STAN Manual

STAN Manual

by Anne-Sophie Valin, Patrick Durand, and Grégory Ranchy

Published May, 9th, 2005

Revision History

Revision 2.0 31/01/2007 Revised by: Laetitia Guillot
Update of the screen printings, page-setting and form.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. What is STAN?

During the past 10 years, genome sequencing projects have provided a huge number of sequence data. As a consequence of their size and complexity, searching for string patterns has become a difficult problem. That problem does not only rely on efficiently handling huge amount of data in computer's memory, it also relies on the kind of formalism used to model sequence patterns. Most frequently, searching for such patterns is based on the use of regular expressions, but such formalism can only capture basic sequence properties. When considering DNA sequences, one has to consider not only the sequence but also their structural features such as repeats, palindromes, stem-loop, hairpin and pseudoknot. The formal modelling of those structures goes beyond regular expressions, and requires languages that are more expressives.

Among the researches done in the field of formal language theory, the pioneering work of David Searls on String Variable Grammars (SVG) is of high interest [*Searls 95*] , [*Searls 02*] . SVG introduces the concept of a variable that can be associated to a string during a pattern search. SVGs can be used to model not only DNA/RNA sequence features, but also structural features such as the ones given above. To our knowledge, the only two tools capable of searching for SVG-based patterns in biological sequences are GenLang [*Dong and Searls 94*] and PatScan [*Dsouza et al.*](Dsouza et al., 1997). However, GenLang is no longer maintained and, because of its time complexity, was restricted to the analysis of medium size sequences (several Mbases). PatScan, on the other hand, does not guarantee to find all occurrences of complex pattern (once a hit is found, it does not check overlapping alternative solutions).

STAN is software especially design to efficiently search large DNA sequences for SVG based nucleotidic and peptidic patterns. STAN's efficiency mainly relies on the data structure it uses to handle DNA sequences: a suffix-tree. STAN's name originates from the usage of that data structure: STAN stands for Suffix-Tree ANalyser.

1.2. Suffix-tree

When searching for patterns in a sequence, the main problem relies on how to access as fast as possible sub-sequences (i.e. words). That problem can be solved using various indexed data structure representations of the sequence to analyse, and more particularly data structures that can be created and scanned online in linear time [*Manber 93*], [*Lefebvre 03*]. Then, searching for patterns can directly be applied on such data structures. In the context of STAN, the dedicated data structure representation of the sequence is a suffix tree.

A suffix tree is a widely used computational data structure that exposes the internal structure of a string in a way suitable to all problems dealing with string/pattern matching.

If we consider a string S that contains m characters, then the suffix-tree T of S is a rooted directed tree with exactly m leaves numbered 1 to m. Each internal node, other than the root, has at least two children and each edge is labelled with a nonempty substring of S. No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix-tree is that for any leaf i, the concatenation of edge-labels on the path from the root to leaf i exactly spells out the suffix S[i..m] [*Gusfield 97*]. Figure 1 shows the suffix-tree for the string 'aggagct'.

The suffix-tree T from a string S is created by successively inserting all suffixes of S in T. Various implementations of this naive method have been proposed ([*McCreight 76*], [*Ukkonen 95*],) and STAN uses the algorithm proposed by Kurtz [*Kurtz 99*] . In addition to be the fastest method to create a suffix-tree, Kurtz algorithm minimizes the space required to store the tree in the computer's memory: Kurtz suffix-tree only requires 10 to 12 times the sequence size.

Locating a sub-string P within string S is quite simple. Starting from the root of suffix-tree T, match the characters of P along the unique path in T until either P is exhausted or no more matches are possible. In the former case, every leaf in the subtree below the point of the last match gives a location of P in T. In the latter case, P does not appear anywhere in T, so as in S. Figures 1-1, 1-2 and 1-3 show examples of locating a sub-string in the string 'aggagct'.

Figure 1-1. Search for G in the suffix-tree of 'aggagct'

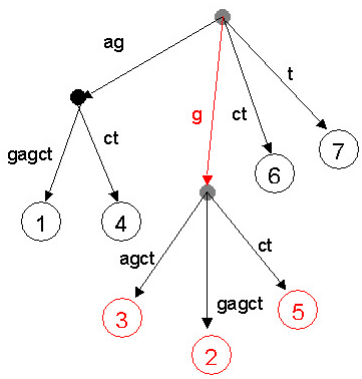


Figure 1-2. Search for AGCT in the suffix-tree of 'aggagct'

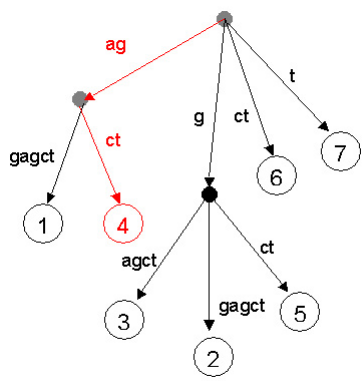
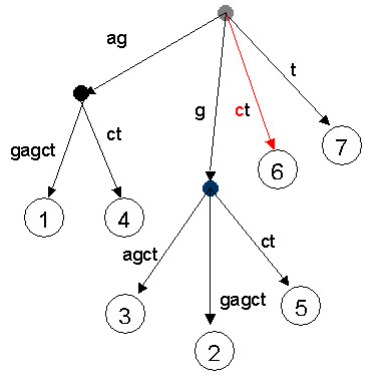


Figure 1-3. Search for CG in the suffix-tree of 'aggagct'



1.3. STAN pattern

STAN pattern describes site/region on nucleotidic or peptidic sequence using a syntax belonging to a particular form of grammar, namely a string variable grammar, or SVG for short [*Searls 95*] , [*Searls 02*] , [*Searls 99*] . SVG is a formalism suitable to describe high-order sequence organization such as copy, palindrome or even structural properties. SVG is a more expressive formalism than the regular-expression grammar that is used to describe Prosite's patterns [*Gattiker 02*].

STAN patterns are made of tokens, separated by a '-' character, belonging to the following language. Letters are taken from standard IUPAC alphabets describing DNA or protein sequences.

- Tokens are made of a single letter or a string of letters.
- Ambiguous tokens are described using either curly braces ({C} for all nucleotides but cytosine) or brackets and pipe characters ([AC|AG] for AC or AG tokens).
- Spacers are described using the syntax 'x(a, b)'. In this syntax, and the following, *a* and *b* denote positive integers, $a \preccurlyeq b$.
- Tokens with insertions and/or deletions are described as 'indel(m, a_i, b_i, a_d,b_d)' where m is a token and a_i and b_i (resp. a_d and b_d) give the minimum and maximum numbers of letter insertions (resp. deletions) allowed in m. Tokens with mismatches are described as 'm:a' where *m* is a token and *a* in the maximum number of letter substitutions allowed in *m*.
- String variables may be introduced in a pattern using identifiers starting with the upper letter X. String variables may be constrained in size using the syntax 'X:[a]' where *X* is a string variable, *a* is the upper limit of X size. Size constraint can also be a range, using the syntax 'X:[a,b]'.
- A token or a string variable may be prefixed with operator '~' to search for the reverse complement. In that way, pattern 'X - ~X' allows to search for palindromes. Finally, mismatches are allowed when defining a string variable ('X - ~X:a' allows to search for palindromes with 'a' letter mismatches).

> **Note:** In order to limit the complexity (search time, number of occurences) of searching for a string variable the size constraint has to be in the range from 4 up to 30 letters.

Table 1-1. Examples of peptidic patterns

Use pattern...	... to search for
LLLVLLL-x(4,20)-LLL	LLLVLLL follows by a spacer of 4 up to 20 amino acids follows by LLL
M-[K|R]-x(3)-[F|L]	a methionine follows by either a lysine or an arginine follows by a spacer of 3 amino acids follows by either a phenylalanine or a leucine. Note: instead of writing [K|R] (resp. [F|L]), [KR] (resp. [FL]) is allowed
C-x(3,7)-C-{C}	two cysteines separated by 3 up to 7 amino acids follows by any amino acids but a cysteine

Table 1-2. Examples of nucleic patterns

Use pattern...	... to search for
ATCGAT:1	ATCGAT exactly or with a single mismatch at any position within ATCGAT
indel(ATCGAT, 0, 0, 1, 2)	ATCGAT exactly or with 1 up to 2 deletions at any position within ATCGAT
TTC-X:[4]-~X	TTC follows by any string of 4 nucleotides, follows by the reverse complement of that string, i.e. a palindrome
ACCG-X:[4]-AT-~X:1-ATT:1	ACCG follows by any string of 4 nucletodies follows by AT, follows by the reverse complement of X, accepting up to 1 mismatch, follows by ATT accepting up to 1 mismatch
TCCTACTATATTATTTGGGAAGTACATTTTAAATGT:9 x(100,4000)-AAATCGT:1-X:[7]-x(4)-~X:5-TTAAAATCTAG:2	TCCTACTATATTATTTGGGAAGTACATTTTAAATGT with 0 up to 9 mismatches, follows by a run of 100 up to 4000 letters, follows by AAATCGT with 0 up to 1 mismatches, follows by any string of 7 nucleotides long, follows by a run of 4 nucleotides, follows by the reverse complement of X with 0 up to 5 mismatches, follows by TTAAAATCTAG with 0 up to 2 mismatches

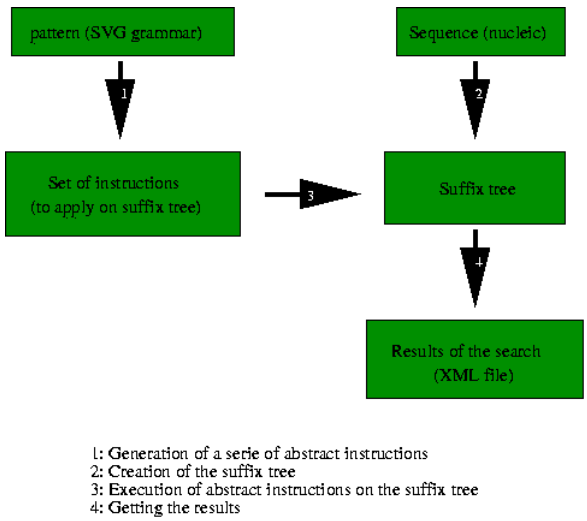
1.4. Implementation of STAN

STAN is implemented using two languages: Prolog and C. Prolog is used to drive the pattern search, whereas the C compiled application computes suffix-tree representations of sequences, handles suffix-tree in computer's memory and executes all the string searches on the suffix-tree.

