No d'ordre: 2009-ISAL-0036 Année 2009

THÈSE

présentée devant

L'Institut National des Sciences Appliquées de Lyon

pour obtenir

Le Grade de Docteur

Spécialité

Informatique

Ecole Doctorale: Informatique et Information pour la Société

par

Ieva Mitašiūnaitė

Mining String Data under Similarity and Soft-Frequency Constraints: Application to Promoter Sequence Analysis

Soutenue publiquement le 19 mai 2009 devant le jury:

Michael R. Berthold	University of Konstanz, D	Examinateur
Jean-Francois Boulicaut	INSA Lyon, LIRIS UMR CNRS 5205	Directeur
Olivier Gandrillon	UCBL, CGMC UMR CNRS 5534	Examinateur
Ross D. King	University of Wales, UK	Examinateur
Dominique Mouchiroud	UCBL, LBBE UMR CNRS 5558	Examinateur
Arno Siebes	Universiteit Utrecht, NL	Rapporteur
Maguelonne Tesseire	CEMAGREF, Montpellier	Rapporteur

Acknowledgements

I would like to thank the members of the thesis committee Prof. Michael R. Berthold, Dr. Olivier Gandrillon, Prof. Ross D. King, Dr. Dominique Mouchiroud, Prof. Arno Siebes and Dr. Maguelonne Teisseire for their interest and the time dedicated to evaluate this thesis. I also thank Prof. Arno Siebes and Dr. Maguelonne Teisseire for their work and the attention put in writing the rapports of this manuscript.

I would like to offer my very sincere thanks to my supervisor Prof. Jean-François Boulicaut who made this PhD research very rewarding in a scientific point of view and in human terms. I am grateful for the continuous care starting from the first days I joined his research group as a Master intern student in September 2004.

The work of this thesis results from the close collaboration between the Turing group directed by Prof. Jean-François Boulicaut and biologist partners. The context of such interdisciplinary collaboration allowed me to learn a lot about biology (for someone graduated in computer science). I would like to thank Dr. Sophie Rome from the group *Adaptations nutritionnelles et mécanismes de l'insulino-résistance* group in INSERM U870/ INRA 1235 laboratory for the patient and very comprehensive answers to my numerous questions about gene regulation, for providing me with books, for her enthusiasm and interest in my Master's internship work. A large part of this thesis work was built on the close collaboration with the group *Bases Moléculaires de l'Autorenouvellement et de ses Altérations*, directed by Dr. Olivier Gandrillon, in CGMC CNRS UMR 5534 laboratory at the University Lyon 1. The application of our contribution to promoter sequence analysis is the fruit of the joint work with Dr. Olivier Gandrillon, Dr. Christophe Rigotti, Laurène Meyniel and Stéphane Schicklin. Last but not least I thank our IQ IST-FET FP6-516169 project partner biologist and computer scientist Prof. Ross D. King from the University of Wales for the enriching discussions, advising, care and the collaboration on mining yeasts genomes.

Since my arrival in Lyon in 2004, the research group *Data Mining and Inductive Databases* (which, starting on October 2006, became the *Modeling and Knowledge Discovery* group, called *Turing*) became by "second home" for 4 years. During the first 2 years I shared the office with the former PhD students and nowadays Doctors

Ruggero G. Pensa and Jérémy Besson. Ruggero and Jérémy, thank you a lot for being friends and for a joyful and supportive daily presence. I would like to offer a special thank to Dr. Christophe Rigotti for communicating the joy of the research work, for patient explanations and collaboration, for the attention and presence in a number of situations. I think of my colleagues Dr. Guillaum Beslon, Dr. Carole Knibbe, Dr. Claire Leschi, Dr. Céline Robardet, Dr. Nazha Selmaoui, my colleagues PhD students Loic Cerf, Antoine Coulon, Dominique Gay, Yolanda Sanchez-Dehesa, my colleagues Master intern students Yann Letrillard, Stefania Loparco, Laurène Meyniel, Stéphane Schicklin and Jana Schmitz – I learned from each of you and it is also because of your presence, sharing and help that I arrived where I am now.

My arrival to Lyon for the Research Master *Mathématiques et informatique du vivant* studies with the specialisation *Méthodes en bioinformatique moléculaire* was the first step towards this thesis. Many persons have contributed so that I could pursue these bioinformatics studies in Lyon. I thank attaché of scientific, academic and technical cooperation Christian Bonneville for his confidence to recommend to attribute me the French Government scholarship. I also thank the assistant in the Cooperation and Cultural Action Center in the French Embassy Giedrė Bernotaitė for her availability and help concerning the administrative and practical issues. I thank Dr. Donatas Kvedarauskas and Dr. Jolita Ralytė for the advices, availability and help to prepare the documents of the candidature for this scholarship. I would like to offer a sincere thank to the responsible of the Research Master *Mathématiques et informatique du vivant* at INSA Lyon Dr. Hubert Charles for the attention and confidence he witnessed being the first person who responded me in Lyon when I was anonymously searching for the Master's internship. I am equally grateful to the responsible of that Master at the University Lyon 1 Dr. Sandrine Charles and the responsible of my Master specialisation *Méthodes en bioinformatique moléculaire* Dr. Dominique Mouchiroud for their care and disponibility when answering my questions.

This thesis is a result of the 4 years spent in Lyon, including not only the work in the laboratory, but also the out-laboratory discussions, evenings and weekends. I think of my flatmates Magdalena Mistak, Elodie Reynold, Valentina Pistoia, Aurore Millet, Laetitia Catala, Alexandra Curien, Asta Krupavičiūtė, Iris Lobecke, Cathrin Vetterlein, Tanja Schilling – thank you for daily sharing and support. I also thank Gérard A. Sanchez for the enriching experience of 3 years of living in *28 rue Billon.* During 2 final years of my PhD studies I attended the weekly lessons of watercolours, and I would like to say a special thank to Jean-Marc Brugeilles and my watercolours colleagues Marie-Ange Andrianoely, Jean-Louis Andrieu, Nicole Gautier, Christiane Heranney and Alain Perrot. I equally thank a former PhD student and nowadays Doctor Romuald Thion and say a sincere *farewell.* My very special thanks goes to Françoise Boulicaut.

The ground for Research Master studies in Lyon was set and, actually, possible because of my previous *Maîtrise* studies in the University of La Rochelle, which

I pursued thanks to Socrates-Erasmus scholarship and program. I would like to thank Dr. Vilius Stakėnas who was responsible for the international relationships at the Faculty of Mathematics and Computer Science in Vilnius University and Rima Masiulienė who worked enthusiastically to launch and develop the academic relations with France. I would like to offer a very sincere thank to my binome-colleague Matthieu Guénégo whose patience while explaining French words in the exercises, joyful presence and collaboration helped me a lot. My special thanks goes to Dr. Arūnas Stočkus, to Manu and to my *Maîtrise* internship supervisor at the University of La Rochelle Dr. Frédéric Bertrand.

I thank my professor of *Bio-cybernetics* at Vilnius University Dr. Vygandas Vanagas for supporting my interest in stydying the living systems and encouraging to pursue my Master cursus in the domain of bioinformatics. The implementation of the algorithms in C and C++ is an important part of the work of this thesis. I would like to thank my professor of *C and C++ Programming* at Vilnius University Viktoras Golubevas for communicating the pleasure of programming what initiated me to develop an understanding and a self-confidence necessary to tackle the difficult programming tasks.

Most of all, my gratitude goes to my family. First, to my parents Antanas and Idalija, and my brother Jonas, who have been very supportive during the years that I have spent abroad to pursue my studies. I sincerely thank Jérémy for the continuous support, encouragement and help during all the year that took to write this manuscript.

Abstract

An inductive database is a database that contains not only data but also patterns. Inductive databases are designed to support the KDD process. Recent advances in inductive databases research have given rise to a generic solvers capable of solving inductive queries that are arbitrary Boolean combinations of anti-monotonic and monotonic constraints. They are designed to mine different types of pattern (i.e., patterns from different pattern languages). An instance of such a generic solver exists that is capable of mining string patterns from string data sets. In our main application, promoter sequence analysis, there is a requirement to handle fault-tolerance, as the data intrinsically contains errors, and the phenomenon we are trying to capture is fundamentally degenerate. Our research contribution to fault-tolerant pattern extraction in string data sets is the use of a generic solver, based on a non-trivial formalisation of fault-tolerant pattern extraction as a constraint-based mining task. We identified the stages in the process of the extraction of such patterns where state-of-art strategies can be applied to prune the search space. We then developed a fault-tolerant pattern match function InsDels that generic constraint solving strategies can soundly tackle. We also focused on making local patterns actionable. The bottleneck of most local pattern extraction methods is the burden of spurious patterns. As the analysis of patterns by the application domain experts is time consuming, we cannot afford to present patterns without any objective clue about their relevancy. Therefore we have developed two methods of computing the expected number of patterns extracted in random data sets. If the number of extracted patterns is strongly different from the expected number from random data sets, one can then state that the results exhibits local associations that are a priori relevant because they are unexpected. Among others applications, we have applied our approach to support the discovery of new motifs in gene promoter sequences with promising results.

Key Words: Inductive databases, constraint-based pattern mining, string patterns, fault-tolerance, patterns relevancy assessment, promoter sequence analysis.

Contents

List of Figures

List of Tables

Introduction

This manuscript presents a research on mining string data under similarity and soft-frequency constraints and its application to promoter sequence analysis. It has been the result of a tight collaboration between a group of computer scientists (TURING group, LIRIS CNRS UMR 5205 at INSA Lyon), working on data mining methods, supervised by Prof. Jean-François Boulicaut, and a group of biologists (BM2A[1] group, CGMC CNRS UMR 5534 at University Lyon 1), working on molecular basis of self-renewal, supervised by Dr. Olivier Gandrillon. This collaboration started 7 years ago. It was for the TURING group an occasion to confront their data mining algorithms and tools to real data sets and to discover new valuable open research questions. The primary interest for biologists was to be able to extract relevant groups of co-expressed genes in the data sets resulting from SAGE experiments [VEVK95]. Another concern was to take in account the additional information on gene regulation mechanisms. The tight link between mathematical and algorithmical methods, employed in the TURING group, and those that appear relevant in the regulation mechanisms in molecular biology, acts as a remarkable catalyst. It is now clear that new data mining tools can help biologists in their complex data analysis tasks. However, while trying to answer specific needs driven by the application, we aim at developing sound and complete generic algorithms. This manuscript will show, in addition to obtained results, the enthusiasm and the curiosity which lead all of those who participated to this research project.

Substring Pattern Mining

The TURING group of LIRIS mainly focus on constraint-based *local pattern* mining. Informally, local patterns describe some properties of a subset of a data, on the contrary to the *global models*, which characterize the entire data. Local patterns, e.g., itemsets and substrings, are of descriptive nature, whereas global models, e.g., clusters and decision trees, are of predictive nature (for more details on local patterns versus global models, see [Han02, HMS01b]).

[1]Bases Moléculaires de l'Autorenouvellement et de ses Altérations, *fr.*

This work considers *string* data sets mining to find *substring* patterns that satisfy the given *constraints*. String is a finite sequence of symbols from a finite alphabet. Substring is a sequence of contiguous symbols in a string. For example, $\mathcal{D} = \{\mathsf{actgcac}, \mathsf{acttgcgac}, \mathsf{gatagata}, \mathsf{tgctgtgtg}, \mathsf{gtcaacg}\}$ is a string data set, containing five strings over an alphabet of four symbols $\{a, c, g, t\}$. The *frequency* of a string pattern ϕ in a string data set is the number of data strings in $\mathcal{D}$ that contain the substring ϕ. For example, in $\mathcal{D}$, the frequencies of the substrings tgc, act and caac are respectively 3, 2 and 1. The principal primitive constraints, which we use to specify the string mining tasks, are the *minimum frequency*, *maximum frequency*, and *syntactic constraints*. Patterns, satisfying the minimal (resp. maximal) frequency constraint, have a frequency that is higher (resp. lower) than a given threshold. A syntactic constraint restricts the form of a pattern and is not related to a data set. For example in $\mathcal{D}$, we may look for the set of string patterns whose the frequency is higher than 1 (minimum frequency constraint), but lower than 4 (maximum frequency constraint) and that contain the symbol "c“ (a syntactic constraint). The substring patterns tgc, act satisfy this constraint. The pattern caac does not satisfy the constraint, because only one data string contains its occurrence. On the other hand, the pattern a is too frequent and do not respect the maximum frequency constraint. As we will see, constraints are the pillars of local pattern mining techniques.

