Int. J. Bioinformatics Research and Applications, Vol.

GRASPm: an efficient algorithm for exact pattern-matching in genomic sequences

Sérgio Deusdado*

Centre for Mountain Research (CIMO),
Polytechnic Institute of Bragança,
5301-854 Bragança, Portugal
E-mail: sergiod@ipb.pt
*Corresponding author

Paulo Carvalho

Department of Informatics,
School of Engineering,
University of Minho,
4710-553 Braga, Portugal
E-mail: pmc@di.uminho.pt

Abstract: In this paper, we propose Genomic-oriented Rapid Algorithm for String Pattern-match (GRASPm), an algorithm centred on overlapped 2-grams analysis, which introduces a novel filtering heuristic – the compatibility rule – achieving significant efficiency gain. GRASPm's foundations rely especially on a wide searching window having the central duplet as reference for fast filtering of multiple alignments. Subsequently, superfluous detailed verifications are summarily avoided by filtering the incompatible alignments using the *idcd* (involving duplet of central duplet) concept combined with pre-processed conditions, allowing fast parallel testing for multiple alignments. Comparative performance analysis, using diverse genomic data, shows that GRASPm is faster than its competitors.

Keywords: pattern-matching; sequence searching and analysis; motif discovery.

Reference to this paper should be made as follows: Deusdado, S. and Carvalho, P. (2009) 'GRASPm: an efficient algorithm for exact pattern-matching in genomic sequences', *Int. J. Bioinformatics Research and Applications*, Vol.

Biographical notes: Sérgio Deusdado was graduated in 1996, in 2002 received his MSc in Computer Science and, in 2008 received his PhD in Bioinformatics from the University of Minho, Portugal. He is currently Lecturer in Computer Science at the Polytechnic Institute of Bragança (IPB) in Portugal and a researcher at IPB's CIMO (Centre for Mountain Research). His research interests include bioinformatics, information theory, pattern-matching, internetworking, groupware and e-learning.

Paulo Carvalho was graduated in 1991 and received his PhD in Computer Science from the University of Kent, Canterbury, UK, in 1997. He is currently Assistant Professor of Computer Communications, Department of Informatics,

the University of Minho, Portugal. His main research interests include broadband technologies, multiservice networks, mobile networks, data traffic analysis, characterisation and modelling.

1 Introduction

The exponential growth of biological databases has renewed the need for faster searching algorithms. The genomic alphabet because of its peculiarities is a challenge to any pattern-matching algorithm.

Pattern searching is the core of bioinformatics, recurrent in genomic information exploration and essential for homology search, motif discovery and phylogeny inferenceapplications. In genomic data, the most relevant patterns are codons, motifs, polyadenylation signals and genes.

Basically, an exact pattern search algorithm intends to find all instances of a string-pattern p of length m in a text x of length n, being $n \geq m$. Strings p and x are built over a finite set of characters in a given alphabet $\sum$, of size σ. Genomic sequences present the succession of symbols of DNA nucleotides in a genome, using a quaternary alphabet $\sum = \{a, c, t, g\}$.

Advanced pattern-matching algorithms operate in two stages or phases: the first phase includes the pre-processing or study of the pattern, and the subsequent phase, named processing phase, includes searching the text or sequence iteratively.

The key to achieve sublinear performance is to pre-process the pattern to collect useful information to avoid further redundant tests, and consequently boosting the detection of pattern occurrences. Using this knowledge, the processing phase is an iterative process of testing and shifting, concerning to efficiently identify exact matches in successive windows over the sequence.

We propose a new algorithm for exact pattern-matching, based on 2-grams (duplets), and adapted to the specificity of the genomic alphabet, using a search logic based on the exploration of multiple alignments in a wide window, selectively tested after verifying a novel compatibility rule. The shift strategy includes two cumulative components, the first is constant and represents $m - 1$ bases, and the second is variable for each duplet, and represents an extra-shift value. The efficiency of the new algorithm is supported by favourable experimental results obtained by comparison against prominent algorithms described in the next section.

2 Survey on exact pattern-matching algorithms

Most pattern-matching algorithms scan the sequence or text iteratively. Each iteration or attempt comprises a window whose length equals the pattern length. The window is aligned with the pattern and each character involved is compared until a failure or a complete match occurs. The new window for next iteration is initiated further in the sequence; the shifted portion is safely ignored based on the results of the pre-processing phase. The cycle ends when the sequence has been integrally searched. Most efficient algorithms perform fewer comparisons and additively benefit from a simpler logic to evaluate the maximum possible window shift for the next iteration.

Alphabet length and searching performance are strongly correlated. Pattern-matching algorithms are mainly focused on 1-grams or characters analysis/comparisons, however efficiency could be gained enlarging the alphabet by using *n*-grams. *N*-grams are sub-strings composed of *N* characters over an alphabet. Consecutive or overlapping *n*-grams are possible. For example, explaining the use of 2-grams in the proposed algorithm, the sequence 'atgc' could be decomposed in three overlapped 2-grams ('at', 'tg' and 'gc'). The DNA alphabet contains 16 different 2-grams.

The idea of using *n*-grams to maximise the alphabet's potential is not new. This concept was already mentioned by Boyer and Moore (1977). By using combinations of symbols as units, it is possible to obtain a new and larger alphabet. Despite the inherent overhead, this transformation generally enhances the algorithm's efficiency (Salmela et al., 2007). In fact, the use of super-alphabets has a positive outcome as demonstrated in Baeza-Yates (1989), Fredriksson (2003), Tarhio and Peltola (1997) and Lecroq (2007), especially when small alphabets are involved.