Prolog runs a SVG interpreter that takes a SVG file as an input. That file actually contains the rules describing the pattern to search for. The SVG interpreter parses the various elements of a pattern (letter, string, string variable, disjunction, etc). For each such element, the interpreter creates an abstract instruction. As a result of that interpretation procedure, an ordered series of abstract instructions is

created (it is worth noting that the current release of STAN does not optimize the abstract program: abstract instructions are executed in the same order they appear in the pattern, while it should be better to first search for the most restrictive elements of a pattern. Such optimizations will be available with future releases of STAN). Then, the interpreter sends those instructions to the C compiled application. In turn, that application converts each abstract instruction into a real function call that executes on the suffix tree. As a result of a search, the C compiled application saves the solutions directly in a XML file on disk: execution of function calls on the suffix tree is done according to the orderered set of abstract instructions. Regarding the suffix tree, those executions consist in following paths made of internal tree's nodes: each function call is executed on the nodes that are solutions of the previous function call (the first function call being executed from the root node).. The following figure summarizes the STAN search procedure:

Figure 1-4. STAN' search procedure



1: Generation of a serie of abstract instructions
2: Creation of the suffix tree
3: Execution of abstract instructions on the suffix tree
4: Getting the results

The creation of the suffix trees for genome sequences is made once, and the binary form of the trees are stored on disk (those trees are only re-created during an update of the genome sequences). In that way, when a user requests to execute a pattern against a genomic sequence, STAN just needs to load the suffix tree in memory. On the other hand, when a user requests to execute a pattern against a sequence he/she provides, STAN computes the suffix tree online (it takes 10 seconds to create the suffix-tree of a 10 Mbases sequence, and construction time is linear regarding sequence size).

To speed up searches on the suffix tree, STAN actually parallelizes its procedure: a sequence is split on several overlapping parts, and a suffix tree is created for each part. String searches are then executed in parallel on the various suffix trees.

Since expressivity of SVG patterns is theoretically high, STAN uses a mechanism to stop a search when the number of occurences of a given pattern exceeds a limit. That limit is fixed by the value of the threshold parameter T. The value of T is the maximum number of accepted occurences of a pattern on each strand (in case of a nucleotidic pattern) or within each translation frame (in case of a peptidic pattern). During a search, when the number of occurences of a pattern exceeds T, STAN stops the search and reports the results found on strands/frames already scanned. It is worth noting that STAN reports an alert message on the result page when the stop mechanism has been applied during a search.

1.5. Performance of STAN

Since STAN searches each pattern's element once at a time, the search complexity mainly relies on the number of elements, and, for each element, on the number of solutions collected during the suffix tree analysis. The search complexity does not depend on the sequence size, except with respect to the number of solutions. Indeed, in the worst case, all positions can be solutions, corresponding to a complete, linear search of the tree. In fact, there exists a slow linear increase of the search time with respect to the length of the sequence due to the split of the sequence in fixed size fragments.

Searching for literal strings is quite immediate on a suffix tree, and the complexity is linear regarding the size of the string to search for.

Searching for ambiguities implies to search various paths in the suffix tree. Such a search could be of exponential complexity with respect to pattern size, but that complexity remains limited by the tree structure complexity which is linear. Complexity of ambiguities searching also greatly depends on the number of possible solutions.

Searching for gaps is quite straightforward when using a suffix tree. STAN searches separately for all solutions of the two elements located on each side of the gap, and it only keeps the solutions satisfying the distance constraint imposed by the gap.

On the other hand, the complexity of searching for strings with errors (insertion, deletion, substitution) and string variables (with/without error/size constraints) is difficult to formally evaluate. However, to get an idea of STAN behaviour during pattern searches, we studied the search of various patterns with increasing complexity. At the same time, we compared STAN performances with related tools, GenLang [*Dong and Searls 94*] and PatScan [*Dsouza et al.*], which are both SVG-based search tools and PatMatch (http://www.arabidopsis.org/cgi-bin/patmatch/nph-patmatch.pl) which uses regular expressions.

All the tests were executed on the Arabidopsis thaliana chromosome 1, which is a 29 millions nucleotides sequence. Four different tests were made, all of them using patterns from known thaliana's transposons:

Table 1-3. A. thaliana transposons used in the test procedure

Transposon name	Grammar definition	Reference
Emigrant MITE	TA-CAGTAAAACCTCTATAAA TTAATA:3-x(0,2500)-TATTAATTT ATAGAGGTTTTACTG:3-TA	Adapted in SVG from [*Santiago 02*]
AtREP3	T-[CT]-x(0,1)-TAC:1-x(2)-TAT-[TA]-AT-[TC]-T-GGGAAG:2-T-ACA:1-TT:1-[AT]-x(0,1)-TAA:1-[ATG]-TGT:2-x(100,3000)-AAATCGT:1-X:[7]-x(4)-~X:5-TTAAAATCTAG:2	Adapted in SVG from [*Kapitonov and Jurka 01*]

1.5.1. Test 1

We searched the thaliana entire sequence with patterns (hereafter called grammars) of increasing complexity. The following table lists the grammars used as well as the number of hits obtained by each tool. Below the table, one can see the search time figure.

Table 1-4. Number of hits for Test 1

Grammar no.	Grammar definition	Nb. of hits found by STAN	Nb. of hits found by PatMatch	Nb. of hits found by PatScan	Nb. of hits found by Genlang
1	CAGTAAAA CCTCTAT AAATTAATA	9	9	9	9
2	CAGTAAAA CCTCTAT AAATTAATA:3	51	51	51	51
3	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,2500)-TATTAA TTTATAGA GGTTTTACTG	8	N/A (1)	6 (3)	8
4	T-[CT]-x(0,1)-TAC:1-x(2)-TAT-[TA]-AT-[TC]-T-GGGAAG:2-T-ACA:1-TT:1-[AT]-x(0,1)-TAA:1-[ATG]-TGT:2-x(100,3000)-AAATCGT:1-x(18)-TTAAA ATCTAG:2	18	N/A (1)	17 (3)	18
5	T-[CT]-x(0,1)-TAC:1 -x(2)-TAT-[TA]-AT-[TC]-T-GGGAAG:2-T-ACA:1-TT:1-[AT]-x(0,1)-TAA:1-[ATG]-TGT:2-x(100,3000)-AAATCGT:1-X:[7]-x(4)-~X:5-TTAAA ATCTAG:2	18	N/A (2)	17 (3)	18

(1) This test cannot be done since PatMatch only allows applying mismatch constraints on the overall grammar, not on particular strings separately.

sftp://bioinfo@genoweb.univ-rennes1.fr/home/genouest/bioinfo/without_ssl/Serveur-

Dev/laeti/outils/patternMatching/HELP/STAN/manuals/uk/user-manual/src/manual.xml (2) This test cannot be done since PatMatch does not allow the use of string variables.

