Context Free Grammars for DNA Annotation

Alexander Hadik

May 12, 2014

An Introduction to Grammars

Many algorithms for DNA analysis approach sequences in a linear fashion. That is, the DNA sequence is treated as a set of separate symbols, with independence applied at most to direct neighbors. We saw such an approach in HMMs, where dependence is found only between two adjacent emissions. Alignment, although not quite as restricted, is constrained to moving in a single direction along a sequence or sequences. Removing this constraint necessarily destroys the optimal substructure that makes alignment an efficient and practical task.

However, there are many more complex forms of dependence within DNA sequences, that can be quantified, but not efficiently modeled using a linear approach. A very common example is that of inverted repeats or palindromes in DNA. It is this feature of DNA that we will use for demonstration throughout this writing. A more thorough description of inverted repeats and palindromes is to follow. For now, let us further define grammars and their hierarchy.

Spoken language is very similar to features of a DNA sequence, as disparate parts of an English sentence are dependent upon each other. Linear approaches are used for modeling English text, often times with humorous results. However more complex forms of dependence capture more complex grammatical structures. English text can be modeling in various forms of complexity with a grammar. For language, a grammar' defines a set of rules by which words can be placed in relation to each other. There are many famous examples of this, such as "colorless green ideas sleep furiously" or "Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo". Neither of these sentences make sense, but both are grammatically correct according to the grammar of the English language. Noam Chomsky is famous for work in this area, recognizing that sequences of words that properly followed a grammar, regardless of their meaning, would be spoken aloud with proper intonation by readers.

Generative Grammars and their Automata

Such a grammar structure, that could be used to produce a sequence of emissions, or words, according to a set of rules, is known as a transformational grammar. A transformational grammar consists of two primary parts; productions and symbols. Productions define rules, and symbols define the different variables that can be used in those rules.

$$\alpha \to \beta$$

In this case, α and β are symbols, and the relation above is a production. Symbols further come in two types; nonterminals and terminals. Nonterminals are variables that can stand in place of a larger collection of terminals, and thus map to a set of nonterminals and terminals. Terminals are variables that stand for a specific emission, such as a word or character. Consider a grammar consisting of the nonterminal S and the terminal a. We can define productions such as

$$S \to aS$$

$$S \to a$$

Such a grammar, when invoked, would generate a sequence of as. Notice that there is a rule for continuing the sequence, and ending it. This is an example of a regular grammar. Chomsky's work with grammars led him to develop a collection of grammar structures that followed a hierarchy, now famously known as the Chomsky hierarchy of transformational grammars. There are four levels to this hierarchy.

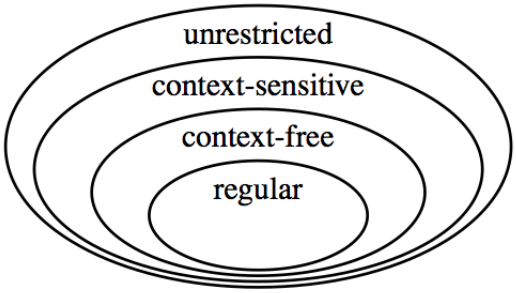
regular Productions must have a single nonterminal on the left, and at most one nonterminal on the right, coupled with terminals if desired. $S \to Sa$ or $S \to a$.

context-free The right hand side of the production can now contain anything, although the left hand side must still contain a single nonterminal.

context-sensitive The left hand side of the production can now contain terminals and nonterminals. This enables complex combinations of nonterminals on the left hand side, but more importantly, terminals as well, placing a context upon the nonterminals. For example $aSb \to aQb$ places this rule within the context of between a and b. However, it is necessary that the left and right hand sides of the production are the same size.

unrestricted There are no restrictions placed upon the grammar. Any production rule is allowed.

These grammar types can be placed in a hierarchy visualized as



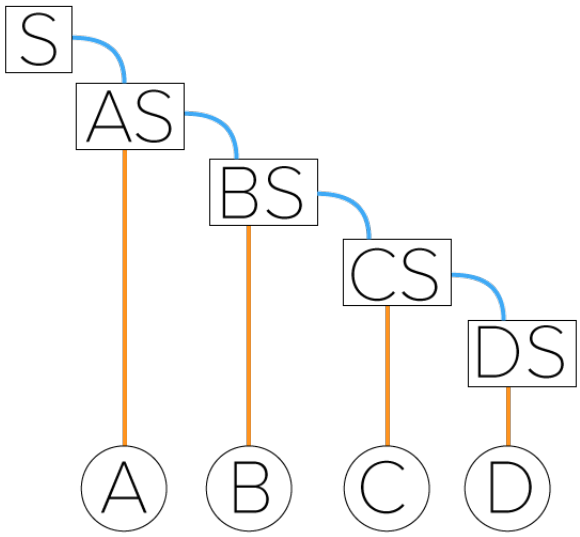
That is, any context-sensitive grammar meets the requirements of an unrestricted grammar, and so on.