Several searching schemes have been developed and refined to improve efficiency. The most common is based on window shifting, being *m* the size of the window and looking for only one pattern occurrence within the window. However, larger searching windows are possible if we consider multiple pattern occurrences within the window (He et al., 2005). The shifting strategy is another important component. Most algorithms use variable shifts provided by a pre-processed shift table, others use a constant shift avoiding recurrent accesses to the shift table (Kim, 1999). The algorithm proposed here is based on 2-grams and uses a hybrid approach, as it includes cumulatively variable and constant shift values to advance a wide searching window.

From classic pattern-matching algorithms, Knuth–Morris–Pratt (KMP) (Knuth et al., 1977) and Boyer–Moore (BM) (Boyer and Moore, 1977) contributions improved significantly this nuclear computation recurrence in the late 1970s. Furthermore, being a heuristic-based algorithm, BM achieved sublinear time complexity in the average case.

BM's variants are known as BM family algorithms. The most relevant ones are Horspool's variant (Horspool, 1980) and Sunday's Quick Search algorithm (Sunday, 1990). Alternative approaches such as suffix automata, bit-parallelism or hashing have originated newer and representative algorithms, respectively, the following examples: Reverse Factor (RF) (Crochemore et al., 1994), Shift-Or (SO) (Baeza-Yates and Gonnet, 1992) and Karp–Rabin (KR) (Karp and Rabin, 1987).

Recent algorithms follow hybrid approaches with refinements, and incorporate the best contributions from past algorithms to achieve better performance, from which SBNDM (Peltola and Tarhio, 2003) and WML (Lecroq, 2007) were developed, constituting significant state-of-the-art examples. Both algorithms were presented as highly efficient, mainly when dealing with short alphabets, e.g., DNA alphabet.

The SBNDM is a bit-parallelism algorithm based on the Backward Nondeterministic DAWG Matching (BNDM) algorithm (Navarro and Raffinot, 1998, 2000), which has been developed from the Backward DAWG Matching (BDM) algorithm (Crochemore and Rytter, 1994). SBNDM is a simplified and faster version of BNDM, mainly because it operates without prefix searching. The advantage of bit-parallelism algorithms stems from fast bit operations in machine words (usually $\omega = 32$ or $\omega = 64$ bits). Since these algorithms need 1 bit per pattern's character, the word length ω is limitative considering the emergent pattern-searching applications. It is possible to search long patterns using bit-parallelism by splitting the pattern and reusing the algorithm for the necessary machine words to cover the entire pattern, but the performance is penalised.

Recently, Lecroq (2007) proposed an adaptation of the Wu and Manber (1994) multiple string matching algorithm to produce a single string matching algorithm, introducing a new searching strategy based on hashing q-grams. Experimental work showed state-of-the-art results for short patterns on small alphabets, however the presented versions take advantage on using q-grams with $3 \leq q \leq 8$.

When evaluating and comparing the performance of pattern-matching algorithms (Lecroq, 1995; Leidig and Trefftz, 2007; Smyth, 2003; Michailidis and Maragaritis, 2001), several variables affect the conclusions. Variables such as alphabet size, pattern size and computer architectures are among the determinant factors to elect the best choice.

3 The new algorithm

3.1 Description

GRASPm is an efficient algorithm that improves exact pattern-matching in genomic sequences. GRASPm could be classified as a heuristic-based algorithm, analysing multiple pattern alignments within a wide search window and using a novel filtering heuristic to maximise efficiency.

GRASPm acts in two sequential phases:

- the pre-processing phase is focused on the pattern study
- the searching phase is committed to selectively scan the sequence aiming to identify pattern replicas in successive iterations.

In more detail, the pre-processing phase is devoted to understand and capture all the peculiarities of the pattern, contributing to optimise pattern recognition in the processing phase and minimising the number of required attempts. As GRASPm elects the duplet as reference (using overlapped 2-grams), initially, the algorithm analyses accurately the pattern, focusing on its duplet composition. The key knowledge gathered in this phase is organised in two tables, an extra-shift table and a compatibility table.

Acting cumulatively with a default window shift, GRASPm uses an extra-shift table. The extra-shift table pre-processing is based on pattern composition, in terms of duplets. For each one of the 16 possible duplets, an extra-shift value is computed based on its last occurrence in the pattern, similar to BMH shift strategy.

The pre-processing phase ends with the compatibility rule analysis, which represents the pattern duplets' classification as compatibles or incompatibles. Compatibility is analysed regarding alignments' co-existence conditions facing the surrounding bases for each pattern duplet, being especially useful when multiple alignments under the same duplet are present. The resulting compatibility table stores the parameters needed to test the compatible alignments in the correct order, and it will be recurrently used later in the processing phase. The compatibility rule details, complemented with a practical example, will be provided further in this section.

The searching phase also includes innovative ideas. The alignments of the pattern within the search window are better explored in GRASPm; the algorithm looks for possible alignments within a larger window having as reference the window's central duplet. The key idea is that being $2(m - 1)$ the search window length, if it contains

a pattern occurrence then the pattern includes necessarily the window's central duplet. A similar concept was already used in Kim (1999) and referred as occurrence rule. This is a powerful concept, as it enables a larger search window and a guaranteed constant shift per pattern of $m - 1$ bases. Additionally, GRASPm uses a variable extra-shift rule to maximise shifts.

A preliminary filter consists of recognising or not the search window's central duplet as a pattern's duplet. As not all partial alignments become total matches, the next filter function aims to discard incompatible partial alignments by applying the compatibility rule. Only compatible alignments are candidates to become full matches; this pre-condition fulfilment generally implies that four bases are already verified. Character comparisons are only used in doubly filtered cases, restricting greatly the character comparisons needed and consequently boosting pattern searching performance.

