Looking for monotonicity properties of a similarity constraint on sequences

Ieva Mitasiunaite
INSA Lyon, LIRIS CNRS UMR 5205
F-69621 Villeurbanne, France
Ieva.Mitasiunaite@insa-lyon.fr

Jean-François Boulicaut
INSA Lyon, LIRIS CNRS UMR 5205
F-69621 Villeurbanne, France
Jean-Francois.Boulicaut@insa-lyon.fr

ABSTRACT

Constraint-based mining techniques on sequence databases have been studied extensively the last few years and efficient algorithms enable to compute complete collections of patterns (e.g., sequences) which satisfy conjunctions of monotonic and/or anti-monotonic constraints. Studying new applications of these techniques, we believe that a primitive constraint which enforces enough similarity w.r.t a given reference sequence would be extremely useful and should benefit from such a recent algorithmic breakthrough. A non trivial similarity constraint is however neither monotonic nor anti-monotonic. Therefore, we have studied its definition as a conjunction of two constraints which satisfy the desired monotonicity properties: a pattern is called similar to a reference pattern x when its longest common subsequence with x (LCS) is large enough (i.e., a monotonic part) and when the number of deletions such that it becomes the LCS is small enough (i.e., an anti-monotonic part). We provide an experimental validation which confirms the added value of this approach on a biological database. Classical issues like scalability and pruning efficiency are discussed.

1. INTRODUCTION

We strongly believe that the inductive database framework can efficiently support complex knowledge discovery (KDD) processes [14, 9, 4]. The so-called inductive queries are declarative queries which express the constraints that have to be satisfied by the solution patterns. Then, typical challenges are (a) to identify useful primitive constraints for various pattern domains (e.g., constraints to express the a priori interestingness of patterns holding in sequence databases), and (b) to be able to design efficient and (when possible) complete solvers for computing every pattern which satisfies a combination of primitive constraints. When feasible, completeness is an invaluable property for supporting end-user driven KDD processes: the semantics of the extracted patterns is formally specified and this can be used during the crucial interpretation phases.

We are interested in sequence database analysis for various application domains (e.g., genomic data analysis, seismic data analysis, alarm data analysis, WWW usage mining). Given potentially huge databases of sequences (i.e., temporal or 1-dimensional spatial data), our current goal is to support the expression and the evaluation of quite general inductive queries returning sequence (string) patterns. The state-of-the-art is that efficient solvers are available for specific conjunctions of primitive constraints. For instance, many solvers have been designed for frequent sequential pattern mining possibly combined with some more or less restricted types of syntactic constraints [25, 17, 26, 12, 19]. Interesting results concern the efficient processing of regular expression constraints (see, e.g., [11, 20, 1]) thanks to ad-hoc optimization strategies. An alternative promising approach has been developed by De Raedt and colleagues which tackles arbitrary Boolean combination of constraints which are either monotonic or anti-monotonic [10, 7, 8]. Indeed, a key issue for designing efficient solvers is to consider constraint properties (like anti-monotonicity and its dual monotonicity property) and their associated opportunities for search space pruning. Many useful primitive constraints are monotonic (e.g., maximal frequency in a data set, enforcing a given sub-string occurrence) or anti-monotonic[1] (e.g., minimal frequency, avoiding a given sub-string occurrence).

Surprisingly, similarity constraints have been seldomly studied by the data mining community while it is one of the core algorithmic issues in bioinformatics. In this paper we consider that a primitive constraint which would enforce enough similarity w.r.t a given reference pattern would be extremely useful across many application domains. The challenge is then to combine such a constraint with other user-defined constraints. For instance, it is interesting to look for sequences of actions on a WWW site which are frequent for a given group of users, infrequent for another group and which are are similar enough to an expected pattern specified by the WWW site designer. In other terms, we want to add a primitive similarity constraint to the available efficient solvers on the string pattern domain, e.g., FAVST [8]. The problem is that a non trivial similarity constraint is neither monotonic nor anti-monotonic and can not benefit from the recent algorithmic breakthrough. Therefore, our technical contribution is to formalise a similarity constraint as a conjunction of two constraints which satisfy the desired

[1]Some other classes have been studied like, e.g., succinctness, which turns to be a conjunction of monotonic and anti-monotonic constraints for which ad-hoc optimizations can be defined.

monotonicity properties: a pattern is called similar to a reference pattern x when its Longest Common Subsequence with x (LCS) is large enough (i.e., a monotonic constraint which enforces similarity with x) and when the number of deletions such that it becomes the LCS is small enough (i.e., an anti-monotonic constraint which prunes too dissimilar candidates). Section 2 provides the needed definitions. Section 3 formalises our similarity constraint. Section 4 is an experimental validation which confirms the added value of this approach on a biological database. Classical issues like scalability and pruning efficiency are discussed. Section 5 is a short conclusion.

