A Constraint Programming Approach for Enumerating Motifs in a Sequence

Emmanuel Coquery[1]
[1]*Université Claude Bernard Lyon 1*
LIRIS - CNRS UMR 5205
F-69622 Villeurbanne Cedex, France
Email: emmanuel.coquery@liris.cnrs.fr

Saïd Jabbour[2] and Lakhdar Saïs[2]
[2]*Université Lille-Nord de France*
CRIL - CNRS UMR 8188
Rue Jean Souvraz, SP-18 F-62300, Lens
Email: {jabbour,sais}@cril.fr

***Abstract*—In this paper we propose a constraint programming approach for enumerating all frequent patterns with wildcards in a given sequence. To reduce the search space, we show that the anti-monotonicity property of frequent patterns can be dynamically encoded using nogood recording based approach. Finally, the constraints network is encoded as a Boolean formula. This last step allows us to exploit the efficiency of modern SAT solvers and particularly their clauses learning component. Preliminary experiments on real world data show the feasibility of our approach in practice.**

***Keywords*-Datamining; Constraint Programming; Boolean satisfiability**

I. Introduction

Frequent sequence-based data mining is the problem of discovering frequent patterns shared across time among an input data-sequence. Pattern discovery in data is widely used in bioinformatics as a way to extract meaningful "knowledge" in large volume of data such as protein motif discovery, gene prediction and sequence alignment. This problem is central to many other applications domains such as database and text mining.

In this paper we consider the pattern discovery problem for a specific class of patterns in a sequence. The data could be represented as a string (a sequence of "solid" characters), while the pattern can be seen as a subsequence with an additional special character called wildcard or joker that match any character [9], [12], [1]. We are particularly interested in enumerating all motifs in a sequence i.e. patterns that occurs at least λ times. The enumeration problem for maximal motifs with wildcards has been investigated recently by several authors [10], [12], [1]. One of the major problem is that the number of motifs can be of exponential size. This combinatorial explosion is tackled using different approaches. For example, in Parida et al [9], the number of patterns is reduced by introducing the maximal non redundant q-patterns (patterns occurring at least q times in a sequence). Harimura and Uno [1] proposed a polynomial delay and polynomial algorithm MaxMotif for maximal pattern discovery of the class of motifs with wildcards.

Our approach for enumerating all frequent patterns with wildcards in a sequence differs from all the previous specialized approaches. It follows the constraint programming (CP) based data mining framework proposed recently by Luc De Raedt et al. in [6] for itemset mining. This new framework offers a declarative and flexible representation model. New constraints often require new implementations in specialized approaches, while they can be easily integrated in such a CP framework. It allows data mining problems to benefit from several generic and efficient CP solving techniques. The authors show, how some typical constraints (e.g. frequency, maximality, monotonicity) used in pattern mining can be formulated for use in CP. This first study leads to the first CP approach for itemset mining displaying nice declarative opportunities without neglecting efficiency. Encouraged by these promising results our first goal is to heavily exploit these declarative languages (constraint programming and Boolean satisfiability (SAT)) and

their associated efficient and generic solving techniques. Our contributions are two folds. First, we propose a CP model to our problem. This model includes different type of constraints. The first one, called support constraint, expresses the inclusion of a candidate pattern at every position of the sequence. The second one, called frequency constraint, encodes that a pattern must appears at least λ times in the sequence. Finally, we propose a Boolean formulation of the obtained constraints network. To this end, we use both the classical encoding of constraint satisfaction problems to a boolean formula in conjunctive normal form and an additional and efficient transformation of the 0/1 linear inequality representing the frequency constraint [2]. This second Boolean formulation offers the possibility to benefit from the recent progress in satisfiability testing. Another important motivation of the Boolean model rises in the ability of modern SAT solvers to efficiently handle nogoods thanks to their clauses learning component. Indeed, to encode anti-monotonicity, we need to maintain dynamically a set of nogoods. Each time a pattern p is shown non frequent, all patterns p' such that $p \subseteq p'$ are eliminated thanks to additional nogoods.

