A Decomposition Approach for Discovering Discriminative Motifs in a Sequence Database

David Lesaint, Deepak Mehta, Barry O'Sullivan, Vincent Vigneron

A Decomposition Approach for Discovering Discriminative Motifs in a Sequence Database

David Lesaint*, Deepak Mehta†, Barry O'Sullivan† and Vincent Vigneron*
*LERIA Université d'Angers
2, bd Lavoisier
49045 Angers cedex 01 France
Email: {lesaint,vigneron}@info.univ-angers.fr
†Insight Centre for Data Analytics
University College Cork, Ireland
Email: {deepak.mehta,barry.osullivan}@insight-centre.org

***Abstract*—Considerable effort has been invested over the years in ad-hoc algorithms for itemset and pattern mining. Constraint programming has recently been proposed as a means to tackle itemset mining tasks within a general modelling framework. We follow this approach to address the discovery of discriminative n-ary motifs in databases of labeled sequences. We define a n-ary motif as a mapping of n patterns to n class-wide embeddings and we restrict the interpretation of constraints on a motif to the sequences embedding all patterns. We formulate core constraints that minimize redundancy between motifs and introduce a general constraint optimization framework to compute common and exclusive motifs. We cast the discovery of closed and replication-free motifs in this framework for which we propose a two-stage approach based on constraint programming. Experimental results on datasets of protein sequences demonstrate the efficiency of the approach.**

I. Introduction

Pattern mining is of critical importance in domains ranging from Bioinformatics to Cybersecurity. Patterns help characterize data sets and many algorithms have been proposed for various mining tasks (e.g., concept learning, clustering) over different datatypes (e.g., item-sets or sequences, labeled or unlabeled data). These algorithms are typically ad-hoc and dedicated to a particular mining task or application domain which makes them hard to adapt when requirements evolve. Knowledge representation frameworks based on constraint programming (CP) have been proposed to address this issue [1]–[3]. CP provides constraint languages to express requirements in a modular way together with generic constraint satisfaction algorithms to solve the resulting problems efficiently. Using CP for data mining aims to facilitate reuse and evolvability without sacrificing computational performances.

This paper introduces a constraint optimization approach for mining discriminative motifs in databases of labelled sequences. Without loss of generality, we shall only consider bipartitioned databases consisting of a "positive class" and a "negative class". In this context, we are interested in computing an n-ary motif that is common and exclusive to the positive class. A n-ary motif consists of n patterns and associates to each pattern a class-wide embedding called a c-block. Every pattern is a sequence of solid characters possibly interspersed with free characters called hash. We require that every c-block preserves the positioning of hashs in its pattern across the positive class, that is, a c-block is a common subsequence subject to gap consistency constraints across positive sequences. This constraint is paramount to tackling alignment problems such as protein sequence alignment where hashs are interpreted as evolutionary point mutations.

On this basis, we adapt closure and coverage constraints on motifs which have been cast in CP frameworks for different data mining problems (singleton [1] and n-ary motifs [2] made of item-sets, singleton motifs of sequential patterns over a single sequence [4], [5] or multiple sequences [6]). We also introduce new constraints, namely, non-inclusion and non-replication between the c-blocks of a motif. These constraints help to reduce redundancy, speed up computation or meet domain-specific requirements. We then propose a framework for motif discovery problems (MDP) that gives full flexibility in the choice and distribution of constraints to satisfy on the positive class or to falsify on the negative class. The framework handles exclusivity as an optimization criterion and uses lexicographic ordering to accommodate additional criteria (motif cardinality, slack, etc).

We cast one particular problem in this framework, namely, the discovery of exclusive, closed and replication-free motifs. The Replication-free MDP (RMDP) enforces motif closure and non-replication over the positive class and minimum coverage of the negative class. This problem is novel from computational and bioinformatics point of views, the closest attempts being those related to the computation of non-sequential motifs for item-sets, singleton and non-discriminative sequential motifs or multiple protein sequence alignments. We introduce a two-stage approach to RMDP when the objective is to maximize exclusivity before minimizing slack and cardinality. This approach assembles closed c-blocks from minimal blocks before determining solution motifs using constraint optimization. In this last stage, minimizing cardinality reduces to a minimum set covering problem. We report experiments on two datasets of protein sequences [7]–[9] that demonstrate the efficiency of the approach.

The paper is organized as follows. Section II presents background definitions, the MDP framework and the RMDP problem. Section III introduces the two-stage approach and constraint optimization method used to solve RMDP. Section IV reports and discusses experimental results. Section V con-

cludes.

II. Definitions

Let Σ be an alphabet which is a finite set of symbols called solid characters. A sequence over Σ is a sequence $s_1 \dots s_n \in \Sigma^n$ for some $n \in \mathbb{N}^*$. The length of a sequence s is denoted by $|s|$. A bipartitioned database D over Σ is the union of two disjoint sets D^+ and D^- of sequences over Σ ($D \subseteq \Sigma^*$ and $D = D^+ \cup D^-$). D^+ (respectively, D^-) is called the positive (resp. negative) class and its elements the positive (resp., negative) sequences of D. A hash is a special character denoted $\#$ that is not in Σ ($\# \notin \Sigma$). A pattern is a sequence over $\Sigma \cup \{\#\}$ of length greater or equal to 2 that starts and ends with a solid character. The following definitions refer to a database D over an alphabet Σ. We shall denote by $[n]$ the range $\{i \mid 1 \leqslant i \leqslant n\}$ for $n \in \mathbb{N}$, and by $[t]$ the range $[|t|]$ for a sequence, pattern or set t.

Definition 1 (Pattern): A pattern is a sequence $p \in \Sigma.(\Sigma \cup \{\#\})^*.\Sigma$. p_i denotes the i-th character of p for all $i \in [p]$, $\pi(p) = \{k \mid p_k \neq \#\}$ denotes the set of positions of solid characters in p, and $\pi(p)(i)$ denotes the position of the i-th solid character in p for all $i \in [\pi(p)]$.

A pattern may be embedded more than once in a sequence. For instance, sequence ADACDCEC embeds pattern A##C#C at locations 1 and 3. A pattern may also be embedded in different sequences of a database, the set of which is called the cover of the pattern. We say that a pattern is positive if every positive sequence embeds it.