Thus far, we've shown how grammars can be used to generate a sequence of emissions based upon a set of rules. However, equally important is to tell if a string of emissions fit the production rules of a grammar. That is, could a given sequence have been generated from a given grammar? The method of answering this question is at heart a parser, and in terms of grammars is referred to as an Automata. There are many forms of automata for different forms of grammar, and the details of each will not be explained here. However, the heart of the process is to link together productions corresponding to the sequence, until no productions can further match the sequence, indicating the sequence can not be generated from the grammar, or until the sequence is full accounted for, indicating it can be produced. The linked productions produce a parse tree, which is a directed graph of productions that produce a given sequence.

Consider for example the sequence $ABCD$ and the productions

$$S \to AS \mid BS \mid CS \mid DS \mid \epsilon$$

where ϵ is the end terminal. The parse tree representing the input sequence takes the form



Probability in Generative Grammars

Thus far we have seen very strict grammars. When productions mapped nonterminals to multiple possible results, as was seen in the parse tree example, there was no mention of how a given result was chosen against the others. This can be resolved by placing a probability weight upon each option. This is a probabilistic grammar. Probabilistic grammars offer several crucial advantages over their counter parts. First of all, the inclusion of weights allows grammars to be formed to follow known parameters of a dataset. Say for example one wanted to generate a series of 1s and 0s at random with 70% 1s. This can be achieved with the following probabilistic regular grammar.

$$S \to 1S \mid 0S$$

$$P(1S) = 0.7$$

$$P(0S) = 0.3$$

Furthermore, probabilistic grammars allow for a more informed parsing of an input sequence. Previously parsing of input was binary; the input does or does not belong to this grammar. Now, with the added information of probability, we can assign a level of significance to our parse. An input that does not fit the grammar has a probability of 0. However, an input that does is parsed and the probability of that parse can be computed, so we can see under a given parse tree, how likely it is that the input sequence would be produced. Very small probabilities indicate that the input sequence contains features that may be worth further analysis.

Clearly, multiple parse trees can be constructed to match the same input sequence. Therefor, a parse tree can be selected based upon the probability that the tree would produce the given input. Often, the parse tree of the highest probability is selected.

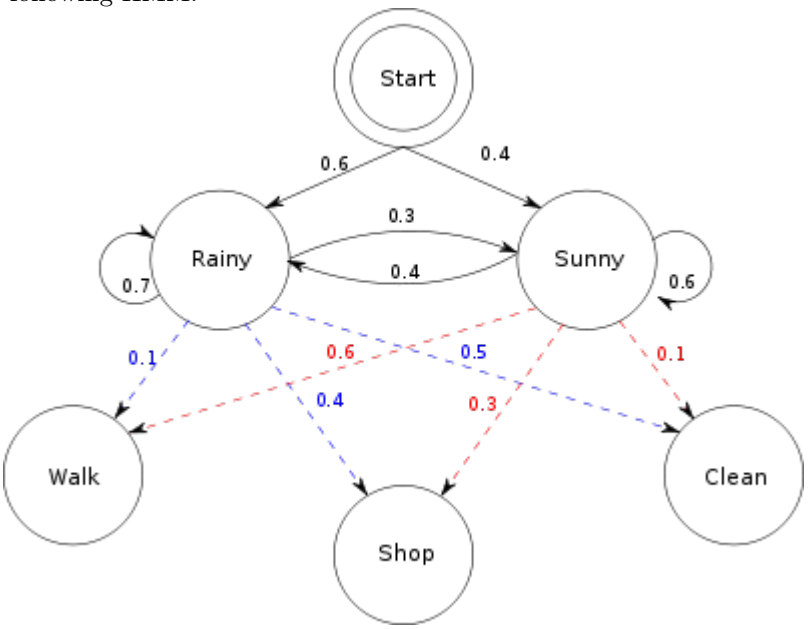
Moving Beyond Linear Data

HMMs

Previously noted were the constraints of linear forms of dependence when generating and annotating sequences of characters. A very popular model for producing and annotating linear data given a probabilistic