The rest of the this paper is organized as follows. After some preliminaries on the problem of enumerating frequent patterns in a sequence (Section II-A), constraint programming and Boolean satisfiability (Section II-B), we introduce our CP formulation in Section III. In Section IV, we propose a transformation to a Boolean formula and show how to encode the anti-monotonicity property. Preliminary experiments are given before concluding.

II. Preliminaries

A. Frequent patterns mining: preliminary definitions

In this section, we give a formal description of the problem of enumerating frequent patterns, possibly interspersed with wildcards symbols, in a sequence. To this end we first introduce some basic definitions and notations according to [9], [12], [1]. Let Σ be an alphabet, which is a finite set of symbols (called solid characters). A string (or simply a sequence) s is a successive sequence of characters $s_1 \dots s_n \in \Sigma^*$ where $s_i \in \Sigma, \forall i \in \{1 \dots n\}$ represents the character at position i in s. The length of the string s is denoted by $|s| = n$. We denote $\mathcal{O} = \{1 \dots n\}$ as the set of positions of the characters in s. A wildcard (or don't care) is an additional symbol $\circ$ not belonging to Σ ($\circ \notin \Sigma$) that matchs any symbol of the alphabet including itself.

Definition 1 (Pattern): A *pattern* over Σ is a string $p = p_1 \dots p_m \in \Sigma \cup \Sigma.(\Sigma \cup \{\circ\})^*.\Sigma$, where $\cup$ and . are the classical regular expression operators of union and concatenation respectively. The first and the last position of a pattern contains a solid character i.e. $p_1 \neq \circ$ et $p_m \neq \circ$

Definition 2 (Inclusion): A pattern p appears in a sequence s at the location $l \in \mathcal{O}$ denoted $p \subseteq_l s$, if $\forall i \in \{1 \dots m\}$, $p_i = s_{l+i-1}$ or $p_i = \circ$. We note $\mathcal{L}_s(p) = \{l \in \mathcal{O} | p \subseteq_l s\}$ the support of p in s. We say that $p \subseteq s$ iff $\exists l \in \mathcal{O}$ such that $p \subseteq_l s$.

Definition 3 (Motif/Frequent pattern): Let s be a sequence and p a pattern. Given a positive number $\lambda \geq 1$, called quorum, we say that p is a motif (or frequent pattern) in s when $|\mathcal{L}_s(p)| \geq \lambda$. The set of all motifs of s for the quorum λ is denoted by $\mathcal{M}_s^\lambda$.

Example 1: Let $s = aababcabcba$ be a sequence, $p = ab \circ b$ a pattern. As $p \subseteq_2 s$, and $p \subseteq_7 s$, we have $\mathcal{L}_s(p) = \{2, 7\}$. If we set the quorum λ to the value 2, then the pattern p is also a motif of s.

Definition 4 (EMS): The problem of Enumerating all Motifs in a Sequence (EMS) can be defined as follows. Given a sequence s and a quorum $\lambda \geq 1$, enumerates all motifs $p \in \mathcal{M}_s^\lambda$.

The following anti-monotonic property can play an important role in reducing the search space. Indeed, if a pattern is shown to be not frequent, all patterns containing it are also not frequent.

Property 1 (Anti-monotonicity): Let p_1 and p_2 be two patterns such that $p_1 \subseteq p_2$. If $|\mathcal{L}_s(p_2)| \geq \lambda$ then $|\mathcal{L}_s(p_1)| \geq \lambda$.

B. Constraint programming and Boolean satisfiability

In this section, we briefly overview constraint programming and Boolean satisfia-

bility, the two most important representation models of problems with constraints.

1) Constraint programming: Constraint programming is a declarative representation model with associated effective solving techniques. CP draws on methods from artificial intelligence, logic programming, and operations research. It has been successfully applied in a number of fields such as scheduling, planning, and computational biology. This paradigm is based on a central notion called a constraint which can be seen as a relation between variables stating the allowed combination of values. A constraint can be expressed extensionally, but usually it is expressed in some chosen language (for example a subset of first order logic, linear inequalities, etc.). A problem can then be modeled as a constraint network $\mathcal{N} = (\mathcal{X}, \mathcal{C})$ made of a finite set of constraints $\mathcal{C} = \{c_1 \dots c_e\}$, where each constraint $c_i \in \mathcal{C}$ is defined on a subset $\mathcal{X}_{c_i} = \{x_{i_1} \dots x_{i_{r_i}}\}$ of a set of variables $\mathcal{X} = \{x_1 \dots x_n\}$. To each variable x_i of $\mathcal{X}$ is associated a finite set of values $dom(x_i) = \{v_1 \dots v_d\}$, called domain of x_i. A solution $\mathcal{I}$ to a constraint network is the assignment of a value v_i to each variable x_i (noted $\mathcal{I}(x_i) = v_i$) such that all the constraints are satisfied.