Definition 2 (Pattern embedding and cover): A pattern p is embedded in a sequence s at location $l \in [s]$, denoted $p \subseteq_l s$, if $p_k = s_{l+k-1}$ or $p_k = \#$ for all $k \in [p]$. The cover of p, denoted $\varphi(p)$, is the set $\varphi(p) = \{s \in D \mid \exists l \in [s],\ p \subseteq_l s\}$.

A block corresponds to the embedding of a pattern in a sequence. A c-block corresponds to the embedding of a positive pattern over its whole cover, that is, it is the choice of one block per sequence in the pattern cover. A motif is a non-empty set of c-blocks and the intersection of their (pattern) covers defines the cover of the motif.

Definition 3 (Block, c-block and motif): A block is a triple (p, l, s) such that s is a sequence of D and p is a pattern embedded in s at location l. A c-block is a pair (p, δ) such that p is a positive pattern and δ is a function embedding p over its cover: $D^+ \subseteq \varphi(p)$, $\delta \in (\varphi(p) \to \mathbb{N}^*)$ and $(p, \delta(s), s)$ is a block for all $s \in \varphi(p)$. The cover of a c-block $c = (p, \delta)$ is $\varphi(c) = \varphi(p)$. A motif Π is a non-empty set of c-blocks and the cover of Π is $\varphi(\Pi) = \bigcap_{c \in \Pi} \varphi(c)$.

Figure 1 shows patterns and blocks for a database of three sequences. For instance, AA is a positive pattern. $(\texttt{AA}, 6, s_1)$, $(\texttt{AA}, 8, s_2)$ and $(\texttt{AA}, 1, s_3)$ are blocks and $(\texttt{AA}, \{6, 8, 1\})$ a c-block for this pattern.[1] $(\texttt{AAC\#E}, \{1, 1\})$ is another c-block but $(\texttt{YE}, \{4\})$ is not since pattern YE is not positive. A possible motif is $\{(\texttt{A\#C}, \{1, 1, 1\}), (\texttt{AAC}, \{1, 1, 1\}), (\texttt{AAC\#E}, \{1, 1\}), (\texttt{AA}, \{6, 8, 1\})\}$ and its cover is D^+.

[1] We represent the embedding function of a c-block by ordering its image consistently with the ordering $\{s_1, s_2, s_3\}$.

	s_1	s_2	s_3
Sequences	AACYEAA	AACZEZZAA	AAC
Blocks	A#C	A#C	A#C
	AAC	AAC	AAC
	AAC#E	AAC#E	
	AA	AA	AA

Fig. 1. Examples of blocks for a database $D^+ = \{s_1, s_2\}$ and $D^- = \{s_3\}$.

We are interested in discovering motifs that are common and exclusive to the positive class, that is, motifs satisfying the prescribed constraints on the positive class (commonality) and falsifying some of the constraints when extended to negative sequences (exclusivity). A motif constraint may then be interpreted over different sets of sequences based on the computation task, i.e., over the positive class for checking commonality or over supersets of the positive class for checking exclusivity.

Definition 4 (Motif constraint): Let C be the set of c-blocks associated to database D. A motif constraint is a mapping $r \in (2^D \times 2^C \to \{false, true\})$. A motif $\Pi \subseteq C$ satisfies r over $X \subseteq D$ if $r(X, \Pi) = true$.

We now introduce motif constraints, namely, non-replication, non-inclusion, closure and coverage whose aim is to eliminate redundant motifs. Motif constraints quantify if the c-blocks of a motif satisfy a given property. Such properties differ in scope, e.g., closure and coverage are checked for each c-block whereas non-replication and non-inclusion are checked for each pair of c-blocks. Properties also differ based on whether they apply to patterns or blocks of c-blocks.

Replication is a pattern subsumption constraint. Informally, a pattern p replicates a pattern p' if p is obtained from p' by replacing hashs with solid characters and/or by adding solid characters and possibly hashs left and/or right. For instance, pattern AAC#E replicates AAC which itself replicates A#C. We say that replication holds between two c-blocks if replication holds between their patterns.

Definition 5 (c-block replication): Let p and p' be two patterns. p replicates p', denoted $p \succeq p'$, if $\exists l \in \mathbb{N}$, $\forall i \in [p']$, $p'_i \neq \# \Rightarrow p'_i = p_{i+l}$. Let $c = (p, \delta)$ and $c' = (p', \delta')$ be two c-blocks. c replicates c', denoted $c \succeq c'$, if $p \succeq p'$.

Inclusion is a block subsumption constraint. Informally, a block b includes a block b' if b and b' are associated to the same sequence s and every solid character in the pattern of b' is found in the pattern of b with the same embedding in s. Note that replication subsumes inclusion. For instance, block $(\texttt{AAC\#E}, 1, s_1)$ in Fig.1 includes $(\texttt{AAC}, 1, s_1)$ which itself includes $(\texttt{A\#C}, 1, s_1)$. However, block $(\texttt{AA}, 1, s_1)$ does not include $(\texttt{AA}, 6, s_1)$. We say that inclusion holds between two c-blocks over $X \subseteq D$ if they have a common sequence in X over which block inclusion holds.

Definition 6 (c-block inclusion): Let $s \in D$ and let $b = (p, l, s)$ and $b' = (p', l', s)$ be two blocks. b includes b', denoted $b \sqsupseteq b'$, if $\{k+l \mid k \in \pi(p)\} \supseteq \{k+l' \mid k \in \pi(p')\}$. Let $c = (p, \delta)$ and $c' = (p', \delta')$ be two c-blocks. c includes c' over $X \subseteq D$, denoted $c \sqsupseteq_X c'$, if $\exists s \in X \cap \varphi(c) \cap \varphi(c')$, $(p, \delta(s), s) \sqsupseteq (p', \delta'(s), s)$.

Closure is a unary constraint on c-blocks that ensures one cannot replicate the pattern of a c-block while preserving its

blocks. Informally, a c-block is closed over $X \subseteq D$ if there is no other c-block including it in each sequence common to X and its cover. For instance, c-block $(\texttt{AAC}, \{1,1,1\})$ in Fig.1 is not closed over D^+ as $(\texttt{AAC\#E}, \{1,1,1\})$ includes it on each positive sequence but it is closed over D.