The main ideas of GRASPm are listed here and conceptually summarised as:

Duplet vision

Aiming to virtually enlarge the genomic alphabet, strings are viewed as duplet based (overlapped 2-grams) instead of character based. A shift table based on duplets produces longer shifts. A pattern p of length m has $m - 1$ overlapped duplets.

Wide search window

The pattern length determines the search window length. Considering $s = m - 1$, the window length is 2s. In the first iteration, the central duplet is positioned at $cd_1 = s$, in subsequent iterations the position of its first base is given by $cd_i = cd_{i-1} + s + \text{extra\_shift}(\text{duplet}((cd_{i-1} + s), (cd_{i-1} + s + 1))$. The GRASPm search window is by definition the greatest subsequence to safely test possible pattern alignments having the window's central duplet as reference. If the pattern p occurs in the search window, p includes necessarily the central duplet initiated at the character $x[cd_i]$ of the window i. As a result, there are a maximum of $m - 1$ possible pattern occurrences within a window.

Involving Duplet of the Central Duplet (*idcd*)

Each window contains an idcd, which is used in GRASPm as a key discriminator for the compatibility rule. It is formed agglutinating the surrounding bases of the central duplet. Formally, $\text{idcd}_i = x[cd_i - 1] + x[cd_i + 2]$. The idcd test implies two simple character comparisons previewed by the compatibility rule. If the pattern exists in a certain window, except when the central duplet is aligned with the first or last duplet of the pattern, the idcd is part of the pattern and could be used to group pattern alignments that share the same central duplet and idcd. To capitalise this feature, during the pre-processing phase, GRASPm simulates all idcd possibilities for all pattern duplets seeking to classify those that do not match the pattern as incompatible alignments. In the processing phase, only the alignments having a viable idcd are effectively tested.

Compatibility rule

In GRASPm, a plausible alignment means that a pattern duplet coincides with the central duplet in the search window. It also means that we need to test the remaining $m - 2$ bases to eventually declare a complete match. Before proceeding with the settings to verify a plausible alignment, GRASPm introduces a second filter based on the analysis of the idcd_i and its compatibility facing all the occurrences of the window's

central duplet in the pattern. Compatible bases, allowing at least one valid alignment, must flank the central duplet. The compatibility rule is a selective rule that discards the incompatible alignments. Except for the first and last duplets, surrounding each pattern duplet exist a preceding base and a succeeding base forming a 'virtual' involving duplet; these bases are relevant because they can be used to preview useful duplet compatibilities in the search phase since future idcds will be aligned with the referred bases. A duplet is compatible with a future idcd if possesses, at least, one compatible alignment. The incompatibility is determined if the idcd of the window does not enfold any duplet in the pattern. In the searching phase, pre-processed compatibilities for each central duplet in analysis are available, allowing selective alignment trials. Each duplet in the pattern is characterised, beyond its occurrences and positioning details, by its compatibility conditions, i.e., the required alignments' indexes in case of tests necessity facing a specific idcd. The compatibility rule represents a good-performance trade-off, as it avoids a huge number of superfluous comparisons, and additively their necessary set-up. GRASPm's major achievement in efficiency relies on this rule.

Guaranteed window shift of m – 1 bases

The search window is, at least, constantly shifted $s = m - 1$ bases per iteration. This is possible since a new pattern instance could only occur, at least, one base ahead of the last possible alignment tested in the previous window. To maximise shift possibilities, GRASPm uses as complement a pre-processed shift table, inspired in the BMH shift table. GRASPm's extra-shift table uses duplets instead of characters, hence providing larger shift values on average. Applying the extra-shift rule to the duplet initiated at $cd_i + s$, the returned extra-shift value is added to s (the default value), constituting the total shift value applicable. For the same sequence, the new algorithm needs a number of iterations considerably lower than the existing algorithms, which only count on the variable shift rule. GRASPm may reach a shift value of $2(m - 1)$ in the best case.

3.2 Methodology

To better explain the underlying methodology of the proposed algorithm, the theory is presented along with a complete and illustrative example. The example considers a pattern p and a sequence x defined as follows:

$P =$ "*atgtgtgcat*"; $m = 10$; $m - 1 = 9$ *overlapped duplets*

$x =$ "*accgtatcattgcccatgtgtgcatgtgccaattctcgagtaccc*"; $n = 45$

3.3 Data structures set-up

As referred, GRASPm is mostly focused on 2-grams and analyses the pattern considering its overlapped duplets. Consequently, it becomes necessary for a data structure (*ituples*, see Table 1) to index and recognise the 16 possible combinations of DNA dinucleotides when the sequence is read. The first column (numbered zero) is exceptionally needed to establish a default value for the idcd's second base when the central duplet coincides with the last duplet of the sequence. Another array (*ibases*, see Table 2) is used to obtain base's indexes dependently of the character's ASCII code. The combination of these data

structures provides a simple way to obtain the index of a duplet. For instance, to obtain the index corresponding to the duplet 'ac' using a C-like syntax, the expression would be: *index = ituples* [*ibases*['*a*']][*ibases*['*c*']];

Table 1 Dinucleotides' indexes table

0		1	2	3	4
		a	c	t	g
aa 1	1 a	aa 1	ac 2	at 3	ag 4
ca 5	2 c	ca 5	cc 6	ct 7	cg 8
ta 9	3 t	ta 9	tc 10	tt 11	tg 12
ga 13	4 g	ga 13	gc 14	gt 15	gg 16

Table 2 Bases' indexes table

		a		c		g		t
ASCII	…	97	…	99	…	103	…	116
Index	…	1	…	2	…	4	…	3

3.4 Pre-processing phase

The study of the pattern comprehends the analysis of its composition regarding the characterisation of its duplets. These properties, namely the number of occurrences and the corresponding positions, are memorised using an array. This table supplies information, for instance, to do the initial filtering in the search phase and facilitates the compatibility rule pre-processing. Considering the DNA alphabet, 16 different duplets are possible, therefore it is required a bidimensional array with 16 columns and a variable number of lines depending on the pattern composition and extension. Table 3 illustrates the study's resulting data for the current pattern example.