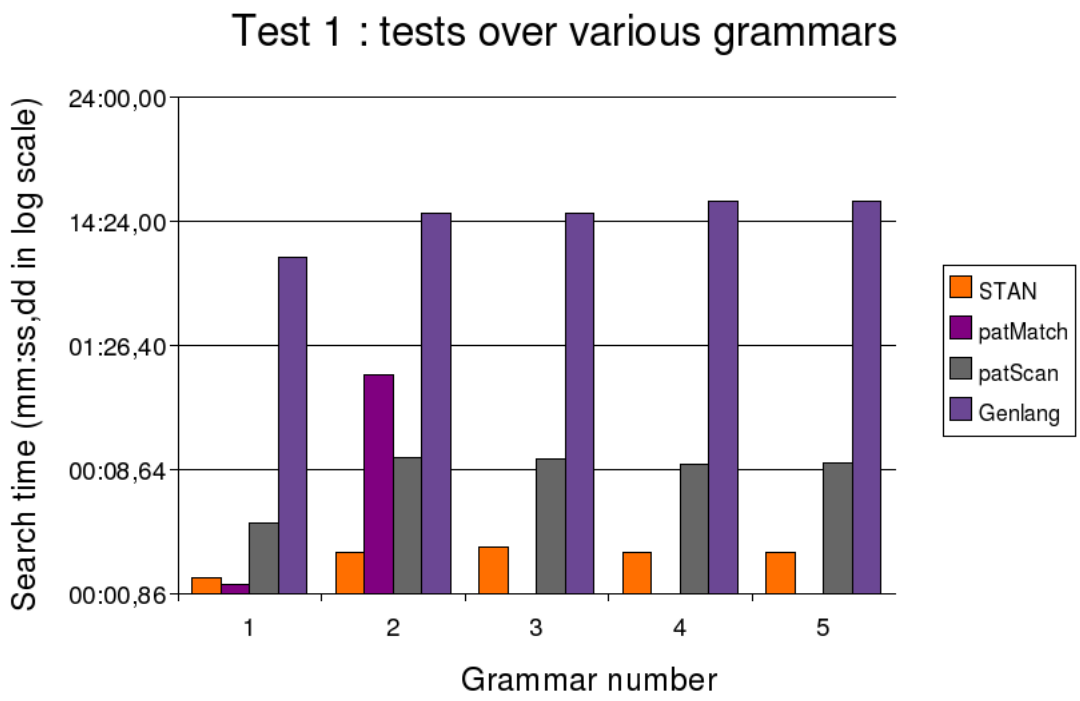
(3) It is worth noting that PatScan does not provide all the possible hits of grammars containing gaps. Once a hit is detected, Patscan sets the current position just after the hit, thus skipping the possible overlapping hits. Genlang and STAN are complete algorithms, finding all possible matches.

Table 1-5. Search time for Test 1

Grammar no.	Grammar definition	Search time for STAN (m:s,d)	Search time for PatMatch (m:s,d)	Search time for PatScan (m:s,d)	Search time for Genlang (m:s,d)
1	CAGTAAAA CCTCTAT AAATTAATA	00:01,17	00:01,01	00:03,16	07:14,30
2	CAGTAAAA CCTCTAT AAATTAATA:3	00:01,87	00:50,20	00:10,64	16:33,80
3	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,2500)-TATTAATTTA TAGAGGTTT-TACTG	00:02,07	N/A	00:10,57	16:22,90
4	T-[CT]-x(0,1)-TAC:1 -x(2)-TAT-[TA]-AT-[TC]-T-GGGAAG:2-T-ACA:1-TT:1-[AT]-x(0,1)-TAA:1-[ATG]-TGT:2-x(100,3000)-AAATCGT:1-x(18)-TTAAA ATCTAG:2	00:01,87	N/A	00:09,58	20:52,40

Grammar no.	Grammar definition	Search time for STAN (m:s,d)	Search time for PatMatch (m:s,d)	Search time for PatScan (m:s,d)	Search time for Genlang (m:s,d)
5	T-[CT]-x(0,1)-TAC:1-x(2)-TAT-[TA]-AT-[TC]-T-GGGAAG:2-T-ACA:1-TT:1-[AT]-x(0,1)-TAA:1-[ATG]-TGT:2-x(100,3000)-AAATCGT:1-X:[7]-x(4)-~X:5-TTAAA ATCTAG:2	00:01,87	N/A	00:09,63	20:56,50

Figure 1-5. Search time for Test 1



1.5.2. Test 2

We searched the thaliana sequence of increasing size with a literal string grammar. The following table lists the search parameters used as well as the number of hits obtained by each tool. Below the table, one can see the search time figure.

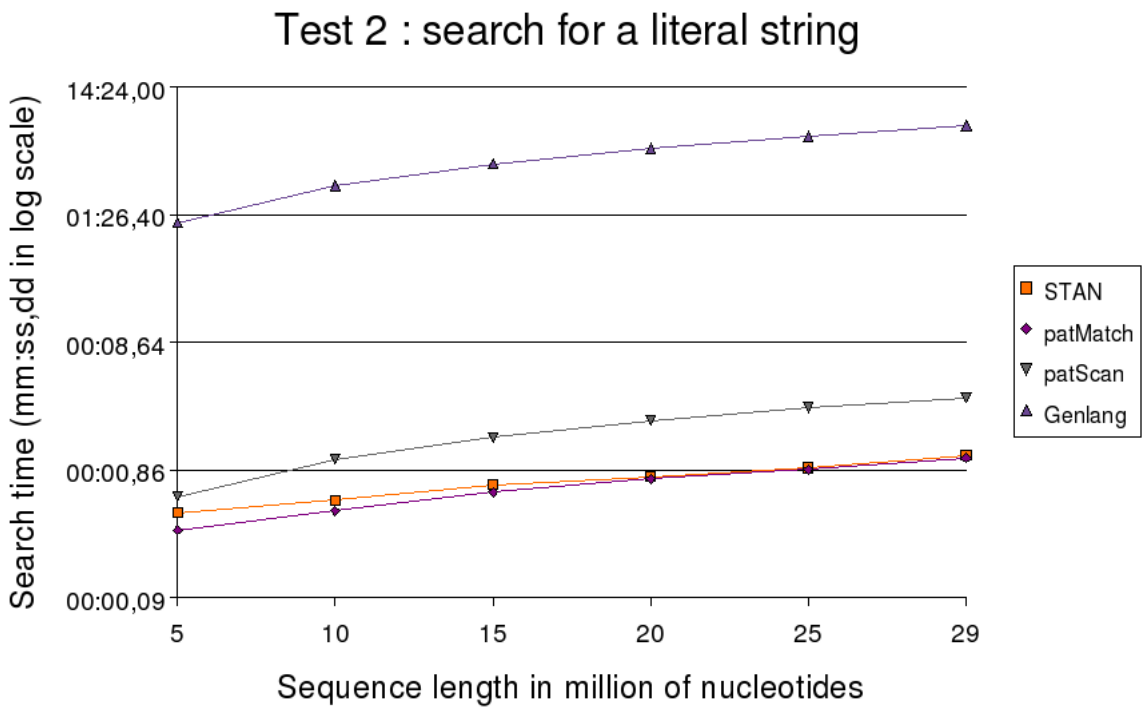
Table 1-6. Number of hits for Test 2

Grammar description	Sequence size (Mbases)	Nb. of hits found by STAN	Nb. of hits found by PatMatch	Nb. of hits found by PatScan	Nb. of hits found by Genlang
CAGTAAAA CCTCTAT AAATTAATA	5	1	1	1	1
CAGTAAAA CCTCTAT AAATTAATA	10	2	2	2	2
CAGTAAAA CCTCTAT AAATTAATA	15	2	2	2	2
CAGTAAAA CCTCTAT AAATTAATA	20	5	5	5	5
CAGTAAAA CCTCTAT AAATTAATA	25	8	8	8	8
CAGTAAAA CCTCTAT AAATTAATA	29	9	9	9	9

Table 1-7. Search time for Test 2

Grammar definition	Sequence size (Mbases)	Search time for STAN (m:s,d)	Search time for PatMatch (m:s,d)	Search time for PatScan (m:s,d)	Search time for Genlang (m:s,d)
CAGTAAAA CCTCTAT AAATTAATA	5	00:00,40	00:00,29	00:00,54	01:14,80
CAGTAAAA CCTCTAT AAATTAATA	10	00:00,50	00:00,41	00:01,05	02:26,00
CAGTAAAA CCTCTAT AAATTAATA	15	00:00,67	00:00,58	00:01,56	03:36,60
CAGTAAAA CCTCTAT AAATTAATA	20	00:00,77	00:00,74	00:02,11	04:46,30
CAGTAAAA CCTCTAT AAATTAATA	25	00:00,90	00:00,87	00:02,68	05:57,60
CAGTAAAA CCTCTAT AAATTAATA	29	00:01,13	00:01,08	00:03,18	07:13,60

Figure 1-6. Search time for Test 2



1.5.3. Test 3

We searched the thaliana entire sequence with a literal string on which we increase the number of mismatches (substitution errors). The following table lists the search parameters used as well as the number of results obtained by each tool. Below the table, one can see the search time figure.