model is a Hidden Markov Model. HMMs have three seminal algorithms, the Forward/Backward algorithms, the Viterbi algorithm, and Expectation Maximization, often using the Baum Welch EM algorithm. The Forward/Backward algorithms find the total probability of generating a sequence given all possible routes of production. The Viterbi algorithm uses the Forward algorithms form to find the route of highest probability. Expectation Maximization is used to find parameters that maximize the probability of producing a given sequence. Each of these algorithms has a counter part for grammars, especially because an HMM can in fact be converted to a regular grammar.

There are two pieces of an HMM. There are two finite collections of states and emissions. Transition probabilities are defined between states, and emission probabilities are defined within states. Let's consider the following HMM.



This HMM has two states, Rainy and Sunny, and three possible emissions; Walk, Shop and Clean. Each states has a unique set of emission probabilities for the emissions, and transition probabilities are defined between each state. This same model can be represented as a regular grammar of the following form.

$$Rainy \rightarrow Walk : Rainy \mid Shop : Rainy \mid Clean : Rainy$$

$$Rainy \rightarrow Walk : Sunny \mid Shop : Sunny \mid Clean : Sunny$$

$$Sunny \rightarrow Walk : Sunny \mid Shop : Sunny \mid Clean : Sunny$$

$$Sunny \rightarrow Walk : Rainy \mid Shop : Rainy \mid Clean : Rainy$$

Each of these productions has a probability assigned to it, matching the probabilities in the HMM seen above. Clearly, HMMs are nothing more than a well defined regular grammar, and therefor, we can use more complex grammars to model more complex forms of data. As a result for each of the seminal HMM algorithms, there exists a counterpart for grammars. The Forward/Backward algorithms are replaced with the Inside/Outside algorithms and the Viterbi Algorithm is replaced with the CYK algorithm. For this project, I implemented a variation of the CYK algorithm tailored somewhat to my particular grammar.

Context Free Grammars and Palindromes

Context Free Grammars, and more usefully, Probabilistic Context Free Grammars, are perhaps one of the best described forms of grammars, and are extremely useful for the modeling of nonlinear relationships. A beautiful example of such a data form is that of the palindrome.

Palindromes have very special significance in DNA sequences. Recall that in English language, a palindrome is a word or series of words that reads the same direction forwards and backwards. That is, the characters form a mirror image.

In a DNA sequence, a palindrome is not a sequence of mirrored characters but instead a sequence for which the reverse complement of the sequence reads the same as the sequence. That is, reading the sequence in the 5' to 3' direction and the sequences complementary strand in the 3' to 5' direction is exactly the same. A simple DNA palindrome would be

```
ATCGAT
TAGCTA
```

Notice that reading the top strand and bottom strand of the above example in reverse directions is the same. Palindromes in DNA serve many purposes. Smaller palindromes serve as restriction sites where restriction enzymes can bind and sheer the DNA in two. Larger palindromes are the remnants of inverted duplications, where a long segment of DNA is replicated and spliced back into the genome in reverse orientation. These larger palindromes have been associated with mental retardation in humans. The work of Beverly Emanuel and Hiroki Kurahashi identified palindromes of about 500 bp on chromosomes 11 and 22 of humans. Carriers of these inverted repeats express reduced fertility, however their offspring express far more serious phenotypes such as mental retardation. These palindromes are quite clearly inverted translocations, as randomly generated palindromes of such length are statistically impossible.

Finally, DNA palindromes have important uses once DNA is translated to RNA. As RNA is single stranded, pieces of an RNA strand that are complementary to each other bind together, forming 3D structures. Identification of this binding allows for prediction of the final 3D shape of an RNA strand.

Thus, we have identified three forms of DNA palindromes and the complications with each.

1. Restriction sites are small and well defined palindromes. Mismatches are not considered as sequences of such short size must necessarily be precise for proper sheering of the DNA.

2. Large inverted repeats can take the form of palindromes when placed next to each other on the chromosome. Palindromes of such size will necessarily contain mismatches, thus proper identification requires room for error.