Definition 7 (c-block closure): A c-block (p, δ) is closed over $X \subseteq D$ if there is no c-block (p', δ') such that $p' \neq p$ and $(p', \delta'(s), s) \sqsupseteq (p, \delta(s), s)$ for all $s \in X \cap \varphi((p, \delta)) \cap \varphi((p', \delta'))$.

We now define motif constraints relatively to some set $X \subseteq D$. First, a motif covers X if its cover includes X. It is closed over X if its c-blocks are closed over X. It is replication-free over X if there is no replication between its c-blocks. It is inclusion-free over X if there is no inclusion between its c-blocks over X. Formally,[2]

$$coverage_X(\Pi) \Leftrightarrow X \subseteq \varphi(\Pi) \quad (1)$$

$$closure_X(\Pi) \Leftrightarrow \forall c \in \Pi,\ c \text{ is closed over } X \quad (2)$$

$$non-replication_X(\Pi) \Leftrightarrow \forall c, c' \in \Pi,\ c \succeq c' \Rightarrow c = c' \quad (3)$$

$$non-inclusion_X(\Pi) \Leftrightarrow \forall c, c' \in \Pi,\ c \sqsupseteq_X c' \Rightarrow c = c' \quad (4)$$

In Fig.1, motif $\{(\texttt{AAC}, \{1,1,1\}), (\texttt{AAC\#E}, \{1,1\})\}$ covers D^+ but not D and it is neither inclusion-free nor closed over its cover. $\{(\texttt{AAC}, \{1,1,1\}), (\texttt{AA}, \{6,8,1\})\}$ covers D and is inclusion-free but neither replication-free nor closed over D^+. $\{(\texttt{AAC\#E}, \{1,1\}), (\texttt{AA}, \{6,8,1\})\}$ covers D^+ and is both inclusion-free and closed over its cover but it is not replication-free.

Many other constraints may be considered which we discuss briefly. These include pattern constraints (e.g., prohibiting some solid characters, bounding the number of hashs and, more generally, enforcing regular expression constraints [6]), constraints on individual c-blocks (e.g., requiring that the set of solid characters mapping to a hash fit into predefined classes of interchangeability), and constraints between c-blocks (e.g., cardinality constraints on pattern replications or duplications). As opposed to itemsets, pattern motifs also lend themselves to positioning or ordering constraints. Blocks indeed map to position intervals and point or interval algebras may be used to enforce a consistent ordering between the c-blocks of a motif. For instance, one may search for "sequential motifs" by imposing a total ordering on c-blocks by the means of precedence constraints on blocks.

We now present the MDP framework to compute discriminating motifs. A MDP is parameterized by a set of constraints and requires that a solution motif satisfy all constraints over the positive class and exclude the largest number of negative sequences. A motif excludes a negative sequence if it cannot be "extended" to that sequence without falsifying some constraint, i.e., any motif sharing the same family of patterns and blocks over the positive class falsifies some constraint when interpretation extends to the negative sequence. In Fig.1 for instance, motif $\{(\texttt{AAC}, \{1,1,1\}), (\texttt{AA}, \{6,8,1\})\}$ satisfies non-inclusion over the positive class and excludes negative sequence s_3 with this constraint.

[2]We abuse notation by using $r_X(\Pi)$ to mean $r(X, \Pi) = true$ and $\neg r_X(\Pi)$ to mean $r(X, \Pi) = false$ for a motif constraint r.

Definition 8 (Exclusion and inclusion counts): Let Π and Π' be motifs. We denote $\Pi =_+ \Pi'$ if there exists a one-to-one mapping $f : \Pi \to \Pi'$ such that $\forall (p, \delta) \in \Pi,\ f((p, \delta)) = (p', \delta') \Rightarrow\ (p = p' \wedge \delta_{|D^+} = \delta'_{|D^+})$. Let R be a set of motif constraints. The exclusion count of Π for R is $exc_R(\Pi) = |\{s \in D^- \mid \forall \Pi' =_+ \Pi, \exists r \in R, \neg r_{\varphi(\Pi) \cup \{s\}}(\Pi')\}|$ and the inclusion count of Π for R is $inc_R(\Pi) = |D^-| - exc_R(\Pi)$.

Note that some constraints can never be falsified on negative sequences if they hold over the positive class for a given motif. This is the case of constraints that are monotonic for set inclusion over sequence sets and constraints whose semantics does not depend on motif cover. Examples include closure (a motif closed over X is closed over all $Y \supseteq X$) and non-replication (a replication-free motif over X is replication-free over all sets of sequences). For this reason but also to clearly separate commonality and exclusivity checks, constraints that should not be checked for exclusivity are made explicit in the MDP framework. While exclusion maximization is the main objective, the MDP framework can accommodate secondary objectives. This is achieved by the means of a user-defined strict weak ordering.

Definition 9 (MDP): A motif discovery problem is a tuple $P = (D, R^+, R^-, \lesssim)$ such that $D = D^+ \cup D^-$ is a bipartitioned sequence database, R^+ and R^- are (possibly empty) sets of motif constraints, and $\lesssim$ is a strict weak ordering[3] over motifs such that $\Pi \lesssim \Pi' \Rightarrow inc_{R^-}(\Pi) \leqslant inc_{R^-}(\Pi')$. A solution to P is a minimal motif for $\lesssim$ that satisfies all constraints of $R^+ \cup R^-$ over D^+.

We present below two measures, namely motif cardinality and slack, that may be combined with the inclusion count. The slack of a motif is the maximum number of consecutive hashs in its patterns.

Definition 10 (Measures): Let Π be a motif. $card(\Pi) = |\Pi|$ is the cardinality of Π. $slack(\Pi) = \max_{(p,\delta) \in \Pi, i \in [|p|-1]} (\pi(p)(i+1) - \pi(p)(i) - 1)$ is the slack of Π.