Another fundamental component of GRASPm is the compatibility table, which is subdivided into 16 sub-tables, one for each possible duplet. Each sub-table contains for each possible idcd, the indexes of the compatible alignments or zero if the alignment is incompatible. In fact, in the next phase, if the idcd of the alignment does not match the idcd in the pattern, then the alignment is incompatible and testing it extensively is avoided. On the contrary, if it is compatible, we will know exactly where to search. In practice, especially when there are multiple alignments for the same duplet, usually few are compatible. As a consequence, the efficiency is improved executing selective tests.

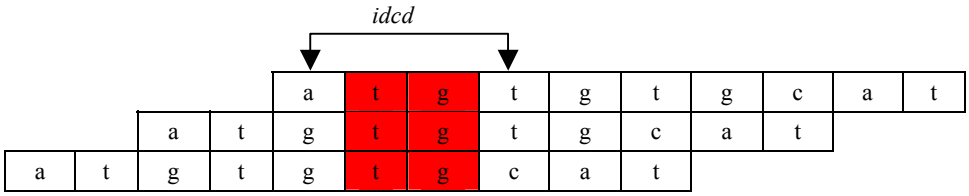
Table 3 Study of pattern's overlapped duplets

Index	1	2	3	4	5	5	7	8	9	10	11	12	13	14	15	15
Duplet	*aa*	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
Occurrences	0	0	2	0	1	0	0	0	0	0	0	3	0	1	2	0
Occurrences' indexes			1		8							2		7	3	
			9									4			5	
												6				

For the ongoing example, after analysing the three different alignments of the duplet 'tg' in the pattern (as shown in Table 4), the compatibility rule states the incompatibility and inhibition for any idcd different from 'at', 'gt', or 'gc'. Analysing in detail each one of the viable alignments and its compatibility conditions, if the idcd is 'at', only the alignment of the central duplet of the window with the same duplet in the pattern initiated at $p[2]$ needs to be tested. Considering an *idcd* = 'gt', a test involving the pattern alignment at 4th base is required and an idcd = 'gc' also implies further verifications, specifically by aligning the pattern at 6th base with cd_i.

The first and last duplets of the pattern are special cases, hence the compatibility rule is only applicable to the base succeeding or preceding the cd_i, respectively.

Table 4 Different alignments of the duplet 'tg' in the pattern (see online version for colours)



Considering the pattern example and its duplets (see Table 3), only five sub-tables will be updated, Table 5 encloses those sub-tables, representing the compatibility rule's pre-processing results. Each sub-table represents a pattern duplet or future *cd* of the search window. The 16 rows of each sub-table represent each eventual idcd. The indexes greater than zero indicate the compatible alignments and represent the position of the duplet from which the pattern should be aligned.

Finally, the extra-shift table is pre-processed. As mentioned before, GRASPm has a regular shift of $m - 1$ bases plus an extra-shift value provided by a pre-processed extra-shift table, the sum will indicate the distance to the next central duplet. The extra-shift rule provides a shift value depending on the duplet initiated at $dc_i + s$ in the sequence. If this duplet has no occurrences in the pattern, then a maximum extra-shift ($m - 1$ bases) is allowed because the next window can be initiated immediately after it. As there are five different duplets in the current pattern, five exceptions to maximum extra-shift may occur. The last duplet in the pattern corresponds to a null extra-shift since the window could not be safely advanced more than the default value. The remaining

pattern duplets provide a variable extra-shift value calculated facing the distance, in duplets, of its last occurrence to the final duplet in *p*. Table 6 shows the pre-processed extra-shift values obtained for the current pattern example.

Table 5 Compatibility rule sub-tables after pattern analysis

"at"

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
aa	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
0	0	0	1	9	9	9	9	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

"tg"

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
aa	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
0	0	2	0	0	0	0	0	0	0	0	0	0	6	4	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

"gt"

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
aa	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

"gc"

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
aa	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

"ca"

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
aa	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Table 6 The resulting extra-shift table

aa	*ac*	*at*	*ag*	*ca*	*cc*	*ct*	*cg*	*ta*	*tc*	*tt*	*tg*	*ga*	*gc*	*gt*	*gg*
9	0	9	9	1	9	9	9	9	9	9	3	9	2	4	9

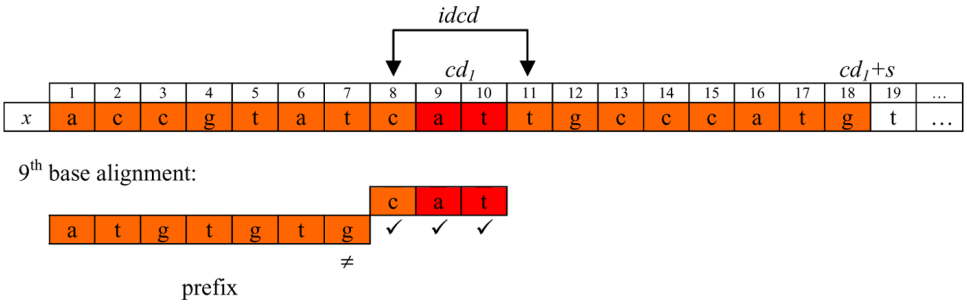
3.5 *Processing phase*

The searching phase is based on alignment probes over the search window, which is iteratively shifted to discover instances of the pattern. The central duplet of each search window serves as axle to align with the pattern whenever the probability of its occurrence still remains after consulting the pre-processed information. If the pattern *p* exists in the search window, *p* includes necessarily the central duplet at cd_i. With this rule in mind, the algorithm will check if the duplet at cd_i has possible and compatible alignments to test, if so, the central duplet and the idcd were already verified and only the remaining characters (prefix and/or suffix) are compared in reverse order looking for a complete match. The searching phase ends when the sequence has been entirely scanned.

