sestoft@dina.kvl.dk

Stochastic grammars and RNA

In a nutshell

- Primary sequence structure is described by hidden Markov models
- Hidden Markov models are stochastic regular grammars
- Context free grammars generalize regular grammars
- Secondary sequence structure (of RNA) is described by stochastic context free grammars

Plan

- Grammars, derivation, parsing, and the Chomsky hierarchy
- Regular grammars and finite state automata
- Context free grammars and reconstruction of derivation trees
- Stochastic regular grammars
- Stochastic context free grammars; the CYK algorithm

A grammar $G = (N, T, R, S)$

$$
\begin{array}{lcl}
S & \longrightarrow & aS \\
S & \longrightarrow & bS \\
S & \longrightarrow & \epsilon
\end{array}
$$

Nonterminal symbols $N = \{S\}$, terminal symbols $T = \{a, b\}$, a start symbol S, and rules R (above).

In biosequence analysis, terminal symbols will be nucleotides or proteins, e.g. $T = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$.

Rules are sometimes called *productions*.

Rules may be written compactly: $S \longrightarrow aS \mid bS \mid \epsilon$ or in Backus-Naur form: $\langle S \rangle ::= a\langle S \rangle \mid b\langle S \rangle \mid \langle empty \rangle$

Derivation, and the language generated by a grammar

Derivation generates a sequence of terminal symbols. Example: $S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abb$

We may write $S \Rightarrow^* abb$ for derivation in zero or more steps.

The *language* generated by a grammar G is $L(G) = \{\alpha \in T^* \mid S \Rightarrow^* \alpha\}$ where S is the start symbol of G.

Example: $L(G) = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$

In biosequence analysis, a 'language' will be a set of DNA or protein sequences.

Grammar classes: the Chomsky hierarchy (ca. 1956)

By restricting the possible form of grammar rules, we get a hierarchy of increasingly powerful grammar classes:

Grammar class	Admissible rules
Regular grammars	$W \longrightarrow aW$ and $W \longrightarrow a$
Context free grammars	$W \longrightarrow \beta$
Context sensitive grammars	$\alpha_1 W \alpha_2 \longrightarrow \alpha_1 \beta \alpha_2$, non-contracting
Unrestricted grammars	$\alpha_1 W \alpha_2 \longrightarrow \gamma$

Above, α, β etc. stand for arbitrary sequences of terminals and nonterminals.

Notes

- Regular grammars and context free grammars are widely used in computer science
- Programming languages are described by the so-called LR or LL subclasses of context free grammars
- Only part of programming language syntax is described by the LR-grammar (e.g. type rules are not).
- Unrestricted grammars are also called term rewrite systems
- Chomsky's original goal was to describe natural languages. He (and everybody else) failed.

Regular grammars and finite state automata

The FMR-1 triplet region contains repeats of the triplets `CGG` and occasionally `AGG`, e.g.

`GCG CTG, GCG CGG CTG, GCG CGG CGG CTG, GCG CGG CGG CTG, ...`

Such regions can be described by the grammar

$$
\begin{array}{lcl}
S & \longrightarrow & gW_1 \\
W_1 & \longrightarrow & cW_2 \\
W_2 & \longrightarrow & gW_3 \\
W_3 & \longrightarrow & cW_4 \\
W_4 & \longrightarrow & gW_5 \\
W_5 & \longrightarrow & gW_6 \\
W_6 & \longrightarrow & aW_4 \mid cW_4 \mid cW_7 \\
W_7 & \longrightarrow & tW_8 \\
W_8 & \longrightarrow & g
\end{array}
$$

The language generated by a regular grammar may be recognized by a finite state automaton (and vice versa).

Regular grammars and regular expressions

It is customary to use *regular expressions* as a shorthand for regular grammars.

Every regular expression corresponds to a regular grammar and vice versa.

The FMR-1 grammar can be written as the regular expression:

$$gcgcgg((a+c)gg)^*ctg$$

In UNIX grep or emacs notation, that is:

```
gcgcgg([ac]gg)*ctg
```

More regular expressions: PROSITE patterns

Regular expressions are used to describe 'signature' conserved protein sequences and their variants.

Brackets `[RK]` indicate choice, braces `{EDRKHPCG}` choice from the complement, `x` matches anything.

(Figure 9.3)

Expressive power in the Chomsky hierarchy

Let $\alpha, \beta \in (N \cup T)^*$ denote arbitrary sequences of nonterminals and terminals.

- Context sensitive grammars can describe copy languages, such as:

  $$\{\alpha\alpha \mid \alpha \in (N \cup T)^*\}$$

  Context free grammars cannot.

- Context free grammars can describe palindrome languages (where α^{-1} is α reversed), such as:

  $$\{\alpha\alpha^{-1} \mid \alpha \in (N \cup T)^*\}$$

  (Context free grammars can describe proper parenthetical nesting, as used in programming languages).