Given a constraint network $\mathcal{N}$, the constraint satisfaction problem (CSP) consists in deciding if $\mathcal{N}$ admits an assignment of values to its variables satisfying all the constraints. To solve such problem, and depending on the chosen representation of constraints, one can use domain specific methods, general methods or a combination of both. The first one include for example a program that solve systems of linear inequalities, while the second category generally refer to backtrack search methods. These last and brute force methods are usually coupled with useful search space reduction techniques usually called constraint propagation, and powerful variable ordering heuristics. See [5], for a detailed review on CP.

2) Boolean Satisfiability: Let $\mathcal{P}$ be a propositional language of formulas $\mathcal{F}_\mathcal{P}$ built in the standard way, using usual connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) and a set of propositional variables. Using linear Tseitin encoding, any Boolean formula $f \in \mathcal{F}_\mathcal{P}$ can be translated to $CNF(f)$ a formula in conjonctive normal form. A *CNF formula* Σ is a set (interpreted as a conjunction) of *clauses*, where a clause is a set (interpreted as a disjunction) of *literals*. A literal is a positive or negated propositional variable. A *unit* clause is a clause containing only one literal (called *unit literal*). An *empty clause*, noted $\bot$, is interpreted as false (unsatisfiable), whereas an *empty CNF formula*, noted $\top$, is interpreted as true (satisfiable). $\Sigma|_x$ will denotes the formula obtained from Σ by assigning x the value *true*. Σ^* denotes the formula Σ *closed under unit propagation*, defined recursively as follows: (1) $\Sigma^* = \Sigma$ if Σ does not contain any unit clause, (2) $\Sigma^* = \bot$ if Σ contains two unit-clauses $\{x\}$ and $\{\neg x\}$, (3) otherwise, $\Sigma^* = (\Sigma|_x)^*$ where x is the literal of a unit clause of Σ. An *assignment* $\mathcal{I}$ of Σ is a function which associates a value $\mathcal{I}(x) \in \{false, true\}$ to some of the variables of Σ. $\mathcal{I}$ is *complete* if it assigns a value to every variable of Σ, and *partial* otherwise. A *model* of a formula Σ is an assignment that makes the formula $true$. *The SAT problem consists in determining if a Boolean formula expressed in CNF admits a model or not?*

Let us informally describe the most important components of the modern SAT solvers. They are based on a reincarnation of the historical Davis, Putnam, Logemann, and Loveland procedure, commonly called DPLL [4]. It performs a backtrack search; selecting at each level of the search tree, a decision variable which is set to a Boolean value. This assignment is followed by an inference step that deduces and propagates some forced unit literal assignments. This is recorded in the implication graph, a central data-structure, which encodes the decision literals together with there implications. This branching process is repeated until finding a model, or a conflict. In the first case, the formula is answered satisfiable, and the model is reported, whereas in the second case, a conflict clause (called learnt clause) is generated by resolution following a bottom-up traversal of the implication graph [8], [14]. The learning or conflict analysis process stops when a conflict clause containing only one literal from the current decision

level is generated. Such a conflict clause asserts that the unique literal with the current level (called asserting literal) is implied at a previous level, called assertion level, identified as the maximum level of the other literals of the clause. The solver backtracks to the assertion level and assigns that asserting literal to *true*. When an empty conflict clause is generated, the literal is implied at level 0, and the original formula can be reported unsatisfiable. In addition to this basic scheme, modern SAT solvers use other components such as activity based heuristics, and restart policies. An extensive overview can be found in [3].

III. A Constraint programming model for EMS

In this section, we present a CP model for EMS. We recall that n and m represent the size of the sequence and the maximal size of motifs respectively.