As referred, the search window in the proposed algorithm is larger than usual, almost the double, and the same applies to the shift values. Following the current example, the window length is $2s = 18$ and is shifted, at least, in each iteration by $s = 10 - 1 = 9$ bases, and in the best case by $2s = 18$ bases. The first search window's central duplet is initiated at position *s* and composed by $x[s] + x[s + 1] = x[9] + x[10]$ = 'at'.

Tracing the algorithm's behaviour in the processing phase, the initial iteration (see Figure 1) for the example in progress presents a search window with the central duplet 'at'. This 2-gram has two occurrences in the pattern, its first and last duplets. Additionally, the compatibility rule (see Table 5) previews two tests for this duplet: the 9th base alignment test, only if the *idcd* is initiated by 'c' (which is the case), and the first base alignment test if the idcd finishes with 'g' (which is not the case).

Figure 1 First iteration status in the proposed example (see online version for colours)

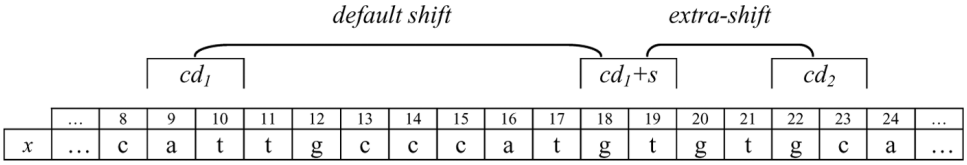


To discover eventual pattern occurrences in the right sequence, back in the compatibility sub-tables pre-processing, the alignments had been registered in the inverse order of appearance in the pattern. In fact, a greater base index alignment corresponds to an earlier pattern occurrence in the sequence. Accordingly, the examination of multiple compatible alignments in the search phase follows the order established in the respective compatibility sub-table. In the present case, having a central duplet = 'at' combined with an $idcd_1$ = 'ct', only the 9th base alignment is compatible.

The pattern is then aligned with the central duplet by the 9th base and, cyclically, character comparisons are performed backwards till a failure or a complete match occurs. Since the bases composing the $idcd_1$ and cd_1 were pre-tested, the next step in GRASPm is to evaluate the length of the suffix, being tested only if it is greater than zero. A complete match in the suffix joint with the existence of unverified characters is followed by the prefix verification. For this alignment, the first character comparison is negative, and having no more alignments to test, this mismatch truncates the iteration.

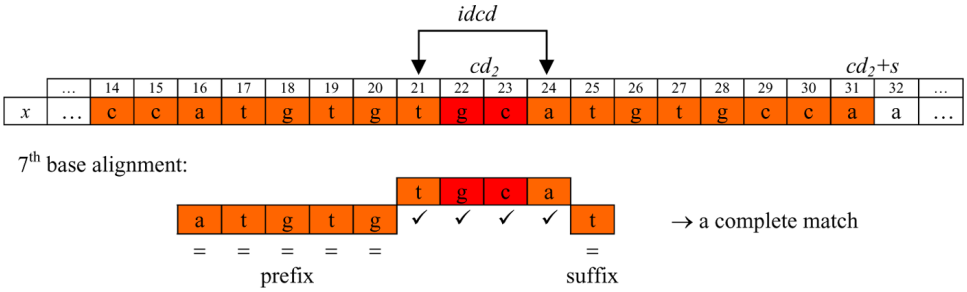
Preparing the next iteration, the next *cd* is computed. The GRASPm's shift rule includes two shift parcels to be added, the default shift $s = m - 1$ and the extra-shift value obtained from the pre-processed extra-shift table. The shift value for the next iteration is $9 + 4 = 13$ since the extra-shift rule for the duplet at position $cd_1 + s$ = 'gt' returns 4, thus, cd_2 will be positioned at the 22nd base (see Figure 2).

Figure 2 GRASPm's extra-shift rule application at the end of the first iteration



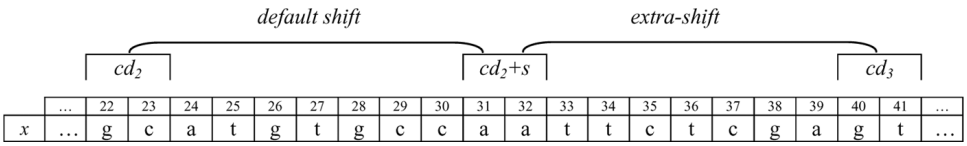
In the second iteration (see Figure 3), the central duplet = 'gc' at cd_2 exists in the pattern and has one possible alignment, which is compatible with the current $idcd_2$ = 'ta', therefore, subsequent character comparisons are necessary. All characters in the suffix and prefix match the sequence alignment, which means the discovery of a pattern occurrence.

Figure 3 Second iteration status (see online version for colours)



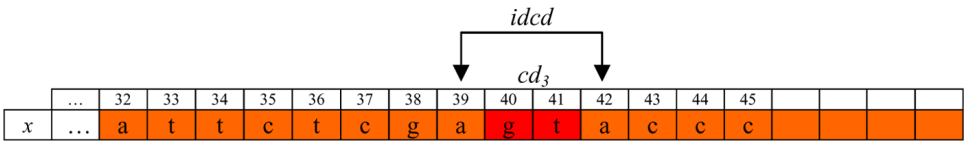
The shift value for the next iteration is maximal ($2s = 18$) since the duplet analysed by the extra-shift rule does not integrate the pattern. In fact, the extra-shift rule for the duplet at position $cd_2 + s$ = 'aa' returns 9, thus, cd_3 will be positioned at the 40th base (see Figure 4).