  Regular grammars cannot.

- Regular grammars can describe languages with uncorrelated repeats, such as:

  $$\{a^n\alpha a^m \mid \alpha \in (N \cup T)^*\}$$

  But they cannot encode the additional requirement $n = m$.

A context free grammar for palindromes over $\{a, b\}$

$$S \longrightarrow aSa \mid bSb \mid aa \mid bb$$

Derivation of the palindrome $aabaabaa$:

$$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aabaabaa$$

Context free grammars and RNA secondary structure

RNA sequences can form 'stem loops' when one part of the sequence matches another part ($a \leftrightarrow u$, $c \leftrightarrow g$).

Three-base RNA stem loops with 'head' $gaaa$ or $gcaa$ can be described by:

$$\begin{array}{lcl} S & \longrightarrow & aW_1u \mid cW_1g \mid gW_1c \mid uW_1a \\ W_1 & \longrightarrow & aW_2u \mid cW_2g \mid gW_2c \mid uW_2a \\ W_2 & \longrightarrow & aW_3u \mid cW_3g \mid gW_3c \mid uW_3a \\ W_3 & \longrightarrow & gaaa \mid gcaa \end{array}$$

Derivation trees and parsing

The derivation from a context free grammar may be shown as a tree.

(The derivation from a regular grammar is a degenerate – linear – tree, not very interesting).

Parsing: find a derivation tree for a given sequence, if any

Given a grammar G and a sequence α,

- is sequence α derivable from G?
- if so, what derivation trees would produce sequence α?

Grammar classes and 'parsers' (recognizers)

For each grammar class there is a characteristic 'machine type' that solves the parsing problem for that class.

More expressive grammar classes require more powerful recognizers (and more time and space).

Let n be the length of the sequence being parsed.

Grammar class	Recognizer	Time complexity	Space complexity
Regular grammars	Finite state automata	Linear	Constant
Context free grammars	Pushdown automata	$O(n^3)$	$O(n^3)$
Context sensitive grammars	Linear bounded automata	NP-complete	PSPACE-complete
Unrestricted grammars	Turing machines	Undecidable	Unbounded

Stochastic regular grammars

For each non-terminal symbol W, assign probabilities to rules of form $W \longrightarrow aW_1$ or $W \longrightarrow a$.

Think of W as a state and a as an emitted symbol.

The stochastic grammar emits symbols on state transitions.

A hidden Markov model (HMM) emits symbols without state transitions, and emits nothing on state transitions.

The two kinds of machines are interconvertible (by introducing extra states and transitions).

Thus alignment of a biosequence α using an HMM is parsing of sequence α using a stochastic regular grammar.

Stochastic context free grammars

For each non-terminal symbol W, assign a distribution to the set of rules $W \longrightarrow \alpha$.

	Simple ('deterministic')	Stochastic
Regular	PROSITE patterns	Primary structure, probabilistically (HMM)
Context free	RNA secondary structure	RNA secondary structure, probabilistically

Chomsky normal form

A grammar is on Chomsky normal form if all rules have form $W \longrightarrow W_1 W_2$ or $W \longrightarrow a$.

Every context free grammar can be transformed to Chomsky normal form.

Finding the most probable parse tree: the CYK algorithm

Dynamic programming (tabulation). Analogous to the Viterbi algorithm, which finds the most probable alignment.

A probabilistic version of the Cocke-Younger-Kasami (CYK) algorithm for context free grammar parsing (ca. 1968).

Input: A stochastic context free grammar G on Chomsky normal form, and a sequence $x = x_{1..L}$.

Let the grammar G have terminal symbols $T = \{V_1, \ldots, V_M\}$ and start symbol $S = V_1$.

Let $e_v(a)$ be p if $V_v \longrightarrow a$ with probability p, and 0 otherwise.

Let $t_v(y,z)$ be p if $V_v \longrightarrow V_y V_z$ with probability p, and 0 otherwise.

Output: A table γ such that $\gamma(i,j,v)$ is the probability of the most probable parse tree that derives $x_{i..j}$ from V_v.

Algorithm: Put $\gamma(i,i,v) = e_v(x_i)$ for $i = 1..L$ and $v = 1..M$.

For $i = 1..(L-1)$ and $j = (i+1)..L$ and $v = 1..M$, put

$$\gamma(i,j,v) = \max_{y,z} \max_{k=i}^{j-1} \left(\gamma(i,k,y) \cdot \gamma(k+1,j,z) \cdot t_v(y,z)\right)$$

The probability of the most probable derivation of x from G is $\gamma(1,L,1)$.

Also, a traceback $\tau(i,j,v)$ is built as in the Viterbi algorithm.