, `g`, and `t`. The first grammar rule will accept a three-base DNA sequence in which the first and the third bases are identical (because they share the same variable, `X`) and the second base is `t`. For instance, this grammar will accept sequences such as `ata`, `ctc`, `gtg`, and `ttt` but will not accept `atg`. The second grammar rule will accept `regionB` if `regionA` is found to be flanked by two identical categories. The third grammar rule specifies a tandem repeat pattern, which is composed of two identical categories.

Gap categories. Gaps are parts of the sequence that are not of interest and/or their significance is still not known. Our definition of a gap in the grammar is a subsequence of some length (comprising zero or more bases) flanked by two subsequences of interest. This is the only place that gaps can be indicated. In the grammar, the symbol `gap` is used to specify a gap of indefinite length. When the gap length is known to be within a range, `gap(LowerLimit, UpperLimit)` is preferred. If the exact gap length is known, `gap(Length)` is more elegant than `gap(Length,Length)`. Some examples of grammar rules with the gap categories are given below.

```
intron  ---> b(g), b(t), gap, b(a), b(g).
contact ---> minus_35, gap(14,19), minus_10.
regionA ---> cat_A, gap(10, no), cat_C.
regionB ---> cat_E, gap(no, 20), cat_G.
regionC ---> cat_H, gap(25), cat_I.
```

In the above example, the first rule specifies a gap of indefinite length between `GT` and `AG`. The second rule specifies a gap of 14–19 bases in length. The third rule specifies a gap of at least 10 bases in length. The fourth rule specifies a gap not longer than 20 bases. The last rule precisely specifies a gap of 25 bases in length.

Key categories and overlaps

Basic Gene Grammars represent overlapping categories and indicate the possible distribution of key categories

Table 1. Arrow notations

Arrow symbol	Arrow body	Arrow head
`--->`	`---`	`>`
`===>`	`===`	`>`
`<---`	`---`	`<`
`<===`	`===`	`<`

in DNA by using arrow symbols. There are four types of arrow symbols in the grammar (Table 1). An arrow symbol is composed of two parts, an arrow body and an arrow head. The arrow body indicates whether the RHS categories are possibly overlapping. The arrow head specifies the distribution of possible key categories in the RHS, i.e. more significant categories on the left (`>`) or more significant categories on the right (`<`). The significant categories contribute more essential characteristics and higher confidence than other categories in the RHS for pattern recognition. The chart parser will index each rule on whichever of the leftmost or rightmost categories is indicated as more significant.

A single-line arrow body (`---`) indicates that the syntactic categories on the RHS of a grammar rule are non-overlapping. A double-line arrow body (`===`) suggests that the RHS syntactic categories are possibly, but not necessarily, overlapping in a physical sequence, regardless of the arrow symbol of the grammar rule. In a grammar rule with an arrow `===>`, each RHS category must start at or after the start of the previous one. In a grammar rule with an arrow `<===`, each RHS category must end at or before the end of the next one.

Approximate sequence patterns

DNA sequences may have sequence variations in the same syntactic category. It is possible to represent an approximate sequence pattern in a grammar rule in which only a subset of the RHS base categories have to match.

Simple match and mismatch. To represent an approximate base pattern with its score of matching, we introduce an operator `#`, which indicates the score of matching, into the LHS category. Two example grammar rules are given below:

```
category # Match --->
   b(g), b(n), b(t), b(t), b(n), b(a), b(a).
category # Match : Match >= 4 --->
   b(g), b(n), b(t), b(t), b(n), b(a), b(c).
```

In the above grammar rules, the variable `Match` stores the number of the bases matched to the given sequence pattern in RHS. The value of `Match` indicates how similar are the sequence and the pattern. There is no constraint formula in the first grammar rule because the LHS category can be accepted unconditionally. The constraint `Match >= 4` in the second grammar rule specifies that the `Match` should be greater than or equal to 4 for the LHS category to be accepted. To represent an approximate base pattern with a possible number of base mismatches, an operator `$` is introduced to indicate the penalty of mismatching, into the LHS category.

Best match of bases. While the simple match or mismatch grammars specify the possible number of the bases matched or mismatched to the given sequence pattern, a best match grammar rule specifies that only the best match of the RHS base categories is allowed when base insertions and deletions are allowable. This is the best match starting from the beginning or the end of the sequence, depending on the direction of the arrow. We introduce an operator `&` into the LHS category. Two example grammar rules are given below:

```
category & (std, 15, Best) --->
  b(g), b(n), b(t), b(t), b(n), b(a), b(c).
category & (std, 15, Best) : Best > 6 --->
  b(g), b(n), b(t), b(t), b(n), b(a), b(c).
```

In the above grammar rule, the matching score scheme (`std`) and the maximum length (`15`) of the final candidate sequence are specified. The variable `Best` stores the number of the nucleotides matched to the given sequence pattern in the RHS. The variable `Best` is regarded as additional information in the first grammar rule. The constraint `Best > 6` in the second grammar rule specifies that the `Best` should be greater than 6 when the LHS category is established.