We now introduce the Replication-free MDP (RMDP) which enforces motif closure and non-replication over the positive class and minimal coverage of the negative class. That is, a solution to a RMDP is a replication-free motif that is closed over the positive class, that covers the smallest number of negative sequences, and that is minimal for the additional criteria if any.

Definition 11 (RMDP): A RMDP is a MDP $(D, R^+, R^-, \lesssim)$ with $R^+ = \{\texttt{closure}, \texttt{non-replication}\}$ and $R^- = \{\texttt{coverage}\}$.

The next result shows that a careful choice of criteria allows to restrict computation to a subset of c-blocks having "maximal" patterns. Let C denote the set of c-blocks associated to a RMDP. We denote $\equiv$ the binary relation defined over C by $c \equiv c'$ iff $c \succeq c' \wedge c' \succeq c$. Note that $\equiv$ is an equivalence relation since $\succeq$ is a pre-order. We say that a c-block $c \in C$ is maximal for replication or $\succeq$-maximal iff for all $c' \in C$, $c' \succeq c \Rightarrow c' \equiv c$. $\equiv$ partitions the set of $\succeq$-maximal c-blocks into equivalence classes. The choice of a representative

[3]A strict weak ordering is a strict partial ordering for which the incomparability relation is transitive.

element in each class determines a *representative set* of $\succeq$-maximal c-blocks which, under certain conditions on the optimization criteria, necessarily includes a solution motif if the problem is satisfiable. In particular, any representative set of $\succeq$-maximal c-blocks includes a solution if inclusion count minimization is the sole objective.

Theorem 1: Let $P = (D, R^+, R^-, \lesssim)$ be a satisfiable RMDP, C be its set of c-blocks, $C_{\succeq_{max}}$ be its set of $\succeq$-maximal c-blocks, $f : C \to C_{\succeq_{max}}$ be a function verifying $f(c) \succeq c$ and $f(c) \equiv f(c') \Rightarrow f(c) = f(c')$ for all $c, c' \in C$, and $F : 2^C \to 2^{C_{\succeq_{max}}}$ be the function defined by $F(\{c_1, \ldots, c_k\}) = \{f(c_1), \ldots, f(c_k)\}$. If $\lesssim$ is equal to $\leqslant$ or if $F(\Pi) \lesssim \Pi$ for all $\Pi \in 2^C$ then P has a solution in $F(2^C)$.[4]

Proof: Let P be a satisfiable RMDP. Since C is finite and $\succeq$ is a preorder, every c-block $c \in C$ may be mapped to at least one c-block $c' \in C_{\succeq_{max}}$ s.t. $c' \succeq c$. Every $\succeq$-maximal c-block may also be mapped to a particular representative of its equivalence class. Therefore there exists a function $f : C \to C_{\succeq_{max}}$ satisfying $f(c) \succeq c$ and $f(c) \equiv f(c') \Rightarrow f(c) = f(c')$ for all $c, c' \in C$. Let $F : 2^C \to 2^{C_{\succeq_{max}}}$ s.t. $F(\{c_1, \ldots, c_k\}) = \{f(c_1), \ldots, f(c_k)\}$ and let $\Pi \in 2^C$. We first show that (a) $inc_{R^-}(F(\Pi)) \leqslant inc_{R^-}(\Pi)$ and (b) $F(\Pi)$ satisfies all constraints of $R^+ \cup R^-$ over D^+.

(a) Since $R^- = \{\texttt{coverage}\}$, $exc_{R^-}(\Pi) = |\{s \in D^- \mid \forall \Pi' =_+ \Pi, \neg\texttt{coverage}(\varphi(\Pi) \cup \{s\}, \Pi')\}| = |\{s \in D^- \mid \forall \Pi' =_+ \Pi, \varphi(\Pi) \cup \{s\} \nsubseteq \varphi(\Pi')\}|$. By definition of $=_+$, $\varphi(\Pi') = \varphi(\Pi)$ for all $\Pi' =_+ \Pi$ so $exc_{R^-}(\Pi) = |D \backslash \varphi(\Pi)|$. For all $c, c' \in C$, $c' \succeq c$ implies $\varphi(c') \subseteq \varphi(c)$. So $\varphi(f(c)) \subseteq \varphi(c)$ for all $c \in C$. Since $\varphi(F(\Pi)) = \bigcap_{c \in \Pi} \varphi(f(c))$, $\varphi(F(\Pi)) \subseteq \varphi(\Pi)$. Therefore $exc_{R^-}(\Pi) \leqslant exc_{R^-}(F(\Pi))$ ie. $inc_{R^-}(F(\Pi)) \leqslant inc_{R^-}(\Pi)$. (b) By definition, every $\succeq$-maximal c-block is necessarily closed over D^+ and $F(\Pi) \subseteq 2^{C_{\succeq_{max}}}$ so $F(\Pi)$ is closed over D^+. Let $c, c' \in F(\Pi)$ s.t. $c \succeq c'$. $c \equiv c'$ since c' is $\succeq$-maximal. By definition of F, there exists $a, a' \in C$ such that $c = f(a)$ and $c' = f(a')$. So $f(a) \equiv f(a')$ which implies $f(a) = f(a')$ by definition of f so $c = c'$. Therefore $F(\Pi)$ is replication-free. Every c-block covers D^+ and so does every motif so $F(\Pi)$ covers D^+. Therefore $F(\Pi)$ satisfies all constraints of $R^+ \cup R^-$ over D^+.

Let Π be a solution to P. If $\lesssim$ is equal to $\leqslant$ then (a) implies $F(\Pi)$ is minimal for $\lesssim$ and (b) implies $F(\Pi)$ satisfies all constraints of $R^+ \cup R^-$ over D^+ so $F(\Pi)$ is also a solution to P. If $F(\Pi') \lesssim \Pi'$ for all $\Pi' \in 2^C$ then $F(\Pi)$ is minimal for $\lesssim$ which together with (2) implies that $F(\Pi)$ is also a solution to P. ∎

By definition, all motifs of $F(2^C)$ are subsets of a representative set of $\lesssim$-maximal c-blocks. This motivates a two-stage approach when the conditions of the theorem hold on $\lesssim$. Precisely, one may compute a representative set of $\lesssim$-maximal c-blocks first before extracting an optimal motif through combinatorial optimization. Consider for instance motif cardinality and slack which respectively decreases and increases through function F (i.e., $|F(\Pi)| \leqslant |\Pi|$ and $slack(F(\Pi)) \geqslant slack(\Pi)$). Solution motifs may be found with this approach if the problem is satisfiable and the additional objectives are to minimize cardinality and/or maximize slack.