The extraction of substring patterns is by nature highly combinatorial. A string of length L over an alphabet of n symbols contains up to $\sum_{i=1}^{L} n^i$ different substrings. In practice, the data sets, we are interested in, may contain thousands or tens of thousands of patterns. Both search-space and solution set can be huge. The solution space is commonly a small portion of the search space. One of the main difference with the problems, tackled in the domain of artificial intelligence, is that the access time to the data set is taken into account in data mining, and makes part one of the main problem, altogether with the size of the search-space. Reducing the cost of data access generally leads to more efficient algorithms. The first concern of the data mining algorithm is to make the extractions tractable in most real-life situations. The second (main) concern is to provide not just patterns, but only the relevant patterns to the data analyst. Indeed, the analysis and the assessment of the extracted patterns is difficult, and the a priori irrelevant patterns must be avoided. Indeed, actionable patterns are quite often hidden amongst many irrelevant ones. The bottleneck of most local pattern extraction methods is the burden of these spurious patterns. Irrelevancy must be understood in terms of both objective and subjective interestingness.

That is where constraints become so crucial. Extraction tasks can be defined in a declarative manner, by specifying the pattern type, the constraints that the patterns must satisfy and the data set being analyzed. The role of constraints is multiple: (1) they enable to reduce the search-space by discarding uninteresting regions, and thereby lead to much more efficient extractions, while keeping the result sound and complete, (2) they reduce the size of the solution set and (3) they improve the relevancy of the extracted patterns w.r.t. to data analyst expectations. It looks like a silver bullet: the pattern relevancy is increased, whereas the extraction time is reduced. The difficulty is that this generally applies to a limited set of constraints that exhibits the (anti-)monotonicity or the associated properties. A constraint is said anti-monotonic (resp. monotonic) w.r.t. a partial relation order on pattern space, if when a pattern does not satisfy the constraint then every "smaller" (resp. "larger") pattern than it does not satisfy it either. It is a simple, but very powerful technique to safely prune search spaces.

If constraint-based data mining is our answer, we have to specify the constraints that define the extraction task. This is not a trivial task. There is often a trade-off between the constraint expressiveness, which is necessary to support subjective interestingness specification, and the possibility to solve that constraint efficiently, e.g., by exploiting the associated (anti-)monotonicity properties to prune the search space. One can search how to express (possibly approximately) these difficult constraints through combinations of constraints that have these good properties. This is the approach we adopted in our doctoral research to tackle the extractions of string patterns under similarity and soft-frequency constraints.

Inductive Databases

In recent years, the data mining research community put considerable effort to develop efficient algorithms to extract various types of patterns in different kinds of large collections of data. However, the data mining step in the Knowledge Discovery in Database (KDD) process [PSF91] is not limited to the application of an algorithm to produce a collection of local patterns or to construct a global model. The process of finding knowledge in data is composed of various steps, and among them one can distinguish the collection of data, data selection and transformation, pattern extractions and/or model construction, result visualisation, selection and assessment. It is an iterative process, rather than a strict sequential one. In addition, it can involve different kinds of data and also different types of local patterns and/or global model. The underlying idea of the *inductive databases* (IDB) framework [IM96, BKM98, BKM99, MT97] is that the tasks of selecting, manipulating and querying data and patterns can be considered as queries, and consequently the whole KDD process can be considered as a process of querying. Concerning patterns, the idea is to see an extraction task as an *inductive query* that declaratively specifies the properties of patterns we are looking for in an intentionally defined pattern space (i.e., generally not materialized).

The pattern extraction task can be formalized as a task of computing the set, denoted $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$, i.e., finding the patterns ϕ in a pattern space $\mathcal{L}$ such that ϕ satisfies the constraint C [MT97]. Thus, the constraint C is a declarative specification of the desired properties on the searched patterns. A task of computing $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$ is called a task of solving an inductive query with the constraint C. As declarative queries are often formulated using constraints, inductive querying is closely related to *constraint-based data mining* [Bay02].

The promise of modeling data mining tasks as inductive queries is to identify simple primitive operations, such that through their combinations high expressiveness can be achieved. This was the case in the field of database systems, where Codd's relational algebra [Cod70] provides a set of simple primitives that allow to express highly complicated tasks. By unifying various data mining tasks under a single theoretical framework and braking them down into common primitives, research efforts can be concentrated to optimize the solvers of the primitives, instead of developing new solvers for each new task and application. The design of efficient solvers is particularly important, because in general, the pattern space $\mathcal{L}$, which is indeed the search space, is very large (and possibly infinite), and the collections of data can also be very large, what means that the cost of solving the constraints, whose evaluation needs to access the data, can be very high.

The complete scope of IDBs is still a research topic and will more likely depend on how all the known and future Data Mining techniques will manage to be incorporated together in a natural and practical way. In past years, significant progress on inductive databases was made within the cInQ project (**c**onsortium on knowledge discovery by **In**ductive **Q**ueries) funded by the EU under the FET open branch of IST [BDM06]. This thesis manuscript presents a work made within the (IQ) project (**I**nductive **Q**ueries for Mining Patterns and Models) which was a follow-up of cInQ funded by the FET open branch of IST as well. As stated in its Technical Annex, the overall goal of the IQ project was to develop a theoretical understanding of inductive querying by (1) developing the applications in the area of bioinformatics, and, (2) further developing the required theory, representations and primitives for local pattern and global model mining, and integrating these into expressive inductive query languages that enable one to discover new knowledge from data in real-life applications.

Generic Solvers

By a *generic solver* we mean a solver capable to evaluate the inductive queries that are arbitrary Boolean compositions of constraints, i.e., arbitrary expressions over constraints using Boolean algebra operations: conjunction, disjunction and negation. On the contrary, an *ad-hoc* solver is a solver, designed to evaluate a particular composition of constraints (e.g., a minimum frequency constraint, a conjunction of minimum frequency constraint with other application specific constraints) on a particular pattern space (e.g., itemset pattern space, string patterns).

The inductive database framework provides a basis to study the properties of different pattern languages, constraints and their different compositions, as well as different types of data sets. Thereby it provides a framework to design efficient generic algorithms that solve inductive queries. A key issue for designing efficient though correct and complete solvers is to exploit the structure of the pattern space, the structure of the solution set and the constraint properties to compute the solution set without exploring the whole search space. Extensive studies of the anti-monotonicity and monotonicity properties have given rise to a general theory [DJDM02, DD03] for mining quite general patterns (i.e., patterns from different pattern language), satisfying an inductive query with the constraint C that is a Boolean combination of anti-monotonic and monotonic primitive constraints. It is based on characterizing the solution set by means of version spaces [Hir91, Hir94, Mit82] and border representations [MT97]. More specifically, this theory concerns the decomposition of such constraint C into a set of sub-constraints C_i, such that $Th(\mathcal{L}, \mathcal{D}, C) = \bigcup_{i=1}^{k} Th(\mathcal{L}, \mathcal{D}, C_i)$, where k is minimal and each $Th(\mathcal{L}, \mathcal{D}, C_i)$ can be represented using a single version space, i.e., $C_i = \mathcal{A} \wedge \mathcal{M}$, where $\mathcal{A}$ denotes an anti-monotonic constraint and $\mathcal{M}$ denotes a monotonic constraint. Such solution set that is a union of version spaces is known as generalized version space [DD03]. This results in an operational procedure for solving arbitrary Boolean inductive queries, since once each C_i is computed, the resulting solution sets can be combined using the set manipulation operations to obtain the solution $Th(\mathcal{L}, \mathcal{D}, C)$ [DD03]. While the latter can be done efficiently, the step of computing $C_i = \mathcal{A} \wedge \mathcal{M}$ is not trivial. The general theoretical framework was instantiated to answer inductive queries on patterns from string pattern language [DJDM02]. One of the key elements of this instance is the algorithm `VST` to efficiently compute patterns, satisfying an arbitrary conjunction of anti-monotonic and monotonic constraints. The algorithm `FAVST` [DD04] efficiently solves the same problem in a distributed environment where a data set access is slow (e.g. over the Internet or an external disk). Our contribution to fault-tolerant pattern extraction is build on the generic solver `FAVST`.

Fault-Tolerance when Mining Strings

It is common that real life data contains errors due to technological issues concerning data collection, storage and transmission. In some application domains they may be also due to somewhat exploratory alphabet design. Also, data representing real world phenomenon is often intrinsically degenerated (in a sense of ability of elements that are structurally different to perform the same function or yield the same output [EG01]). For example, many variants of string patterns in DNA sequences can be binding sites for the same transcription factor or many web site browsing sequences can lead to the accomplishment of the same task. To capture knowledge when working with such data, a *fault-tolerance* is needed.

A motivation for fault-tolerance came from many application domains, the principal ones being *signal processing*, *error correction*, *text and information retrieval.* One of the largest application areas remains *computational biology.* Biological sequences can be considered as strings over specific alphabets, e.g., DNA sequences can be seen as strings over four letter nucleotide alphabet and protein sequences can be seen as strings over twenty letter amino acids alphabet. Exact matching is rarely convenient when analyzing biological sequences, since they typically contains errors due to sequencing technologies and, in addition to this, they are known to be intrinsically degenerated.

The approaches to the needed fault-tolerance when searching for regularities in strings come from different research domains, such as approximate string matching, bioinformatics and data mining. They are formulated and presented in a domain specific manner and terms. This presents a serious impediment to knowledge transfer and exchange, and prohibits the cross-fertilisation among these domains. Our contribution consists of formalizing the different approaches to fault-tolerance in the unified constraint-based mining terms thereby putting them into the inductive databases framework.

From a constraint-based data mining point of view, the notion of fault-tolerance when mining string data implicitly embraces two distinct problems. The first one is to find patterns that are *similar* to a given entity of reference. The second problem is to find unknown patterns that capture *soft regularities*, i.e., not exactly repeating regularities. This problem can be formulated as a problem of extracting patterns satisfying a minimum frequency constraint, when a pattern occurrences in data are the entities acknowledged to be *similar* to the pattern. Note that a *relation of similarity*, either between two patterns, or between a pattern and its occurrence in data, is present in the formulation of both problems and is central when handling the fault-tolerance.

We concentrate on *deterministic* approaches that employ correct and complete strategies to extract fault-tolerant patterns in string data, on the contrary to *probabilistic* approaches. Thus we do not cover non-deterministic approaches, such as Hidden Markov Models (HMMs), Bayesian networks, weight matrices, profiles, etc. A major part of considered approaches were inspired when trying to answer a particular class of biological problems, but their are not tied to any particular problem and can be directly applied to a variety of application domains.

Similarity and Soft-Frequency Constraints

The goal of this PhD work was to develop a *correct and complete generic solver* that is able to handle the fault-tolerance and thereby can be applied in biological sequence analysis. A state of art on the existing approaches to tackle the similarity constraint and the fault-tolerant patterns extraction is that efficient *ad-hoc solvers* are available to tackle specific combinations of primitive constraints. A key issue for designing efficient *generic solver* is to exploit the opportunities for search space pruning, associated to constraint properties (like anti-monotonicity and its dual monotonicity property) [DJDM02, DD03]. It is far more complex and generally not feasible to consider generic solvers for constraints that do not have these good (anti-)monotonicity properties.

In most of the application domains, the notion of similarity between two entities ε_1 and ε_2 informally means a "small difference" between ε_1 and ε_2. Obviously, the property "small difference" should not be propagated too far, i.e., the relation of similarity should not be transitive. A similarity constraint, establishing a non-transitive similarity relation between two entities, is fundamentally neither monotonic nor anti-monotonic, since a non-transitive similarity relation can not be isomorphic to a generalisation relation. Due to this property, a fault-tolerant pattern extraction can not benefit from recent algorithmic breakthrough in generic solver design.

Therefore, our approach, published in [MB06, MB07], is to formulate the similarity and soft-frequency constraints so that they can be handled by the state-of-the-art search space pruning strategies, i.e., as Boolean combinations of anti-monotonic and monotonic primitive constraints. We found such expression using the longest common subsequence (LCS) of two strings, which is the longest pairing of their matching symbols, allowing all possible interruptions in either of the strings. We formulate a similarity constraint between a string of reference σ and a candidate string pattern ϕ as a conjunction of two constraints having the desired (anti-)monotonicity properties: a string pattern ϕ is similar to a reference string σ, if its LCS with σ is large enough (a monotonic constraint) and if the number of deletions, necessary to perform on ϕ in order to obtain that LCS, is small enough (an anti-monotonic constraint). Concerning the soft-frequency constraints, notice that one needs to find all *soft-occurrences* of a pattern ϕ (i.e., all entities in data that are *similar* to a pattern ϕ) in order to evaluate its soft-frequency. This means to evaluate a similarity constraint where the pattern ϕ in question is a string of reference. To accomplish this we use our previous contribution on similarity constraint, expressed as a conjunction of two (anti-)monotonic constraints that bear on the LCS. Remark however, that even if we can locate the soft-occurrences of a pattern ϕ efficiently, the associated minimum (resp. maximum) soft-frequency constraint is not guaranteed to be anti-monotonic (resp. monotonic), since there is no guaranteed stable relation between the string pattern ϕ substrings and superstrings and the cardinality of the sets of their soft-occurrences (and thus the satisfaction of the soft-frequency constraint). Yet, interestingly, when using the defined similarity constraint to acknowledge a substring σ as a soft-occurrences of a pattern ϕ, the associated soft-frequency constraints are guaranteed to be (anti-)monotonic, provided that a (sensible) condition on similarity constraint parameters is satisfied.