As an approximate sequence will probably not be the same as the given sequence pattern in terms of length (because of insertions or deletions), the maximum length of the allowable sequence should be given. We can specify one of a variety of matching score schemes for best match, similar to the score schemes used in sequence alignment by dynamic programming. To specify a matching score scheme in a grammar, we need to add a predicate as exemplified below:

```
% scheme(+Name,+MatchWeight,+MismatchWeight,
%        +InsWeight,+DelWeight).

scheme(std,  2,  0, -1, -1).
```

In a matching score scheme, we should specify the name of the scheme (`Name`) and the weights for matches (`MatchWeight`), mismatches (`MismatchWeight`), insertions (`InsWeight`), and deletions (`DelWeight`) of bases.

Overlapping approximate categories. To represent multiple approximate sequence categories that are overlapping

with one another, we need to write an independent rule for each approximate category and then combine overlapping categories by an additional rule using a double-line arrow symbol. For example,

```
cat1#X : X>2 ---> b(a), b(n), b(c), b(n), b(a).
cat2#Y : Y>3 ---> b(t), b(n), b(g), b(n), b(t).
siteA  : A+B>8 ===> cat1#A, cat2#B.
```

`cat1#A` and `cat2#B` are overlapping approximate cateogries in `siteA`. Their simple match scores are the criteria for accepting category `siteA` in the third grammar rule.

Constraints

Constraints, which determine whether the LHS category can be established, may have multiple variables for evaluation:

```
siteA : A+B <5 --->
        b(g), cat1$A, b(t), cat2$B, b(c).
```

In the above example, `A+B < 5` is a condition for establishing the syntactic category `siteA`. The values of `A` and `B` will be taken from `cat1$A` and `cat2$B` respectively. During parsing, the equation `A+B < 5` will be evaluated when all the RHS categories have been found. To handle complicated evaluation, it is possible to have multiple constraint formulae in Prolog syntax. For example,

```
siteA : (A+B < 5, P+Q > 6) --->
        cat1$A, cat2$B, cat3$P, cat4$Q.
```

Positional information

Information about the absolute position of one or more categories in a DNA sequence can be important for recognizing a syntactic category. We may use position variables to retrieve positional information and require position values as a kind of constraint. A position variable or value is preceded by an operator '`@`', which means 'located *at*'.

```
siteA ---> @10, cat2, cat3, @30.
siteA ---> @P1, cat2, cat3, @P1+20.
siteA : P2-P1<25 --->
           @P1, cat2, cat3, @P2.
```

In the above example, the first grammar rule specifies the absolute positions of the start and end points of `siteA` in the DNA sequence. The second grammar rule only defines the length of `siteA` to be 20, not the absolute positions of the starting and ending points. The third grammar rule relaxes the positional constraints but stipulates a range of possible lengths (less than 25 bases) for `siteA`.

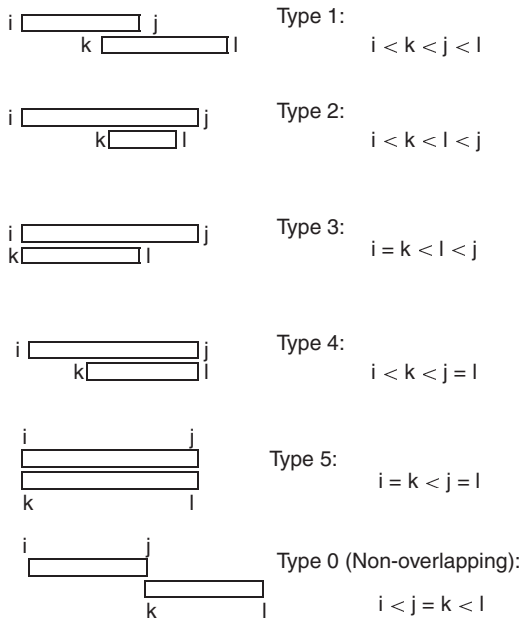

Fig. 2. Some possible types of overlapping and the relative positions of the overlapping categories.

The positional information is also useful in specifying particular ways categories can overlap. For instance, Type I overlapping as illustrated in Figure 2 can be specified as follows:

```
siteA(I,J) ---> @I, cat1, @J.
siteB(K,L) ---> @K, cat2, @L.
region1 :(K>I, J>K, L>J) ===>
      siteA(I,J), siteB(K,L).
```

DNA-CHARTPARSER

Chart parsing stores intermediate parsing data in a *chart* to avoid redundant searches (Gazdar and Mellish, 1989). Its parsing algorithm can be viewed as a kind of dynamic programming, which is widely used in sequence alignment. We extended the Gazdar and Mellish chart parser (Gazdar and Mellish, 1989) to develop the DNA-ChartParser. This section describes chart recognition, which is easily supplemented with additional facilities to support parsing.