[4] $F(2^C)$ denotes the image of F.

This is no longer true if motif cardinality has to be maximized or slack minimized. The next section addresses such a case. Specifically, the objective is to minimize both cardinality and slack.

III. A Decomposition Approach to Solving RMDP

We describe an approach to solve RMDP where we consider an additional constraint where slack is upper-bounded by some predefined parameter. The lexicographic objective is to first maximize exclusion then minimize slack and then minimize cardinality. We also investigate a variant of the problem where the minimal cardinality and slack criteria are swapped in the lexicographic objective function.

Solving RMDP involves finding a motif consisting of one or more c-blocks that are closed, replication-free and exclusively covering D^+. From a computational viewpoint, the main bottleneck is that a solution motif may contain an exponential number of c-blocks and this number is not known a priori. This contrasts with itemset or single pattern computation problems as in that case the size of the solution is bounded by the maximum number of items in any transaction. However, in our case as the number of patterns could be exponential modelling the whole problem as a giant monolithic Constraint Optimization Problem (COP) is space-wise challenging. Therefore, we propose a two-step approach where first we compute all c-blocks over D^+ that are closed and replication-free and then for each sequence in D^- we determine c-blocks that do not cover it and then compute an optimal motif for D^+.

As memory requirements may remain prohibitive, we implement a lazy approach where the value of slack is incremented only if required. More precisely, we vary the value of slack starting from 0 to a maximum allowed value, and compute the c-blocks and an optimal motif at each step. We exit the loop as soon as all negative sequences are excluded. We describe below the method for computing an optimal motif for a given value of slack.

A. Computing Closed C-blocks

The pseudo-code for computing maximal c-blocks is shown in Algorithm 1. We compute closed c-blocks in three successive steps:

1) We start by computing all "positive blocks" of length 2, i.e., blocks whose patterns are positive and only have 2 solid characters We further enforce the constraint that the cover of the pattern of each block is D^+. This is done by first selecting the sequence having the minimum length and then verifying for each valid block of length 2 whether its pattern is occurring in all the sequences or not. Once done, we know all the minimal length blocks and their locations in the smallest sequence of the positive class (Lines 1–2). The pseudo-code for this task is depicted in Algorithm 2.
2) We then compute all inclusion-maximal positive blocks for the shortest sequence, i.e., positive blocks that are not included in any other positive block. The idea is to build a lattice of blocks bottom-up based on the inclusion relation. The procedure is iterative

and starts with the minimal length positive blocks. Each iteration merges every possible pair of blocks verifying that the resulting block is positive and within the allowed slack before eliminating any block that is not inclusion-maximal in the set computed so far. We remark that the number of non-inclusive blocks defines a bound on number of non-inclusive closed c-blocks. The pseudo-code for this task is depicted in Algorithm 3.

3) We finally compute a set of inclusion-maximal closed c-blocks (Line 4). The pseudo-code for this task is depicted in Algorithm 4.

Algorithm 1
`ComputeClosedCBlocks(`D^+`,maxSlack)`

Require: D^+ is the positive class
Require: `maxSlack` is the maximum allowed slack
1: $s \leftarrow \arg\min_{t \in D^+} |t|$
2: `minbs` $\leftarrow$ `FindAllMinimalBlocks(`D^+`, s, maxSlack)`
3: `maxbs` $\leftarrow$ `FindAllMaximalBlocks(`D^+`, s, maxSlack, minbs)`
4: `maxcbs` $\leftarrow$ `FindMaximalCBlocks(`D^+`, s, maxbs)`
5: **return** `maxcbs`

Algorithm 2 computes all minimal blocks of the sequence s of class D^+, denoted by `minbs`, such that the cover of each pattern of each block is D^+. For each allowed value of slack (Line 2) and for each block of s having pattern p and length 2 (Line 3), the algorithms checks if the cover of p is D^+ (Line 4). If it finds such a block then the set `minbs` is updated (Line 5). Let $n = |D^+|$. Let $\underline{d}$ and $\overline{d}$ be the minimum and maximum lengths of the sequences of D^+. The maximum value of slack is bounded by $\underline{d} - 2$ as sequence s is smallest in class D^+. Also, the maximum number of minimal blocks for any given slack value is bounded by $\underline{d} - 1$. The task of verifying whether the cover of the block is occurring is D^+ is bounded by $n\,\overline{d}$. Therefore, the worst-case complexity of `FindAllMinimalBlocks` is $\mathcal{O}(n\,\underline{d}^2\,\overline{d})$.

Algorithm 2
`FindAllMinimalBlocks(`D^+`,s,maxSlack)`

Require: D^+ is the positive class
Require: s is a positive sequence
Require: `maxSlack` is the maximum allowed slack
1: `minbs` $\leftarrow \emptyset$
2: **for all** $\sigma \in$ [`maxSlack`] **do**
3: **for all** $b = (p, l, s)$ s.t. $|\pi(p)| = 2 \wedge \sigma = slack(p)$ **do**
4: **if** $\varphi(p) = D^+$ **then**
5: `minbs` $\leftarrow$ `minbs` $\cup \{b\}$
6: **return** `minbs`

Algorithm 3 computes all maximal blocks of sequence $s \in D^+$, denoted by `maxbs`, such that the cover of the pattern of each block is D^+ itself. The general idea is to build a lattice of blocks based on inclusion relation. Initially, `maxbs` is initialized to the given set of minimal blocks (Line 1). In each iteration (Lines 2–17), the blocks of the current top layer, denoted by `oldbs`, are considered to construct new blocks for the next layer, denoted by `newbs`. Initially, `oldbs` is set to `maxbs` and `newbs` is the empty set (Lines 3–4). Each pair of blocks of the current layer (Line 5) are composed to generate a new block b with pattern p (Line 6–7) and slack σ (Line 8). If the value of the slack is less than the maximum allowed slack and if the block b is new (Line 9) then the algorithm checks whether the cover of the pattern p is D^+ (Line 10). If it is true then the new block b is added to the set `newbs` (Line 11). If at least one new block is found (Line 12) then the algorithm checks for the maximality of the blocks in `oldbs`. Therefore, the algorithm checks the inclusion relation between the blocks of `oldbs` and those of `newbs` (Lines 13–16). First, `maxbs` is set to `newbs` (Line 13) and a block $b \in$ `oldbs` is added to `maxbs` if it is not included in any block of `oldbs` (Line 15–16). The algorithm terminates when it fails to find any new maximal c-block (Line 17).