Table 1-8. Number of hits for Test 3

Grammar no.	Grammar definition	Nb. of hits found by STAN	Nb. of hits found by PatMatch	Nb. of hits found by PatScan	Nb. of hits found by Genlang
1	CAGTAAAA CCTCTAT AAATTAATA	9	9	9	9
2	CAGTAAAA CCTCTAT AAATTAATA:1	24	24	24	24
3	CAGTAAAA CCTCTAT AAATTAATA:2	35	35	35	35

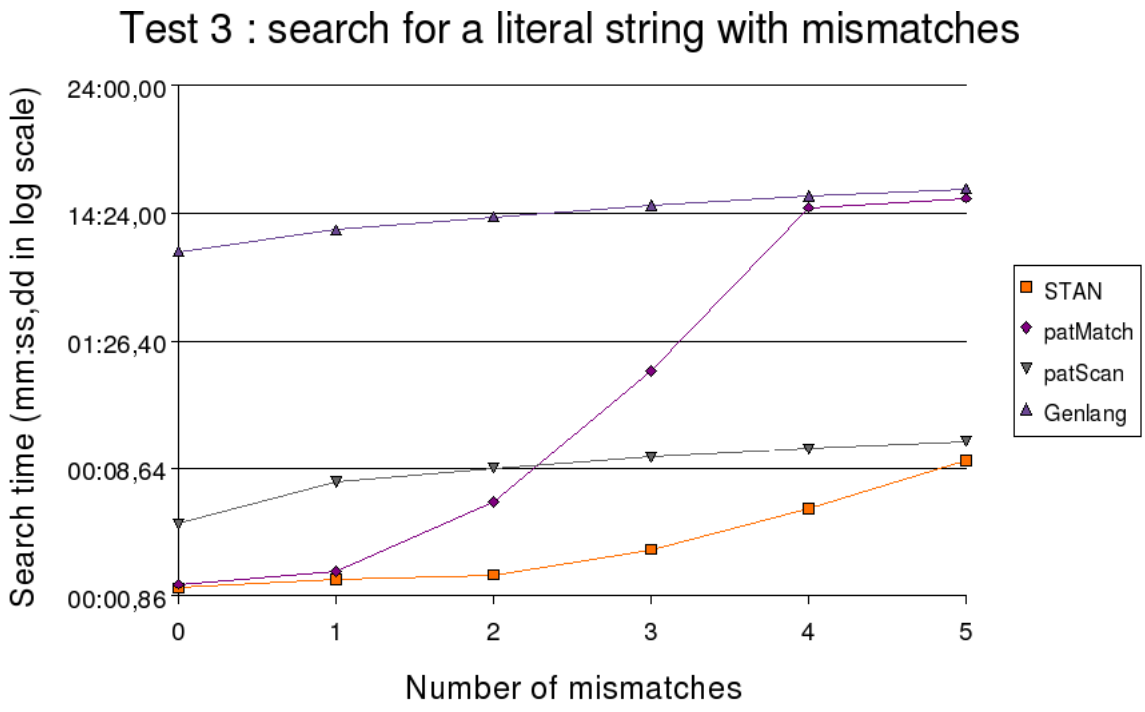
Grammar no.	Grammar definition	Nb. of hits found by STAN	Nb. of hits found by PatMatch	Nb. of hits found by PatScan	Nb. of hits found by Genlang
4	CAGTAAAA CCTCTAT AAATTAATA:3	51	51	51	51
5	CAGTAAAA CCTCTAT AAATTAATA:4	61	61	61	61
6	CAGTAAAA CCTCTAT AAATTAATA:5	104	100 (1)	100 (1)	104

(1) It is worth noting that PatScan does not provide all the possible hits of grammars containing substitutions errors. Once a hit is detected, Patscan and PatMatch sets the current position just after the hit, thus skipping the possible overlapping hits. Genlang and STAN are complete algorithms, finding all possible matches.

Table 1-9. Search time for Test 3

Grammar no.	Grammar definition	Search time for STAN (m:s,d)	Search time for PatMatch (m:s,d)	Search time for PatScan (m:s,d)	Search time for Genlang (m:s,d)
1	CAGTAAAA CCTCTAT AAATTAATA	00:01,00	00:01,07	00:03,18	07:13,60
2	CAGTAAAA CCTCTAT AAATTAATA:1	00:01,17	00:01,35	00:06,82	10:39,40
3	CAGTAAAA CCTCTAT AAATTAATA:2	00:01,27	00:04,73	00:08,74	13:28,30
4	CAGTAAAA CCTCTAT AAATTAATA:3	00:02,00	00:50,21	00:10,61	16:33,70
5	CAGTAAAA CCTCTAT AAATTAATA:4	00:04,20	16:00,29	00:12,33	19:35,80
6	CAGTAAAA CCTCTAT AAATTAATA:5	00:09,97	18:59,23	00:14,17	22:28,50

Figure 1-7. Search time for Test 3



1.5.4. Test 4

We searched the thaliana entire sequence with a grammar containing a gap of increasing size. The following table lists the search parameters used as well as the number of results obtained by each tool. Below the table, one can see the search time figure.

Table 1-10. Number of hits for Test 4

Grammar no.	Grammar definition	Nb. of hits found by STAN	Nb. of hits found by PatMatch	Nb. of hits found by PatScan	Nb. of hits found by Genlang
1	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,500)-TATTAA TTTATAGA GGTTTTACTG	6	N/A	5 (1)	6
2	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,1000)-TATTAA TTTATAGA GGTTTTACTG	8	N/A	6 (1)	8

Grammar no.	Grammar definition	Nb. of hits found by STAN	Nb. of hits found by PatMatch	Nb. of hits found by PatScan	Nb. of hits found by Genlang
3	CCAGTAAAA CCTCTAT AAATTAATA:3-x(0,1500)-TATTAA TTTATAGA GGTTTTACTG	8	N/A	6 (1)	8
4	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,2000)-TATTAA TTTATAGA GGTTTTACTG	8	N/A	6 (1)	8
5	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,2500)-TATTAA TTTATAGA GGTTTTACTG	8	N/A	6 (1)	8

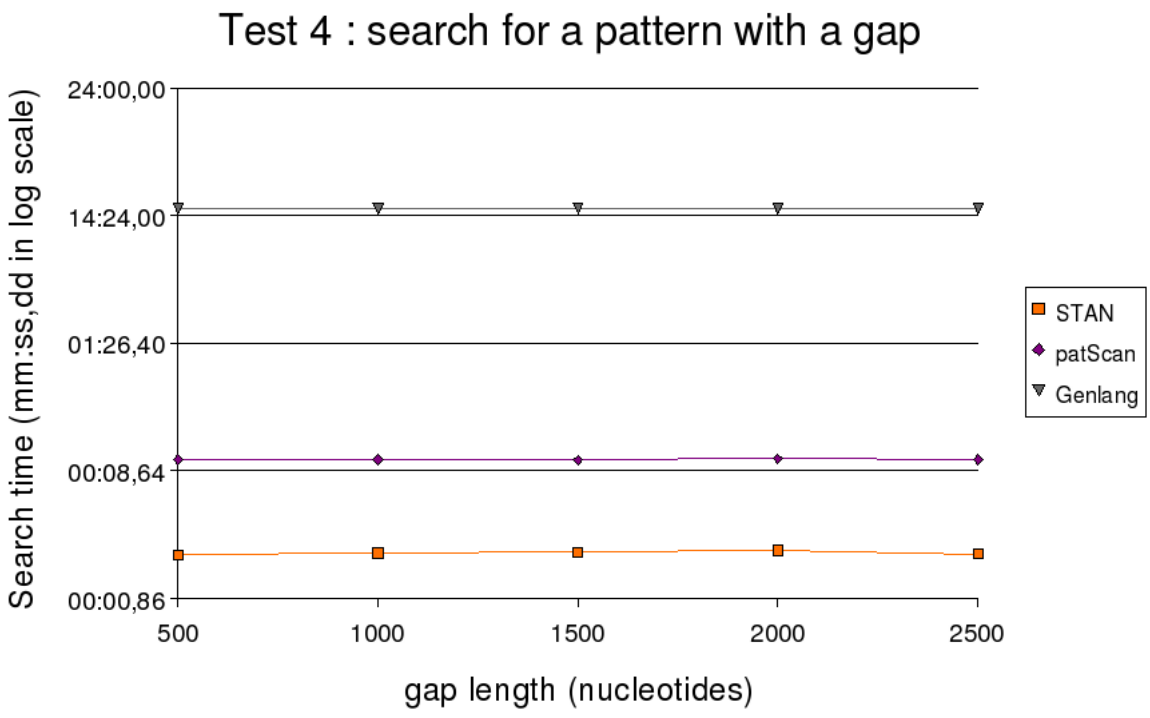
(1) It is worth noting that PatScan does not provide all the possible hits of grammars containing gaps and substitutions errors. Once a hit is detected, Patscan sets the current position just after the hit, thus skipping the possible overlapping hits. Genlang and STAN are complete algorithms, finding all possible matches.

Table 1-11. Search time for Test 4

Grammar no.	Grammar definition	Search time for STAN (m:s,d)	Search time for PatMatch (m:s,d)	Search time for PatScan (m:s,d)	Search time for Genlang (m:s,d)
1	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,500)-TATTAA TTTATAGA GGTTTTACTG	00:01,90	N/A	00:10,65	16:18,90
2	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,1000)-TATTAA TTTATAGA GGTTTTACTG	00:01,97	N/A	00:10,59	16:16,70

Grammar no.	Grammar definition	Search time for STAN (m:s,d)	Search time for PatMatch (m:s,d)	Search time for PatScan (m:s,d)	Search time for Genlang (m:s,d)
3	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,1500)-TATTAA TTTATAGA GGTTTTACTG	00:02,00	N/A	00:10,48	16:19,00
4	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,2000)-TATTAA TTTATAGA GGTTTTACTG	00:02,07	N/A	00:10,75	16:19,60
5	CAGTAAAA CCTCTAT AAATTAATA:3-x(0,2500)-TATTAA TTTATAGA GGTTTTACTG	00:01,93	N/A	00:10,65	16:19,30

Figure 1-8. Search time for Test 4



Chapter 2

Using STAN web interface

2.1. Principles

2.1.1. Main page

STAN's main page (Figure 2-1) contains a search form divided in two sections: search parameters on the left and help messages on the right. Help messages are hidden until some values are entered in form's fields. As an example, the STAN pattern help is displayed as soon as you type in a pattern in the appropriate field, as illustrated on Figure 2-2. Figure 2-3 shows an example of syntax error message.

Search parameters are organized in three sections: User parameters, Pattern parameters and Target Sequence.