The chart and agenda

The *chart* of a chart parser is a modified well-formed substring table (WFST) including not only the intermediate results but also the hypotheses for parsing. While the chart accumulates the intermediate data during chart parsing, an *agenda* temporarily stores the data that are still to be added to the chart. The data in the agenda are arranged according to their priority for processing. A chart is usually represented as a set of structures, each of which has the

following attributes:

<Start, Finish, Label, Found, ToFind>

This structure is called an *edge*. The attributes Start and Finish respectively store the starting and ending chart positions. The chart positions are equivalent to the positions in the string, i.e. DNA sequence. Label is equivalent to the LHS of a particular grammar rule. Found is the sequence of RHS categories which have been found. ToFind is the sequence of RHS categories which are being looked for. We can also represent an edge in the following structure:

<Start, Finish, Label → Found . ToFind>

where the edge attribute Label is equivalent to the LHS of a grammar rule. Found is the RHS category set recording the categories which have been found. ToFind is the RHS category set recording the categories which are still to be found so that the LHS category can be established.

An edge with an empty ToFind is called an *inactive (complete) edge*. An edge with a non-empty ToFind is called an *active (incomplete) edge*.

To support parsing in a preferred direction corresponding to the significance order of key categories, the DNA-Chart Parser adds an attribute Arrow to the basic edge structure. Hence, each edge in the DNA-ChartParser consists of six attributes:

<Start, Finish, Label, Arrow, Found, ToFind>

where the order of attributes Found and ToFind depends upon the direction of Arrow. The possible values of Arrow are →, ⇒, ←, ⇐, and ↔. For the active edges indicating a left-to-right parsing direction, the notations are as follows:

<Start, Finish, Label → Found . ToFind>

Since an inactive edge can be used in both the left-to-right and right-to-left parsing, we use a bidirectional arrow '↔' in this case:

<Start, Finish, Label ↔ AllRHSCategories>

In an inactive edge, all the RHS categories are found.

Fundamental rules of chart parsing

To make use of the chart data, we need to apply the fundamental rules of chart parsing. The fundamental rule of left-to-right chart parsing is as follows:

Fundamental Rule: Left-to-Right
If the chart contains edges
<i, j, A → W1 . B W2> and
<j, k, B ↔ W3>,
where A and B are categories and W1, W2 and W3 are sequences of zero or more categories, then add edge:
<i, k, A → W1 B . W2>
to the agenda.

We can represent the edge structure as follows:

```
edge(Start,Finish,Label,Arrow,Found,ToFind).
```

where `Start` and `Finish` are respectively the starting and ending positions of the found fragement of a category `Label`. In the edge notation, the subcategories before the dot (.) have been found and the subcategories after the dot (.) is still to be found to fully establish the complete category. To represent the dot, we use two separate lists, i.e. `Found` and `ToFind`. The found subcategories are stored in `Found` list while the subcategories to be found are stored in `ToFind` list. For example, according to the fundamental rule of chart parsing, if the chart contains the following two edges:

```
edge(10, 14, region1, --->,
     [[10,cat1,14]], [siteA,siteB]).
edge(14, 18, siteA, <-->,
     [[14,cat2,16],[16,cat3,18]],[]).
```

then the following edge is added to the chart:

```
edge(10, 18, region1, --->,
     [[10,cat1,14],[14,siteA,18]],[siteB]).
```

In this way, every step of the parsing process can be recorded as an edge in the chart.

The right-to-left rule of chart parsing is a mirror image of the left-to-right rule. Details about the mechanism of chart parsing can be found in books on natural language processing (Gazdar and Mellish, 1989).

Process of DNA chart parsing

The present DNA-ChartParser provides bottom-up chart parsing and breadth-first search. The category matching in the DNA-ChartParser is based on the Prolog unification mechanism. Before initialization of chart parsing, both the chart and agenda are usually empty. At initialization, the inactive edges representing each base category in the input DNA sequence will be generated during lexical lookup and added to the agenda thereafter.

The chart parser adds one agenda edge at a time to the chart according to their order appearing in the agenda, until the agenda becomes empty. An agenda edge is only added to the chart if it is not subsumed by an existing chart edge.

Upon the addition of an inactive edge to the chart, for each active chart edge that is waiting for something matching the category of the newly added inactive edge, the fundamental rule of chart parsing is applied to add

a resulting edge to the agenda. In addition, for each grammar rule with arrow `--->` or `===>` where the leftmost RHS category matches with the label of the newly added edge, an active edge is added to the agenda; for each grammar rule with arrow `<---` or `<===` where the rightmost RHS category matches the label of the newly added edge, an active edge is added to the agenda.

Upon the addition of an active edge to the chart, for each inactive edge that matches the category required by the newly added active edge, the fundamental rule of chart parsing is applied to add a resulting edge to the agenda.