To enumerate all motifs of arbitrary length, we first need to set the length of the motif to the upper bound $m = n - \lambda + 1$. Secondly, we suppose that the sequence is completed on the right hand side with $m - 1$ wildcards.

Let us take again the example 1. The maximum length of the motifs is set to $m = 9$. Indeed, as $\lambda = 3$ and the size of the sequence is $n = 11$, then the patterns of size greater than 9 are not frequent. The number of wildcards added at the end of the sequence is $m - 1 = 8$. This completion allows us to search for all motifs of different size. Indeed, the string a$\circ\circ\circ\circ\circ\circ\circ\circ$ of size 9 representing the pattern a of size 1 (a pattern starts and ends with a solid character) appears 5 times in s, if s is completed with 8 wildcards; 2 times otherwise.

We now introduce our CP model for enumerating all motifs in a sequence s given a quorum λ. The problem is defined as a constraint network $\mathcal{N}_s^{co}(\lambda) = (\mathcal{X}, \mathcal{C})$.

Variables and domains: To match a candidate pattern to a subsequence of s, we introduce two types of variables:

- $\mathcal{P} = \{p_1 \dots p_m\}$ where p_i for $1 \leq i \leq m$ represents the ith symbol of the candidate pattern p. The set of variables $\mathcal{P}$ represents the candidate pattern p. Each $p_i \in \mathcal{P}$ can be either instantiated with $a \in \Sigma$ or $\circ$ i.e. $dom(p_i) = \Sigma \cup \{\circ\}$.
- $\mathcal{B} = \{b_1 \dots b_n\}$ where b_k for $1 \leq k \leq n$ is a Boolean variable. $b_k = true$ if the pattern p is located at the position k in s; $false$ otherwise. The set of variables $\mathcal{B}$ represents the support $\mathcal{L}_s(p)$.

An instantiation of the variables $\mathcal{P}$ of a candidate pattern p to $a_1 \dots a_m$ represents a pattern $a_1 \dots a_l$ such that $a_l \neq \circ$ and for all $l < i \leq m$, $a_i = \circ$ i.e. l is the last position of a solid character in $a_1 \dots a_m$.

Constraints: We define the location constraint (1) to express that p is located in s at position k:

$$loc(k, p, s) = \bigwedge_{i=1}^{m} (p_i = \circ \vee s_{k+i-1} = p_i) \tag{1}$$

The support contraint (2) expressing the location of p at each position k in s:

$$supp(p, s) = \bigwedge_{k=1}^{n} (b_k \Leftrightarrow loc(k, p, s)) \tag{2}$$

The support constraint (2) is sufficient for the enumeration all the patterns. To search only for frequent patterns, we introduce the frequency constraint (3) which express that p must occurs at least λ times in s.

$$freq(p, s) = \sum_{k=1}^{n} b_k \geq \lambda \tag{3}$$

The last unary constraint expresses that the first symbol of p is a solid character:

$$p_1 \neq \circ \tag{4}$$

The constraint (4) can be omitted by eliminating $\circ$ from $dom(p_1)$.

Constraints network: The EMS CP model, can be represented as constraint network $\mathcal{N}_s(\lambda) = (\mathcal{X}, \mathcal{C})$ where:

- $\mathcal{X} = \mathcal{P} \cup \mathcal{B}$
- $\mathcal{C} = supp(p, s) \wedge freq(p, s) \wedge p_1 \neq \circ$

Definition 5: Let $\mathcal{I}$ be a solution of $\mathcal{N}_s(\lambda)$. We define $pattern(\mathcal{I}) = \mathcal{I}[p_1] \dots \mathcal{I}[p_l]$ such that $\mathcal{I}[p_l] \neq \circ$ and $\forall i \in \{l+1 \dots m\}$, $\mathcal{I}[p_i] = \circ$.

Property 2: $\mathcal{I}$ is a solution of $\mathcal{N}_s(\lambda)$ iff $pattern(\mathcal{I})$ is a motif of s.

Proof: consequence of the encoding. ■

A. Anti-monotonicity: a nogood-based approach

To exploit the anti-monotonic property, one need to prove that a given pattern p is not frequent. Consequently, each time a pattern is proven non frequent, we dynamically add new constraints called nogoods to the constraints database.