Algorithm 3
`FindAllMaximalBlocks(`D^+`,s,maxSlack,minbs)`

Require: D^+ is the positive class; s is a positive sequence; `maxSlack` is the maximum allowed slack; `minbs` is the set of minimal blocks for s with positive cover.
1: `maxbs` $\leftarrow$ `minbs`
2: **repeat**
3: `oldbs` $\leftarrow$ `maxbs`
4: `newbs` $\leftarrow \emptyset$
5: **for all** $\{b' \leftarrow (p', l', s), b'' \leftarrow (p'', l'', s)\} \subseteq$ `oldbs` **do**
6: {For a block $b = (p, l, s)$ we denote $\mu(b) = \{l + k | k \in \pi(p)\}$}
7: $b \leftarrow (p, \min(l', l''), s)$ such that $\mu(b) = \mu(b') \cup \mu(b'')$
8: $\sigma \leftarrow slack(p)$
9: **if** $\sigma \leqslant$ `maxSlack` $\wedge\ b \notin$ `newbs` $\wedge\ b \notin$ `oldbs` **then**
10: **if** $\varphi(p) = D^+$ **then**
11: `newbs` $\leftarrow$ `newbs` $\cup \{b\}$
12: **if** `newbs` $\neq \emptyset$ **then**
13: `maxbs` $\leftarrow$ `newbs`
14: **for all** $b \in$ `oldbs` **do**
15: **if** $\nexists b' \in$ `newbs` such that $b \sqsubseteq b'$ **then**
16: `maxbs` $\leftarrow$ `maxbs` $\cup \{b\}$
17: **until** `newbs` $= \emptyset$
18: **return** `maxbs`

Complexity of `FindAllMaximalBlocks`. The number of iterations performed by the loop in Line 2 is bounded by $\underline{d}$. The reason is that the maximum length of the blocks put in `newbs` keeps on increasing at each iteration and $\underline{d}$ is the size of the sequence for which we are computing all maximal blocks. Let m be the maximum number of blocks found in any iteration. The number of iterations performed by the loop in Line 5 is therefore bounded by m^2. The cost of checking if a given block b already exists is m and the cost of checking if there exists a block b in all the sequences of the class is $n\,\overline{d}$. Thus, the overall complexity is $\mathcal{O}(\underline{d}\,m^2(m + n\,\overline{d}))$.

Algorithm 4 computes a set of inclusion-free closed c-blocks of class D^+, denoted by `maxcbs`. The maximum number of c-blocks is bounded by $|$`maxbs`$|$ as each $b \in$ `maxbs` of the sequence p can be associated with at most one c-block. Therefore, for each such block (Line 2), the algorithm tries to find a c-block (Lines 3–12). It first creates an empty function δ (Line 3) and then set the location l to for sequence s (Line 4). It then tries to find a block in each sequence t such that its pattern is equivalent to that of p and it is not included by any block of any sequence of previously computed c-blocks (Line 6–7). If it manages to find such an embedding function then the set `maxcbs` is updated (Line 10–11). Let $m = |$`maxbs`$|$ be the maximum number of maximal blocks. The cost of finding a given maximal block in each sequence is $\overline{d}$ and the task of checking whether the block has already been discovered before is m. Therefore, the complexity of the algorithm is $\mathcal{O}(mn\,(\overline{d} + m))$.

B. Computing Optimal Discriminative Motifs

Once we have a set of non-inclusive closed c-blocks for a given value of maximum slack, we have to determine those

Algorithm 4 `FindMaximalCBlocks`(D^+,s,maxbs)

Require: D^+ is the positive class; s is a positive sequence; maxbs is the set of inclusion-maximal blocks amongst the blocks for s having positive cover.

```
1: maxcbs ← ∅
2: for all (p, l, s) ∈ maxbs do
3:     ∀t ∈ D+ δ(t) ← ∅
4:     δ(s) ← l
5:     cbFound ← TRUE
6:     for all t ∈ D+ such that t ≠ s do
7:         if ∃(p, l′, t) such that
           ∀(p′, δ′) ∈ maxcbs (p, l′, t) ⋤ (p′, δ′(t), t) then
8:             δ(t) ← l′
9:         else
10:            cbFound ← FALSE
11:    if cbFound then
12:        maxcbs ← maxcbs ∪ {(p, δ)}
13: return maxcbs
```

which are optimal and, in particular, discriminative. In other words we have to extract a replication-free and optimal motif from this set of c-blocks. We address this subproblem as a constraint optimization problem.

a) Notations.: Let A be the set of the patterns associated with non-inclusive and closed c-blocks for a class D^+. Notice that A now contains replication free patterns. For each sequence $s \in D^-$, we compute the subset E_s of A that does not cover s. This set is denoted by $E_s \subseteq A$. The exclusion count is computed by checking whether E_s is empty or not. The objective is to select a minimum number of patterns from A such that at least one from each non-empty set E_s is selected.

b) Variables and Constraints.: For each pattern of non-inclusive and closed c-block $j \in A$, a Boolean variable x_j is created that denotes whether j is part of the motif or not. For each sequence $i \in D^-$ such that $E_i \neq \emptyset$, we want to select at least one pattern that does not cover i, i.e., $\sum_{j \in E_i} x_j \geqslant 1$. The slack of the motif is the maximum of the slacks of selected patterns, i.e., $\texttt{slack} = \max_{j \in A}(\sigma(j).y_j)$ where $\sigma(j)$ denotes the slack of c-block j.

c) Objective.: The lexicographic objective is to minimize the value of slack following by minimizing the number of maximal c-blocks associated with the patterns that are selected for class D^+

$$\min \sum_{j \in A} x_j + \alpha \cdot \texttt{slack}$$

α is the coefficient whose value is set in such a way the it first minimises the slack and then the cardinality of the motif. Notice that the formulation is equivalent to that of a minimum set covering problem.