Parsing gap categories

We use bounded gap categories to illustrate the parsing of gap categories. A bounded gap category is a gap category with specific lower and upper limits of gap length. The left-to-right parsing techniques for combining two categories with an intervening bounded gap are as follows:

If the chart contains edges
$<$i, j, A $\rightarrow$ W1 . gap(Lower,Upper) B W2$>$ and
$<$k, l, B $\leftrightarrow$ W$>$
where Lower $\leqslant$ k – j $\leqslant$ Upper, then add edges:
$<$i, l, A $\rightarrow$ W1 gap(Lower,Upper) B . W2$>$ and
$<$j, k, gap $\leftrightarrow$ gap$>$
to the agenda.

The right-to-left parsing is a mirror structure of the left-to-right parsing.

Parsing overlapping categories

Overlapping categories are syntactic categories that are overlapping partially or completely. A method of combining two overlapping categories in the left-to-right direction is as follows:

If the chart contains edges:
$<$i, j, A $\Rightarrow$ W1 . B W2$>$ and
$<$k, l, B $\leftrightarrow$ W3$>$,
where (1) A and B are categories, (2) W1, W2 and W3 are (possibly empty) sequences of categories, (3) i $\leqslant$ k $\leqslant$ j, and (4) m is the maximum value of j and l, then add edge:
$<$i, m, A $\Rightarrow$ W1 B . W2$>$
to the agenda.

The right-to-left parsing has a mirror structure to the left-to-right parsing.

Approximate pattern matching

Approximate Pattern Matching (APM) allows mismatches, insertions, and deletions of bases; thus, the fundamental rules of chart parsing for perfect matching are not applicable. By definition, every base category is eligible to invoke an APM rule because every subsequence is a possible candidate of an approximate sequence pattern. To avoid doing unnecessary APM, the APM rules are invoked only when a category adjacent to an approximate sequence pattern has been found. Best match scoring is explained to exemplify APM. As we use breadth-first search, the edges for all the bases are available in the chart during best matching. The best match score is calculated according to a basic sequence match algorithm (Pearson and Miller, 1992), which gives the best possible similarity score in compliance with the set of match, mismatch, insertion, and deletion scoring parameters. For left-to-right parsing:

If the chart contains an edge:
$<$i, j, A $\rightarrow$ W1 . X&Y W2$>$ and
there is a grammar rule
X&(Scheme,MaxLen,Y) : constraints $\rightarrow$ W
and l is the length of the DNA subsequence matched with W after doing best matching and l $\leqslant$ MaxLen and r is the best match score W against the sequence from j to j+l according the scoring Scheme, and the constraints are satisfiable, then add edges:
$<$i, j+l, A $\rightarrow$ W1 X&r . W2$>$ and
$<$j, l, X&r $\leftrightarrow$ W$>$
to the agenda.

The right-to-left parsing has a mirror structure to the left-to-right parsing.

Evaluation of constraint formulae

In the notation for an edge, the constraint formulae attaches to the LHS category. The constraint formulae are evaluated just before adding the relevant inactive edge into the agenda to ensure that all the edges added to the agenda are eligible for further processing.

Evaluation of positional information

Positional information is stored in a dotted rule in the form '`@X`', where '`@`' is an operator and `X` can be an integer or a variable. For processing a left-to-right edge having an '`@X`' as the first data item of the attribute ToFind, `X` will be unified with the value of Finish. For processing a right-to-left edge having an '`@X`' as the first data item of the attribute ToFind, `X` will be unified with the value of Start. If the unification is successful, the instantiated edge will be added to the agenda.

GRAMMARS OF *E.COLI* PROMOTERS

This section exemplifies the use of BGGs in representing knowledge of some DNA sequences, particularly the sequences of *E.coli* promoters. The knowledge to be represented is obtained from human and machine empirical learning.

```
promoter ===> contact, conformation.
contact  ---> minus_35, gap(15,19), minus_10.
minus_35 ---> b(t),b(t),b(g),b(a),b(c),b(a).
minus_10 ---> b(t),b(a),b(t),b(a),b(a),b(t).
conformation ---> . . . .
```

Fig. 3. A partial grammar of consensus sequences.

Features of *E.coli* promoters

A promoter enables the initiation of a gene expression after binding with an enzyme called RNA polymerase, which moves bidirectionally in searching for a promoter and starts making RNA according to the DNA sequence at the transcription initiation site following the promoter (Mishra and Chatterji, 1993; Lewin, 2000). The most significant patterns in *E.coli* promoter sequences are the −10 and −35 regions, which are approximately at the region of 10 bases and 35 bases before the transcription initiation site. The spacing (gap) between the −10 and −35 regions is not fixed, ranging from 15 to 19 bases. The −35 and −10 sequences together are the contact region for RNA polymerase.

Grammars of consensus sequences

A consensus sequence represents the common patterns of a group of aligned sequences. The consensus sequences for the −35 region and the −10 region of the *E.coli* promoter are respectively `TTGACA` and `TATAAT`. A Basic Gene Grammar making use of these consensus sequences of the *E.coli* promoters is shown in Figure 3.