Figure 2-1. STAN's main page

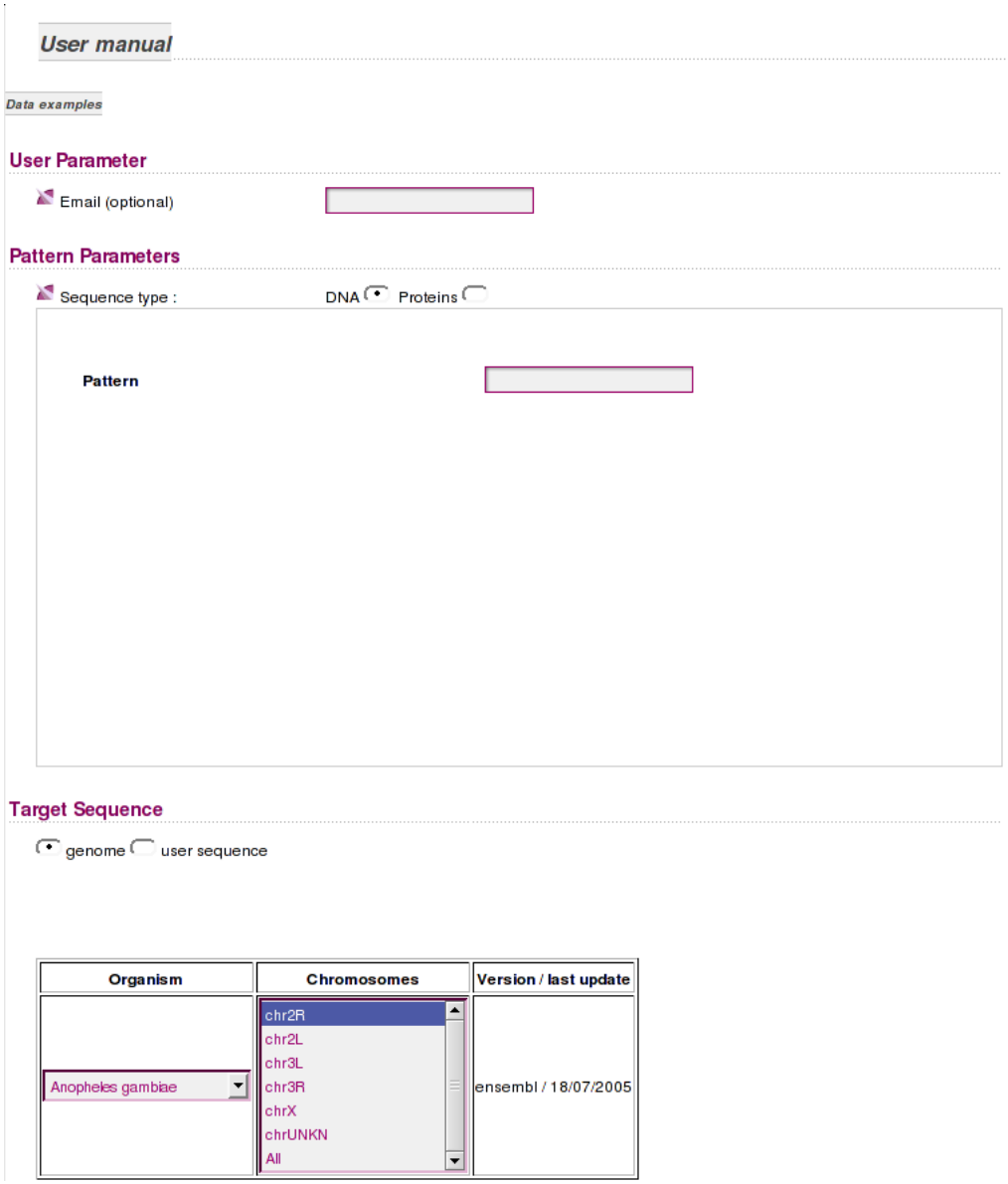
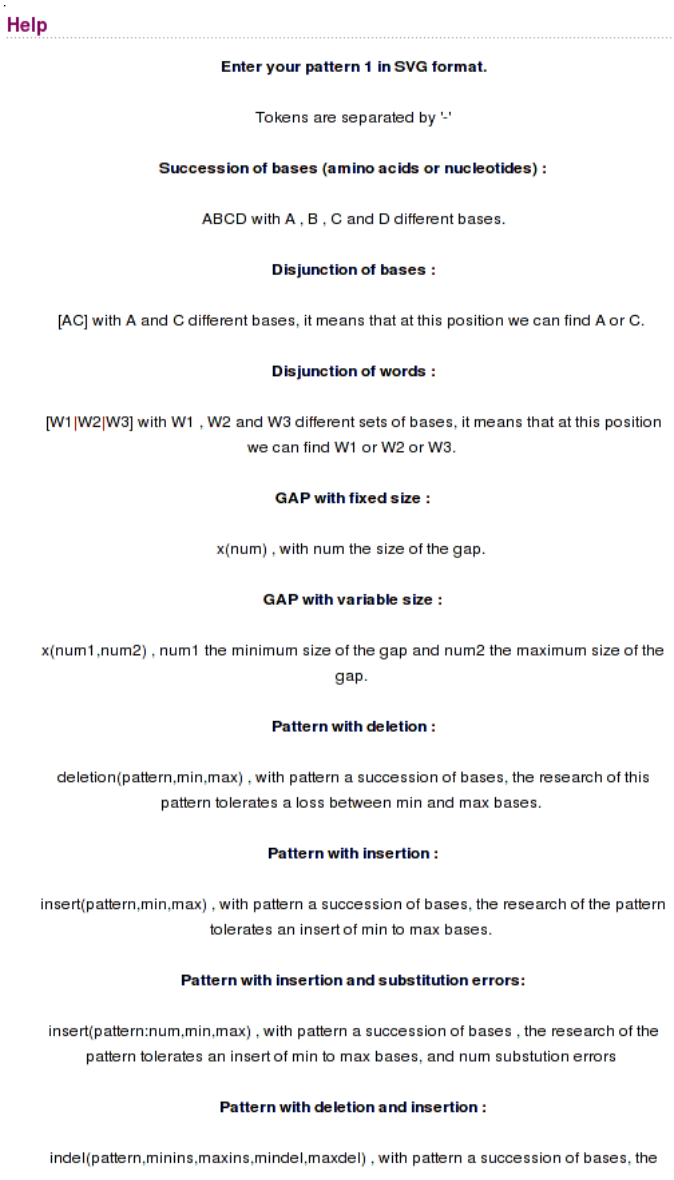


Figure 2-2. STAN pattern help



Help

Enter your pattern 1 in SVG format.

Tokens are separated by '-'

Succession of bases (amino acids or nucleotides) :

ABCD with A , B , C and D different bases.

Disjunction of bases :

[AC] with A and C different bases, it means that at this position we can find A or C.

Disjunction of words :

[W1|W2|W3] with W1 , W2 and W3 different sets of bases, it means that at this position we can find W1 or W2 or W3.

GAP with fixed size :

x(num) , with num the size of the gap.

GAP with variable size :

x(num1,num2) , num1 the minimum size of the gap and num2 the maximum size of the gap.

Pattern with deletion :

deletion(pattern,min,max) , with pattern a succession of bases, the research of this pattern tolerates a loss between min and max bases.

Pattern with insertion :

insert(pattern,min,max) , with pattern a succession of bases, the research of the pattern tolerates an insert of min to max bases.

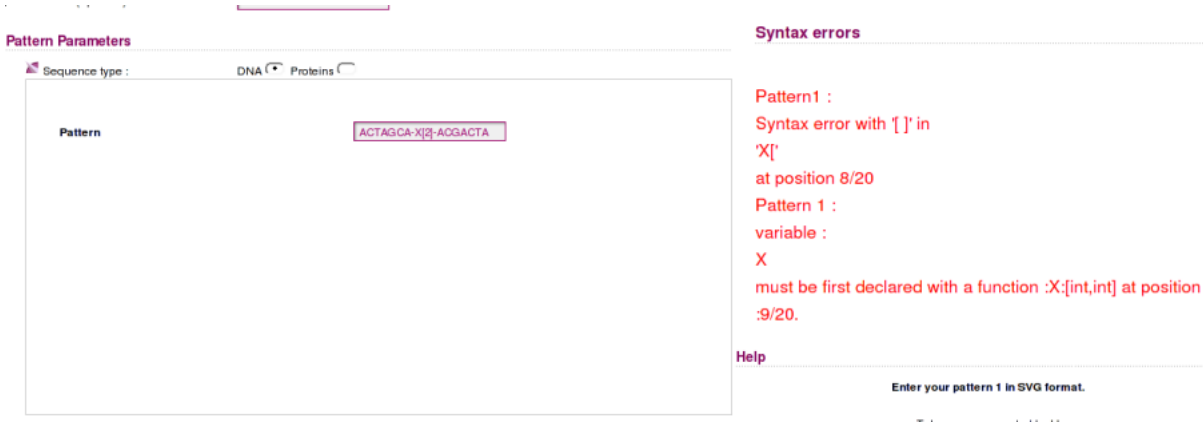
Pattern with insertion and substitution errors:

insert(pattern:num,min,max) , with pattern a succession of bases , the research of the pattern tolerates an insert of min to max bases, and num substution errors

Pattern with deletion and insertion :

indel(pattern,minins,maxins,mindel,maxdel) , with pattern a succession of bases, the

Figure 2-3. STAN pattern help



2.1.2. User parameters

In this section, you can enter your email address that is used by STAN to send you a message as soon as the search is done. The email message contains a reference to the result file kept on STAN system during 10 days before deletion. Entering an email address is optional, but we recommend you do so. Indeed, you have to keep in mind that the search for a pattern may require several hours of computation. In that case, you may have some problems to get back your result file if you close your browser while a search is under execution. Providing your email address ensures that STAN always informs you when a search has been executed.