Based on these contributions we designed and implemented the generic solvers

`Marguerite-`{`Sim`,`SoftFr`} that solve arbitrary combinations of similarity and soft-frequency constraints with other (anti)-monotonic constraints. These solvers are built on and extends the generic solver `FAVST` [DD04], which instantiates the efficient generic strategies [DJDM02, DD03] to mine string patterns in string data sets.

Twilight Zone

Fault-tolerance is a first step toward actionable patterns, when mining real-world data (i.e., containing errors) that represents (degenerated) phenomenons. However, it does not tell anything about the relevancy of the extracted patterns. Indeed, relevant patterns are quite often hidden amongst many irrelevant ones. The bottleneck of most local pattern extraction methods is the burden of the spurious patterns. Irrelevancy must be understood in terms of both objective and subjective interestingness. For instance, most of the time, it is important to avoid to provide or, even better, to compute the known patterns. Data miners cannot afford to present (hundred of) thousands of patterns to the data expert, hoping that she/he will find the gold nuggets. For example, it would be of great interest to know whether a given pattern or a collection of patterns is statistically unexpected w.r.t the mining task (i.e., input data set and constraint thresholds values). However, the constraints that rely on statistical measures are known to be difficult to push into the extraction phase. Specifically, they cannot be fully/easily described by means of combinations of monotonic and anti-monotonic constraints.

To try to solve this problem, we decided to assess local patterns relevancy adapting the principle of the Twilight Zone studied in [KP02b]: comparing the number of extracted patterns in the input data set with the expected number of patterns extracted in random data sets exhibiting the same features (i.e., structural properties of the data set, e.g., size of the data set, number of sequences, length of the sequences, etc) as the original data set. One difficulty here was to estimate the number of local patterns that satisfy a given constraint in a random data set, exhibiting some features. Estimating the expected number of patterns that satisfy a constraint is in general much more difficult than estimating the probability that a given pattern satisfies such a constraint. This second problem has received a lot of attention, on the contrary to the first one. The estimate of the number of extracted patterns might be difficult as soon as we have at hand the combinations of many primitive constraints, each of these constraints requiring at least one parameter (i.e., threshold) value. We have a limited insight about the number of patterns, satisfying the constraints, in the multidimensional parameter (threshold) space. For example, the number of patterns, satisfying a minimum frequency constraint with a given frequency threshold, depends on the pattern size (length), allowed in the pattern size (length) constraint. A common practice is to count the number of patterns obtained for a few different parameter settings and to guess what could be the interesting parameter values for a deeper investigation. In simple contexts, e.g., when considering a single minimum frequency constraint, a limited number of trials may be sufficient. This is obviously not the case when considering a conjunction of primitive constraints giving rise to a large multidimensional parameter space: we cannot afford to run hundreds or thousands of experiments to probe such a space.

We proposed two solutions to this difficult problem. First, we have studied the computation of an analytical estimate of the expected number of extracted patterns thanks to known features of the input data set [MRS$^+$08]. To simplify the task, we consider data strings of the same length and we suppose that they are composed of independent and uniformly distributed symbols (i.e., having the same occurrence probability), and that the overlapping of the occurrences of the patterns has a negligible impact on the number. From these assumptions, we derived analytical formulas to compute, given threshold values, the expected number of patterns satisfying frequency and soft-frequency constraints. At this end, we obtain the probability that a pattern of a given length satisfies a minimum and maximum (soft-)frequency constraints, according to chosen threshold values. The results of experimental validation are encouraging, yet the drawback is that for each constraint and each combination of constraints a new probability has to be computed (if possible). In other words, it works well for some particular constraints, but it cannot be considered as an ultimate solution within a generic framework. Next, we studied an alternative approach in [BRMB08]. We consider here pattern space sampling. Instead of computing the estimate on abstract patterns, we generate a sampling of patterns and then we compute their probability to satisfy the given constraint on random data sets, exhibiting the same features as the original data set. The use of real patterns ((instead of abstract ones) facilitates the computations and it allows to take into account a broader class of constraints. For instance, arbitrary conjunctive syntactic constraints can be used.

Genomic Data Analysis

Understanding the regulation mechanisms of the gene expression remains one of the major challenges in molecular biology. One of the elements, through which the regulation works, is the initiation of the transcription by the interaction between gene promoter elements at the level of DNA sequence and multiple activator and repressor proteins, called Transcription Factors (TF). This interaction occurs when a TF binds on its binding site on a gene promoter. Numerous efforts have given rise to a variety of computational methods to discover putative Transcription Factors Binding Sites (TFBSs) in sets of promoters of co-regulated genes. Among them two families can be distinguished: statistical or stochastic approaches, and combinatorial approaches [VMS99a]. Concerning the family of statistical and stochastic approaches, a recent review of the most widely used algorithms exhibits rather limited results [TLB+05], and concludes to the necessity to go on exploring alternative methods. There are several reasons for their limited success, but it seems that the difficulty to separate the patterns from the *random background* is among the principal ones.

We focused on the family of combinatorial approaches that aims at an exhaustive motif extraction without *a priori* hypothesis on the underlying stochastic process. According to [KP02a], probably the best tools for finding consensus based motifs in DNA sequences are the combinatorial pattern-driven algorithms that test all the 4^l different patterns of length l (DNA sequences are composed of 4 nucleotides: adenine, guanine, thymine, cytosine), and then score each pattern by the number of approximate occurrences and find the high-scoring patterns. The exhaustive search through all these 4^l patterns becomes impractical for large l, but the length of binding sites in promoter sequences is estimated to be between 5 and 15 base-pairs (bp) [Bul03] and the mean of these lengths in $Transfac^{®}$ [MFG+03] is 14.3 bp with standard deviation 4.7 bp [FWV+05]. Thus, combinatorial methods are of great promise and it is worth to test our approach to fault-tolerant string pattern extraction under constraints for the discovery of the putative TFBS.

Having in mind the difficulties to model statistically the biological randomness, we propose to postpone the phase of significant pattern selection, based on a statistical measure, and to use beforehand the supplementary biological information to constraint the search and thereby reduce the number of extracted patterns. This additional information comes in the form of a second data set representing a somehow *opposite biological situation*. To collect this information, the method starts with a classical operation used in molecular biology: the identification for differentially expressed genes. This allows to obtain two groups of genes, from which one can derive two opposite data sets, composed of the promoters of these genes. To look for the putative TFBSs regulating the overexpressed genes, we choose the first set (the promoters of the over-expressed genes) to be used as a positive set, and the second set as a negative one. Then our method consists in finding the patterns occurring on at least $minFr$ promoters from the positive set and on at most $maxFr$ promoters from the negative set, where the parameter $minFr$ (resp. $maxFr$) is supposed to be a large (resp. small) frequency constraint threshold value. The originality of the proposed method w.r.t the other combinatorial algorithms, which allow to extract patterns from several data sets (e.g., SPEXS [BJVU98b] or DRIM [ELYY07]), is that the maximal frequency threshold is set explicitly. This is particularly interesting, when there is a clear semantic cut between positive and negative data sets, and the negative data set has an opposite biological sense (presence/absence of a mutation; addition or not of a given drug, etc.), and does not just represent random background. Two kinds of patterns are handled by our method: patterns having exact matches in the sequences and patterns having approximate matches (i.e., within a given Hamming distance). Interestingly, in both cases, the enrichment of the pattern discovery context, using a negative data set, reduces the size of the solution set by several orders of magnitude. Even then, the set of the extracted patterns remains large, and thus we develop an approach to tune the parameters to focus on a manageable and potentially interesting set of patterns. We select the exceptional patterns based on the measure of *subtlety* [KP02b]: a pattern ϕ is considered to be subtle (i.e., inside the Twilight Zone) if we expect that some random patterns could occur at least as often as ϕ in the positive data set and at the same time no more often than ϕ in the negative data set. In other terms, an exceptional pattern must *not* be subtle, i.e., inside the Twilight Zone. Then, we verify which exceptional patterns are known TFBSs. Identification of the TFs that can bind on the patterns specific to the positive data set can help to discover new regulators of the concerned biological process. Patterns that do not correspond to known TFBSs are of course interesting since they can reveal unknown elements of regulation.

The method was developed in collaboration with the biologist group BM2A (CNRS UMR 5534 in the Center for Molecular and Cellular Genetics), supervised by Dr. Olivier Gandrillon. This group work on molecular basis of self-renewal, which is a characteristic property of stem cells. Deregulation of this process occurs frequently during cancer generation. We applied the previously described method to analyze two different sets of promoters. The first set consists of the promoters of the genes that are either repressed or not by the transforming form of the v-erbA oncogene. The second set consists of the genes, such that their expression varies between self-renewing and differentiating progenitors. The biological meaning of the found TFBSs was assessed by the BM2A group, and, for one TF, its biological involvement is demonstrated. In addition to this, the results of this study also provided our biologist collaborators with the new hypothesis and insights in the self-regulation mechanism. This work is published in [MRS$^+$08]. A survey paper emphasizing the inductive querying scenario that supports such a discovery process is [RMB$^+$08].

We started to study a second application domain related to genomic data analysis. Comparative genomics can be seen as an attempt to take advantage of the information provided by the signatures of selection to understand the function and evolutionary processes that act on genomes. One of the comparative genomics tools is to identify the common parts of genomes of the different biological species. These common parts can be found by conjunctive correct and complete fault-tolerant pattern extraction in the genomes in question. We considered the extraction of frequent *InsDels* fault-tolerant patterns[2] that are common to the four species of yeasts: *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus*. To extract such patterns, we applied our generic solver `Marguerite-SoftFr`. This opening to an application to comparative genomics was carried out thanks to a collaboration with Prof. Ross D. King (Department of Computer Science, University of Aberystwyth) in the context of the IQ project.

[2]See Section 5.3.2 in Page 105.

Structure of the Manuscript

We now sum up the structure of this manuscript. The first part presents the inductive databases framework, a state-of-the-art on constraint-based pattern mining, recent advances on generic inductive query solvers, and finally an original survey of proposals for a declarative specification of fault-tolerant pattern mining tasks. Part 2 is dedicated to our research contribution. First, it describes our results on a decomposition of similarity and soft-frequency constraints to handle fault-tolerance. Then, it is concerned by our proposals for probabilistic approaches that can assess pattern relevancy thanks to the Twilight Zone idea. Part 3 describes the application of the aforesaid methods on real data sets for the discovery of new transcription factor binding sites and an opening to application to comparative genomics. Last Part concludes and opens on the future research directions.

We assume the reader knows the classical machine learning and data mining techniques which are well described in popular textbooks, for instance [HMS01a, TSK06, BH03].

Part I

State of Art

Chapter 1

Inductive Databases and Inductive Queries

Numerous effort put in data mining research during recent years have resulted in its rapid and successful development. However, the generally accepted theoretical framework for data mining is still lacking and remains a major research priority. One of the most promising approach to this task is taken by inductive databases. Many knowledge discovery from data processes can be based on data mining tasks which extract *patterns* from *data*. We thus first provide the definitions that are useful to formalize many data mining problems, and then introduce the underlying framework of inductive databases.

1.1 Objects and Data

Definition 1.1 (Object, Universe) *An object, denoted X, is an element of a given set, denoted $\mathcal{U}$ and called universe.*

We do not precise here what constitutes the universe. The general framework that we will present is suitable for a wide range of objects of different nature, and the particular universe can be specialized for each application.

Object is an important term because it is a basis for defining the notion of *frequency* (see Definition 1.19 in Page 21), which is employed to a great extent in data mining.

In this thesis we will focus on objects from the universe of *strings*.