Let $\mathcal{N}_s(\lambda)$ be the constraint network obtained by our EMS CP model. We assume that $\mathcal{I}$ is the current instantiation such that the frequency constraint is violated and $p' = \{p'_1, \ldots, p'_k\} = pattern(\mathcal{I})$. Let $i_1, \ldots i_l$ be the ordered sequence of positions in p' such that $\forall 1 \leq j \leq l, p'_{i_j} \neq \circ$. The following nogoods are added dynamically to the constraints database in order to avoid future patterns p such that $p' \subseteq p$.

$$antiMon(p', p) = \bigwedge_{x=1}^{m-i_l+1} \bigvee_{y=1}^{l} (p'_{i_y} \neq p_{i_y+x-1}) \quad (5)$$

These nogoods state for each position x in p s.t. $p' \not\sqsubseteq_x p$, at least one of the solid characters in p' is not matched in p at the expected position $i_y + x - 1$.

IV. Boolean satisfiability model

In this section, we show how the EMS CP model (Section III) can be encoded as a Boolean formula in CNF. This Boolean model is motivated by several important features of SAT solvers. Our goal is to benefit from the (1) *impressive progress in Boolean Satisfiability checking* [3]. The scalability of modern SAT solvers can be related to both algorithmic improvements and to (2) *their ability to exploit the hidden structures of the problem instance*. By structure, we understand the *dependencies between variables*, which can often appear through Boolean functions. One particular example being the well known notion of *strong backdoors* [13] that can be informally defined as subset of variables such that any assignments of these variables leads to a tractable sub-formula.

Another important motivation of this Boolean model rises in the ability of modern SAT solvers to (3) *efficiently handle nogoods thanks to their clauses learning component*.

The features (2) and (3) are of particular interest in our context. Indeed, if we take a closer look to the EMS CP model, we can see that the set of variables representing the patterns $\mathcal{P}$ is clearly a strong backdoor set. Indeed, when such variables are assigned, the values of the other Boolean variables are trivially deduced. The constraint (2) expresses such dependencies (the value of the variable b_k depends on the value of $loc(k, p, s)$). Consequently, one can enumerate only on $\mathcal{P}$. Hence, the complexity is exponentially bounded by the size of $\mathcal{P}$. This is clearly an interesting information that we provide to the SAT solver. It is important to note that the exploitation of the anti-monotone property in our approach requires the use of the strong backdoors (pattern variables). By branching on these pattern variables, the extraction of the non frequent pattern can be done in a simple way as the pattern variables are the first assigned in the current branch. Another reason, is that when a conflict occurs we can also determine if such a conflict is encountered because of the frequency constraint.

To encode our EMS CP model into a Boolean formula, we use both the direct encoding of constraint satisfaction problems to a CNF formula and an additional and efficient transformation to CNF of the 0/1 linear inequality representing the frequency constraint (3). Suppose that $\Sigma = \{a_1 \ldots a_{|\Sigma|}\}$. The Boolean formula is defined as follows:

Variables: for each $p_i \in \mathcal{P}$ we associate $|\Sigma| + 1$ boolean variables $\{p_{i_1} \ldots p_{i_j} \ldots p_{i_{|\Sigma|+1}}\}$, where $p_{i_j}, 1 \leq j \leq |\Sigma|$ expresses that $p_i = a_j$ and $p_{i_{|\Sigma|+1}}$ expresses that $p_i = \circ$. The number of boolean variables is $|\mathcal{B}| + |\mathcal{P}| \times (|\Sigma| + 1)$.

Clauses : are obtained as follows:

- *Domains encoding*: expresses that a

given variable must be assigned to exactly one value. For each variable $p_i \in \mathcal{P}$ with $dom(p_i) = \Sigma \cup \{\circ\}$ we introduce two kind of constraints. The $atLeastOne$ $(p_{i_1} \vee \cdots \vee p_{i_{|\Sigma|+1}})$ expresses that each p_i must be assigned to at least one value, while $atMostOne$ $\bigwedge_{1 \leq j < k \leq |\Sigma|+1} (\neg p_{i_j} \vee \neg p_{i_k})$ encodes that each p_i must be assigned to at most one value.