2. PROBLEM SETTING

Definition 1 (Basic notions on strings). *Let Σ be a finite alphabet, a string σ over Σ is a finite sequence of symbols from Σ and Σ^* denotes the set of all strings over Σ. Σ^* is our language of patterns $\mathcal{L}$ and we consider that the mined data set denoted r is a multi-set*[2] *of strings built on Σ. $|\sigma|$ denotes the length of a string σ and ϵ denotes the empty string. We note σ_i the i^{th} symbol of a string σ. A sub-string σ' of σ is a sequence of contiguous symbols in σ, and we note $\sigma' \sqsubseteq \sigma$. σ is thus a super-string of σ', and we note $\sigma \sqsupseteq \sigma'$. We assume that, given a pattern $\phi \in \mathcal{L}$, the supporting set of sequences in r is denoted by $ext(\phi, r) = \{\sigma \in r \mid \phi \sqsubseteq \sigma\}$.*

Example 1. *Let $\Sigma = \{a, c, g, t\}$. acct, acgct, ϵ are examples of strings over Σ. Examples of sub-strings for acgct are a and gct. aacgctg is an example of a super-string of acgct. If r is $\{acttc, agttca, ttacg\}$, $ext(ttc, r) = \{acttc, agttca\}$.*

Definition 2 (Constraints/inductive queries). *A constraint is a predicate that defines a property of a pattern and evaluates either to true or $false$. An inductive query on $\mathcal{L}$ and r with parameters p is fully specified by a constraint Q and its evaluation needs the computation of $\{\phi \in \mathcal{L} \mid Q(\phi, r, p) \text{ is true}\}$ [16]. In the general case, Q is a Boolean combination of the so-called primitive constraints.*

Definition 3 (Generalisation/specialisation). *A pattern ϕ is more general than a pattern ψ (denoted $\phi \succeq \psi$) iff $\forall r\ ext(\phi, r) \supseteq ext(\psi, r)$. We also say that ψ is more specific than ϕ (denoted $\psi \preceq \phi$). Two primitive constraints can be defined: $MoreGeneral(\phi, \psi)$ is true iff $\phi \succeq \psi$ and $MoreSpecific(\phi, \psi)$ is true iff $\phi \preceq \psi$.*

For strings, constraint $SubString(\phi, \psi) \equiv \phi \sqsubseteq \psi$ (resp., $SuperString(\phi, \psi) \equiv \phi \sqsupseteq \psi$) are instances of $MoreGeneral$ (ϕ, ψ) (resp., $MoreSpecific(\phi, \psi)$). In other terms, $\forall \phi, \psi \in \mathcal{L}$, $\phi \succeq \psi$ iff $\phi \sqsubseteq \psi$.

Definition 4 (Some primitive constraints). *Typical syntactic constraints are $MinLen(\phi, len) \equiv |\phi| \geq len$ and $MaxLen(\phi, len) \equiv |\phi| \leq len$. Assume that $Fr(\phi, r)$ denotes the number of strings in r that are super-strings of ϕ, i.e., $|ext(\phi, r)|$. Given a threshold value f, $MinFr(\phi, r, f) \equiv Fr(\phi, r) \geq f$ (resp. $MaxFr(\phi, r, f) \equiv Fr(\phi, r) \leq f$) denotes a minimal (resp. maximal) frequency constraint in r.*

[2]Data may contain multiple occurrences of the same sequence.

Example 2. *If $r = \{acg, act, gt, t, gt\}$ and $\Sigma = \{a, c, g, t\}$, $Fr(acg, r) = 1$, $Fr(gt, r) = 2$, $Fr(ag, r) = 0$, and $Fr(\epsilon, r) = 5$. $MinFr(gt, r, 2)$, $MaxFr(acg, r, 2)$, $MoreGeneral(t, gt)$, and $MinLen(acg, 3)$ are examples of satisfied constraints. $Q \equiv MinFr(\phi, r, 2) \wedge MaxFr(\phi, r, 4) \wedge MinLen(\phi, 2)$ is an example of an inductive query whose solution set is $\{ac, gt\}$.*

Definition 5 ((Anti-)monotonicity [10]). *Let r be a data set, $\mathcal{L}$ be the pattern language and p be parameters. A constraint Q is anti-monotonic iff $\forall r$ and $\forall \phi, \psi \in \mathcal{L}$, $\phi \succeq \psi \Rightarrow Q(\psi, r, p) \rightarrow Q(\phi, r, p)$. Dually, a constraint Q' is monotonic iff $\phi \preceq \psi \Rightarrow Q'(\psi, r, p) \rightarrow Q'(\phi, r, p)$. Note that conjunctions and disjunctions of anti-monotonic (resp. monotonic) constraints are anti-monotonic (resp. monotonic).*

Example 3. *SuperString, MinLen, and MaxFr are monotonic constraints. SubString, MaxLen, and MinFr are anti-monotonic ones.*

In most of the application domains, the notion of similarity between two objects o_1 and o_2 informally means a "small difference" between o_1 and o_2. We denote that relation $sim(o_1, o_2)$. Obviously, the property "small difference" should not be propagated too far, i.e., the relation $sim(o_1, o_2)$ should not be transitive.

Monotonicity properties and their associated efficient pruning strategies exploit the generalisation relation property (see Definition 3) which establishes a lattice structure on the search space. We observe that a constraint establishing a relation (e.g., a similarity constraint) is anti-monotonic or monotonic if the associated relation is isomorphic to a generalisation relation. Let us emphasize that a similarity constraint establishing a non-transitive similarity relation between two objects is fundamentally neither monotonic nor anti-monotonic, since a non-transitive similarity relation can not be isomorphic to a generalisation relation.