Despite the fact that a consensus sequence does represent a common pattern for the group of sequences it is not useful for the recognition of variable DNA sequence patterns unless a facility for approximate pattern matching can be provided (Mehldau and Myers, 1993).

To do approximate pattern matching, we need to find the *best match score* for each possible subsequence and then compare the score to a cutoff value (threshold) when applying grammar rules. The calculation of the best match score is based on a pre-defined scoring scheme which we illustrate using a hypothetical example. Suppose we would like to use a scoring scheme '`ecoli`', in which the score for each *match*, *mismatch*, *insertion*, and *deletion* is respectively 12, −1, −9, and −9, as stipulated in the following predicate:

```
scheme(ecoli, 12, -1, -9, -9).
```

A grammar rule for representing the consensus sequence of the −10 region may be written as follows:

```
minus_10 & (ecoli,7,Score):Score>=20 --->
      b(t), b(a), b(t), b(a), b(a), b(t).
```

The above grammar rule specifies that the best match score `Score` is calculated according to the scoring scheme `ecoli`; the maximum length of the qualified sequence is 7; the best match score (`Score`) must be equal to or greater than 20; and the consensus sequence is `TATAAT`.

Grammars of weight matrices

A weight matrix, also called a frequency matrix, is more informative than a consensus sequence in representing a DNA sequence pattern. A weight matrix is a two-dimensional representation of a sequence pattern where one axis represents the position in the pattern, and the other axis provides the frequency of occurrence of each of the nucleotides at each position in the pattern (Rice *et al.*, 1991).

One method for obtaining an estimated score is by performing a calculation according to the following equation (Harr *et al.*, 1983):

$$\text{Score} = \prod_{i=1}^{m} \frac{n_i}{a_i} = \frac{n_1 \times n_2 \times \cdots \times n_{m-1} \times n_m}{a_1 \times a_2 \times \cdots \times a_{m-1} \times a_m}$$

where n_i is the score for base at position i taken from the weight matrix, a_i is the score for the most frequent base at position i, and m is the number of bases in the sequence pattern. The equation gives a value of 1 for a perfect match.

To do this calculation by using Basic Gene Grammars, we need to have a category that can carry a weight for each base occurrence. A calculation performed on the weights of the bases within a particular region will give an estimated score. If the estimated score is greater than the pre-defined cutoff value, then the region is classified as a qualified region. A possible category is as follows:

```
freq(Position, Weight)
```

where `Weight` stores the weight (or frequency ratio) of a base at a particular position (`Position`). To use the method suggested in Harr *et al.* (1983), we use this special category in the grammar rules to represent a weight matrix provided in Lisser and Margalit (1993). The grammar rules in Figure 4 illustrate a possible representation of the frequency ratios of bases in the −10 region. These grammar rules represent the probabilities of base existence in the −10 region of the *E.coli* promoters. Each of these grammar rules suggests the frequency ratio (the second argument of the LHS category) of a particular base at a specific position (the first argument of the LHS category) in the −10 region. To obtain the frequency ratios as suggested in Harr *et al.* (1983), we calculate the ratio of the frequency of a particular base at a specific position to the highest frequency at that position. For example, the frequency of base `A` at the first position (`m10_pos1`) of the

```
freq(m10_pos1,  5/77) ---> b(a).
freq(m10_pos2, 76/76) ---> b(a).
. . .
freq(m10_pos1, 10/77) ---> b(c).
freq(m10_pos2,  6/76) ---> b(c).
. . .
freq(m10_pos1,  8/77) ---> b(g).
freq(m10_pos2,  6/76) ---> b(g).
. . .
freq(m10_pos1, 77/77) ---> b(t).
freq(m10_pos2, 12/76) ---> b(t).
. . .
```

Fig. 4. A grammar of a weight matrix.

−10 region is 5 and the most frequent base (i.e. base T) at such position is 77. The frequency ratio of the base A to the most frequent base T is 5/77. The other grammar rules for storing other base occurrence weights can be written in the same way.

To calculate a score of a possible candidate of the −10 region and compare the score to a cutoff value, we can use a grammar rule with a constraint formula as follows:

```
minus_10 : A*B*C*D*E*F>=0.002 --->
   freq(m10_pos1,A), freq(m10_pos2,B),
   freq(m10_pos3,C), freq(m10_pos4,D),
   freq(m10_pos5,E), freq(m10_pos6,F).
```

where the cutoff value is assumed to be 0.002 and the score is the product of the frequency ratios of the actual bases within the −10 region.

Grammars of extracted knowledge

Some machine learning approaches such as neural networks are powerful in their ability to acquire knowledge by learning from data but their knowledge representation is difficult to comprehend and manipulate by humans. Knowledge extraction from such machine learning systems can facilitate explicit representation and processing (including combination) of knowledge, e.g. in logic grammars. In this section we show how the output from several different machine learning systems can be represented in our BGG notation. In the next section we take advantage of this to perform parsing using combined grammars.