Definition 1.2 (Basic Notions on Strings) *Let Σ be an alphabet of symbols. Any finite sequence $\sigma = \sigma_1 \dots \sigma_n$, $n \in \mathbb{N}$, of symbols from Σ is called a string*[1] *over the alphabet Σ. The length of a string σ, denoted $|\sigma|$, is a number of symbols it contains. An individual symbol on the position i in a string σ is denoted by σ_i. The set of all possible strings over the alphabet Σ is denoted by Σ^*. A substring σ' of a string σ is a sequence of contiguous symbols in σ, denoted $\sigma' \sqsubseteq \sigma$. Then σ is a superstring of σ', denoted $\sigma \sqsupseteq \sigma'$.*

Example 1.1 *Let $\Sigma = \{a, c, g, t\}$ be an alphabet.* acctg *is an example of string over Σ. Its length is* $|\mathsf{acctg}| = 5$*.* acc *is an example of a substring of* acctg*, denoted* $\mathsf{acc} \sqsubseteq \mathsf{acctg}$*.*

Definition 1.3 (String Universe) *Let Σ be an alphabet. Consider the universe of all strings constructed from the symbols of Σ, denoted $\mathcal{U}_\Sigma$. An object $X \in \mathcal{U}_\Sigma$ is a string over Σ.*

Definition 1.4 (Data) *Data on the universe $\mathcal{U}$, denoted $\mathcal{D}$, is a collection of objects X, organized in a specific way.*

Data can be organized in the various forms, e.g., a multi-set[2], a sequence of events, a stream, a graph, etc. In this manuscript we focus on mining *string data*. It can be organized into a multi-set of strings, called *string data set*, or consist of one string, called *data string*.

Depending on the application context, it is convenient to see such data organized either as a multi-set of *string objects* or as a collection of *substring objects*.

Definition 1.5 (String Object) *Let $\mathcal{D}$ be string data. String objects, denoted S, are elements of that set.*

Example 1.2 *Consider the alphabet of four symbols $\Sigma_4 = \{a, c, g, t\}$ and the universe $\mathcal{U}_{\Sigma_4}$ of all strings over Σ_4. Consider a string data set* $\mathcal{D} = \{\mathsf{agtac}, \mathsf{aaac}, \mathsf{aaac}\}$ *on $\mathcal{U}_\Sigma$. There are three string objects in $\mathcal{D}$:* $S_1 = \mathsf{agtac}$, $S_2 = \mathsf{aaac}$, *and* $S_3 = \mathsf{aaac}$*.*

Definition 1.6 (Substring Object) *Let $\mathcal{D}$ be a string data. Substring objects, denoted s, are substrings of the strings that compose that set. Note, that they are stored intentionally in $\mathcal{D}$.*

[1]Also known as a *sequence* or a *word* in a literature

[2]Data may contain multiple instances of the same object

Example 1.3 *Consider the alphabet Σ_4, the universe $\mathcal{U}_{\Sigma_4}$ and the string data set $\mathcal{D}$ from Example 1.2. $s_1 =$ ag, $s_2 =$ agt, $s_3 =$ tac, $s_4 =$ aaa are several examples of substring objects in $\mathcal{D}$.*

Remark 1.1 *Note that string and substring objects are elements of the same universe $\mathcal{U}_\Sigma$.*

1.2 Patterns

The notion of pattern is central to data mining, since they are what data mining aims to discover. As an object, in a high level of abstraction, a pattern is an element of a pre-specified set.

Definition 1.7 (Pattern, Pattern Language) *A pattern ϕ is an element of a given set $\mathcal{L}$, which is called a pattern space, a search space or a pattern language. In this manuscript, we use the term pattern space or search space when referring to the set of all possible patterns (see Section 2.1 in Page 29), otherwise, especially when referring to the syntactic properties and expressiveness of the patterns, we use the term pattern language.*

In the following we define several pattern languages that will be used throughout this manuscript. Our thesis focuses on string patterns, and therefore we first define the string pattern language.

Definition 1.8 (String Pattern Language) *Let Σ be an alphabet. String pattern language, denoted $\mathcal{L}_\Sigma$, is the set of all possible strings Σ^* over the alphabet Σ.*

This thesis concerns mainly string patterns but we will also discuss concepts that have been studied in the popular context of 0/1 data analysis, e.g., the famous pattern language of *itemsets* which has been extensively studied by the data mining community. The pattern space of itemsets can be arranged in a particularly advantageous manner to exploit the resulting lattice structure (see Example 2.5 in Page 33).

Definition 1.9 (Basic Notions on Itemsets) *Let i be a literal, called item. Let $I = \{i_1, \ldots, i_n\}$, $n \in \mathbb{N}$, be a set of items, called domain of items. An itemset $\mathcal{I}$ is a subset of I. The size of an itemset $\mathcal{I}$, denoted $|\mathcal{I}|$, is the number of items it contains. A powerset of I, i.e., a set of all its subsets, is denoted 2^I.*

Example 1.4 *Consider an item domain $I = \{A, B, C, D\}$. $\mathcal{I}_1 = \{A, C\}$, $\mathcal{I}_2 = \{A, B, C, D\}$ are several examples of itemsets on I.*

Definition 1.10 (Itemset Pattern Language) *Let I be a domain of items. Itemset pattern language, denoted $\mathcal{L}_I$, is a powerset of I.*

String data mining is one of the domains of sequential data mining research. Sequence patterns were extensively studied since their introduction in [AS95]

Definition 1.11 (Basic Notions on Sequences) *A sequence is an ordered list of elements. These elements can be either simple items from a fixed set of literals (i.e., alphabet Σ or domain of items I), or itemsets on I. Sequence is denoted $\Phi = \langle \mathcal{I}_1, \ldots, \mathcal{I}_n \rangle$, $n \in \mathbb{N}$, where $\mathcal{I}_i$ is the i-th element of the sequence. The length of a sequence, denoted $|\Phi|$, is the number of elements it contains. Consider another sequence $\Psi = \langle \mathcal{Y}_1, \ldots, \mathcal{Y}_m \rangle$, $m \leq n$, composed of elements $\mathcal{Y}_i$. Ψ is a subsequence of Φ, if there exists integers $1 \leq j_1 \leq \ldots \leq j_m \leq n$ such that $\mathcal{Y}_1 \subseteq \mathcal{I}_{j_1}, \ldots, \mathcal{Y}_m \subseteq \mathcal{I}_{j_m}$. Then, Φ is a supersequence of Ψ.*

Definition 1.12 (Sequence Pattern Language) *Let I be a domain of items. Sequence pattern language, denoted $\mathcal{L}_{\mathcal{S}}$, is a set of all possible sequences that can be constructed over I.*

Example 1.5 *Let $I_4 = \{A, B, C, D\}$ be domain of items. $\Phi = \langle \{C\}\{A, D\}\{B, C\} \rangle$, $\Psi = \langle \{C\}\{B\} \rangle$ and $\Xi = \langle \{C\}\{A, C\}\{B\} \rangle$ are several examples of sequences over I_4. Ψ is an subsequence of Φ, but Ξ is not.*

Remark 1.2 *Note that a string is also a sequence. Therefore, sequence pattern mining algorithms can be often applied to mine string data. Remark, however, that sequence is not a string, and a subsequence is not a substring.*

Patterns, describing the properties of a subset of a data are called *local patterns*, and patterns characterizing the entire data are called *global patterns* or *global models*. Local patterns, e.g., itemsets, association rules, formal concepts, functional or inclusions dependencies, datalog queries, substrings, episodes, episode rules, subgraphs, etc., are typically used for descriptive purposes, whereas global patterns or models, e.g., clusters, decision trees, support vector machines, etc., are also used to support predictive ones. For more details concerning local patterns versus global patterns and global models see [HMS01b, Han02, MBS05, BMS07]. In this thesis we consider only local patterns, and whenever there is no ambiguity, we use the term *pattern* to denote *local pattern*.

A relation between patterns and objects is established by a *match function*.

Definition 1.13 (Match Function) *Given a universe $\mathcal{U}$ and a pattern language $\mathcal{L}$, a match function is a Boolean function: $\mathcal{U} \times \mathcal{L} \rightarrow \{\mathsf{true}, \mathsf{false}\}$. Given a pattern $\phi \in \mathcal{L}$*

and an object $X \in \mathcal{U}$, we say that ϕ matches X if and only if $match(\phi, X) =$ true, otherwise we say that ϕ does not match X.

String data can be considered as a collection of string objects or a collection of substring objects. Consequently, depending on the desired semantics of patterns, through the match function they can be associated either to string or to substring objects. Match function is defined through some binary relation R. We will say that a pattern ϕ match a string object S, if and only if S contains a substring σ that is in relation R with ϕ. We will also say that a pattern ϕ match a substring object s, if and only if s is in relation R with ϕ. Then, we call the substring σ (in the case of match with a string object) and the substring object s the *occurrences* of the pattern ϕ.

Following is the exact string patterns match function on string and substring objects, defined through an identity relation.

Definition 1.14 (Exact String Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, S be a string object and s be a substring object from $\mathcal{U}_\Sigma$. Let ϕ be a pattern from a string pattern language $\mathcal{L}_\Sigma$.*

1. *Define an exact string match function for string patterns $match_{\sqsubseteq,S}(\phi, S)$ to evaluate* true*, if and only if there exists a string σ, such that $\sigma \sqsubseteq S$ and σ is equal to ϕ.*
2. *Define an exact match function for string patterns $match_{\sqsubseteq,s}(\phi, s)$ to evaluate* true*, if and only if s is equal to ϕ.*

Example 1.6 *Consider the alphabet Σ_4, the universe $\mathcal{U}_{\Sigma_4}$ and the string data set $\mathcal{D}$ from Example 1.2 in Page 16. Consider the string pattern language $\mathcal{L}_{\Sigma_4}$ over Σ_4 and the exact string match function. The pattern $\phi =$ ag exactly matches the string object $S_1 =$ agtac, but it does not match the string object $S_2 =$ aaac. Also, the pattern $\phi =$ ag exactly matches the substring object $s_1 =$ ag, but it does not match the substring object $s_2 =$ agt from Example 1.3 in Page 17.*

In Example 1.6, we have $\mathcal{L}_{\Sigma_4} = \mathcal{U}_{\Sigma_4}$. There is no requirement that a pattern space $\mathcal{L}$ be equal to a universe $\mathcal{U}$. In the following we give two examples of cases, where pattern languages are different from the universe.

The first example considers a pattern language that is a subset of $\mathcal{L}_{\Sigma_4}$.

Example 1.7 *Let $\Sigma_2 = \{a, g\}$ be the alphabet and $\mathcal{L}_{\Sigma_2}$ be a string pattern lnaguage over Σ_2. Consider the string data set $\mathcal{D}$ on the universe $\mathcal{U}_{\Sigma_4}$ from Example 1.2* *in Page 16. The only patterns from $\mathcal{L}_{\Sigma_2}$ that match string objects in $\mathcal{D}$ are* a*,* aa*,* aaa*,* g*,* ag*. Note that the pattern language $\mathcal{L}_{\Sigma_2}$ is more restrictive than $\mathcal{L}_{\Sigma_4}$ from Example 1.6.*

The second example considers regular expression pattern language, which is a superset of $\mathcal{L}_{\Sigma_4}$.

Definition 1.15 (Regular Expression) *Regular expression, denoted re, is an expression over a given alphabet Σ using the well established set of regular expression operators [LP81], e.g., the implicit concatenation operator, the disjunction denoted by the sign* | *, or the grouping, denoted by the parentheses* () *.*

Definition 1.16 (Regular Expression Pattern Language) *Let Σ be an alphabet. Regular expression pattern language, denoted $\mathcal{L}_{re}$, is a set of all possible regular expressions over the alphabet Σ.*

Regular expression *match* function is defined through a classical regular expression matching.

Definition 1.17 (Regular Expression Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, S be a string object and s be a substring object from $\mathcal{U}_\Sigma$. Let ϕ be a pattern from regular expression pattern language $\mathcal{L}_{re}$.*

1. *Define a regular match function $match_{re,S}(\phi, S)$ to evaluate* true*, if and only if there exists a string σ, such that $\sigma \sqsubseteq S$ and σ belongs to the language generated by a regular expression ϕ.*
2. *Define a regular match function $match_{re,s}(\phi, s)$ to evaluate* true*, if and only if s belongs to the language generated by a regular expression ϕ.*

Example 1.8 *Consider the alphabet Σ_4 and the string data set $\mathcal{D}$ from Example 1.2 in Page 16. Let $\mathcal{L}_{re_4}$ be a regular expression pattern language over the alphabet Σ_4. Pattern* ag(t|c) *from $\mathcal{L}_{re_4}$ matches the string object* agtac *and the substring object* agt*. Note that the pattern language $\mathcal{L}_{re_4}$ is more expressive than the pattern language $\mathcal{L}_{\Sigma_4}$ from Example 1.6.*

1.3 Constraints

Data mining aims to discover patterns that are *interesting*. We specify the a priori interesting patterns by declaring what properties, or in other words, what constraints these patterns must satisfy.