C. An Example

In this section we illustrate the complete approach by showing the trace for solving RMDP for a toy problem instance. The instance containing 2 positive sequences and 2 negative sequences is shown in Table I.

TABLE I. A TOY PROBLEM INSTANCE

Class	Sequence-id	Sequence
D+	s_1	AABAXCDDC
D+	s_2	AACAYYCEEC
D-	s_3	AABA
D-	s_4	CDDC

The trace of the approach described in the earlier sections for solving RMDP for the toy instance (Table I) is depicted in Table II. There are three columns. The first column correspond the maximum slack value, the second step denotes the name of the step and last column is the output of the step. As we have implemented a lazy approach the slack is incremented in step of 1 if required. For each slack first the minimal set of blocks of the shortest sequence is computed using Algorithm 2, the maximal set of blocks of the shortest sequence is computed using Algorithm 3, a set of maximal c-blocks is computed using Algorithm 4, and an optimal motif is computed by solving the COP using complete search. The optimal motif is $\{c_2, c_3\}$ is which excludes all the negative sequences. The minimal slack of motif is 2 and its minimal cardinality is 2. Note that $\{c_1, c_3\}$ is also optimal as it has the same objective value.

TABLE II. TRACE OF THE APPROACH FOR FINDING AN OPTIMAL SOLUTION OF RMDP

Slack	Step	Description and Output
0	Algorithm 2	Computes block b_1=AA for s_1
	Algorithm 3	Computes inclusion-maximal blocks b_1=AA
	Algorithm 4	Computes c-blocks c_1=(AA,1,1)
	COP	Finds a motif containing c_1 which does not exclude s_3
1	Algorithm 2	b_1=AA and b_2=A#A
	Algorithm 3	b_3=AA#A (AA and A#A are merged and then eliminated)
	Algorithm 4	c_2=(AA#A,1,1)
	COP	Finds motif c_2 that does not exclude s_3
2	Algorithm 2	b_1=AA, b_2=A#A and b_3=C##C
	Algorithm 3	b_4=AA#A (AA and A#A are merged and then eliminated) and b_3=C##C
	Algorithm 4	c_2=(AA#A,1,1) and c3=(C##C,6,7)
	COP	Finds motif $\{c_2, c_3\}$ which excludes all negative sequences since c_2 excludes s_4 and c_3 excludes s_3

IV. EMPIRICAL RESULTS

In this section, we present results to demonstrate the effectiveness of our approach. We investigated with two databases: Late Embryogenesis Abundant Proteins (LEAP) and Small Heat Shock Proteins (SHSP). The LEAP database [7] contains 1066 proteins partitioned into 12 classes while the SHSP database [8] contains 2244 proteins partitioned into 23 classes. The results are presented in Tables III and IV, respectively.

All algorithms are written in Java. The experiments were carried out on a Quad Core CPU running Linux with 3.8 GB of RAM and 2.66 GHz processor. All the experiments were run to completion. The lexicographic objective function was to maximise the exclusion count followed by minimizing slack and then minimise the cardinality of the motif. `cid` denotes the id of the class, `#proteins` the number of proteins in each class, $\underline{d}$ the minimum size of the protein while $\overline{d}$ denotes the maximum size of the protein of a given class.

We found that there are no motifs that can exclude all the foreign proteins (i.e., negative sequences) for 6 out of 12 classes of LEAP and 3 out of 23 classes of SHSP. In the tables, `nonexp` denotes the number of foreign proteins that an optimal motif was not able to discriminate for a given class of a given database. For class 11 of SHSP, it was not possible to discriminate any foreign protein. The slack, cardinality and length measures of the motifs are also depicted in the columns labelled as `slack`, `card`, and `length`. Computation time is

TABLE III. RESULTS OF LEAP PROTEINS OBTAINED USING RMPD

cid	#proteins	$\underline{d}$	$\overline{d}$	#nonexp	card	slack	length	time1	time2
1	177	117	507	35	13	14	29	66775	237
2	96	122	338	3	9	27	25	276043	402
3	29	86	186	0	1	0	6	92	124
4	83	81	625	690	1	3	2	6745	8
5	60	83	217	0	3	2	8	311	121
6	202	66	843	258	3	6	6	4079	18
7	53	95	341	0	1	4	4	127	23
8	184	136	411	84	2	1	4	7016	17
9	67	78	144	0	1	2	4	45	15
10	76	88	173	2	7	46	18	49962	674
11	24	159	278	0	5	1	13	358	292
12	15	71	117	0	2	0	6	55	57

TABLE IV. RESULTS OF SHSP PROTEINS OBTAINED USING RMPD

cid	#proteins	$\underline{d}$	$\overline{d}$	#nonexp	card	slack	length	time1	time2
1	237	130	163	0	10	10	20	4122	511
2	107	129	174	0	3	2	7	760	258
3	47	165	328	0	5	3	13	1215	388
4	16	119	173	0	3	0	10	251	226
5	14	172	203	0	2	0	6	356	275
6	65	127	248	0	7	8	15	3569	531
7	80	163	266	0	4	3	9	1112	412
8	15	115	146	0	6	1	14	597	357
9	146	121	170	0	4	2	10	236	437
10	294	120	498	1747	1	1	2	5471	9
11	295	130	316	1949	-	-	-	4815	0
12	257	149	269	0	8	25	16	3800	274
13	119	174	271	0	1	1	4	420	332
14	25	145	178	0	1	0	5	473	467
15	25	108	262	0	1	0	7	492	554
16	31	114	154	0	1	0	4	215	257
17	69	102	131	0	3	0	7	96	104
18	154	107	277	77	1	0	2	7765	36
19	23	194	220	0	4	1	9	665	483
20	90	214	298	0	9	3	18	656	251
21	96	161	344	0	3	1	6	217	149
22	13	332	453	0	2	0	7	1474	1392
23	26	79	127	0	2	1	5	342	161

given in milliseconds. The times for computing all maximal c-blocks and an optimal motif for a given class are shown in the columns `time1` and `time2`, respectively.