Knowledge-based neural networks. Previous efforts in using neural networks to recognize *E.coli* promoters by perceptrons gave accuracy rates ranging from 75 (Nakata *et al.*, 1988) to 80% (Horton and Kanehisa, 1992). By using a multi-layer perceptron with a hidden layer, 92% of the *E.coli* promoter sequences were accurately recognized (Shavlik *et al.*, 1992).

The internal decision procedure of trained neural networks is not directly accessible by human users. Extracting rules from the trained neural networks can assist human inspection of the implicitly encoded knowledge. Where this is possible, we would expect that the knowledge extracted from a neural network could be represented by grammars and processed in a parsing system.

```
promoter <--- b(t), gap(1), b(b), b(h),
              gap(20), b(h), gap(9),
              b(v), gap(1), start.
promoter <--- b(g), b(h), gap(20), b(w),
              gap(11), start.
```

Fig. 5. A grammar of induce-net extracted rules.

The Knowledge-based Artificial Neural Network (KBANN) approach not only extracts rules from a neural network but also makes use of domain knowledge (Towell and Shavlik, 1993). Using a rules-to-network translator, KBANN uses the domain theory for constructing an initial network topology. After training the neural network with a number of positive and negative examples, KBANN uses a network-to-rules translator to extract rules from the trained neural networks.

An example for a KBANN extracted rule (with simplified notation) is as follows:

```
minus10  if  1.5 < nt('CA---T').
```

where the function 'nt' in the KBANN rules counts the number of perfectly matched bases, which is equivalent to simple approximate matching in Basic Gene Grammars.

The sequence pattern 'CA---T' can be represented in the following *simple match* grammar rule:

```
minus10 # X : X>1.5  --->
  b(c), b(a), b(x), b(x), b(x), b(t).
```

In the above grammar rule, the operator # indicates that the rule is a *simple match* rule. The variable X stores the simple match score as a result of doing a simple approximate pattern matching. Unlike the category b(n), the category b(x) is a special category that will not be counted into the simple match score. The constraint 'X > 1.5' is satisfiable only when two or more bases on DNA are found matched with the base categories in this sequence pattern, excluding the category b(x).

Besides KBANN, Induce-Net has also been used in the recognition of *E.coli* promoters. Induce-Net is a neural network model for inducing symbolic knowledge from examples by exploiting the certainty-factor-based activation function (Fu, 1999). The rules extracted from trained Induce-Net can be represented in Basic Gene Grammars (Figure 5).

HCV Rule-induction. As the HCV rule-induction algorithm (Wu, 1992, 1993) gave higher accuracy in recognizing *E.coli* promoters than the reported result of the better

```
promoter <--- b(v), gap(7) , b(k), b(b),
              b(k), gap(20), b(r), gap(12),
              start.
promoter <--- b(k), gap(1) , b(b), gap(2),
              b(d), gap(18), b(h), gap(9),
              b(v), gap(1) , start.
promoter <--- b(t), gap(26), b(t), gap(4),
              b(t), gap(6), start.
```

Fig. 6. A grammar of HCV rules.

known ID3 rule-induction algorithm (Shavlik *et al.*, 1992), we consider HCV induced rules for our grammars. A set of grammar rules based on the HCV *if–then* rules for recognizing a small set of promoter sequences obtained from the Machine Learning Repository, University of California at Irvine, are shown in Figure 6. The grammar for a large set of promoter sequences (Lisser and Margalit, 1993) has also been obtained.

Classification and regression trees. Classification and Regression Tree (CART) analysis was used for recognizing the *E.coli* promoter sequences and a classification result similar to decision trees was obtained (Walker, 1992). The −10 region of an *E.coli* promoter is assumed to be the DNA subsequence from position −13 to position −8. For instance, the CART tree will classify a DNA sequence as a promoter if there are base T at position −13 and base A at position −12 of the DNA sequence. Each path through the CART tree from the topmost node to a promoter or non-promoter terminal node can be regarded as a rule or a rule chain for classification. The CART tree was generated according to a set of *E.coli* promoter sequences which were pre-aligned to reveal the −10 region. A CART tree can be represented in grammar rules with similar features of the HCV grammars.

Combined grammars for recognition

Many formulations of explicit knowledge for recognizing *E.coli* promoter sequences can be represented simply by Basic Gene Grammars, without needing machine learning software or requiring tedious programming. The grammars enable us to parse a large real-world data set (Lisser and Margalit, 1993), which we used as unseen examples to test the effectiveness of learning systems. Machine learning approaches such as HCV, KBANN, Induce-Net achieved higher than 95% in the accuracy on the data sets used for development but not for the unseen real-world examples, for which only 54–57% accuracy was achieved. Table 2 shows our results for KBANN (omitting HCV and InduceNet which were similar). These parsings reflect the inadequacy of human/machine learning methods (and data) for recognizing *E.coli* promoters. Since individual grammars perform poorly but we have (in BGGs) a single system for representing them all, it is possible to experiment with combinations of grammar rules from different sources to see if diversity improves performance. Combination may either be disjunctive (recognition by one single grammar is sufficient) or conjunctive (recognition requires agreement by all grammars). Both can be investigated with either whole grammars or portions of grammars. As expected, disjunctively combined grammars achieve better sensitivity while conjunctively combined grammars achieve better specificity and predictive values. We found that the disjunctive combination of KBANN, Weight Matrix, and 35W_10K grammars achieves better accuracy than individual grammars (Table 2).