3. Reverse complementary RNA takes the form of a palindrome, however the two sides are likely not adjacent to each other. That is, additional DNA sequence may be found between the two halves. Combined with potential mismatch, identifying such a structure is even more complicated.

Motivation

Identification of restriction sites clearly does not require a PCFG approach. Let us propose a naive algorithm for identifying palindromes in a DNA sequence.

For a limited number of palindromes in a given sequence, this naive algorithm runs in nearly linear time. However, from the different types of palindromes of use identified above, this algorithm clearly does not hold. Allowing for mistakes or for separation of halves by an arbitrary distance, as is needed in motivations 1 and 2, cannot be achieved by such a naive algorithm.

Implementation

Generation

Based on these motivating principles, a Context Free Grammar is a superior alternative for palindrome generation and identification. For simplicity, I have developed and implemented a grammar that looks only for exact palindromes. That is it does not allow for errors or gaps. However, the premise is that such a grammar could easily be extended to meet the more robust requirements laid out above without the need to implement entirely different algorithms for each modified grammar.

The generative Probabilistic Context Free Grammar takes the form of

Algorithm 1 Naive Palindrome Identification

```
SEQ:=input
palindrome:=null
prevChar:=SEQ[0]
for     i=1 to length of SEQ
        currChar:=SEQ[i]
        if      currChar is complementary to prevChar
                then    palindrome:=prevChar,currChar
                while   characters are complementary
                        move    forward from currChar and backward from prevChar.
                        if      characters are complementary
                                then    add to palindrome
                        else    break
```

Palindrome Generation

Termination $P \rightarrow \epsilon$

Extension $P \rightarrow APT \mid TPA \mid CPG \mid GPC$

Random Sequence Extension

Extension $Q \rightarrow AQ \mid TQ \mid CQ \mid GQ$

Transition $Q \rightarrow PQ$

Termination $Q \rightarrow \epsilon$

Starting Production $S \rightarrow Q$

Each of the above rules is assigned a probability.

Palindrome

Termination 35%

Extension 65%

Random

Extension 95%

Base 25% per base

Transition 4.9%

Termination 0.1%

The purpose of this grammar is to model restriction sites, and restriction sites are typically of length 4 to 8 base pairs. Therefor, I wanted any palindrome created to have a high probability of being at least 4 base pairs and a low probability of being more than 8 base pairs. A geometric distribution was used to select the probabilities. First, the probabilities for the Q extension were determined somewhat arbitrarily. A high weight was placed on extension so that palindromes would be separated in generated sequences. With weights

selected, the probability of a palindrome of length 4 or greater being created by chance could be modeled by the expression

$$P(length) = .95^{length} \times .25^{\frac{length}{2}}$$

The length is divided in two for the base selection because only the second half of the palindrome must match a specific pattern. The first half can be created entirely randomly. For a length of 6, the above expression works out to ~0.011. Thus, I needed to select my probabilities for P such that a palindrome of length 6 or greater would have a probability higher than 1.1%. A geometric distribution was used, with "success" defined as a termination of the palindrome. Thus, the probability of a termination after 3 nestings of the palindrome needed to be less than 1.1%. Solving

$$P(X = 3) = (1 - p)^3 p < 0.01$$

produced a value of p at ~0.3. Thus, the probability of extending a palindrome would be designated 70% and the probability of terminating the palindrome would be designated 30%. Experimentation revealed that a probability of nesting at 65% produced the best results.

The above grammar is what has been implemented. However, the addition of just two rules extends this grammar to allow for mistakes and random gaps. For mistakes, the production

$$P \rightarrow APC \mid TPA \mid CPG \mid GPC$$

can be added, with a small probability assigned to each production. Therefor, a parse tree would place low probability on such mistakes, but not reject the parse outright. For gaps, the production $P \rightarrow Q$ can be added so that palindromes can transition back to random generation instead of nesting.

With these productions and probabilities defined, I implemented an algorithm for the random generation of sequences that include embedded palindromes. Algorithm 2 is described below.

This generation algorithm is contained in the Generator subclass of the CFG package found in the accompanying code. It can be run from the command line with the bash script `./generate`. Returned to standard out is a sequence of DNA with palindromes embedded.