Definition 1.18 (Constraint) *Given a pattern language $\mathcal{L}$, a constraint C is a Boolean function: $\mathcal{L} \rightarrow \{\mathsf{true}, \mathsf{false}\}$. We say that a pattern $\phi \in \mathcal{L}$ satisfies a constraint C if and only if $C_\phi = \mathsf{true}$. Constraints are usually parametrised. A constraint C on a pattern ϕ with parameters $p_1, p_2, \ldots, p_n$, $n \in \mathbb{N}$, is denoted $C_\phi(p_1, p_2, \ldots, p_n)$.*

Constraints allow not only to select interesting patterns, but also to prune the pattern space $\mathcal{L}$, and thereby render the pattern extraction more efficient. When a constraint is exploited to prune a search space, we say that it is *pushed.* Depending on the extraction phase, in which a constraint is exploited for pruning, we say that it is pushed more or less *deeply.* If the search space is small (this is usually a case when it is already restricted to a particular collection of patterns), a constraint can be evaluated by testing it for every pattern in that space. In that case, we say that a constraint is *post-processed.*

In the following we give several examples to illustrate the notion of constraint. We first define the constraint of frequency, which, due its remarkable pruning efficiency and semantic capacity to select the interesting patterns, is very important in data mining.

Definition 1.19 (Frequency) *Given a pattern ϕ and data $\mathcal{D}$, the frequency of ϕ in $\mathcal{D}$ is the number of objects that ϕ matches :*

$$\mathsf{Fr}(\phi, \mathcal{D}) = |\{X \in \mathcal{D} \mid match(\phi, X)\}|$$

Definition 1.20 (Minimum/Maximum Frequency Constraint) *Given data $\mathcal{D}$, a pattern $\phi \in \mathcal{L}$ and the thresholds $minFr, maxFr \in \mathbb{N}$, the minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ evaluates to true if and only if $\mathsf{Fr}(\phi, \mathcal{D}) \geq minFr$. Similarly, the maximum frequency constraint $\mathsf{MaxFr}_\phi(maxFr, \mathcal{D})$ evaluates to true if and only if $\mathsf{Fr}(\phi, \mathcal{D}) \leq maxFr$. When referring to both minimum and maximum frequency constraints, we abbreviate and call them frequency constraints.*

Definition 1.21 (Exact String Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, S be a string object and s be a substring object from $\mathcal{U}_\Sigma$. Let ϕ be a pattern from a string pattern language $\mathcal{L}_\Sigma$.*

1. *Define an exact string match function for string patterns $match_{\sqsubseteq,S}(\phi, S)$ to evaluate true, if and only if there exists a string σ, such that $\sigma \sqsubseteq S$ and σ is equal to ϕ.*

2. *Define an exact match function for string patterns $match_{\sqsubseteq,s}(\phi, s)$ to evaluate true, if and only if s is equal to ϕ.*

Definition 1.22 (Exact-frequency) *Frequency that is computed using the exact string match function* $match_{\sqsubseteq,\{S,s\}}(\phi,\{S,s\})$ *is called exact-frequency, denoted* $\mathsf{Fr}_{\sqsubseteq,\{S,s\}}(\phi,\mathcal{D})$*. One of the contribution of this thesis concerns a soft-frequency. Whenever it is clear from the context, which one is considered, we will abbreviate and refer to the exact-frequency as frequency.*

Definition 1.23 (Minimum/Maximum Exact-Frequency) *Minimum (resp. maximum) frequency constraint, which bears on exact-frequency* $\mathsf{Fr}_{\sqsubseteq,\{S,s\}}(\phi,\mathcal{D})$ *is called minimum (resp. maximum) exact-frequency constraint, denoted* $\mathsf{MinFr}_{\phi}^{\sqsubseteq,\{S,s\}}(minFr,\mathcal{D})$ *(resp.* $\mathsf{MaxFr}_{\phi}^{\sqsubseteq,\{S,s\}}(minFr,\mathcal{D})$*). One of the contribution of this thesis concerns a minimum/maximum soft-frequency constraints. Whenever it's clear from the context, which one is considered, we will abbreviate and refer to the minimum/maximum exact-frequency constraints as minimum/maximum frequency constraints.*

Example 1.9 *Let* Σ_4 *be an alphabet* $\{a,c,g,t\}$*,* $\mathcal{L}_{\Sigma_4}$ *be a string pattern language on* Σ_4 *and* $\mathcal{D}$ *be a string data set from Example 1.2 in Page 16. Then, with the exact string match function on string objects,* $\mathsf{Fr}_{\sqsubseteq,S}(\mathsf{tac},\mathcal{D})=1$*,* $\mathsf{Fr}_{\sqsubseteq,S}(\mathsf{aa},\mathcal{D})=2$*,* $\mathsf{Fr}_{\sqsubseteq,S}(\mathsf{ac},\mathcal{D})=3$*, and* $\mathsf{MinFr}_{\mathsf{aa}}^{\sqsubseteq,S}(2,\mathcal{D})$ $\mathsf{MinFr}_{\mathsf{ac}}^{\sqsubseteq,S}(2,\mathcal{D})$ $\mathsf{MaxFr}_{\mathsf{tac}}^{\sqsubseteq,S}(1,\mathcal{D})$ *are several examples of satisfied minimum and maximum frequency constraints. Similarly, with the exact string match function on substring objects,* $\mathsf{Fr}_{\sqsubseteq,s}(\mathsf{tac},\mathcal{D})=1$*,* $\mathsf{Fr}_{\sqsubseteq,s}(\mathsf{aa},\mathcal{D})=4$*,* $\mathsf{Fr}_{\sqsubseteq,s}(\mathsf{ac},\mathcal{D})=3$*, and* $\mathsf{MinFr}_{\mathsf{aa}}^{\sqsubseteq,s}(3,\mathcal{D})$*,* $\mathsf{MaxFr}_{\mathsf{tac}}^{\sqsubseteq,s}(2,\mathcal{D})$ *are examples of satisfied constraints.*

Following are several examples of so called *syntactic* constraints (see Section 2.2.1.2), that allows to delimit the form of putatively interesting patterns.

Definition 1.24 (Minimum/Maximum Length) *Let* $\mathcal{L}$ *be a pattern language,* ϕ *be a pattern from* $\mathcal{L}$*, and* $l\in\mathbb{N}$ *be a threshold. Let* $|\phi|$ *denote a length of a pattern* ϕ*. Define a minimum length constraint* $\mathsf{MinLength}_{\phi}(l)$ *to evaluate* **true** *if and only if* $|\phi|\geq l$*. Similarly, define a maximum length constraint* $\mathsf{MaxLength}_{\phi}(l)$ *to evaluate* **true** *if and only if* $|\phi|\leq l$*.*

Example 1.10 *Let* Σ_4 *be an alphabet* $\{a,c,g,t\}$*. Consider a string pattern language* $\mathcal{L}_{\Sigma_4}$ *on* Σ_4*. Then,* $|\mathsf{c}|=1$*,* $|\mathsf{agat}|=4$*, and* $\mathsf{MinLength}_{\mathsf{agat}}(4)$*,* $\mathsf{MaxLength}_{\mathsf{c}}(3)$ *are examples of satisfied minimum and maximum length constraints.*

Definition 1.25 (Regular Expression Constraint) *Let* $\mathcal{L}$ *be a string or sequence pattern language,* ϕ *be a pattern from* $\mathcal{L}$ *and* re *be a regular expression (see Definition 1.15 in Page 20). Define a regular expression constraint* $\mathsf{MatchRE}_{\phi}(re)$ *to evaluate* **true** *if and only if the pattern* ϕ *belongs to the language generated by the regular expression* re*.*

Remark 1.3 *Note that a regular expression constraint $\mathsf{MatchRE}_{\phi}(re)$ allows to specify a subset of string language or, equivalently, a regular family of sequence patterns.*

Example 1.11 *Let Σ_4 be an alphabet $\{a, c, g, t\}$, and $\mathcal{L}_{\Sigma_4}$ be a string pattern language on Σ_4. Consider a regular expression `aa?g(t|c)`. Then $\mathsf{MatchRE}_{agc}($`aa?g(t|c)`$)$ is an example of satisfied and $\mathsf{MatchRE}_{aggt}($`aa?g(t|c)`$)$ is an example of not satisfied regular expression constraint.*

1.4 Inductive Databases

In recent years, the data mining research community put considerable effort to develop efficient algorithms to extract various types of patterns in different kinds of large collections of data. However, the data mining step in the Knowledge Discovery in Database (KDD) process [PSF91] cannot be limited to the application of a given algorithm to produce a collection of local patterns or to construct a global model. In real life applications, the process of knowledge discovery in data is intrinsically interactive and iterative: it needs for many related extractions that treat not only different kinds of data, but also different pattern and model types. For instance, consider the following sequence of tasks to find, among genes and biological situations, some subsets of genes that are generally over-expressed in some subsets of situations [BPB$^+$07]:

- Collect the gene expression profiles[3] over several biological situations.
- Discretize the gene expression profiles in the form of a Boolean matrix, where genes are the columns and situation are the rows, so that a cell in a matrix contains a 1 if the corresponding gene is over-expressed in a corresponding situation, or a 0 if not. This matrix is called the Boolean expression matrix.
- Extract the formal concepts which hold in such a data [Wil82] (i.e., the largest sets of over-expressed genes associated to the largest set of biological situations, in which their over-expression is observed) and such that minimum size constraints are satisfied.
- Perform a hierarchical clustering of the extracted formal concepts.
- Select a cluster (i.e., a node in the hierarchy).
- To assess the strength and the pertinence of the association, visualize the frequencies of the genes/situations associations in this cluster.

[3] In the field of molecular biology, a gene expression profile is a measure of the activity of thousands of genes at once, creating a global picture of cellular activity.

- Continue the exploration by selecting different clusters.

To study and support such knowledge discovery processes, the framework of *inductive databases* (IDB) was suggested in [IM96]. A early and simple formalisation was then proposed in [BKM98, BKM99, MT97]. An IDB instance contains data and patterns. The underlying idea of IDB is that the tasks of selecting, manipulating and querying data and patterns can be considered as queries, and consequently the whole KDD process can be considered as a process of querying. The queries on data and sets of already extracted patterns can be formulated in Structured Query Language (SQL) and handled by a Database Management System (DBMS). Concerning patterns, the idea is to see an extraction task as an *inductive query* that declaratively specifies the properties of patterns we are looking for in a (possibly not materialized, i.e., stored intentionally) pattern space $\mathcal{L}$. The *declarative* specification of an extraction task instead of the ad-hoc *procedural* constructs is appealing, because it holds the promise of the discovery of an algebra for data mining. The actual situation in data mining has much in common with the situation in databases before the discovery of Codd's relational algebra in 1970, when one disposed of many ad hoc proposals, each adapted to a particular application domain. The discovery of IDB query language that conceptually integrates the querying mechanisms and the primitives for treating simultaneously the data and patterns would allow to formalise the KDD processes in the form of sequences of queries that satisfy a closure property: every query takes and produces an instance of inductive database. Thus it would be an equivalent to relational algebra in databases theory for the extended framework of knowledge discovery. However, the discovery of such language remains a long term goal. Several specialized query languages have been proposed and implemented, e.g., `DMQL` [HFW$^+$96], `MSQL` [IV99], `MINE RULE` [MPC96, MPC98], `XMine` [BCKL02] (see also [BBMM04, BM05, Mas05] for a state of art). These data mining query languages have been studied and compared during the first EU funded project on inductive databases, namely the cInQ project. The perspective of an abstract inductive query language in IDB framework is proposed in [Rae02] and it has been developed further within the IQ project. Indeed, in this project, a couple of inductive query languages have been proposed like `IQL` [NR07] or Mining Views [BCF$^+$08].

1.5 Inductive Queries

A pattern extraction task can be formalized as the computation of $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$, i.e., finding patterns $\phi \in \mathcal{L}$ such that ϕ satisfies the constraint C [MT97]. The constraint C appears as the declarative specification of the subjective interestingness for pattern relevancy. We say that computing $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$ requires to solve the inductive query under the constraint C and this is called *constraint-based data mining*. The promise of modeling data mining tasks as inductive queries is to identify simple primitive operations, such that, through their combinations, high expressiveness can be achieved. This was the case in the field of database systems, where Codd's relational algebra [Cod70] provides a set of simple primitives that allow to express highly complicated tasks. By unifying various data mining tasks under a single theoretical framework and braking them down into common primitives, research efforts can be concentrated to optimize the solvers of the primitives, instead of developing new solvers for each new task and application. A design of efficient solvers is particularly important, because in general, the pattern space $\mathcal{L}$, which is indeed the search space, is very large (and possibly infinite), and the collections of data can also be very large, what means that the cost of solving the constraints, whose evaluation needs to access the data, can be very high.