2.1.3. Pattern parameters

Pattern parameters consist of the pattern sequence type (either DNA or protein) and the pattern itself.

For a DNA pattern, the pattern can be entered in the appropriate field using nucleotides alphabet along with STAN syntax elements (see above).

A peptidic pattern can be described with 1 up to 5 parts using the amino acids alphabet along with STAN syntax elements (see above). When describing a pattern with more than one part, you can also specify the spacer size between two consecutive parts. When using a multi-parts pattern, STAN works as follows. Each part is search for separately within the six translation frames of the DNA sequence. STAN only keeps combinations of pattern's parts satisfying spacer constraints. Such a multi-parts pattern can be useful to locate pattern spanning the multiple exons of genes.

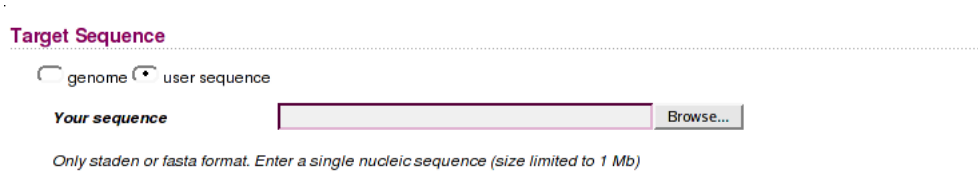
When searching for a peptidic multi-parts pattern, STAN optimizes the search by separately looking for each part in parallel. Then, STAN produces all possible occurences of the complete pattern using a dichotomic sort of the positions of the various parts' occurences while taking into account spacer size constraints.

2.1.4. Target sequence

The pattern can be searched for either within a genome sequence of within a user-defined one.

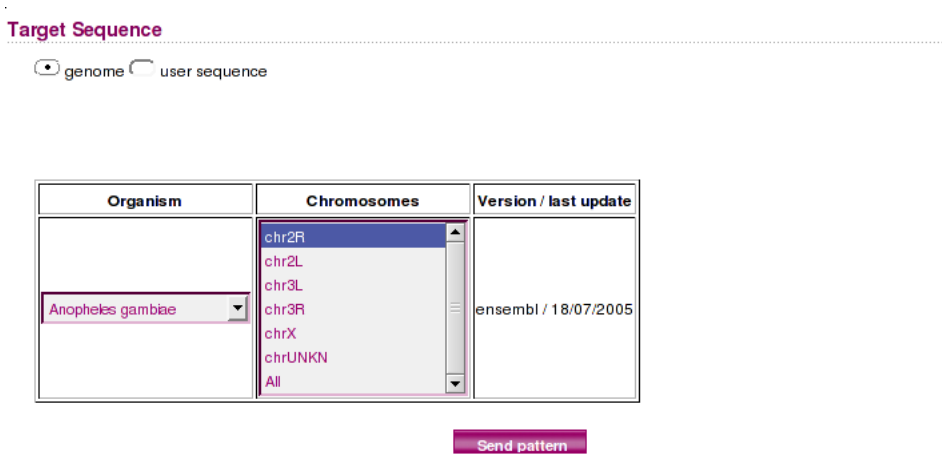
To enter your own nucleic sequence, select the check box called 'user sequence', then enter the file name in the appropriate text field (see figure below) or use the [Browse...] button. Please note that the sequence size cannot exceed one million nucleotides and have to be nucleic (staden or fasta format).

Figure 2-4. Entering a user sequence



To search for a pattern within a genome, select the check box called 'genome'. For a given organism, you can select either all chromosome sequences or only some of them. The set of chromosome sequences are globally referred to as the target genome.

Figure 2-5. Selecting a genome sequence



Note: If you want to search for a pattern within a genome not listed here, you can contact our Web Master and ask to add that genome on the list.

For a DNA pattern, STAN searches the target genome in both normal and reverse complement directions.

For a peptidic pattern, STAN searches the six translation frames of the target genome. Pattern parts are separately searched for in the six translation frames with respect of their order and spacers size.

As soon as all search parameters are edited, you can click on the [Submit query] button to go to the next page, 'Formatting result'.

2.1.5. Formatting result

On the 'Formatting result' page, you can select which sequence elements will be reported in the result file. For a given pattern, it is possible to report in the result the genomic sequence matched by the pattern as well as the sequences located upstream and downstream the match. For a peptidic pattern having several parts, it is also possible to report in the result the genomic sequences associated to the spacers. All those sequences can be reported either directly as DNA sequences or as translated protein sequences.

Figure 2-6. Formatting results

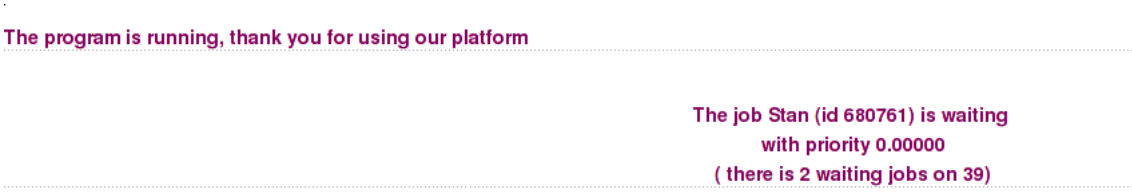
Since expressivity of SVG patterns is theoretically high, STAN uses a mechanism to stop a search when the number of occurences of a given pattern exceeds a limit. That limit is fixed by the value of the threshold parameter T. The value of T is the maximum number of accepted occurences of a pattern on each strand (in case of a nucleotidic pattern) or within each translation frame (in case of a peptidic pattern). During a search, when the number of occurences of a pattern exceeds T, STAN stops the search and reports the results found on strands/frames already scanned. It is worth noting that STAN reports an alert message on the result page when the stop mechanism has been applied during a search.

As soon as all search parameters are edited, you can click on the [Submit query] button to start the search and proceed to the 'Waiting' page.

2.1.6. Waiting page

During a search, STAN successively displays the following pages:

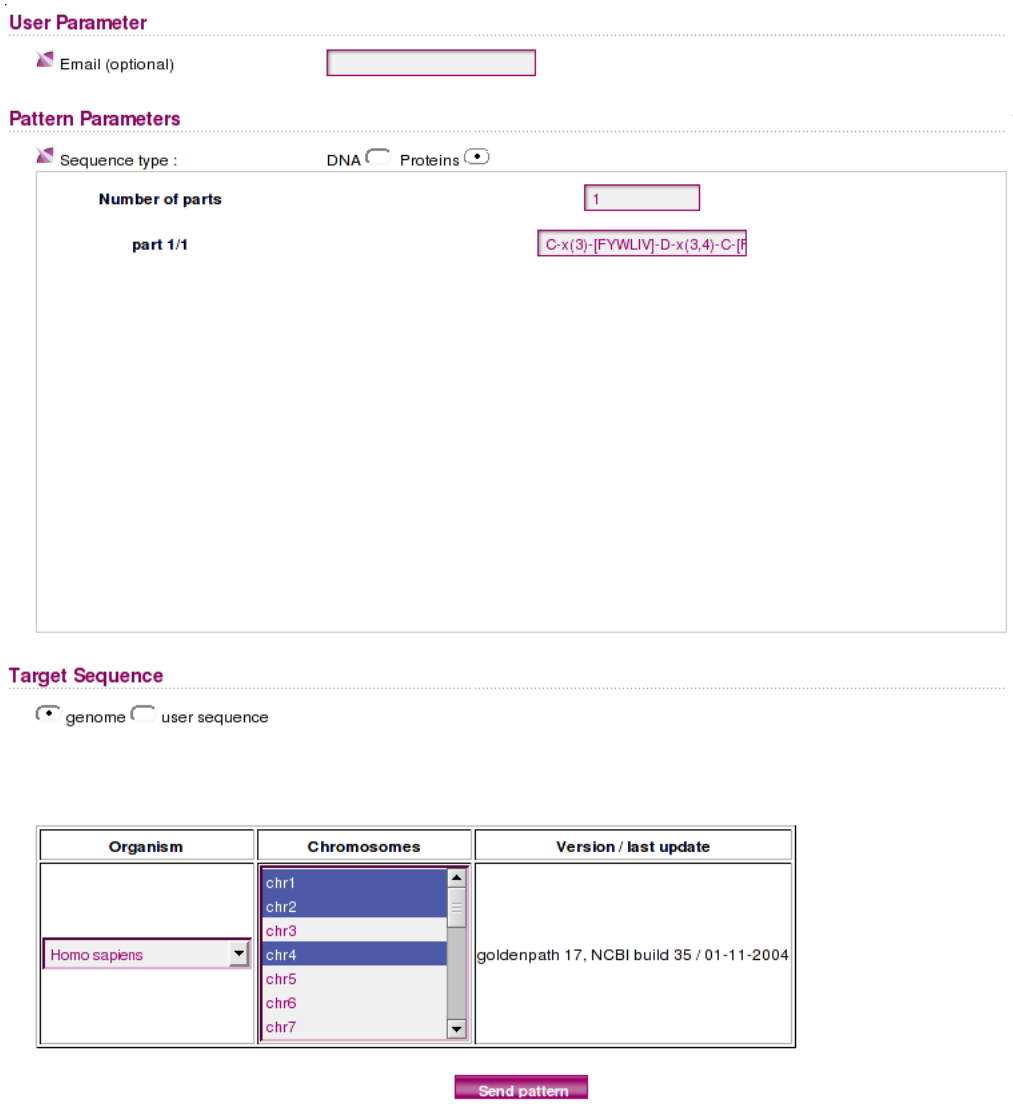
Figure 2-7. Standard waiting page



2.2. Basic example

To illustrate the usage of STAN's web interface, we are going to search the human chromosomes 1, 2 and 4 for the G-protein coupled receptors family 2 pattern 'C-x(3)-[FYWLIV]-D-x(3,4)-C- [FW]-x(2)-[STAGV]-x(8,9)-C-[PF]' (Cf. figure 2-11). This pattern is referenced by the Prosite database as entry PS00649. You can refer to Prosite web site (http://www.expasy.org/prosite/) for more information.

Figure 2-8. Search of pattern in human chromosomes



2.2.1. Fill in the main page

2.2.1.1. Enter your email address

On the STAN's main page, start to fill in the search form by entering you email address.

2.2.1.2. Enter the pattern

The pattern we want to search for is constituted of a single peptidic signature. So, under section 'Pattern Parameters', select 'Protein' sequence type, the default value of field 'Number of parts' is '1', so you don't have to change the value of this field. You can now enter the pattern (see above) in the field 'Pattern, part 1/1'. (Cf. 2-8)

2.2.1.3. Select the target sequence

Finally, you have to select the target genome by selected the appropriate values under the 'Target Genome' section. In this example, select 'Human' in the list of organisms, then select chromosomes 1, 2 and 4 in the list of chromosomes. To execute the latter selection, start by pressing the [Ctrl] key on your keyboard, then select the three values with the mouse while keeping [Ctrl] key pressed (Cf. 2-8).

Click on the [Submit Query] button.

2.2.2. Format the result

From the main page, you now have the 'Formatting Result' page that allows you to choose which components to include in the results of a STAN search (Cf. Figure 2-9).

Figure 2-9. Formatting Result

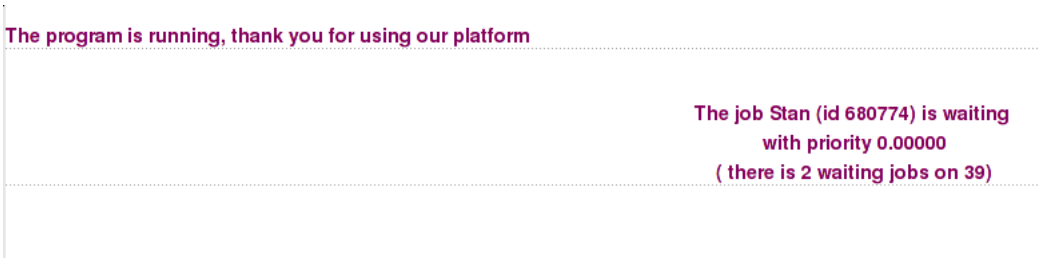
For each database sequence that will be matched by your pattern, you can choose to report the following data in the result file:

- the sequence located upstream the pattern hit,
- the sequence matched by the pattern itself,
- the sequence located downstream the pattern hit.

Sequences may be reported in the result file either as DNA sequences or as protein sequences. In our example, we just select the check box to get the DNA sequence matched by our pattern (Cf. figure 2-9):

Click on the [Submit Query] button to start the search and get the following page:

Figure 2-10. Searching for the pattern



Your Web browser automatically updates that page until the search is done. Please note that for a time consuming search, you can close that window and wait until STAN sends you an email. Using the content of the email, you will be able to get your results.

In this example, the search could be done within a few minutes, so keep the Web browser opened until you end up with the following page:

Figure 2-11. Result page

You are now invited to choose the format to display the results of the pattern search. You can choose one of the following formats:

- Web Pages. This default format can be used to display the search results in your Web browser.
- Spreadsheet format (Excel). This format allows you to get a result file that can be imported in spreadsheet software, such as Excel or Open Office.
- XML. This is the native format of STAN search results. Experienced users can use it to further process the results.

In this example, keep "Web Pages" selected then click on the [Submit Query] button.

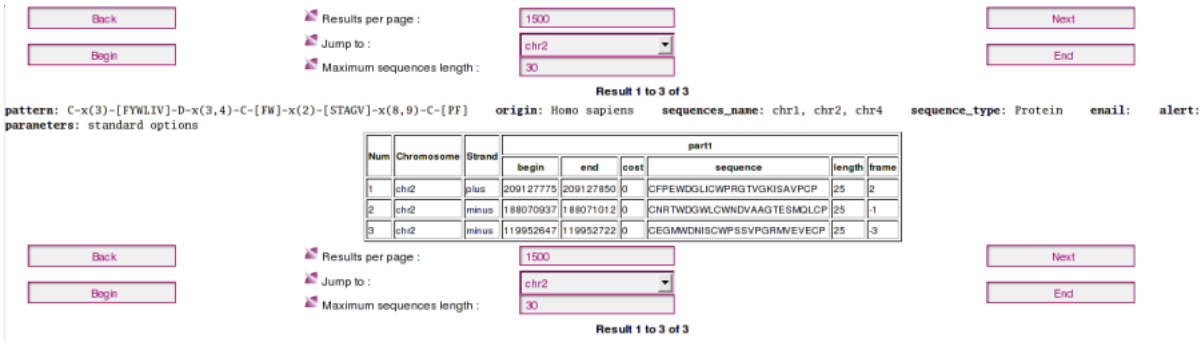
2.2.3. Analyse the result

Search results are displayed as a table where each row corresponds to a single hit of the pattern in the target genome. Columns mainly give information about the location of a pattern: chromosome number, strand, positions and frame. The latter only appears for translated DNA sequences. In addition to these data, the table displays the sequences as requested on page Format the result. In our example, we have chosen to only view the sequence matched by the pattern.

Parameter 'Maximum sequence length' (displayed at the top of the result page) limits the size of the sequences displayed in the table. When a sequence has a size above the value of parameter 'Maximum

sequence length', that sequence is partially displayed: only left and right parts are presented, with '...' between the parts (see examples on Figure 2.12).

Figure 2-12. HTML Result page



2.3. Complex example: searching for multi-parts patterns

In the context of searching genome sequences for peptidic patterns, STAN allows the definition of multi-parts patterns. Such a pattern is made of from 1 up to 5 parts, separated by spacers. Each part is written using the STAN pattern syntax, as described earlier, while spacers are used to specify the minimum and maximum distances (in nucleotides) between two consecutive parts. During a search for such a multi-parts pattern, STAN tries to locate each part of the pattern within the 6 translation frames of the scanned DNA sequence, while taking into account the size of the spacers. As a result, the search procedure may return pattern hits where parts can be located on different translation frames. Multi-parts patterns can be useful to locate patterns that could span protein-coding genes having an intron/exon structure.

Note: multi-parts patterns are only available when searching for peptidic based patterns.