Figure 4 GRASPm's extra-shift rule application at the end of the second iteration



In the third and last iterations (see Figure 5), the central duplet = 'gt' at cd_3 has multiple occurrences in the pattern, however the compatibility rule discards all the alignments in the presence of the current $idcd_3$ = 'aa'. The compatibility rule has a major role in GRASPm, through a single and simple operation numerous character comparisons are encapsulated, leading to a parallel searching capability, which means a breakthrough facing the *modus operandi* of existing algorithms.

Figure 5 Third iteration status (see online version for colours)



As cd_4 would exceed the sequence length, the searching phase is concluded.

3.6 Implementation

To clarify the implementation details of the proposed algorithm regarding the search phase, an implementation proposal, in C language, is presented in Figure 6.

Figure 6 GRASPm's search phase: an implementation proposal in C language

```
void GRASPm_Search(char *seq, long n, char *pat, unsigned int m)
{
long cd;
int s,id,idcd,ia,cal,ias,j,suf;

 s=m-1;
 cd=s-1;
 while (cd<n-1)
 {
   id=ituples[ibases[seq[cd]]][ibases[seq[cd+1]]];
   if (duplets_in_pattern[id][1]!=0)
     {
      idcd=ituples[ibases[seq[cd-1]]][ibases[seq[cd+2]]];
      ia=1;
      while ((cal=compatibilities[id][idcd][ia])>0)
       {
        suf=m-cal-2;
        ias=cd+suf+2;
        j=0;
        while ((j<suf) && (seq[ias-j]==pat[s-j])) j++;
        if (j>=suf)
         {
          j+=4;
          while ((j<m) && (seq[ias-j]==pat[s-j])) j++;
          if (j>=m) printf("\nPattern at base %d.", ias-s);
         }
        ia ++;
       }
     }
   cd+=s+xshift[ituples[ibases[seq[cd+s]]][ibases[seq[cd+s+1]]]];
  }
}
```

The $m = 2$ case deserved a special attention to enhance efficiency when searching dinucleotide patterns. This particular case is treated based on a simple idea:

differentiating dinucleotides constituted by different bases or equal bases, deciding an appropriated searching advance in accordance with the rightmost base analysis. In fact, a new instance of a same-base dinucleotide may only occur two bases ahead of the previous mismatch. A similar logic is applied in mixed-base dinucleotides. This strategy does not guarantee an optimal advance, but its simplicity grants remarkable performance, also beating existing algorithms in this particular case.

3.7 Complexity analysis

The GRASPm's pre-processing phase includes mainly four components:

- data structures set-up
- pattern study
- compatibility rule pre-processing
- extra-shift rule pre-processing.

The overall space and time complexity in this phase is predominantly influenced by the third component.

Pre-processing phase's Space complexity ((*i*) + (*ii*) + (*iii*) + (*iv*)):

$$O(m) + O(\sigma) + O(\sigma^2 m) + O(\sigma^4(m-1)) + O(\sigma^2).$$

Pre-processing phase's Time complexity ((*i*) + (*ii*) + (*iii*) + (*iv*)):

$$O(m) + O(\sigma) + O(2(m-1)) + O(m^2) + O(2m).$$

Time complexity in the search phase is in the best case $O(n/(2(m-1)))$, this case occurs when the central duplet is always inexistent in the pattern and the extra-shift is maximal. In the worst case, the extra-shift contribution is always null and each window requires the examination of the maximum number of alignments ($m - 1$). As in the worst case the pattern is uniform, only two operations are needed to overcome the two filters (central duplet and idcd) requirements for all the alignments within the window and, in this manner, several bases are verified in parallel. Each alignment requires a set-up (two operations maximum) to evaluate the suffix and initiate backward character comparisons. Finally, the remaining character comparisons needed for each alignment are, in the worst case, $m - 4$. Therefore, the time complexity in this case is:

$$O((n/(m-1))(2+2+(m-1)(m-4)) = O(n(4/(m-1)+m-4)),$$

thus $O(nm)$ in the worst case.

The new algorithm's complexity analysis suggests sublinear average-case behaviour, even so this assumption requires further theoretical analysis.

4 Experimental results and comparisons

Regarding bioinformatics, and specifically in genomic applications ($\sigma = 4$), BNDM and SO families are considered among the most efficient pattern-matching algorithms. To benchmark the performance of the proposed algorithm, for simplicity reasons,

all comparisons have been confined to three reference searching algorithms: the BMH algorithm, because it is generically considered the fastest of all classical algorithms; the SBNDM algorithm, pointed out as highly efficient regarding exact pattern-matching in biological sequences; the WML, presented in 2007 as the most efficient, considering pattern-matching in small alphabets contexts.

We have put our best efforts to assure fair and equal conditions to our experiments, in this way we have gathered genuine and optimised versions, written in C language, of the algorithms used in performance comparison. An adapted version of BMH to use 2-grams, either during pre-processing or searching, was produced based on the implementation publicised in Lecroq (1995), and genuine implementations, gently provided by the authors, were used to test SBNDM and WML algorithms. As the WML algorithm was originally presented to use q-grams with $3 \leq q \leq 8$, an adaptation was made to use 2-grams to keep equality. A new version of SBNDM, capable of analysing 2-grams, was also produced. GRASPm was equally written in C language, the source code is available for academic purposes by request. The applications were compiled with *gcc* (version 3.4.2) using full optimisation –O3.