Not all inductive queries can be evaluated. In the following we describe some typical bottlenecks and known solutions to tackle them.

Restricting an Inductive Query Consider a domain of items I_{20000} containing 20 000 items, e.g., a collection of French articles in Wikipedia[4] at the end of 2003. Let $\mathcal{L}_{I_{20000}}$ denote an itemset pattern language over I_{20000}. Then, the inductive query that asks for all the patterns $\phi \in \mathcal{L}_{\mathcal{I}_{20000}}$ containing 15 items can not be solved, because there are about 10^{60} of them, what is far too many[5]. Such an inductive query is not selective enough, and it is hard to imagine the possible optimisations to solve it, or to compute an approximation of the asked collection of patterns. A classical approach is to render the evaluation feasible by specifying a more selective inductive query, e.g., given data on Wikipedia usage logs, by adding an additional constraint on minimum number of users that have consulted the demanded subsets of articles, i.e., a minimum frequency constraint (see Definition 1.20 in Page 21).

Constraint Relaxation If an inductive query can not be solved efficiently, because the constraint C does not have good properties that enable to employ clever pattern space $\mathcal{L}$ pruning strategies, a solution can be to apply constraint C relaxation techniques. Note that, somewhat surprisingly, this is indeed the contrary to the previous approach. The idea is that having found a constraint C' that is less selective than C (i.e., $Th(\mathcal{L}, \mathcal{D}, C) \subset Th(\mathcal{L}, \mathcal{D}, C')$) and that have the desired properties enabling to solve it efficiently, we can compute the set $Th(\mathcal{L}, \mathcal{D}, C')$ and then post-process that set to retain only the patterns that satisfy C. SPIRIT algorithms family [GRS99] is a famous example of regular expression constraint relaxation when mining sequences under a minimum frequency constraint and a regular expression constraint (see Definition 1.25 in Page 22).

[4]Wikipedia (http://www.wikipedia.org/) is a multilingual, web-based, free content encyclopedia project.

[5]To compare, there are 10^{50} atoms in the Earth and 10^{57} atoms in the Solar system.

Query Execution Plan Optimisation The efficiency of solving an inductive query that is a Boolean composition of several constraints generally depends on the order in which these constraints are pushed, and how deeply they are pushed into the extraction phase (for an explication, what we mean by *pushing* the constraints, see Page 21). A classical approach is to base on constraints properties (see Section 2.2.2 in Page 36) to choose the appropriate pushing strategy, e.g., the different SPIRIT algorithms, depending on the regular expression constraint relaxation C', enforce either the pruning by C' or by minimum frequency constraint. The efficiency of a particular SPIRIT algorithm depends on the selectivity of a given C'. Yet, the selectivity of the constraint is not known *a priori* and depends on data. The algorithm RE-Hackle [AB03] tackles this problem by applying dynamical pruning strategy: the tradeoff between the minimum frequency constraint and the relaxed regular expression constraint C' is evaluated during the extraction phase, and then the appropriate constraint is pushed for pruning. To find an optimal query evaluation plan is indeed a typical tasks of IDB management system. It is an optimisation problem, similar to the query evaluation problems in relational databases. This problem becomes particularly interesting when considering interactive querying sessions, which are intended to be common when working with IDB. During such sessions, a user formulates rough query to get an insight what patterns hold in his/her data. Then, based on the obtained results, he/she refines the query. The querying process iterates, until the satisfactory result is obtained. Incremental techniques can be designed for a clever reuse of previous query evaluations (see, e.g., several chapters from [MLK04] and [BDM06].

Local Optimisation Consider a set of items I. Let pattern language $\mathcal{L}_{part}$ be the partitions of I into disjoint subsets. The inductive query that asks a $\phi \in \mathcal{L}_{part}$ such that the intra-subset distance is minimal, becomes impossible to solve when the number of items in I reaches 10. In this case, local optimisation techniques employed by, e.g., K-Means or hierarchical clustering algorithms, allows to compute the partitions that are expected to be close to the optimal ones.

Remark that there is no mean to specify precisely the quality of the patterns extracted using the local optimisation techniques. In the following, the approaches, when extracted patterns do not satisfy exactly the constraints of an inductive query or when not all patterns satisfying the constraint are extracted, are denoted *heuristic* approaches[6]. On the contrary, when the constraints of an inductive query are selective enough or they have good properties enabling to efficiently prune the pattern space, the approaches to extract all the patterns that satisfy the constraints, denoted *correct and complete* approaches[7], can be designed.

[6]Heuristic algorithm takes shortcuts when exploring a search space, leading to the loss of an undefinable part of the solution [VMS99b].

[7]Correct and complete algorithms, also known as *combinatorial* or *exhaustive* algorithms, explore exhaustively all the search space unless there is a guarantee that some its subspace can be safely

An obvious advantage of correct and complete approaches is that the extracted patterns are formally characterized and this is invaluable when trying to interpret the computed solutions.

In this thesis, we will consider inductive queries on string data sets. Furthermore, among the pattern domains on string data, we will consider mainly substring patterns and the primitive constraints which can specify substring relevancy. In other terms, we will not discuss further the many other pattern domains which have been studied to support knowledge discovery from sequential data, for instance sequential patterns (see, e.g., [AS95, SA96, PHW02]), episode and episode rules (see, e.g., [MTV97, MR04]), composite episodes [BS07], etc. For recent survey on this important research area, see, e.g., [MTP04] and more recently [Tei07].

discarded without a loss of any solution element.

Chapter 2

Constraint-Based Pattern Mining

We have seen that pattern extraction can be performed by means of inductive query solving, i.e., computing $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$. A naive approach to solve this task is to generate every pattern ϕ from the pattern language $\mathcal{L}$ and check whether the constraint C_ϕ is satisfied. However such approach is rarely feasible, because a pattern space $\mathcal{L}$ is not only very large in most cases, but it can also be infinite (e.g., a string pattern language $\mathcal{L}_\Sigma$). In addition to this, the evaluation of C_ϕ can be expensive, especially when the collection of data $\mathcal{D}$ is large.

The constraint-based pattern mining research domain consists of searching for efficient strategies to evaluate the constraints by a clever $\mathcal{L}$ traversal that enables to find the solution set without exploring the whole pattern space $\mathcal{L}$. The key issues for designing such efficient strategies is to exploit the *structure of the pattern space* $\mathcal{L}$, the *constraint types and their properties*, and the *structure of the solution set.* These three issues are explored in the following sections.

2.1 Structure of the Pattern Space

To establish a structure in a pattern space $\mathcal{L}$, we require that the *generality* and *specificity* relations be defined on every pair of patterns from the pattern language $\mathcal{L}$.

Definition 2.1 (Generality/Specificity Relation) *Let $\mathcal{L}$ be a pattern language, $\mathcal{U}$ be a universe, ϕ and ψ be patterns from $\mathcal{L}$, X be an object from $\mathcal{U}$ and $\mathcal{U} \times \mathcal{L} \rightarrow \{\mathit{true}, \mathit{false}\}$ be a match function. Pattern ϕ is said to be more general than pattern*

ψ (denoted $\phi \succeq \psi$) if and only if $match(\psi, X) \Rightarrow match(\phi, X)$. In that case we also say that pattern ψ is more specific than pattern ϕ (denoted $\psi \preceq \phi$).

Example 2.1 *Let Σ_4 denote the alphabet $\{a, c, g, t\}$. Let $\mathcal{U}_{\Sigma_4}$ be universe of strings over alphabet Σ_4 and $\mathcal{L}_{\Sigma_4}$ be string pattern language over alphabet Σ_4. Consider exact string match function $match_{\sqsubseteq,S}$ or $match_{\sqsubseteq,s}$ (see Definition 1.21 in Page 21). Then the pattern $\phi =$ gt is more general than the pattern $\psi =$ gtt, denoted gt $\succeq$ gtt. Correspondingly, the pattern $\psi =$ gtt is more specific than the pattern $\phi =$ gt, denoted gtt $\preceq$ gt.*

The generality relation $\succeq$ induces a structure in the pattern space $\mathcal{L}$. Methods to efficiently compute $Th(\mathcal{L}, \mathcal{D}, C)$ rely on that structure. Note that the relation of generality is a pre-order relation[1]. Because of symmetry, the same properties are shared by a relation of specificity. Remark that the generality relation may not be anti-symmetric, and thus it is not necessarily a partial order.[2]

Example 2.2 *Let a set of propositional logic formulas, denoted $\mathcal{L}_f$, be a pattern language. Let a set of truth assignments, denoted $\mathcal{U}_f$, be a universe. Define a match function $match_f(\phi, X)$ to evaluate true if and only if a propositional logic formula ϕ evaluates to true under a truth assignment X.*

$\phi_1 = x$, $\phi_2 = x \wedge (x \vee y)$, $\phi_3 = x \vee y$ and $\phi_4 = y \vee \neg y$ are several patterns from $\mathcal{L}_f$ and $X_1 = \{x \mapsto \textsf{true}, y \mapsto \textsf{true}\}$, $X_2 = \{x \mapsto \textsf{true}, y \mapsto \textsf{false}\}$, $X_3 = \{x \mapsto \textsf{false}, y \mapsto \textsf{true}\}$ and $X_4 = \{x \mapsto \textsf{false}, y \mapsto \textsf{false}\}$ are several objects from $\mathcal{U}_f$. Then we have

$$\begin{aligned}
match_f(\phi_1, X_1) &= match_f(\phi_1, X_2) = \\
match_f(\phi_2, X_1) &= match_f(\phi_2, X_2) = \\
match_f(\phi_3, X_1) &= match_f(\phi_3, X_2) = \\
match_f(\phi_3, X_3) &= \\
match_f(\phi_4, X_1) &= match_f(\phi_4, X_2) = \\
match_f(\phi_4, X_3) &= match_f(\phi_4, X_4) = \textsf{true}
\end{aligned}$$

and

$$\begin{aligned}
match_f(\phi_1, X_3) &= match_f(\phi_1, X_4) = \\
match_f(\phi_2, X_3) &= match_f(\phi_2, X_4) = \\
match_f(\phi_3, X_4) &= \textsf{false}.
\end{aligned}$$

Observe that $\phi_1 \succeq \phi_2$, $\phi_2 \succeq \phi_1$, $\phi_3 \succeq \phi_1$, $\phi_3 \succeq \phi_2$, $\phi_4 \succeq \phi_1$, $\phi_4 \succeq \phi_2$ and $\phi_4 \succeq \phi_3$. Note that we have $\phi_1 \succeq \phi_2$ and $\phi_2 \succeq \phi_1$, but $\phi_1 \neq \phi_2$ and thus the relation $\succeq$ defined in this example is not anti-symmetric and thus it is not a partial order.

[1] A pre-order relation (or quasi-order) is a binary relation that is reflexive and transitive.

[2] A partial order is a binary relation that is reflexive, transitive and anti-symmetric.

We seek for a generality relation that is a partial order, because it allows to organize a pattern space $\mathcal{L}$ into an efficiently exploitable structure. We can derive an equivalence relation $\sim$ in $\mathcal{L}$ and use the resulting quotient set[3] as the pattern language. For this pattern language we can derive the generality relation that is a partial order.

Given a match function that relates patterns from a pattern language $\mathcal{L}$ to objects from a universe $\mathcal{U}$, consider two patterns ϕ and $\psi \in \mathcal{L}$, such that $\phi \succeq \psi$ and $\psi \succeq \phi$. Then from Definition 2.1 it follows that for any object X in the universe $\mathcal{U}$, ϕ matches X if and only if ψ also matches X. These two patterns can be thus considered equivalent w.r.t. the universe $\mathcal{U}$ and the function *match*.

Definition 2.2 (Quotient Set) *Let $\mathcal{L}$ be a set with a pre-order relation $\succeq$. Define an equivalence relation $\sim$ by:*

$$\phi \sim \psi \text{ if and only if } \phi \succeq \psi \text{ and } \psi \succeq \phi$$

Denote the equivalence class containing ϕ by $\phi/\sim$ and the quotient set by $\mathcal{L}/\sim$.