Although one can compute all non-inclusive and closed c-blocks with respect to a given value of maximum slack and then compute an optimal motif using the COP formulation presented in Section III, the memory requirement could be huge. We therefore considered a lazy approach where the value of slack was incremented only if required. More precisely, we vary the value of slack starting from 0 to a given maximum value of the slack, compute the c-blocks and an optimal motif. We stop incrementing the value of slack as soon as `nonexp` is 0. However, when the objective is `cs`, one can stop incrementing slack when the value of `nonexp` is 0 and when the length of the motif is 1. As there is a trade-off between the cardinality of the motif and its slack, for some classes we need to try more values of slacks when the objective function is `cs`. Consequently, the total time requirement for computing all maximal c-blocks for `cs` is more when compared to that of `sc`. Currently, when the value of slack is incremented, the entire problem procedure restarts from the beginning. Nevertheless, it is possible to integrate the two sub-tasks in a more incremental way which could reduce the total time. Overall, the results suggest that the presented approach is indeed scalable for handling large size instances.

As mentioned before creating a single COP model is space-wise challenging. To verify this we also modelled the problem using Minizinc [10] but the resulting models were prohibitively huge and could only be applied to toy problem instances. Tools like Miningzinc (CP for Data mining) were also considered but they did not provide any specific string computation functionalities to solve our problem.

We would like to remark that the solutions RMPD are motifs that do not enforce a total order on c-blocks. Therefore, they are more likely to compute discriminative motifs. To illustrate this a motif for one of the classes of the SHSP database is depicted in Table V. Each column is a pattern followed by their starting positions in the corresponding protein of the class. It is clear that the c-blocks are not following any consistent ordering throughout the proteins.

V. CONCLUSION

We have introduced a constraint optimization framework to compute discriminative n-ary motifs in databases of labelled sequences. This problem is of particular interest in bioinformatics to discover motifs in protein sequences or to align sequences. The framework relies on the notion of c-block to enforce consistency of pattern embeddings. It allows to state constraints on motifs such as coverage, closure, non-inclusion, or non-replication in order to minimize redundancy or meet domain-specific requirements. The framework uses a lexicographic optimization scheme to combine measures on

TABLE V. AN OPTIMAL MOTIF FOR CLASS 19^{th} OF DATABASE SHSP. THE CARDINALITY OF THE MOTIF IS 4, SLACK IS 1 AND LENGTH IS 9.

protein	PE#V	D#K	Q#S	CS
1	149	90	8	159
2	140	91	8	3
3	140	91	8	3
4	140	91	8	3
5	140	91	8	3
6	147	98	8	157
7	140	91	8	3
8	140	91	8	3
9	147	98	8	157
10	144	75	8	3
11	151	127	69	156
12	137	83	157	142
13	144	75	8	154
14	152	98	8	157
15	141	87	8	146
16	148	89	8	17
17	150	71	8	17
18	140	91	8	3
19	141	122	189	151
20	141	122	189	151
21	140	91	8	3
22	134	85	172	144
23	141	92	81	151

motifs such as slack or exclusion count. A replication-free motif computation problem has been cast in this framework and a two-stage method presented. Experimental results on protein datasets have shown its efficiency.

Future work involves casting new types of constraints on motifs, improving algorithmic efficiency and scalability (dedicated data structures, global constraints for exclusivity, etc.), carrying out experimental comparisons with ad-hoc algorithms in Bioinformatics (e.g., multiple sequence alignment), and addressing other mining tasks (e.g., clustering).

ACKNOWLEDGMENT

This publication has emanated from research supported in part from Ulysses 2013 research award. The Insight Centre for Data Analytics is supported by Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289.

REFERENCES

[1] T. Guns, S. Nijssen, and L. D. Raedt, "Itemset mining: A constraint programming perspective," *Artif. Intell.*, vol. 175, no. 12-13, pp. 1951–1983, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2011.05.002

[2] ——, "k-Pattern Set Mining under Constraints," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 2, pp. 402–418, 2013. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.204

[3] J. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, and S. Loudni, "A constraint language for declarative pattern discovery," in *Proceedings of the ACM Symposium on Applied Computing, SAC*, 2012, pp. 119–125. [Online]. Available: http://doi.acm.org/10.1145/2245276.2245302

[4] E. Coquery, S. Jabbour, L. Saïs, and Y. Salhi, "A SAT-Based Approach for Discovering Frequent, Closed and Maximal Patterns in a Sequence," in *ECAI*, 2012, pp. 258–263. [Online]. Available: http://dx.doi.org/10.3233/978-1-61499-098-7-258

[5] E. Coquery, S. Jabbour, and L. Sais, "A Constraint Programming Approach for Enumerating Motifs in a Sequence," in *Data Mining Workshops (ICDMW)*, 2011, pp. 1091–1097. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ICDMW.2011.10

[6] J.-P. Métivier, S. Loudni, and T. Charnois, "A Constraint Programming Approach for Mining Sequential Patterns in a Sequence Database," in *Workshop Languages for Data Mining and Machine Learning of the ECML/PKDD (LML'13)*, 2013, pp. 1–15.

[7] G. Hunault and E. Jaspard, "The Late Embryogenesis Abundant Proteins Database," 2013. [Online]. Available: http://forge.info.univ-angers.fr/~gh/Leadb

[8] ——, "The Small Heat Shock Proteins Database," 2013. [Online]. Available: http://forge.info.univ-angers.fr/~gh/Shspdb

[9] ——, "LEAPdb: a database for the late embryogenesis abundant proteins," *BMC Genomics*, vol. 11, no. 1, p. 221, Apr. 2010, PMID: 20359361. [Online]. Available: http://www.biomedcentral.com/1471-2164/11/221/abstract

[10] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "MiniZinc: Towards a Standard CP Modelling Language," in *Principles and Practice of Constraint Programming - CP*, 2007, pp. 529–543. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74970-7_38