Many approaches can be used to define similarity between strings. It has been extensively studied in the bioinformatics area (see, e.g., [6] for a survey). [24] has proposed the definition of a reflexive, symmetric but not transitive relation R on Σ s.t. two strings are said to be similar if their corresponding symbols are in relation by R. Another approach is to use one of the classical distances (e.g., Hamming [23] or edit distance [15]) and to assume that two strings are similar if the distance between them is at most some positive integer threshold. More sophisticated approaches are based on words instead of single symbols. This is the typical approach of the famous BLAST program [2]. Another approach to similarity constraint is fault tolerant pattern mining, where tolerance to exceptions (say noise) is achieved by means of soft matching (see, e.g., [22, 21]). Expressing similarities by means of regular expressions can be used as well. Mining sequences which are frequent and satisfy a regular expression has been studied [11, 1]. Also, mining patterns satisfying a minimal frequency constraint in conjunction with a symbol-based similarity has been considered in [5]. These later contributions are typical of data mining algorithmic research: combining an anti-monotonic constraint like the minimal frequency with others which are neither monotonic nor anti-monotonic (e.g., regular expressions, similarity constraints) can be tackled by means of ad-hoc techniques, often based on relaxation strategies. Looking for generic strategies, an important contribution concerns the efficient evaluation of combinations of anti-monotonic and monotonic constraints

[10, 7] and it has been optimized for string mining [8]. To the best of our knowledge, the associated FAVST algorithm is one of the most advanced proposals for constraint-based string mining. Therefore, instead of looking for ad-hoc relaxation strategies, our goal is to study how to design a natural decomposition of a similarity constraint into a combination of monotonic or anti-monotonic constraints. If possible, such a decomposition might enable not only an efficient exploitation of the primitive similarity constraint itself but also of arbitrary combinations of such a similarity constraint with other anti-monotonic and/or monotonic constraints.

3. DEFINITION OF A SIMILARITY CONSTRAINT

Our goal is to formally define a semantics of a similarity constraint $Sim(\phi, x)$ where x is the reference string pattern and ϕ is a candidate string pattern. Let us assume that a measure $s(\phi, x)$ is associated to such a constraint so that $Sim(\phi, x) \equiv s(\phi, x) \leq t$ (resp. $s(\phi, x) \geq t$), where $s(\phi, x)$ is a distance measure (resp. a similarity measure) and t is an integer threshold. Typical measures are the edit and Hamming distances, the episode distance, or the longest common subsequence distance. Generally, distance functions are dependent of the operation costs (e.g., insertion, deletion, substitution, transposition) needed to convert one string into another. We consider operations where "errors" on one string w.r.t the other are encountered. That notion of error is convenient to interpret in different application domains, e.g., data transmission, text editing, mutational events on biological sequences. Thus, it makes sense to express a similarity constraint in terms of acceptable errors.

Given ϕ and x, the string editing problem consists of transforming ϕ into x by performing series of weighted edit operations on ϕ for an overall minimum cost. An edit operation on ϕ can be the deletion of a symbol, the substitution of a symbol with another one, or the insertion of a symbol.

EXAMPLE 4. *Let $\Sigma = \{a, c, g, t\}$ and assume a function $s(\phi, x)$ based on an edit distance (i.e., the minimal number of insertions, deletions and substitutions to get identical strings). Let $x = aactcgc$ and $t = 2$, then $Sim(\phi, x)$ is true with, e.g., $\phi_1 = aactc$, $\phi_2 = actcg$, $\phi_3 = taactcgcc$ and $false$ with, e.g., $\phi_4 = actc$, $\phi_5 = ct$, $\phi_6 = ataactcgcc$. Note that $Sim(\phi, x)$ is neither anti-monotonic (ϕ_4 is more general than ϕ_1) nor monotonic (ϕ_6 is more specific than ϕ_1).*

DEFINITION 6 (LONGEST COMMON SUBSEQUENCE). *Let x be a string over a finite alphabet Σ. A subsequence of x is any string w that can be obtained from x by deleting zero or more (not necessarily consecutive) symbols. More formally, w is a subsequence of x if there exists integers $i_1 < i_2 < \ldots < i_n$ s.t. $w_1 = x_{i_1}, w_2 = x_{i_2}, \ldots, w_n = x_{i_n}$. w is a Longest Common Subsequence (LCS) of x and ϕ if it is a subsequence of x, a subsequence of ϕ, and its length is maximal. We denote that $|w| = lcs(\phi, x)$ and, in general, w is not unique.*

A substitution can be always achieved by one deletion and one insertion. If insertions and deletions have unit costs, and if the cost of substitution is higher that 2, then an optimal sequence of edit operations will always avoid substitutions and produce x from ϕ only by means of insertions and deletions. Then, the pairs of matching symbols in an optimal editing script constitute a LCS of ϕ and x. Note that with such an edit distance e, $lcs(\phi, x)$, $|\phi|$ and $|x|$ are such that $e = |\phi| + |x| - 2 \times lcs(\phi, x)$. A study concerning string editing and longest common subsequences can be found in, e.g., [3].

It makes sense to evaluate the similarity between ϕ and x in terms of their LCS [18]. One possible approach is to consider the largest number of symbols of one sequence that can be matched with those of a second sequence enabling any interruption in both sequences, i.e., $lcs(\phi, x)$.