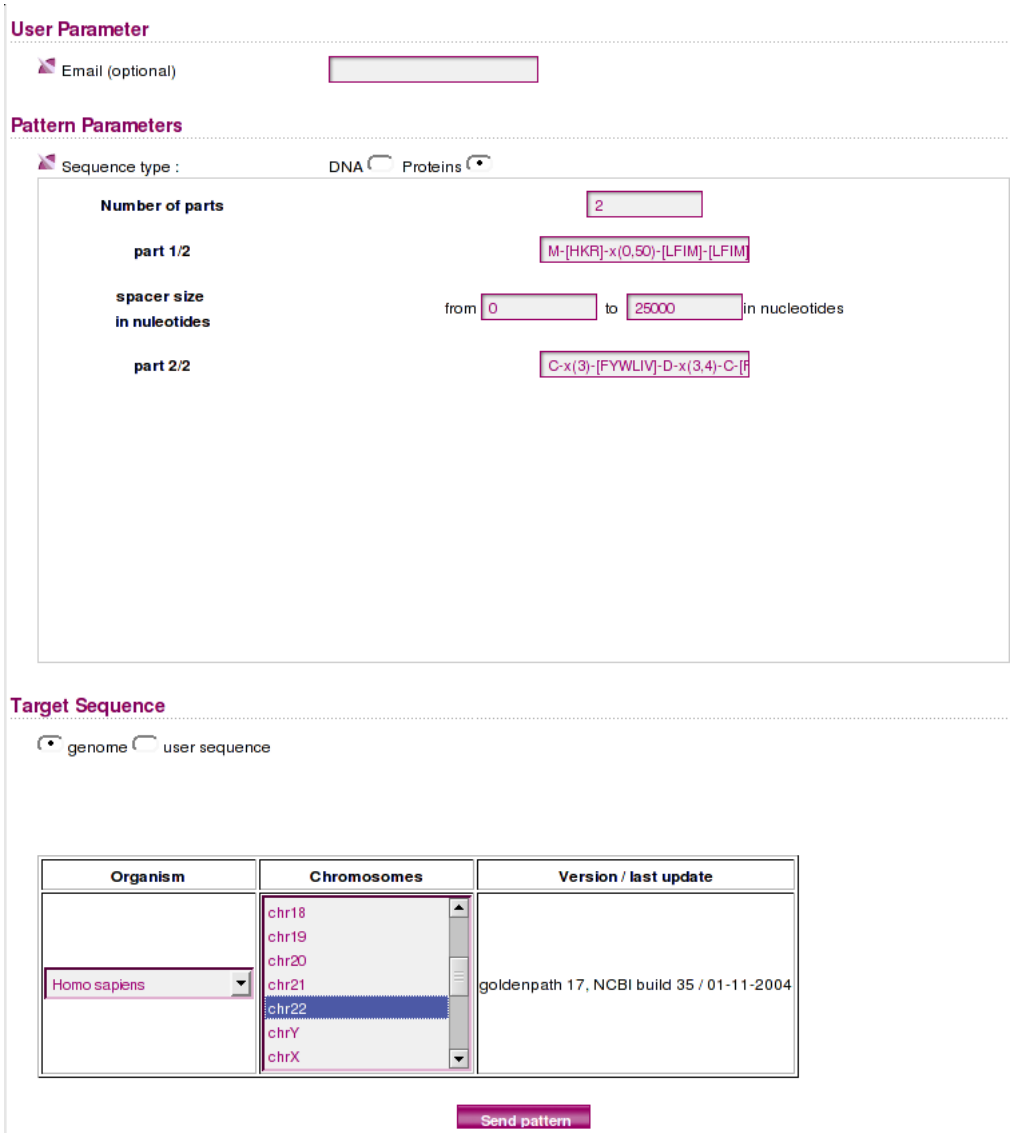
In the following example, we are going to search the human chromosome 22 for a 2 parts pattern. The two parts are defined as M-[HKR]-x(0,50)-[LFIM]-[LFIM]-[LFIM] and C-x(3)-[FYWLIV]-D-x(3,4)-C-[FW]-x(2)-[STAGV]-x(8,9)-C-[PF], respectively. A spacer sizing from 0 to 25000 nucleotides separates them.

2.3.1. Fill in the main page

Filling in the main page of STAN is done as described on page Fill in the main page (Basic example). Then, enter the number of parts of the pattern. Finally, enter the text of the 2 parts as well as the size

of the spacer using the appropriate fields as shown on the following figure:

Figure 2-13. Entering a multi-parts pattern in STAN main page



Note: when entering the various elements (parts and spacers) of a multi-parts pattern, be sure to enter them in the appropriate fields by order: part 1 first, then first spacer, then part 2, then second spacer and so on.

After entering the pattern, select the human chromosome 22 in the section Target Genome of the STAN main page, and then click on the button [Submit Query].

2.3.2. Format the result

From the main page, you now have the 'Formatting Result' page that allows you to choose which components to include in the results of a STAN search. In this example, we choose to include in the results the 3000 nucleotides of the nucleotidic sequence located upstream the first part, the peptidic

sequence of pattern part 1, the nucleotidic sequence of the spacer and the peptidic sequence of pattern part 2.

Figure 2-14. Formatting the result for a two-part pattern

After formatting the result, click on the button [Submit Query] to start the search.

2.3.3. How the search for a multi-parts pattern is performed?

Using a multi-parts pattern, STAN works as follows. In a first step, it searches for matches of the first pattern part in the six frames translation of the target nucleotidic sequence. The result of the search, which consists in a list of positions, is stored in a table. Then, STAN does the same search step up to the fifth pattern part. It is worth noting that a pattern part search is executed independently from the other pattern parts. In a second step, STAN computes all the combinations of the result tables created during step 1, while taking into account spacer sizes. Each valid combination will be reported as a hit in the result file.

Since such a search may result in the production of possible huge amounts of hits, STAN uses a threshold value, T that fixes the maximum number of positions that are stored in each table created during step 1. Thus, when the search of a pattern part produces T positions, STAN stops the search for that part.

Threshold T is also used to limit the computation performed during step 2: if the number of hits reaches T, then STAN stops the search. As a result, a STAN search cannot produce more than T hits.

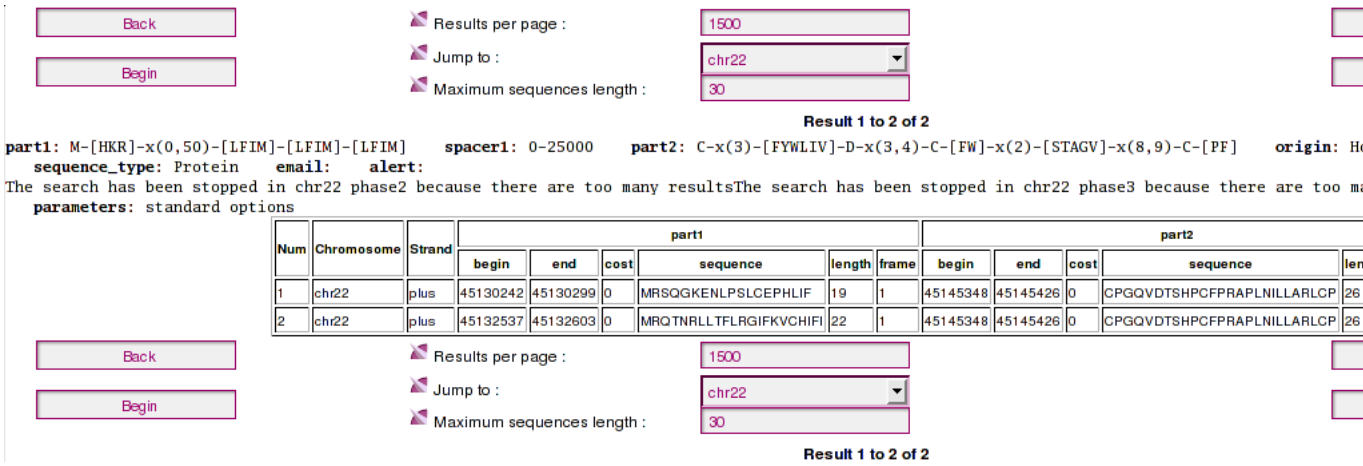
In the current version of STAN, default value for T is 700.

Note: in order to increase the search speed of STAN, it runs on a multi-processors computer where each pattern part search is perfomed on a separate processor.

2.3.4. Viewing the results with a Web browser

The following figure shows the results for the two-part pattern previously defined:

Figure 2-15. Two-parts pattern search results



It is worth noting that this example illustrates a search for which threshold T has been reached (see alert messages above the hit table). So, it is very important to note that this table does not produce all the possible hits of the pattern against human chromosome 22.

Bibliography

[Gusfield 97] D. Gusfield, 1997, *Algorithms on strings, trees and sequences*, Cambridge University Press, page 90.

[McCreight 76] E.M. McCreight, 1976, *A space-economical suffix-tree construction algorithm*, J. ACM., 23, 262-272.

[Ukkonen 95] E. Ukkonen, 1995, *On-line construction of suffix-trees*, Algorithmica, 14, 249-260.

[Kurtz 99] S. Kurtz, 1999, *Reducing the space requirement of suffix-trees*, Soft. Pract. Exper., 29, 1149-1171.

[Dong and Searls 94] S. Dong, D. Searls, 1994, *Gene structure prediction by linguistic methods*, Genomics, 23, 540-551.

[Searls 95] D. Searls, 1995, *String Variable Grammar: a logic grammar formalism for the biological language of DNA*, J. Logic Programming, 14, 73-102.

[Searls 99] D. Searls, 1999, *Formal language theory and biological macromolecules*, Series in Discrete Mathematics and Theoretical Computer Science, 47, 117.

[Searls 02] D. Searls, 2002, *The language of genes*, Nature, 420, 211-217.

[Dsouza et al.] M. Dsouza, N. Larsen, R. Overbeek, 1997, *Searching for pattern in genomic data*, Trends Genet, 13(12), 497-498.

[Kapitonov and Jurka 01] V. Kapitonov, J. Jurka, 2001, *Rolling-circle transposons in eukaryotes*, Proc. Natl. Acad. Sci. USA, 98, 8714-8719.

[Santiago 02] N. Santiago, C. Herraiz, J.R. Goni, X. Messeguer, J.M. Casacuberta, 2002, *Genome-wide Analysis of the Emigrant Family of MITEs of Arabidopsis thaliana*, Mol. Biol. Evol., 19(12), 2285-2293.

[Gattiker 02] A. Gattiker, E. Gasteiger, A. Bairoch, 2002, *ScanProsite: a reference implementation of a PROSITE scanning tool.*, Appl. Bioinformatics, 1(2), 107-108.

[Manber 93] U. Manber, 1993, *Suffix arrays: A new method for on line string searches*, SIAM Journal on Computing, 5, 935-948.

[Lefebvre 03] A. Lefebvre, 2003, *FORRepeats: detects repeats on entire chromosomes and between genomes*, Bioinformatics, 19, 319-326.