With generation of palindrome sequences completed, of use is the ability to parse an input sequence and assign a most probable parse tree to the input sequence. Of interest is the observation of where nested palindrome productions are used, signifying that in this most probable parse tree, a palindrome has been found that is more likely to be the result of a palindrome production than random chance. The motivation here is the ability to assign significance to identified palindromes. Clearly, we don't want to mark every pair of complementary bases as a palindrome. We want to identify only those that are statistically significant.

Parsing

The CYK algorithm is the seminal grammar algorithm for producing the parse tree of highest probability. This is a dynamic programming algorithm that uses all possible subsets of emissions to construct all possible parse trees. Then traceback selects the most probable parse tree. The dynamic programming table is of size n^2 where n is the length of the sequence. However, due to further computation needed for each cell, the runtime of the algorithm approaches $O(n^3)$. Certainly not the most efficient, however quite efficient in comparison to alternative approaches. We'll explain the CYK algorithm with an example. Consider a grammar of the following form and the input sequence *Jeff trains geometry students*.

$\Longrightarrow$ $S \rightarrow N\ V_P$
$VN \rightarrow N\ N$
$V_P \rightarrow V\ N$
$N \rightarrow$ `students` | `Jeff` | `geometry` | `trains`
$V \rightarrow$ `trains`

We create a DP table with each row incrementally larger such that the first column and last row are each the length of the input.

Algorithm 2 Random Sequence Generation with Embedded Palindromes

seq:=”

activeProduction:=S

SeqQueue:={}

generateProduction(production)

select at random from possibleRules attribute of production
set rule attribute of production to selected production

end

evaluate(production)

{productions} = right hand side of production
return {productions}

end

Enqueue activeProduction to SeqQueue

while SeqQueue is not Empty

activeProduction = pop from SeqQueue
generateProduction(activeProduction)
if activeProduction is nonterminal
evaluate(activeProduction)
push evaluated activeProduction to front of seqQueue
if activeProduction is of type terminal
concatenate lable of terminal to working end of seq

end

len				
4	Jeff trains geometry students			
3	Jeff trains geometry	trains geometry students		
2	Jeff trains	trains geometry	geometry students	
1	Jeff	trains	geometry	students
	Jeff	trains	geometry	students

first word in substring

In each cell, subsets of the input of incrementally smaller size are placed, as seen above. Then, the DP table is filled row by row bottom to top. To each cell, productions are assigned that could generate the subset of the cell. For this example, the last row is filled like so

4				
3				
2				
1	N	N,V	N	N
	Jeff	trains	geometry	students

first word in substring

Then the second to last row is filled in as

4				
3				
2	N	N,V_P	N	
1	N	N,V	N	N
	Jeff	trains	geometry	students

first word in substring

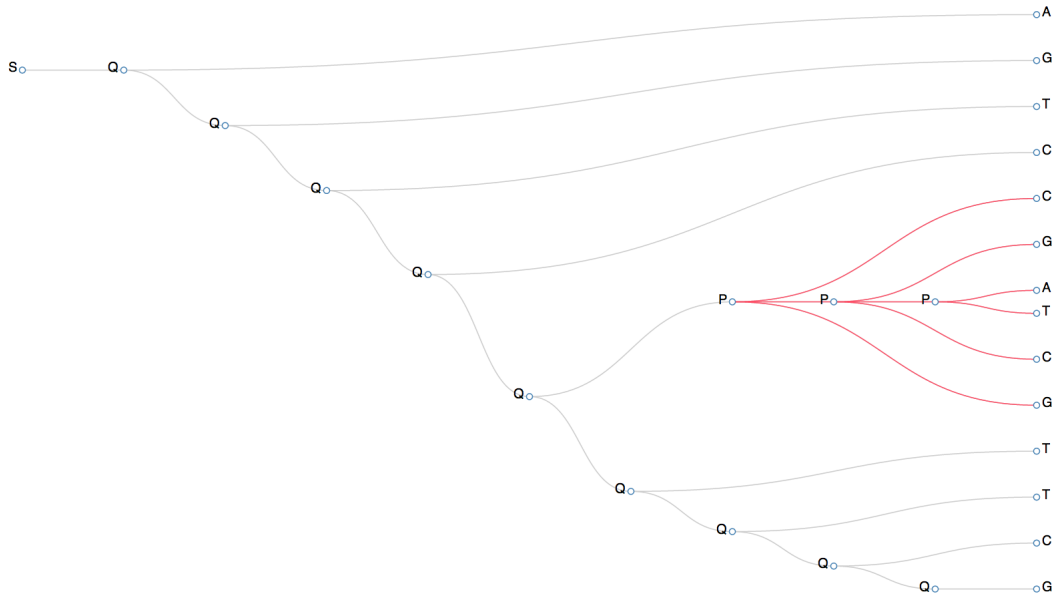
Finally the completed table takes the form