LEMMA 1. *Assume two strings $x, \phi \in \mathcal{L}$, $\phi' \sqsubseteq \phi$, w one LCS of ϕ and x, and w' one LCS of ϕ' and x. We have $|w| = lcs(\phi, x) \geq lcs(\phi', x) = |w'|$.*

DEFINITION 7 (MINIMUM LCS LENGTH CONSTRAINT). *Let x be the reference pattern, ϕ be a candidate pattern from $\mathcal{L}$, and l be a threshold value. The minimum LCS length constraint is defined as $MinLCS(\phi, x, l) \equiv lcs(\phi, x) \geq l$.*

PROPOSITION 1. *$MinLCS(\phi, x, l)$ is a monotonic constraint.*

EXAMPLE 5. *Assume $x = tctggga$. Patterns $\phi_1 = gcggga$ and $\phi_2 = ctggaga$ satisfy $MinLCS(\phi, x, 5)$: $lcs(\phi_1, x) = |cggga| = 5$ and $lcs(\phi_2, x) = |ctggga| = 6$. Pattern $\phi_3 = attagtgttttgggg$ also satisfies it: $lcs(\phi_3, x) = |ttggg| = 5$.*

It illustrates that a constraint $MinLCS(\phi, x, l)$ enables to specify a degree of similarity (i.e., a minimum number of matching symbols), and thus to capture patterns which are similar to the reference one. Let us notice however that $MinLCS(\phi, x, l)$ does not restrict the dissimilarity of a candidate. Thus, we would like to add a second constraint that would bound the number of "errors" within a candidate.

DEFINITION 8 (MAX DELETIONS CONSTRAINT). *If x is the reference pattern, ϕ is a candidate pattern from $\mathcal{L}$, and d is a threshold value, we can get a subsequence of ϕ by deleting from it a certain number of symbols (see Definition 6). The number of deletions applied on ϕ to get its LCS with x is $dels(\phi, x) = |\phi| - lcs(\phi, x)$. The Maximum Deletions constraint is defined as $MaxDels(\phi, x, d) \equiv dels(\phi, x) \leq d$.*

PROPOSITION 2. *$MaxDels(\phi, x, d)$ is an anti-monotonic constraint.*

DEFINITION 9 (A SIMILARITY CONSTRAINT). *Our similarity constraint for a pattern ϕ w.r.t. a reference pattern x is defined as $C_{sim}(\phi, x, l, d) \equiv MinLCS(\phi, x, l) \wedge MaxDels(\phi, x, d)$, where d and l are user-defined thresholds whose value can be tuned according to $|x|$.*

EXAMPLE 6. *Continuing Example 5, patterns ϕ_1 and ϕ_2 satisfy $C_{sim}(\phi, x, 5, 1)$. Pattern $\phi_4 = gcgggta$ satisfies C_{sim} $(\phi, x, 5, 2)$ since $lcs(\phi_4, x) = |cggga| = 5$. Pattern ϕ_3 does not satisfy neither $C_{sim}(\phi, x, 5, 1)$ nor $C_{sim}(\phi, x, 5, 2)$.*

REMARK 1. *The length of a pattern ϕ satisfying $C_{sim}(\phi, x, l, d)$ is at least l and at most $|x| + d$. Note that even though the maximal length of a pattern ϕ satisfying $C_{sim}(\phi, x, l, d)$ can be inferred from $|x|$ and d, the fact that ϕ satisfies $MinLCS(\phi, x, l) \wedge MaxLen(\phi, |x| + d)$ does not imply that it satisfies $C_{sim}(\phi, x, l, d)$. Let $x = agcgac$, $\phi_5 = gagataga$, $l = 4$, and $d = 2$. $MinLCS(\phi_5, x, 4) \wedge MaxLen(\phi_5, 8)$ is satisfied but $C_{sim}(\phi_5, x, 4, 2)$ is not satisfied since $lcs(\phi_5, x) = |agga| = 4$ and $dels(\phi_5, x) = 4$.*

REMARK 2. *The constraint $MaxDels(\phi, x, d)$ enables to prune candidate patterns ϕ that have already collected too many "errors" and have no chance to become similar to x. Consider pattern ϕ_3 from Example 5, its sub-string $\phi'_3 =$ attagt and $C_{sim}(\phi, x, 5, 2)$ (see Example 6). The anti-monotonic sub-constraint $MaxDels(\phi'_3, x, 2)$ is not satisfied, since $lcs(\phi'_3, x) = |tta| = |ttg| = 3$ and thus, $dels(\phi'_3, x) = 3$, so any further super-string of ϕ'_3 can be pruned.*

4. EXPERIMENTS

The FAVST algorithm [8] is among the best algorithms for mining strings which satisfy conjunctions of anti-monotonic and monotonic constraints. It uses a Version Space Tree (VST) [10] designed to index a version space of strings. VST is based on a less compact form of a suffix tree, called a suffix trie. A trie is a tree whose each edge is labelled with a symbol from the alphabet Σ. The labels on every edge, emerging from a parent node, must be unique. Each node n in a trie uniquely represents a string $\delta(n)$ composed of the symbols on the path from the root to node n. The root represents the empty string ϵ. A suffix trie is a trie with the following properties:

- For each node n and for each suffix t of $\delta(n)$, there is also a node n' representing t in the trie, i.e., $t = \delta(n')$.
- Each node n has a suffix link suffix(n) $= n'$, where $\delta(n')$ represents the suffix of $\delta(n)$ obtained by dropping its first symbol. As the root represents ϵ, having no suffix, suffix(*root*) is defined to be equal to $\perp$.