- *Constraints encoding* : support constraint (2) is a boolean formula that can be translated in usual way to CNF. For the 0/1 linear inequality encoding the frequency constraint (3), there exists several efficient transformation to CNF (e.g. [2]). In our implementation, we exploit the BoolVar/PB java library [1] dedicated to the translation of pseudo-Boolean constraints into CNF.

V. Implementation and preliminary experiments

A. Implementation details

In this section we describe our SAT based solver for EMS (Algorithm 1). It is based on a model of CDCL SAT algorithm proposed in [11]. Algorithm 1 (`SatEms+AM`) includes all the basic components of modern SAT solvers and integrates the anti-monotone property (AM) with strong backdoors ($\mathcal{P}$). It takes as input a CNF formula Σ and a set $\mathcal{P}$ of pattern variables and returns the set $\mathcal{M}$ of motifs. The algorithm is based on variable assignments called *decisions* ($\mathcal{D}$) followed by unit propagation. `SatEms+AM` starts with the following four empty sets (lines 1-4): decision literals ($\mathcal{D}$), motifs ($\mathcal{M}$), learnt clauses database (Δ) and the anti-monotone nogoods database (Γ). Then, it iterates until finding all the motifs. In each iteration, the conjunction of Σ, $\mathcal{D}$, Δ, and Γ are checked for inconsistency using unit propagation (line 7). If unit propagation finds an inconsistency ($\mathcal{S}^*==\bot$, line 8), the algorithm does one of the two things:
(1) The decision sequence $\mathcal{D}$ is empty, the algorithm terminates by returning the set of all motifs $\mathcal{M}$ (line 8).
(2) The decision sequence $\mathcal{D}$ is not empty, a clause α is generated by classical conflict analysis (line 9) and added to the learnt clauses database Δ (line 10). As a conflict occurs, a non frequent pattern p is then generated (line 11) and the set of associated anti-monotone nogoods Θ are built (line 12). From the nogood representing the non frequent pattern p we apply the classical conflict analysis and generate an additional learnt clause β (line 13). This last learnt clause together with the set of nogoods Θ are added to the anti-monotone nogoods database Γ (line 14). Then a level n is computed based on α and β by taking the minimum of their assertion levels (line 15). The algorithm then erases all decisions made after level n, and moves on to the next iteration.

If unit propagation detects no inconsistency, the solver makes a decision by selecting a literal l from the backdoor set (line 18), and adds it to the decision sequence (line 24). If no such literal is found, a motif p is then extracted from the set of decisions and added to the set of motifs $\mathcal{M}$ (line 19). To avoid enumerating the same models several times, in line 20, a nogood σ is generated from the found motif and added to the original formula Σ (line 21). Search restarts by setting the set of decision to an empty set (line 22).

Some of the basic SAT functions of Algorithm 1 such us $analyzeConflict()$ and $level()$ are informally described in Section II-B. We will now provide some missing details of the other specific functions :

- $extractPattern(\mathcal{D}, \mathcal{P})$: given a set of decisions $\mathcal{D}$ and a set of pattern variables $\mathcal{P}$, a pattern is extracted according to Definition 5.
- $antiMonotone(p)$: generates a set of anti-monotone nogoods (Equation 5).
- $extractNogood(p)$: let $p = d \circ c$ be a motif of s, the extracted nogood is $(\neg p_{1_d} \vee \neg p_{2_\circ} \vee \neg p_{3_c})$.