Performance tests were executed using a system based on an Intel Pentium IV processor, 3.4 GHz, 8 KB L1 + 512 KB L2 cache and 1 GB DDR-RAM, under Windows XP Professional SP2 OS. Execution times were collected (in ms) using the *timeGetTime*() function, provided by an OS library. The run time measured for each algorithm includes both pre-processing and processing phases.

The sequences used for testing were the complete *E. coli* genome (*prokaryote*), with about 4.6 Megabases, and the Human Chromosome 1 (*eukaryote*), with nearly 250 Megabases. A pattern collection containing 700 different patterns, based on 100 samples by length class, with $m = 2, 4, 8, 16, 32, 64$ and 128, were randomly generated and stored in a file for test purposes. Owing to word length limitations, SBNDM2 was only tested for patterns with $m \leq 32$. The mean execution times were used to establish the following performance comparison.

According to the results, GRASPm reveals supremacy in all pattern categories and extensions tested. The performance improvement is more evident as the pattern length increases, the minimum is 10% when $m = 4$. For patterns with eight bases, GRASPm is on average 15% more efficient; with $m = 16$ and $m = 32$ the efficiency gains are about 20%. For 64 base patterns, thus with $m > \omega$, GRASPm performs three times better than its closest competitor. For long patterns, the algorithm's performance is even more convincing, e.g., to search 128 bases patterns is five times faster using GRASPm. Figure 7 presents the comparative execution times of the involved algorithms when searching for different pattern lengths in two different sequences:

a the *E. coli* genome

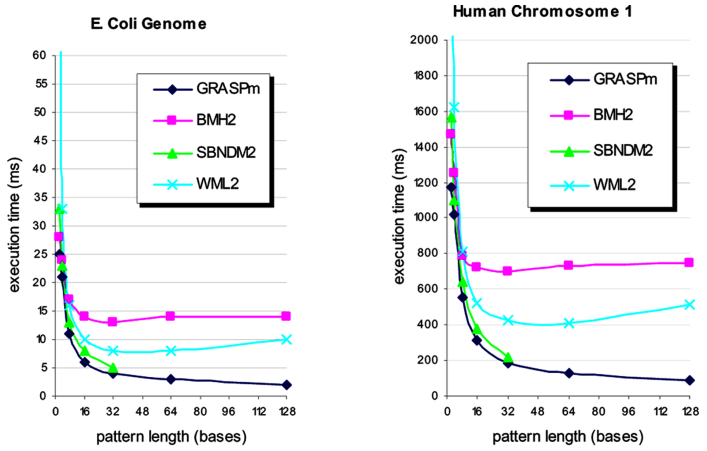
b the Human Chromosome 1.

SBNDM2 algorithm, like any bit-parallelism algorithm, has the performance peak coincident with ω, the tested version, similar to the original, was not prepared to deal with patterns larger than ω. However, the comparisons validity remains untouched as such a version will always include more decisions and consequently more overhead.

Considering genomic searches, WML algorithm needs to be implemented with 3-grams to be comparable, in terms of performance, to GRASPm using 2-grams. BM family algorithms need a 4-grams implementation to be comparable, in terms of performance, to SBNDM2.

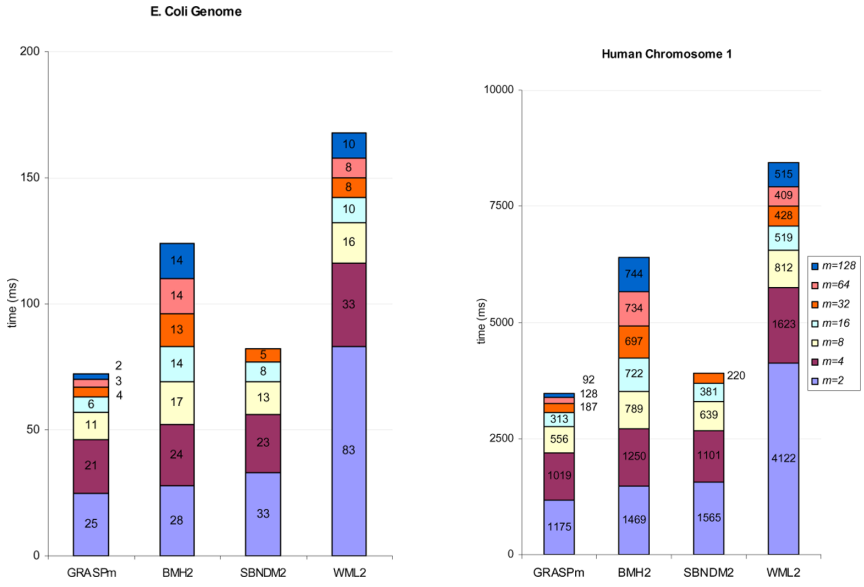
To complement the performance analysis, we suggest looking at the execution time results by other perspective (see Figure 8), i.e., illustrating the sum of mean execution times, for each algorithm, searching all pattern length categories using the *E. coli* sequence (a) and the Human Chromosome 1 sequence (b).

Figure 7 Comparative execution times vs. pattern length (see online version for colours)



(a) (b)

Figure 8 Comparative sums of mean execution time per pattern length (see online version for colours)



(a) (b)

Analysing the algorithms' behaviour and performance when scaling the pattern, only GRASPm and SBNDM2 take continued advantage on ascending pattern length. Additionally, GRASPm evinces better performance stability for each pattern length independently of its composition, which is not true for the other contenders that have significant standard deviations.

The innovative compatibility rule and its inherent selectivity, along with the searching scheme, based on wide searching windows, allow multiple alignments, concentrate and justify the GRASPm power. The extra-shift table corresponds to a small performance gain. Concretely, based on additional experimental results without this table, GRASPm would be, on average, 5% slower in patterns with $m < 50$, but 5% faster in larger patterns.