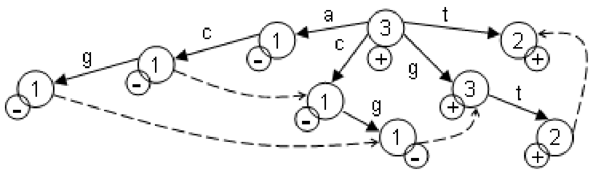

Figure 1: An example of a Version Space Tree

EXAMPLE 7. *Assume $\Sigma = \{a, c, g, t\}$ and a database $r = \{acg, gt, gt\}$. Figure 1 presents a VST for r, with frequency values given inside each node. The dashed arrows indicate the suffix links (the suffix links of root children are omitted). The labels are stuck to the nodes. This VST is labelled for $Q \equiv MinFr(\phi, r, 2)$. Here, the nodes labelled with $\ominus$ are left for illustrative reasons, though the branches containing only $\ominus$ nodes are to be pruned.*

Compared to classical suffix tries, VST have some specific properties:

- it is constructed from a set of strings, instead of a single string;
- as an intermediate computation result, the counts of occurrences of each substring $\delta(n)$ are stored in the corresponding nodes n. In addition to this, each node n is labelled with either $\oplus$, indicating that $\delta(n)$ satisfies a constraint Q, or with $\ominus$, indicating that $\delta(n)$ does not satisfy a constraint Q.
- chains of nodes with only one out-going edge are not coalesced into a single edge label, since the frequency counts and the labels need to be stored for each substring represented by VST.

Note that a memory needed for a VST is not directly dependent on the size of a database but rather on the number of the different string patterns occurring in a database.

To provide an empirical evaluation of our similarity constraint $C_{sim}(\phi, x, l, d)$, we have implemented FAVST in C. We have processed databases of human promoter sequences[3] on a Pentium(R) 4CPU 3.00GHz processor and 1GB main memory.

4.1 Pushing $C_{sim}(\phi, x, l, d)$

Since our similarity constraint is expressed as a conjunction of anti-monotonic and monotonic sub-constraints, it is possible to push it deeply into the extraction phase. The objective of this section is to assess the added value w.r.t. resolving a similarity constraint by post-processing a complete solution set calculated beforehand. We have performed experiments to compare a similarity based pruning and a similarity post-processing. Upstream promoter sequences of the human genome (20647 sequences of length 5000, 101MB) have been processed. Various extractions have been performed using the reference patterns x of different lengths and the more or less selective similarity constraints $C_{sim}(\phi, x, l, d)$ (see first four columns of Table 1). The smaller the parameter l and the larger the parameter d, the less selective is the similarity constraint. We have started with a reference pattern of length 6 (this was identified as a preferred minimal length for the ongoing biological application). We have augmented the reference pattern's length until the extraction became unfeasible.

The space of investigated d and l values is limited by semantical issues. Firstly, a similarity constraint is intended to enforce enough similarity, and large d and small l values do not imply a similarity anymore. Secondly, the parameters l and d are related (see Definition 8). Enforcing $dels(\phi, x) \leq d$ also means that at least $(|x| - d)$ symbols of ϕ constitute a $lcs(\phi, x)$. Thus, d specifies a lower bound for $lcs(\phi, x)$ which can be raised by a larger l in $MinLCS(\phi, x, l)$. Thirdly, note that the meaning of "large" and "small" parameter value is $|x|$ dependent. Hence, we consider that it is illustrative to present the results of experiments by means of a table for l and d values enabling to capture a similarity. The behaviour for other parameter sets can be induced, e.g., large $|x|$ and small l values would result in a huge (unless pattern length is limited by a rather small d) set of solutions, as almost every pattern becomes similar to x, especially when an alphabet is small.

A similarity constraint expressed as a conjunction of a monotonic and an anti-monotonic one can by resolved by the FAVST algorithm. The anti-monotonic $MaxDels(\phi, x, d)$ can be pushed during the VST construction phase, and the monotonic $MinLCS(\phi, x, l)$ is exploited afterwards. Pushing the $MaxDels(\phi, x, d)$ is handled similarly to the *Max-Len*(ϕ, len) constraint in [8]. Minor algorithmical changes comes to the fact that if the current node represents a pattern $\phi = \phi_1\phi_2 \ldots \phi_n$ satisfying a constraint $MaxDels(\phi, x, d)$, and if an extension $\tilde{\phi} = \phi_1\phi_2 \ldots \phi_n\phi_{n+1}$ does not satisfy

[3]Available from http://hgdownload.cse.ucsc.edu/goldenPath /hg17/bigZips/

Table 1: Scalability study

C_{sim}	$\|x\|$	d	l	len	Number of nodes		Time	
					VST for p.p.	VST for C_{sim}	VST for p.p.	VST for C_{sim}
$C_{1_{sim}}$	6	1	4	7	21844	527	23s	1min 40s
$C_{2_{sim}}$	7	1	5	8	87380	642	53s	1min 49s
$C_{3_{sim}}$	7	2	5	9	349524	5280	1min 17s	2min 17s
$C_{4_{sim}}$	10	1	8	11	5386756	5836	1min 59s	2min 50s
$C_{5_{sim}}$	10	2	8	12	18143975	54524	2min 36s	3min 42s
$C_{6_{sim}}$	10	3	8	13	-	406623	-	4min 47s
$C_{7_{sim}}$	15	1	13	16	-	106455	-	5min 07s
$C_{8_{sim}}$	15	2	12	17	-	1024215	-	6min 48s
$C_{9_{sim}}$	20	1	18	21	-	1861901	-	8min 16s
$C_{10_{sim}}$	25	1	23	26	-	1027140	-	12min 55s
$C_{11_{sim}}$	30	1	28	31	-	-	-	-

that constraint, its immediate suffix $\phi_2 \ldots \phi_n \phi_{n+1}$ does not necessarily satisfy $MaxDels(\phi, x, d)$ (though any prefix of $\tilde{\phi}$ does).