B. Preliminary experiments

In this section, preliminary results of our approach on some real world data are given. We consider sequences from two application domains:
Bioinformatics: proteinic data encoded as

[1] BoolVAR/PB : http://boolvar.sourceforge.net/

Algorithm 1: SatEms+AM

Input: a CNF formula Σ, and a set $\mathcal{P}$ of pattern variables
Output: a set of all motifs $\mathcal{M}$

1 $\mathcal{D} \leftarrow \emptyset$; /* Decision literals */
2 $\mathcal{M} \leftarrow \emptyset$; /* Set of all motifs */
3 $\Delta \leftarrow \emptyset$; /* Learnt clauses */
4 $\Gamma \leftarrow \emptyset$; /* Anti monotone nogoods */
5 **while** (true) **do**
6 $\mathcal{S} \leftarrow (\Sigma \wedge \mathcal{D} \wedge \Delta \wedge \Gamma)$;
7 **if** $(\mathcal{S}^* = \bot)$ **then**
8 **if** $(\mathcal{D} = \emptyset)$ **then return** $\mathcal{M}$;
9 $\alpha \leftarrow analyzeConflict(\mathcal{S}, \mathcal{D})$
$\Delta \leftarrow \Delta \cup \{\alpha\}$;
10 $p \leftarrow extractPattern(\mathcal{D}, \mathcal{P})$
$\Theta \leftarrow antiMonotone(p)$
$\beta \leftarrow analyzeConflict(p, \mathcal{S}, \mathcal{D})$;
11 $\Gamma \leftarrow \Gamma \cup \Theta \cup \{\beta\}$;
12 $n \leftarrow min\{level(\alpha), level(\beta)\}$;
13 $\mathcal{D} \leftarrow \mathcal{D}_n$; /* n first decisions */
14 **else**
15 **if** $((l \leftarrow$ decide$(\mathcal{P}))=$ null) **then**
$p \leftarrow extractPattern(\mathcal{D}, \mathcal{P})$,
$\mathcal{M} \leftarrow \mathcal{M} \cup \{p\}$;
16 $\sigma \leftarrow extractNogood(p)$;
17 $\Sigma \leftarrow \Sigma \cup \{\sigma\}$;
18 $\mathcal{D} \leftarrow \emptyset$;
19 **else**
20 $\mathcal{D} \leftarrow \mathcal{D} \cup \{l\}$;

a sequence of amino-acid of arbitrary length[2].

Computer security: user data drawn from the command histories of UNIX computer users[3] [7].

These experiments aim to show the (1) feasibility of our approach, and to analyze the (2) power and weakness of our implementation. We compare two versions of Algorithm 1 based on MiniSAT 2.2 [4]:

1) SatEms+AM: a full version of the Algorithm 1. It integrates the anti-monotone property together with strong backdoors set $\mathcal{P}$.
2) SatEms: a basic version of Algorithm 1. It can be obtained from SatEms+AM by eliminating $\mathcal{P}$ from the function $decide(\mathcal{P})$, thus making the solver branch on any variables in Σ, and by deleting the lines 11-14, and substituting the line 15 with $n \leftarrow level(\alpha)$.

We conducted two kind of experiments. In the first one, we illustrate the evolution of computation time while varying the length of the input sequence. The different sequences are build from the *User data* by taking the first k characters, where k is varied from 200 to 1600 by a step of 200. As in [1], to get approximately the same number of motifs, for each sequence i (dot in the figure), we use a quorum proportional to its length ($\lambda_i = \frac{length_i}{10}$).

A comparison between SatEms and SatEms+AM is depicted in Figure 1. The results clearly show that applying the anti-monotone property with additional domain knowledge leads to dramatic improvements. SatEms+AM outperform the black-box version of SatEms. This experiment demonstrates the feasibility of our approach even if the number of added anti-monotone nogoods is huge. Similar results are obtained on all the available sequences.

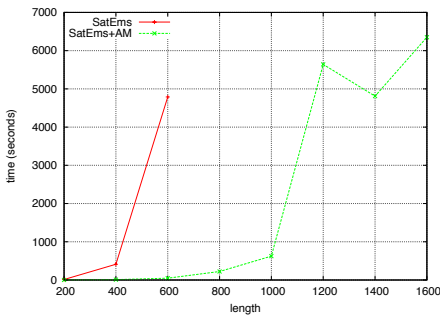

Figure 1. User data : time Vs length

In the second experiment, we consider proteinic data of fixed length and we measure the evolution of computation time with respect to the quorum. The quorum evolves linearly ($\lambda_0 = 10$ and $\lambda_i = \lambda_{i-1} + 10$). The results are depicted in Figure 2. The considered sequence is of length 675 characters. Again SatEms+AM outperform SatEms by several order of magnitude. The hardest instances are obtained for small values of the quorum. Obviously smaller is the value of the quorum, greater is the number of motifs.