Definition 2.3 (Generality Relation on Quotient Set) *Define a generality relation $\succeq^{\sim}$ on $\mathcal{L}/\sim$ by*

$$\phi/\sim \succeq^{\sim} \psi/\sim \text{ if and only if } \phi \succeq \psi.$$

Example 2.3 *Continuing from Example 2.2, we have $\phi_1 \sim \phi_2$. These groups ϕ_1 and ϕ_2 are into the same equivalence class in $\mathcal{L}_f/\sim$, while ϕ_3 and ϕ_4 belongs to two other equivalence classes, i.e., $\phi_1/\sim = \phi_2/\sim \neq \phi_3/\sim \neq \phi_4/\sim$. They are ordered as $\phi_4/\sim \succeq^{\sim} \phi_3/\sim \succeq^{\sim} \phi_1/\sim$.*

The reflexivity and transitivity of $\succeq^{\sim}$ follow from the reflexivity and transitivity of $\succeq$. Also note that given $\phi/\sim \succeq^{\sim} \psi/\sim$ and $\psi/\sim \succeq^{\sim} \phi/\sim$, we have $\phi/\sim = \psi/\sim$. Thus the relation $\succeq^{\sim}$ is anti-symmetric, what also means that it is a partial order relation.

Definition 2.4 (Match Function on Quotient Set) *Define a match function on quotient $\mathcal{L}/\sim \times \mathcal{U} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ by*

$$match_{\sim}(\phi/\sim, X) \equiv match(\phi, X).$$

[3]The set of all equivalence classes in a set A, w.r.t. the given equivalence relation $\sim$, is called the quotient set of A by $\sim$, usually denoted $A/\sim$.

Example 2.4 *Continuing from Example 2.3, we have*

$$\begin{aligned}
match_{\sim}(\phi_1/\sim, X_1) &= match_{\sim}(\phi_1/\sim, X_2) = \\
match_{\sim}(\phi_3/\sim, X_1) &= match_{\sim}(\phi_3/\sim, X_2) = \\
match_{\sim}(\phi_3/\sim, X_3) &= \\
match_{\sim}(\phi_4/\sim, X_1) &= match_{\sim}(\phi_4/\sim, X_2) = \\
match_{\sim}(\phi_4/\sim, X_3) &= match_{\sim}(\phi_4/\sim, X_4) = \mathsf{true}
\end{aligned}$$

and

$$\begin{aligned}
match_{\sim}(\phi_1/\sim, X_3) &= match_{\sim}(\phi_1/\sim, X_4) = \\
match_{\sim}(\phi_3/\sim, X_4) &= \mathsf{false}.
\end{aligned}$$

Thus from a pattern language $\mathcal{L}$ and a generality relation $\succeq$, which is a pre-order, we can always derive another pattern language $\mathcal{L}/\sim$ with the relation $\succeq^{\sim}$, which is a partial order. Also, this new pattern language $\mathcal{L}/\sim$ is related to the universe $\mathcal{U}$ by the quotient match function $match_{\sim}(\phi/\sim, X)$, which can be always derived from the *match* function on $\mathcal{L} \times \mathcal{U}$. Therefore, if a generality relation $\succeq$ on a given pattern language $\mathcal{L}$ is not a partial order, we can use the constructed equivalence class $\mathcal{L}/\sim$ and the generality relation $\succeq^{\sim}$ correspondingly as a pattern language and a generality relation. Thus in the following we assume that the relations $\succeq$ and $\preceq$ are partial orders. Notice that, e.g., the generality relation defined as a substring relation $\sqsubseteq$ on a substring pattern language is a partial order.

A generality relation $\succeq$ that is a partial order allows to arrange the pattern space $\mathcal{L}$ in the form of an acyclic directed graph. This graph is exploited to design efficient algorithms that walk through the pattern space to find the patterns that satisfy the given constraints.

Definition 2.5 (Partially ordered graph) *Let $\mathcal{L}$ be a set with a relation $\succeq$ that is a partial order. The partially ordered graph of $\mathcal{L}$ is an acyclic directed graph (V, E) where*

- *$V = \mathcal{L}$ is a set of vertices*
- *$E \subseteq V \times V$ is a set of edges. An edge exists between vertices ϕ and ψ, if and only if $\phi \succeq \psi$ and $\nexists \nu \in \mathcal{L}$ such that $\nu \neq \phi$, $\nu \neq \psi$ and $\phi \succeq \nu \succeq \psi$.*

In other words, for each pattern $\phi \in \mathcal{L}$ there is a corresponding vertex in the partially ordered graph. Each edge in the graph corresponds to a pair of patterns $\phi \succeq \psi$ without any other pattern in between. If there is a pattern ν between ϕ and ψ, i.e., if $\phi \succeq \nu \succeq \psi$, then the relation $\phi \succeq \psi$ is represented by the directed path from ϕ

through ν to ψ, taking advantage of the transitivity of the relation $\succeq$. Also note that the acyclicity of the partially ordered graph follows from the anti-symmetry property of the partial order $\succeq$.

In the following we give two examples of partially oriented graphs. The first one establish a structure on the itemset pattern language (see Definition 1.10 in Page 18) and the second one - on the string pattern language (see Definition 1.8 in Page 17).

Example 2.5 *Let $I_4 = \{A, B, C, D\}$ be domain of items. Consider an itemset pattern language $\mathcal{L}_{I_4}$ on I_4. The partially ordered graph of $\mathcal{L}_{I_4}$ is depicted in Figure 2.1. Note that there is no edge from, e.g., the itemset $\{A, D\}$ to the itemset $\{A, B, C, D\}$, because there is an itemset $\{A, B, D\}$, such that $\{A, D\} \succeq \{A, B, D\} \succeq \{A, B, C, D\}$, and an itemset $\{A, C, D\}$, such that $\{A, D\} \succeq \{A, C, D\} \succeq \{A, B, C, D\}$. Notice that $\mathcal{L}_{I_4}$ contain a most general element (the empty set) and a most specific element (the itemset $\{A, B, C, D\}$). Also, for every pair of itemsets $\phi, \psi \in \mathcal{L}_{I_4}$ there exists a unique least general element $\nu \in \mathcal{L}_{I_4}$ such that $\nu \succeq \phi$ and $\nu \succeq \psi$. The analogous condition also hold for the relation of specificity $\preceq$. Thus $\mathcal{L}_{I_4}$ is a lattice and takes advantage of the nice properties of lattices.*

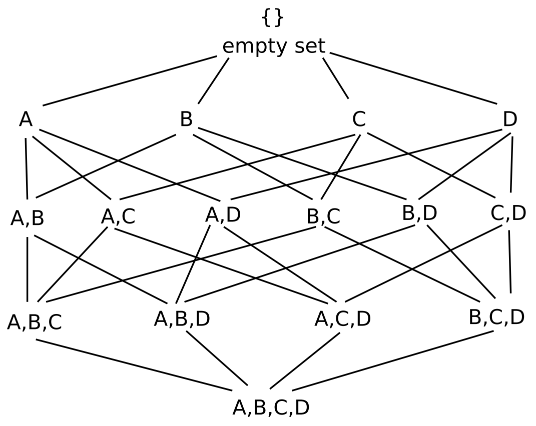

Figure 2.1: Partially ordered graph of $\mathcal{L}_{I_4}$ that is a lattice

Example 2.6 *Let $\Sigma_4 = \{a, c, g, t\}$ be an alphabet and $\mathcal{L}_{\Sigma_4}$ be a string pattern language on Σ_4. Several topmost levels of the partially ordered graph of $\mathcal{L}_{\Sigma_4}$ are depicted in Figure 2.2. The most general element of $\mathcal{L}_{\Sigma_4}$ is the empty string. Note that the pattern space $\mathcal{L}_{\Sigma_4}$ is infinite and thus it is impossible to draw the complete partially ordered graph. Remark that $\mathcal{L}_{\Sigma_4}$ is not a lattice: not only it does not have a most specific element, but also the condition that for every pair of patterns there must exist a unique least general element is not satisfied, e.g., both* ac *and* ca *are the least general elements for the pair of strings* aca *and* cac.

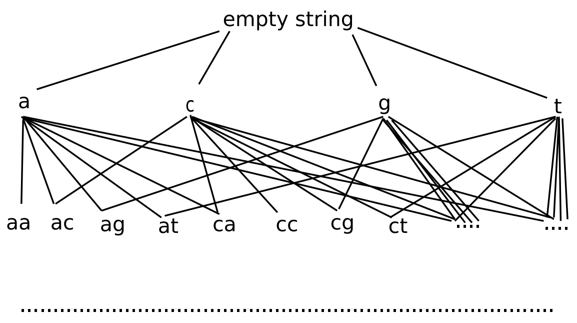

Figure 2.2: Partially ordered graph of $\mathcal{L}_{\Sigma_4}$

2.2 Constraint Types and Properties

The generalisation of data mining tasks to the task of finding the set $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$ [MT97] opens a possibility to distinguish the type of the constraint C and to study their properties. Then the corresponding strategies to efficiently handle the constraints according to their type and/or properties can be designed.

2.2.1 Constraint Types

According to the restriction nature, one can distinguish three types of constraints: data-dependent, syntactic and optimisation constraints.

2.2.1.1 Data-dependent Constraints

Constraints, whose evaluation requires to refer to data, are called data-set dependent constraints.

Definition 2.6 (Data-dependent Constraint) *Let $\mathcal{L}$ be a pattern language, ϕ be a pattern from $\mathcal{L}$, $\mathcal{U}$ be a universe containing objects X, $\mathcal{D}$ be data on $\mathcal{U}$, $match(\phi, X)$ be a match function mapping patterns to objects and C_ϕ be a constraint on ϕ. We say that a constraint C_ϕ is a data-dependent constraint if and only if the evaluation of C_ϕ requires to find the objects $X \in \mathcal{D}$ such that $match(\phi, X) =$ true.*

Minimum and maximum frequency constraints (see Definition 1.20) are classical examples of data-dependent constraints. The notion of frequency is very important in data mining. On one hand, minimum frequency constraint pruning thanks to its anti-monotonicity property (see Section 2.2.2.1 in Page 37) allows to reduce significantly the search space and thus render the correct and complete extractions (see discussion on Page 26) feasible. `A priori` [AS94] is one of the first and most famous data mining algorithms that efficiently exploits the minimum frequency constraint to extract association rules. On the other hand, frequency constraints are semantically important. They are often used to characterize interesting and/or exceptional patterns. Consider, for example, a problem of finding active molecular substructures. A possible solution is to search for the molecular substructures that are frequent in the active molecules and infrequent in the inactive ones, what, in data mining terms, means extracting the substructures that satisfy a minimum frequency constraint in a set of active molecules and a maximum frequency constraint in a set of inactive molecules [KRH01].

2.2.1.2 Syntactic Constraints

On the contrary to the data-dependent constraints, the syntactic constraints are constraints whose evaluation do not require to refer to data. Syntactic constraints restrict the form of the patterns, and thus allows to specify an a priori interesting subset of a pattern language.

Definition 2.7 (Syntactic Constraint) *Let $\mathcal{L}$ be a pattern language, ϕ be a pattern from $\mathcal{L}$ and C_ϕ be a constraint on ϕ. We say that a constraint C_ϕ is a syntactic constraint if and only if the evaluation of C_ϕ does not require to refer at a data set $\mathcal{D}$. Consequently, the corresponding $Th(\mathcal{L}, \mathcal{D}, C)$ does not depend on data $\mathcal{D}$.*

Example 2.7 *Minimum and maximum length constraints (see Definition 1.24 in Page 22) and the regular expression constraint (see Definition 1.25 in Page 22) are syntactic constraints.*

2.2.1.3 Optimisation constraints

It is often useful to associate an evaluation function to define pattern semantics [BKM99]. Constraints that evaluate to true for patterns having optimal evaluation function values are called optimisation constraints.

Definition 2.8 (Evaluation function) *Let $\mathcal{D}$ be data, $\mathcal{L}$ be a pattern language and ϕ be a pattern from $\mathcal{L}$. Let r be a set of result values. Evaluation function $e(\mathcal{D}, \phi)$ maps each pair $(\mathcal{D}, \phi)$ to an element of r.*

Example 2.8 *Let $\Sigma_4 = \{a, c, g, t\}$ be an alphabet and $\mathcal{L}_{\Sigma_4}$ be a string pattern language on Σ_4. Consider a string data set $\mathcal{D} = \{\mathsf{agt}, \mathsf{agc}, \mathsf{gt}, \mathsf{gt}\}$. Let ϕ be a pattern* *from $\mathcal{L}_{\Sigma_4}$. Define an evaluation function $e(\mathcal{D}, \phi)$ be equal to pattern's ϕ frequency (see Definition 1.19 in Page 21), with a set of result values $r = \{0, \ldots, |\mathcal{D}|\} = \{0, \ldots, 4\}$. Then, e.g., $e(\mathcal{D}, \mathsf{gt}) = 3$.*

Since an evaluation function is intended to define pattern semantics, its choice depends on the problem one is trying to answer through pattern extraction. For example, statistical significance [WAG84, RF98a] and information content [Sta89, WFHW96] are widely used evaluation functions when mining patterns in DNA sequences.