5 Discussion and conclusions

This paper has proposed GRASPm, an efficient algorithm for genomic exact pattern-matching. Its underlying logic is innovative, as it introduces a novel searching strategy, maximising the efficiency of searching multiple alignments within a search window by introducing a new filtering heuristic, the compatibility rule. The compatibility rule is based on a compatibility table obtained owing to an enhanced pre-processing phase, proactively extracting decisive knowledge from the pattern. Basically, the proposed algorithm requires more pre-processing time but clearly takes the benefits in the searching phase. The GRASPm algorithm uses a deeply selective testing methodology, inhibiting the overhead of worthless alignments, and frequently analysing compatible alignments in parallel.

The new algorithm was tested and compared with the best pattern-match algorithms available, but limiting the use of *n*-grams to 2-grams to assure fairness and keep the algorithms as generalists as possible. Considering bioinformatics, codon searches are common and a 4-grams based algorithm cannot search codons. The experimental results show GRASPm as the fastest algorithm, widening its advantage as pattern length increases. GRASPm is a highly efficient searching algorithm, as it reduces the number of character comparisons and the runtime period required, converging faster. Overall, GRASPm demonstrates optimised pattern recognition owing to deeply selective searches.

The new algorithm's complexity analysis suggests sublinear average-case behaviour. GRASPm performance measurements were very close to logarithmic curves when testing ascending pattern lengths. Comparatively with other approaches, the new algorithm depends more on space complexity. However, the amount of memory required is negligible considering today's hardware capabilities.

In summary, GRASPm represents an evolution in genomic exact pattern-matching efficiency. The performance improvement over concurrent approaches is enabled by a renewed searching strategy including a novel and efficient compatibility heuristic as secondary filter.

Acknowledgement

This work was supported in part by a grant from the Portuguese government – PRODEP/Action 5.3.

References

Baeza-Yates, R.A. (1989) 'Improved string searching', *Software-Practice and Experience*, Vol. 19, pp.257–271.

Baeza-Yates, R.A. and Gonnet, G.H. (1992) 'A new approach to text searching', *Commun. ACM*, Vol. 35, pp.74–82.

Boyer, R.S. and Moore, J.S. (1977) 'A fast string searching algorithm', *Commun. Assoc. Comput. Mach.*, Vol. 20, pp.762–772.

Crochemore, M. and Rytter, W. (1994) *Text Algorithms*, Oxford University Press, New York.

Crochemore, M., Czumaj, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W. and Rytter, W. (1994) 'Speeding up two string matching algorithms', *Algorithmica*, Vol. 12, pp.247–267.

Fredriksson, K. (2003) 'Shift-or string matching with super-alphabets', *Information Processing Letters*, Vol. 87, pp.201–204.

He, L., Fang, B. and Sui, J. (2005) 'The wide window string matching algorithm', *Theoretical Computer Science*, Vol. 332, pp.391–404.

Horspool, R.N. (1980) 'Practical fast searching in strings', *Software – Practice and Experience*, Vol. 10, pp.501–506.

Karp, R.M. and Rabin, M.O. (1987) 'Efficient randomized pattern-matching algorithms', *IBM J. Res. Dev.*, Vol. 31, pp.249–260.

Kim, S. (1999) 'A new string-pattern matching algorithm using partitioning and hashing efficiently', *Journal of Experimental Algorithmics* (*JEA*), Vol. 4, p.2-es.

Knuth, D.E., Morris, J.H. and Pratt, V.R. (1977) 'Fast pattern matching in strings', *SIAM J. Comput.*, Vol. 6, pp.323–350.

Lecroq, T. (1995) 'Experimental results on string matching algorithms', *Software – Practice and Experience*, Vol. 25, pp.727–765.

Lecroq, T. (2007) 'Fast exact string matching algorithms', *Information Processing Letters*, Vol. 102, pp.229–235.

Leidig, J. and Trefftz, C. (2007) 'A comparison of the performance of four exact string matching algorithms', *IEEE International Conference on Electro/Information Technology*, IEEE, Chicago, IL, USA, pp.333–336.

Michailidis, P.D. and Maragaritis, K.G. (2001) 'On-line string matching algorithms: survey and experimental results', *Int. J. Computer Mathematics*, Vol. 76, No. 4, pp.411–434.

Navarro, G. and Raffinot, M. (1998) 'A bit-parallel approach to suffix automata: fast extended string matching', *Lecture Notes in Computer Science*, Vols. 1448–1998, pp.14–33, http://www.springerlink.com/content/5238211580894375/

Navarro, G. and Raffinot, M. (2000) 'Fast and flexible string matching by combining bitparallelism and suffix automata', *ACM Journal of Experimental Algorithms*, Vol. 5, pp.1–36.

Peltola, H. and Tarhio, J. (2003) 'Alternative algorithms for bit-parallel string matching', *SPIRE '03, 10th Symposium on String Processing and Information Retrieval*, *Lecture Notes in Computer Science*, Springer, Vol. 2857, pp.80–94.

Salmela, L., Tarhio, J. and Kytöjoki, J. (2007) 'Multipattern string matching with q-grams', *Journal of Experimental Algorithmics* (*JEA*), Vol. 11.

Smyth, B. (2003) *Computing Patterns in Strings*, Pearson Addison-Wesley, Essex, England.

Sunday, D.M. (1990) 'A very fast substring search algorithm', *Commun. Assoc. Comput. Mach.*, Vol. 33, pp.132–142.

Tarhio, J. and Peltola, H. (1997) 'String matching in the DNA alphabet', *Software-Practice and Experience*, Vol. 27, pp.851–861.

Wu, S. and Manber, U. (1994) *A Fast Algorithm for Multi-Pattern Searching*, TR-94-17, Department of Computer Science, University of Arizona, Tucson.