To accomplish a post-processing approach we have employed the FAVST to extract all patterns satisfying $MaxLen$ $(\phi, len = |x| + d)$ (since $C_{sim}(\phi, x, l, d)$ implies $|\phi| \leq |x| + d$).

We focus on a VST construction phase for this is the most expensive and crucial for the FAVST. Once a VST is available, it can always be further pruned using any anti-monotonic or monotonic constraint by a simple tree traversal. The scalability study of both approaches is presented on the last four columns of the Table 1. The columns labelled "VST for p.p." correspond to a VST construction when only $MaxLen(\phi, len)$ is used. The patterns stored in this VST are destined for a subsequent similarity post-processing. The columns labelled "VST for C_{sim}" correspond to a VST construction when exploiting $MaxDels(\phi,$ $x, d)$. This VST is intended to be pruned by the monotonic $MinLCS(\phi, x, l)$ to get every pattern ϕ which satisfies C_{sim} (ϕ, x, l, d).

The power of our anti-monotonic similarity sub-constraint pruning is promising. It scales much better on a size of a VST, when $|x|$ and d of $C_{sim}(\phi, x, l, d)$ increase. Moreover, it enables to go far away beyond the limits of a post-processing approach. Starting from $C_{6_{sim}}$, a VST construction while exploiting only $MaxLen(\phi, len)$ pruning is no longer possible on our machine (needed memory exceeds 1GB), whereas exploiting $MaxDels(\phi, x, d)$ reduces a size of VST to 406623 nodes such that it requires approximately 20MB of memory. For $C_{11_{sim}}$, a VST construction turned out to be impossible even with our anti-monotonic similarity sub-constraint pruning. These experiments were performed on a rather large database. Solving $C_{11_{sim}}$ for promoters of chromosomes X and Y only (1004 sequences of length 5000) is feasible (3843999 nodes, 58s) when employing $MaxDels(\phi, x, d)$ pruning. Yet, it failed with similarity post-processing approach since memory required still exceeds 1GB. VST construction using $MaxDels(\phi, x, d)$ pruning takes more time due to a LCS computation. We have employed a classical dynamic programming approach of time complexity $O(nm)$ [13]. There is clearly a room for improvements on such a computation (see, e.g., [3] for a survey). To summarize, the time needed for pushing $MaxDels(\phi, x, d)$ is acceptable on even large databases since it enables extractions that would have been impossible otherwise.

4.2 Selectivity of $C_{sim}(\phi, x, l, d)$ and impact of $MinFr(\phi, r, f)$ pruning

Pushing the $C_{sim}(\phi, x, l, d)$ efficiently prunes the search space, and we study further its selectivity. $C_{sim}(\phi, x, l, d)$ intrinsically specifies the lower and upper bounds for length of patterns belonging to a solution. Thus, it makes sense to investigate the selectivity of $C_{sim}(\phi, x, l, d)$ by comparing it with the selectivity of $C_{len} = MinLen(\phi, l) \wedge MaxLen(\phi, |x|$ $+ d)$. The results of corresponding experiments are presented in three first columns of Table 2. The 1st column gives a similarity constraint (see the first four columns of Table 1). The 2nd column gives the number of patterns satisfying $C_{sim}(\phi, x, l, d)$. The 3rd column gives the number of patterns satisfying a corresponding C_{len}. Clearly, $C_{sim}(\phi, x, l, d)$ is quite selective, and much more powerful than just length's limitations it induces. Observe that the selectivity of $C_{sim}(\phi, x, l, d)$ is not linearly related to its pruning capacity, i.e., a larger solution set can be stored in a smaller VST (see the 7th column of Table 1 and the 2nd column of Table 2 for $C_{3_{sim}}$ and $C_{4_{sim}}$ or $C_{5_{sim}}$ and $C_{7_{sim}}$, knowing that $C_{7_{sim}}$ gave 598 patterns). Large $|x|$ and small d can require many nodes to store a quite restricted solution set (think about combinatorial issues, number and length of branches in a VST).

It is interesting to consider also a frequency based pruning which is known to be quite efficient. Altogether with $MaxLen(\phi, |x| + d)$ it could render the similarity post-processing approach feasible. We have performed experiments to evaluate $MinFr(\phi, r, f)$ impact when extracting patterns for a post-processing approach and having combined with $C_{sim}(\phi, x, l, d)$. The 4th column of Table 2 gives a percent and an absolute value of a minimum frequency threshold. The 5th (resp. 6th) column gives the number of patterns satisfying $C_{sim}(\phi, x, l, d)$ (resp. $C_{len} = MinLen(\phi, l) \wedge$ $MaxLen(\phi, |x| + d)$) extracted with $MinFr(\phi, r, f)$ pruning, and also its percentage of the corresponding solution without $MinFr(\phi, r, f)$ pruning. The 7th column gives the number of nodes in a VST pruned by $MaxLen(\phi, len)$ and $MinFr(\phi, r, f)$ (thus, intended for a similarity post-processing) and the percentage it constitutes of a corresponding VST without $MinFr(\phi, r, f)$ pruning.