Parsing efficiency

This work has concentrated on supporting expressive power in the grammar formalism rather than optimising the implementation exhaustively. A standard chart parser achieves polynomial (rather than exponential) complexity in the length of the string because there is a fixed number of possible edges that can be stored between any two chart positions, adding an edge only leads to a polynomial number of new edges being considered and the new implications of an edge are only considered once. As long as the feature arguments of categories all have a fixed set of possible values that increases in size at most in a way that is polynomial in the length of the string, our extensions maintain all of the properties of a standard chart parser, and so retain polynomial complexity. For simple grammars like CART and HCV, parsing a sequence in the dataset took one or two seconds of CPU time (on a Dell Inspiron 3700 Pentium III-500 notebook computer) when running SICStus Prolog under Linux. While the processing of overlapping categories and approximate pattern matching will lower the efficiency of chart parsing, the performance is improved by finding key categories in a preferred direction. For KBANN grammars which contain overlapping categories and approximate pattern matching, parsing a sequence in the dataset took nearly 1 min. For parsing with all individual and combined grammars of weight matrix and KBANN in a batch (Table 2), processing a sequence took 125.64 ± 0.59 s of CPU time. The processing of gap categories has no adverse effect on the overall performance. The use of arbitrary Prolog code in constraint formulae and the variable categories with arguments should be kept minimal in order to avoid performance degradation of the chart parsing. Although the implementation of the DNA-ChartParser has not been optimised for parsing large sequence databases, the DNA-ChartParser is practical for significantly sized tasks such as testing different grammars in the recognition of *E.coli* promoters.

Table 2. Parsing result of 300 *E.coli* promoters and ten sets of 300 non-promoter random DNA sequences. 35W_10K is a conjunctive combination of the −35 region category specified in Weight matrix (W) grammar and the −10 region category specified in KBANN (K) grammar. K/W/35W_10K grammar is a disjunctive combination of the KBANN, Weight matrix, and 35W_10K grammars. Specificity is a measure of the incidence of negative results in testing all the non-promoter sequences, i.e. (True Negatives/(False Positives + True Negatives)) × 100. Sensitvity is a measure of the incidence of positive results in testing all the promoter sequences, i.e. (True Positives /(True Positives + False Negatives)) × 100. Accuracy is measured by the number of correct results, the sum of true positives and true negatives, in relation to the number of tests carried out, i.e. ((True Postives + True Negatives)/Total) × 100. The predictive value of a positive test is the measure of all positive results which are true positives, i.e. (True Positives /(True Positives + False Positives)) × 100. The data are expressed as average percentage ± standard deviation. As the same full set of positive examples was used in all trials with different set of negative examples, the standard deviation of Sensitivity is zero.

Grammars	Specificity	Sensitivity	Accuracy	Predictive value
KBANN (K)	96.87 ± 0.69	15.67 ± 0.00	56.37 ± 0.29	83.44 ± 3.02
Weight matrix (W)	94.17 ± 1.42	49.33 ± 0.00	71.75 ± 0.71	89.48 ± 2.27
35W_10K	87.16 ± 2.16	46.67 ± 0.00	66.92 ± 1.08	78.53 ± 2.98
K/W/35W_10K	82.23 ± 2.31	68.67 ± 0.00	75.45 ± 1.18	79.50 ± 2.23

SIGNIFICANCE OF THE WORK

Basic Gene Grammars were found useful in representing many kinds of non-trivial human and machine devised knowledge of DNA sequences, particularly the sequences of *E.coli* promoters. The present study is the first demonstration of a single grammar formalism that is able to represent so many kinds of knowledge about *E.coli* promoters, including human-devised domain theory, weight matrices, approximate pattern matching, and the knowledge discovered by symbolic and neural network learning. Combinations of multiple grammars achieve either higher sensitivity or higher specificity (but not both) for *E.coli* promoter recognition. We also found a specific combination of grammars achieving better accuracy than any single grammar.

ACKNOWLEDGEMENTS

We thank LiMin Fu for the induce-net extracted rules, Hanah Margalit for the *E.coli* promoter sequences, Jude Shavlik for the KBANN extracted rules, and Xindong Wu for the HCV software. Their material and advice were very useful for our parsing experiments.

REFERENCES

Clocksin,W. and Mellish,C. (1994) *Programming in Prolog*. 4th edn, Springer, Berlin.

Collado-Vides,J. (1992) Grammatical model of the regulation of gene expression. *Proc. Natl Acad. Sci. USA*, **89**, 9405–9409.