4	N,S			
3	N,S	N,V_P		
2	N	N,V_P	N	
1	N	N,V	N	N
	Jeff	trains	geometry	students

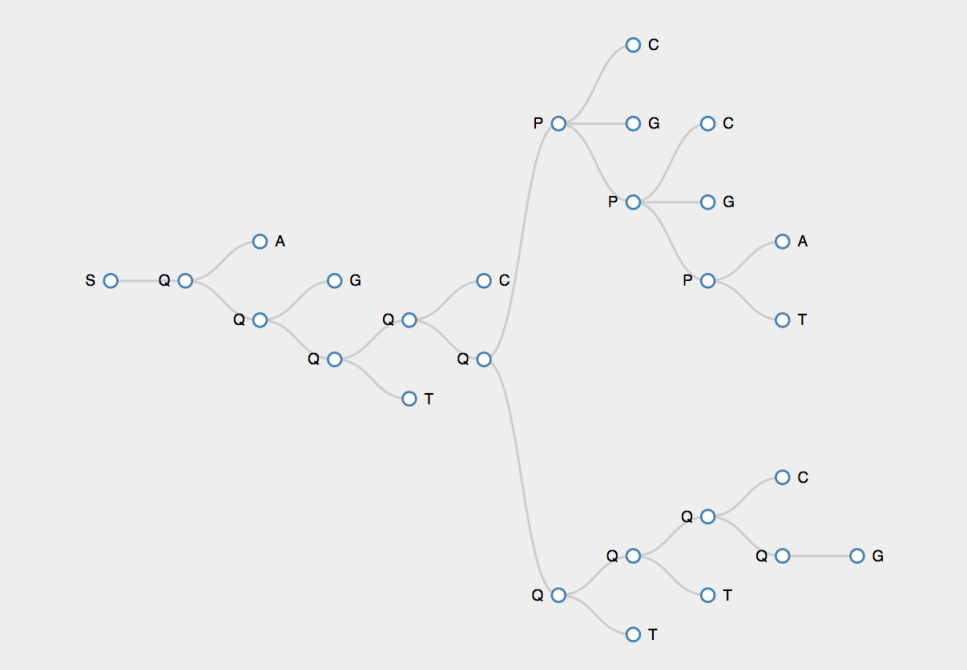
first word in substring

Now traceback is carried out. Starting from the top, each production in each cell is given a pointer to the productions in the lower rows that it would produce. With all pointers filled out, the set of pointers with the highest probability is selected as the most likely parse tree. For the grammar implemented in this project, I made some simplifications to make the algorithm less complicated. I was able to avoid robust back tracing of the table by making assumptions that hold true for this specific grammar. However, with a fully implemented CYK algorithm, the additions to this grammar mentioned above could be used, and the CYK algorithm would work regardless. Thus is the beauty of CFGs for applications such as this, where the alternative might be naive algorithms that work for only specific cases.

My implementation of CYK is contained in the Parser subclass of the CFG package. It can be run with the bash script `./parse` and a single command line argument which is the sequence of DNA to be parsed. The parse tree calculated by the algorithm is parsed to JSON format and saved to a text file in the root of the palindromeCFG directory as a `data.json` file. The tree can be visualized in a web browser by opening the `visualize.html` or `explore.html` file with a browser on a server, local or remote. The Visualize page shows the tree in fully expanded form with the input sequence aligned vertically on the right. Identified palindromes are highlighted in red.



For a more interactive exploration of the parse tree, the Explore page presents a zoomable tree that can be collapsed or expanded.



Bibliography

- Durbin R., Eddy S.R., Krogh A., Mitchison G. *Biological Sequence Analysis,* Cambridge University Press, 1998
- Smith R.G, *Meeting DNA palindromes head-to-head,* Genes and Development, Vol 22, 2008
- Davies J., *Combining Hidden Markov Models and Stochastic-Context Free Grammars,* Worcester College, 2006
- Dyrka W., Nebel J.C., *A probabilistic context-free grammar for the detection of binding sites from a protein sequence,* BMC Systems Biology, 2007