$MinFr(\phi, r, f)$ prunes quite efficiently (7th and 6th columns of Table 2 and 6th column of Table 1), especially when $|x|$ becomes large. Thus, pushing a $MinFr(\phi, r, f)$ to an extraction for post-processing would enable it to go further, as it is

Table 2: Selectivity of C_{sim} and impact of $MinFr(\phi, r, f)$ pruning

C_{sim}	Nb of patterns C_{sim}	Nb of patterns C_{len}	fr	Nb of patterns C_{sim} & $MinFr$	Nb of patterns C_{len} & $MinFr$	Nb of nodes in VST for p.p with $MinFr$
$C_{3_{sim}}$	3196	349184	0.05% = 10	3196 = 100%	348259 = 99.7%	348599 = 99.7%
			1% = 206	3042 = 95.2%	235575 = 67.5%	235915 = 67.5%
			5% = 1032	2245 = 70.2%	67052 = 19.2%	67392 = 19.3%
$C_{4_{sim}}$	1132	5364912	0.05% = 10	1124 = 99.2%	3688030 = 68.7%	3709874 = 68.9%
			1% = 206	635 = 56.1%	325866 = 6.1%	347695 = 6.5%
			5% = 1032	44 = 3.9%	53086 = 1%	73090 = 1.4%
$C_{5_{sim}}$	15965	18122131	0.05% = 10	13765 = 86%	6579752 = 36.3%	6601596 = 36.4%
			1% = 206	1133 = 7%	337960 = 1.9%	359789 = 2%
			5% = 1032	74 = 0.5%	55316 = 0.3%	75320 = 0.4%

exactly with large $|x|$ where a post-processing approach encountered its limits. But note that efficient $MinFr(\phi, r, f)$ pruning also means removing a great number of patterns from a solution set (see 2nd and 5th columns). This is the intended behaviour in many data mining applications. Our goal here is however to capture fault-tolerant patterns in the data w.r.t. a reference pattern and, clearly, even infrequent occurrences can be interesting. In other words, pushing $MinFr(\phi, r, f)$ might enable to perform a similarity post-processing, but the price to pay can be the loss of many relevant patterns.

Finally, observe the 7th column of Table 1 and the 5th and 7th columns of Table 2. Let us emphasize that even having enabled large frequency thresholds (e.g., 5%) that prune lots of patterns similar to a reference one (e.g., 99.5% for $C_{5_{sim}}$), the number of nodes in a VST for post-processing is still greater than in a VST we get without $MinFr(\phi, r, f)$, but with $MaxDels(\phi, x, d)$ pruning through a deep push of a $C_{sim}(\phi, x, l, d)$.

4.3 Empirical validation

We have defined a similarity constraint as a conjunction of two sub-constraints that enables efficient pruning, and, what is equally important, can be arbitrary combined with other (anti-)monotonic constraints and solved by a generic solver (e.g., FAVST). Such a definition is, however, only valuable if it captures a useful and intuitive similarity measure. To study this empirically, we have stated that after having perturbed data by some noise, a pertinent similarity constraint should enable to find the perturbed regularities that held in the data initially.

For experiments we have used the *chrXchrY* promoter sequences of chromosomes X and Y (1004 sequences of 1000 nucleotides). Introducing z% of noise means that each symbol undergoes an error with a probability $z/100$. In case of an error event, we assume that a deletion of a symbol, its substitution by a different one, or an insertion of a supplementary symbol are equally possible. Also, in a case of an insertion or a substitution, any other symbol has an equal probability to be chosen.

We have considered a pattern of length 10 that is present on 25 sequences of *chrXchrY*, once on each of them. That pattern is used as a reference pattern x for the $C_{sim}(\phi, x, l, d)$. Then, we introduced 5% noise on the data to get *noisedchr-XchrY*. In *noisedchrXchrY*, pattern x occurs on 15 sequences where it occurs in *chrXchrY*, and on one sequence which does not contain it in *chrXchrY*. We consider that x occurs if it occurs on the same sequence, but we enabled a relatively small shift in the position, given insertions and deletions. Among patterns satisfying $C_{sim}(\phi, x, 9, 1)$ in *noisedchrXchrY*, there are all 25 (possibly shifted) occurrences of x where it was present in *chrXchrY*, and 57 other patterns. It is encouraging that $C_{sim}(\phi, x, l, d)$ enables to recover all patterns. Yet, the number of false positives is quite large. Notice however that the problem of discriminating between patterns that are perturbed occurrences of a reference pattern, patterns that are perturbed or not perturbed occurrences of those that were similar to a reference pattern in the original data, and patterns that have become similar to the reference pattern due to a noise, is different from our current goal, i.e., designing a constraint which captures a similarity.