Collado-Vides,J. (1996) Integrative representations of the regulation of gene expression. In Collado-Vides,J., Magasanik,B. and Smith,T. (eds), *Integrative Approaches to Molecular Biology.* MIT Press, Boston, MA, pp. 179–203.

Durbin,R., Eddy,S., Krogh,A. and Mitchison,G. (1998) *Biological Sequence Analysis*. Cambridge University Press, Cambridge.

Fu,L. (1999) Knowledge discovery by inductive neural networks. *IEEE Trans. Knowl. Data Eng.*, **11**, 992–998.

Gazdar,G. and Mellish,C. (1989) *Natural Language Processing in Prolog*. Addison-Wesley, Reading, MA.

Gribskov,M. (1992) The language metaphor in sequence analysis. *Computers Chem.*, **16**, 85–88.

Harr,R., Haggstrom,M. and Gustafsson,P. (1983) Search algorithm for pattern match analysis of nucleic acid sequences. *Nucleic Acids Res.*, **11**, 2943–2957.

Horton,P. and Kanehisa,M. (1992) An assessment of neural network and statistical approaches for prediction of *E.coli* promoter sites. *Nucleic Acids Res.*, **20**, 4331–4338.

Ji,S. (1999) The linguistics of DNA: words, sentences, grammar, phonetics, and semantics. *Ann. New York Acad. Sci.*, **870**, 411–417.

Lewin,B. (2000) *Genes VII*. Oxford University Press, Oxford.

Lisser,S. and Margalit,H. (1993) Compilation of *E.coli* mRNA promoter sequences. *Nucleic Acids Res.*, **21**, 1507–1516.

Mehldau,G. and Myers,G. (1993) A system for pattern matching applications on biosequences. *Comput. Appl. Biosci.*, **9**, 299–314.

Mellish,C. (1989) Some chart based techniques for parsing ill-formed input. In *Proceedings of 27th Annual Meeting of the Association of Computational Linguistics,* pp. 102–109.

Mishra,R. and Chatterji,D. (1993) Promoter search and strength of a promoter: two important means for regulation of gene expression in *Escherichia coli*. *J. Biosci.*, **18**, 1–11.

Murakami,K. and Takagi,T. (1998) Gene recognition by combination of several gene-finding programs. *Bioinformatics*, **14**, 665–675.

Nakata,K., Kanehisa,M. and Maizel,J. (1988) Discriminant analysis of promoter regions in *Escherichia coli* sequences. *Comput. Appl. Biosci.*, **4**, 367–371.

Pearson,W. and Miller,W. (1992) Dynamic programming algorithms for biological sequence comparison. In Brand,L. and Johnson,M. (eds), *Methods in Enzymology,* Vol. 210, Academic Press, New York, pp. 575–601.

Pereira,F. and Warren,D. (1980) Definite clause grammars for language analysis. *Artif. Intell.*, **13**, 231–278.

Rice,P., Elliston,K. and Gribskov,M. (1991) DNA. In Girbskov,M. and Devereux,J. (eds), *Sequence Analysis Primer,* chapter 1, Stockton Press, pp. 1–59.

Rosenblueth,D., Thieffry,D., Huerta,A., Salgado,H. and Collado-Vides,J. (1996) Syntactic recognition of regulatory regions in *Escherichia coli*. *Comput. Appl. Biosci.*, **12**, 415–422.

Searls,D. (1989) Investigating the linguistics of DNA with definite clause grammars. In Lusk,E and R.,O. (eds), *Logic Programming: Proceedings of the North America Conference on Logic Programming,* Vol. 1, Association for Logic Programming, pp. 189–208.

Searls,D. (1992) The linguistics of DNA. *Am. Sci.*, **80**, 579–591.

Searls,D. (1993) The computational linguistics of biological sequences. In Hunter,L. (ed.), *Artificial Intelligence and Molecular Biology,* chapter 2, MIT Press, Boston, MA, pp. 47–120.

Searls,D. (1997) Linguistic approaches to biological sequences. *Bioinformatics*, **13**, 333–344.

Searls,D. and Noordewier,M. (1991) Pattern-matching search of DNA sequence using logic grammars. In *Proceedings of 7th Conference on Artificial Intelligence Applications,* pp. 3–9.

Shavlik,J., Towell,G. and Noordewier,M. (1992) Using neural networks to refine existing biological knowledge. *Int. J. Genome Res.*, **1**, 81–107.

Towell,G. and Shavlik,J. (1993) Extracting refined rules from knowledge-based neural networks. *Mach. Learn.*, **13**, 71–101.

Walker,M. (1992) Probability estimation for classification trees and DNA sequence analysis, *PhD Thesis*, Department of Computer Science and Medicine, Stanford University.

Wu,X. (1992) HCV user's manual. *Technical Paper No. 9*, Department of Artificial Intelligence, University of Edinburgh.

Wu,X. (1993) Knowledge acquisition from data bases, *PhD Thesis*, Department of Artificial Intelligence, University of Edinburgh.