To analyze the behavior of our approach with respect to the number of motifs, we show in Figure 3 (in log scale) the evolution of the ratio between CPU time and the number of models while varying the quorum. We used the same sequence as in Figure 2. The good news is that the ratio time/#motifs do not evolves significantly with SatEms+AM. Its performance seems to be less sensitive with respect to the

[2]http://www.biomedcentral.com/1471-2105/11/175/additional/
[3]http://kdd.ics.uci.edu/databases/UNIX_user_data/
[4]MiniSAT: http://minisat.se/

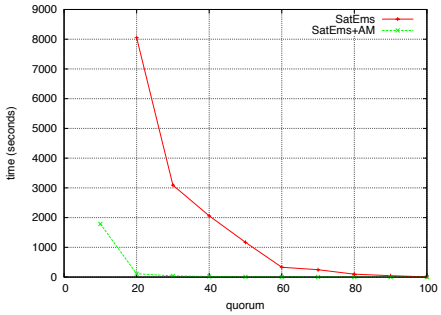

Figure 2. Bioinfo: time Vs quorum

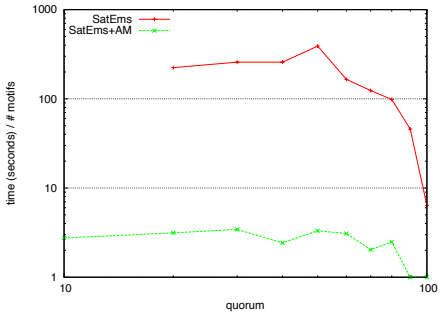

Figure 3. Bioinformatics: time/#motifs Vs quorum

number of motifs.

VI. Conclusion and Future Works

In this paper, we proposed a first constraint programming and Satisfiability based approach for enumerating motifs with wildcards in a sequence. We have shown how such CP model can be translated to Boolean Satisfiability. The experiments on real data sequences show the feasibility of our approach in practice. This preliminary work opens several perspectives. We plan to extend our framework to the problem of enumerating maximal motifs. This issue is under investigation. Nogoods will play again a central role. Finally, the next step is to compare with specialized datamining approaches.

References

[1] Hiroki Arimura and Takeaki Uno. An efficient polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. Journal of Combinatorial Optimization, 13, 2007.

[2] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-boolean constraints into cnf. In SAT'2009, pages 181–194, 2009.

[3] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. Handbook of Satisfiability, volume 185 of Frontiers in AI and Applications. IOS Press, 2009.

[4] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. Communications of the ACM, 5(7):394–397, 1962.

[5] Rina Dechter. Constraint Processing. Morgan Kaufmann publishers, 2003.

[6] T. Guns L. De Raedt and S. Nijssen. Constraint programming for itemset mining. In ACM SIGKDD, pages 204–212, 2008.

[7] Terran Lane. Filtering techniques for rapid user classification. In AAAI-98/ICML-98 Joint Workshop on AI Approaches to Time-series Analysis, pages 58–63, 1998.

[8] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In Proceedings of IEEE/ACM CAD, pages 220–227, 1996.

[9] Laxmi Parida, Isidore Rigoutsos, Aris Floratos, Dan Platt, and Yuan Gao. Pattern discovery on character sets and real-valued data: Linear bound on irredundant motifs and an efficient polynomial time algorithm. In ACM-SIAM Symposium on Discrete Algorithms, pages 297–308, 2000.

[10] Laxmi Parida, Isidore Rigoutsos, and Dan Platt. An output-sensitive flexible pattern discovery algorithm. In CPM'2001, pages 131–142, 2001.

[11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning sat solvers with restarts. In CP'2009, pages 654–668, 2009.

[12] Nadia Pisanti, Maxime Crochemore, Roberto Grossi, and Marie france Sagot. Bases of motifs for generating repeated patterns with wild cards. IEEE/ACM TCBB'2003, 2:2005, 2003.

[13] R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In IJCAI'2003, pages 1173–1178, 2003.

[14] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in Boolean satisfiability solver. In IEEE/ACM CAD'2001, pages 279–285, 2001.