An important application of $C_{sim}(\phi, x, l, d)$ is the discovery of putative transcription factor binding sites in promoters. For instance, it is interesting to know the binding sites that are frequent in the promoter sequences r_1 of one set of genes and unfrequent in the promoter sequences r_2 of another set of genes[4]. The solution to the inductive query $Q \equiv MinFr(\mu, r_1, f_1) \wedge MaxFr(\mu, r_2, f_2)$ contains a priori interesting patterns μ, i.e. putative binding sites, for further investigation. There is a sequence variability among the binding sites of a given transcription factor F, i.e., some errors are tolerated. To know whether binding sites for F are really over-represented in r_1 and under-represented in r_2, we need to find patterns "similar" to μ (i.e., those that are still recognized by F). In a case where μ is an unknown (or not well annotated) binding site, we do not know which errors are tolerated. Given widely accepted biological hypothesis that sequence elements having some functional role are conserved, it makes sense to employ the $C_{sim}(\phi, \mu, l, d)$. By $MinLCS(\phi, \mu, l)$ sub-constraint one specifies a number of symbols that must be matched allowing all possible interruptions in either of the strings. $MinLCS(\phi, \mu, l)$ limits the number of deletions and substitutions and $MaxDels(\phi, \mu, d)$ limits the number of insertions performed on the candidate.

5. CONCLUSION

Constraint-based mining techniques on sequence databases is an important step towards inductive databases for many application domains (molecular biology, WWW usage mining, data stream mining). The idea is that expert data-

[4]This can be used to understand regulation differences in front of some diseases or stress.

owners can express declarative queries (combinations of constraints) on sequential patterns holding in their data: solvers are then used to compute, when feasible, the whole solution set. While many ad-hoc approaches have been proposed for exploiting similarity constraints, the fundamental lack of monotonicity property of such constraints prevents from the design of generic but also efficient solvers which could process arbitrary combinations of constraints involving similarity ones. Based on the generic solver FAVST which computes complete collections of strings satisfying conjunctions of monotonic and/or anti-monotonic constraints, we have proposed a definition of a similarity constraint w.r.t a given reference pattern as a conjunction of two constraints which satisfy the desired monotonicity properties. We provided a preliminary experimental validation which confirms the added value of this approach on a biological database. A real-world application for understanding the action of insulin on human gene regulation by means of inductive queries on regulated gene promotor sequences is ongoing.

Acknowledgements. The authors thank Sophie Rome for stimulating discussions on biological applications of our proposal. This research is partly funded by ACI CNRS MD 46 Bingo and by EU contract IST-FET IQ FP6-516169 (FET arm of the IST programme).

6. REFERENCES

[1] H. Albert-Lorincz and J.-F. Boulicaut. Mining frequent sequential patterns under regular expressions: a highly adaptive strategy for pushing constraints. In *Proceedings SIAM DM 2003*, pages 316–320, 2003.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[3] A. Apostolico. String editing and longest common subsequences. In *Handbook of Formal Languages*, volume 2 Linear Modeling: Background and Application, pages 361–398. Springer-Verlag, 1997.

[4] J.-F. Boulicaut. Inductive databases and multiple uses of frequent itemsets: the cInQ approach. In *Database Technologies for Data Mining - Discovering Knowledge with Inductive Queries*, pages 1–23. Springer-Verlag, 2004.

[5] M. Capelle, J.-F. Boulicaut, and C. Masson. Mining frequent sequential patterns under a similarity constraint. In *Proceedings IDEAL'02*, pages 1–6. Springer-Verlag, 2002.

[6] M. Crochemore and M.-F. Sagot. Motifs in sequences: Localization and extraction. In *Handbook of Computational Chemistry*, pages 47–97. Marcel Dekker, New York, 2004.

[7] S. Dan Lee and L. De Raedt. An algebra for inductive query evaluation. In *Proceedings IEEE ICDM'03*, pages 147–154, 2003.

[8] S. Dan Lee and L. De Raedt. An efficient algorithm for mining string databases under constraints. In *Proceedings KDID'04*, pages 108–129. Springer-Verlag, 2004.

[9] L. De Raedt. A perspective on inductive databases. *SIGKDD Explorations*, 4(2):69–77, 2003.

[10] L. De Raedt, M. Jaeger, S. Dan Lee, and H. Mannila. A theory of inductive query answering. In *Proceedings IEEE ICDM'02*, pages 123–130, 2002.

[11] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In *Proceedings VLDB '99*, pages 223–234. Morgan Kaufmann Publishers Inc., 1999.

[12] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings ACM SIGKDD'00*, pages 355–359, 2000.

[13] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *CACM*, 18(6):341–343, 1975.

[14] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *CACM*, 39(11):58–64, 1996.

[15] V. Levenshtein. Binary codes capable of corresting spurious insertions and deletions of ones. *Probl. Inf. Transmission*, 1:8–17, 1965.

[16] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[17] F. Masseglia, F. Cathala, and P. Poncelet. The PSP approach for mining sequential patterns. In *Proceedings PKDD'98*, pages 176–184. Springer-Verlag, 1998.

[18] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48(3):443–453, March 1970.

[19] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings IEEE ICDE'01*, pages 215–224, 2001.

[20] M. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *Proceedings ACM CIKM'02*, pages 18–25, 2002.

[21] M.-F. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. In *Proceedings CPM '96*, pages 186–208. Springer-Verlag, 1996.

[22] M.-F. Sagot, A. Viari, and H. Soldano. A distance-based block searching algorithm. In *Proceedings ISMB'95*, pages 322–331, 1995.

[23] D. Sankoff and J. Kruskal. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, Mass. Addison-Wesley, 1983.

[24] H. Soldano, A. Viari, and M. Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognition Letters*, 16(3):233–246, 1995.

[25] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings EDBT '96*, pages 3–17. Springer-Verlag, 1996.

[26] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.