Definition 2.9 (Optimisation Constraint) *Let $\mathcal{L}$ be a pattern language, ϕ be a pattern from $\mathcal{L}$, $\mathcal{D}$ be data and $e(\mathcal{D}, \phi)$ be an evaluation function. We say that a constraint C_ϕ is an optimisation constraint if and only if it evaluates to* **true** *when $e(\mathcal{D}, \phi)$ has an optimal value.*

Many learning algorithms seek to compute global models that satisfy an optimisation constraint. For example, a typical task of clustering is to find clusters such that the inter-cluster distance is maximal and intra-cluster distance is maximal. Similarly, a typical task of classification problems is to build a classifier that maximizes an accuracy, or equivalently, minimizes an error rate. In most cases to find a correct and complete solution is too much computationally expensive. Then, the heuristic approaches devise techniques that are able to extract the patterns having a *good*, i.e., near to the optimal, evaluation function value.

Optimisation constraints are also difficult to exploit when mining local patterns, and thus they are often handled by post-processing that filter out patterns having not optimal evaluation function values. Some notable exceptions include algorithms to mine association rules [AIS93] that are optimal w.r.t. a variety of evaluation functions, e.g., [RJBA99], or mining maximal frequent itemsets, e.g., [BCFY05].

A slight variation of optimisation constraints for local patterns are so called *top-k* constraints. Top-k constraint asks $k \in \mathbb{N}$ patterns having the best evaluation function values. For example, [HWLT02] proposes an algorithm to extract the k itemsets with the largest frequency values. Similarly, [TYH03, TYH05] proposes an algorithm to extract the k sequence patterns with the largest frequency values.

2.2.2 Constraint Properties

The idea behind efficient search space pruning strategies is to safely ignore large pattern space areas. The safety of such pruning is assured by constraint properties that guarantee that no pattern satisfying the constraint condition can exist in the discarded subspace. According to these formal properties, constraints can be attributed to the classes. In the following, we present several such classes that were extensively explored by the data mining community.

2.2.2.1 Anti-monotonic and monotonic constraints

The class of anti-monotonic constraints is one of the easiest to handle, and a number of algorithms, `A priori` [AS94] being among the first ones, exploit the property of anti-monotonicity to prune the pattern space and thus to render the complete and correct extractions feasible and efficient.

Definition 2.10 (Anti-monotonicity) *Let $\mathcal{L}$ be a pattern language. A constraint C is anti-monotonic if and only if for any $\phi, \psi \in \mathcal{L}$ we have that if $\phi \succeq \psi$, then $C_\psi \Rightarrow C_\phi$.*

The pruning by an anti-monotonic constraint is based on the following remark.

Remark 2.1 *Note that, if a constraint C is anti-monotonic, then, for any $\phi \succeq \psi$, if C_ϕ is not satisfied, then C_ψ can not be satisfied.*

Example 2.9 *Let $\mathcal{L}$ be a pattern language, ϕ be a pattern from $\mathcal{L}$, $\mathcal{D}$ be data and $minFr, l \in \mathbb{N}$ be thresholds. The minimum support constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ (see Definition 1.20 in Page 21) and the maximum length constraint $\mathsf{MaxLength}_\phi(l)$ (see Definition 1.24 in Page 22) are anti-monotonic constraints.*

Monotonicity is a property dual to anti-monotonicity.

Definition 2.11 (Monotonicity) *Let $\mathcal{L}$ be a pattern language. A constraint C is monotonic if and only if for any $\phi, \psi \in \mathcal{L}$ we have that if $\phi \preceq \psi$, then $C_\psi \Rightarrow C_\phi$.*

The exploitation of a monotonic constraint is usually based on the following remark.

Remark 2.2 *Note that, if a constraint C is monotonic, then, for any $\phi \preceq \psi$, if C_ϕ is not satisfied, then C_ψ can not be satisfied.*

Example 2.10 *Let $\mathcal{L}$ be a pattern language, ϕ be a pattern from $\mathcal{L}$, $\mathcal{D}$ be data and $maxFr, l \in \mathbb{N}$ be thresholds. The Maximum frequency constraint $\mathsf{MaxFr}_\phi(maxFr, \mathcal{D})$ (see Definition 1.20 in Page 21) and the minimum length constraint $\mathsf{MinLength}_\phi(l)$ (see Definition 1.24 in Page 22) are monotonic constraints.*

The duality between monotonic and anti-monotonic constraints has been observed early in [BJ00] with a detailed presentation in [Jeu02]. It has motivated the study of the relationship between border-based algorithms and the popular version spaces (see next section), giving rise to the state-of-the-art generic solvers discussed in the next chapter.

2.2.2.2 Succinct constraints

A property of succinctness was defined for constraints on itemset patterns [NLHP98]. It is an easy to exploit property that characterizes constraints for which pruning can be done without candidate pattern generation and testing w.r.t. data $\mathcal{D}$. Therefore, succinct constraints are syntactic constraints. Notice that according to [Jeu02], succinct constraints can be expressed as conjunctions of monotonic and anti-monotonic constraints.

Definition 2.12 (Succinctness) *Let I be a domain of items, and $\mathcal{I}$ be an itemset on I (see Definition 1.9 in Page 17). Let $\mathcal{L}_I$ be an itemset pattern language. Let π_p be a selection predicate, i.e., any predicate allowed to appear as a parameter of a selection operation in relational algebra. $\pi_p(I)$ returns the items that satisfy predicate p. Then,*

- *an itemset $\mathcal{I} \subseteq I$ is a succinct set if it can be expressed as $\pi_p(I)$;*
- *$SP \subseteq 2^I$ is a succinct powerset if there is $n \in \mathbb{N}$ succinct sets $\mathcal{I}_1, \ldots, \mathcal{I}_n \subseteq I$ such that SP can be expressed in terms of the strict powersets of $\mathcal{I}_1, \ldots, \mathcal{I}_n$ using the union and minus set operators;*
- *a constraint C is a succinct constraint if $Th(\mathcal{L}_I, C)$ is a succinct powerset.*

Example 2.11 *Let domain of items I be products sold in a supermarket, which can be attributed to categories such as vegetables, fruits, cereals, etc. Consider a constraint C that asks all itemsets that contain at least one vegetable item and at least one fruit item. Let $\pi_{vegetable}(I)$ be a selection predicate that returns the items that are vegetables, $\pi_{fruit}(I)$ be a selection predicate that returns the items that contain fruits and $\pi_{\neg vegetable \wedge \neg fruit}(I)$ be a selection predicate that returns the items that are neither vegetables nor fruits. Then, let I_1, I_2 and I_3 be respectively the sets $\pi_{vegetable}(I)$, $\pi_{fruit}(I)$ and $\pi_{\neg vegetable \wedge \neg fruit}(I)$. The constraint C is succinct because its $Th(\mathcal{L}_I, C)$ can be expressed as $2^I - 2^{I_1} - 2^{I_2} - 2^{I_1 \cup I_3} - 2^{I_2 \cup I_3}$.*

2.2.2.3 Convertible constraints

Before formally defining the notion of convertible constraints, we propose to consider the following example.

Example 2.12 *Let the domain of items I be products sold in a supermarket. Let prices be associated with each such item. Consider a minimum average constraint C that is satisfied if and only if the average prices of the items in a itemset $\mathcal{I}$ is greater than θ, $\theta \in \mathbb{N}$. Consider an itemset $\mathcal{I}_1 = \{tomatos, apples\}$ and an itemset $\mathcal{I}_2 = \{tomatos, apples, peaches\}$. Suppose that the constraint $C_{\mathcal{I}_1}$ is* true *for a given θ. Then, we cannot say if $C_{\mathcal{I}_2}$ is* true*, because the price of peaches can increase as well as decrease the average price of items. Similarly, given that $C_{\mathcal{I}_1}$ is* true *we can not say if the $C_{\mathcal{I}_2}$ is* true*. Therefore, C is neither monotonic, nor anti-monotonic. Neither it is succinct. However, observe that if we arrange the items in the price-descending order, the average of the items prices in a itemset is always smaller or equal than the average of items prices in its prefix itemset.*

A notion of convertible constraints on itemsets was introduced in [PH00, PHL01]. The idea is that, given a constraint that is neither anti-monotonic nor monotonic, i.e., a constraint that does not establish a relation between itemset subsets, supersets and a satisfaction of constraint, we can arrange the items in an itemset so that there is a stable relation between a constraint satisfaction and an itemset prefix.

Definition 2.13 (Itemset Prefix) *Let R be a total order over the items from the domain I and $m, n \in \mathbb{N}$ be such that $m \leq n$. Consider the itemset $\mathcal{I}_1 = \{i_1, \ldots, i_m\}$ and the itemset $\mathcal{I}_2 = \{i_1, \ldots, i_n\}$ such that the items in $\mathcal{I}_1$ and $\mathcal{I}_2$ are listed according to the order R. Then, the itemset $\mathcal{I}_1$ is called a prefix of the itemset $\mathcal{I}_2$ w.r.t. R.*

Definition 2.14 (Convertible Constraint) *Let $\mathcal{L}_I$ be an itemset pattern language, and $\phi \in \mathcal{L}_I$ be an itemset pattern. A constraint C is convertible anti-monotonic if and only if there is an order R on items such that whenever an itemset ϕ satisfies C, any itemset prefix of ϕ also satisfies C. A constraint C is convertible monotonic if there is an order R on items such that whenever an itemset ϕ violates C, any itemset prefix of ϕ also violates C. A constraint is convertible whenever it is convertible monotonic or convertible anti-monotonic.*

Remark 2.3 *(Anti-)monotonic constraints are trivially convertible (anti-)monotonic.*

Example 2.13 *The minimum average constraint C from Example 2.12 is convertible anti-monotonic w.r.t. the price-descending order R. A maximum average constraint that is satisfied if and only if average of prices of items in a itemset $\mathcal{I}$ is smaller than v, $v \in \mathbb{N}$, is convertible monotonic w.r.t the same order R.*

Although the notion of convertible constraints allows to push deeply into extraction phase a number of useful and otherwise hard to exploit constraints, it cannot be easily generalized to other pattern languages, e.g., string or sequence patterns, where an order of pattern elements is pre-composed and can not be freely re-arranged. An interesting exception is `Galibot` algorithm [CBM02b], where a similarity to a pattern of reference constraint is relaxed to a convertible anti-monotonic constraint (see Section 4.2.3 for more details).

2.2.2.4 Prefix-monotone constraints

A notion of prefix-monotone constraints on sequence patterns (see Definition 1.12 in Page 18) was introduced in [PHW02] with an objective to cover a large part of useful constraints that can be pushed deeply using the proposed generalized strategy.

We first define a notion of sequence prefix.

Definition 2.15 (Sequence Prefix) *Let $I = \{i_1, \dots, i_k\}$, $k \in \mathbb{N}$, be a domain of items and R be a total order over them. Since the item ordering in an itemset is not important to sequential patterns, any order R can be considered. For example, let R be the alphabetical order. The fact that the item i_i is before the item i_j w.r.t. the order R is denoted $i_i < i_j$. Consider a sequence $\Phi = \langle \mathcal{I}_1, \dots, \mathcal{I}_n \rangle$, $n \in \mathbb{N}$, and a sequence $\Psi = \langle \mathcal{I}_1, \dots, \mathcal{I}_k, \mathcal{Y} \rangle$, $k \in \mathbb{N}$. Ψ is a prefix of Φ, if*

- $k < n$
- $\mathcal{Y} \subseteq \mathcal{I}_{k+1}$
- *for each $i \in \mathcal{Y}$ and for each $i' \in \mathcal{I}_{k+1} - \mathcal{Y}$, $i < i'$*

If Ψ is a prefix of Φ, it is also a subsequence of Φ.

Example 2.14 *Let $I = \{A, B, C, D\}$ be a domain of items. Let R be an alphabetical order. The sequence $\Psi = \langle \{C\}\{A, B\} \rangle$ is a prefix of sequence $\Phi = \langle \{C\}\{A, B, D\}\{B, C\} \rangle$, but the sequence $\Xi = \langle \{C\}\{A, D\}$ is not a prefix of Φ.*

Definition 2.16 (Prefix-monotone Constraint) *Let $\mathcal{L}_\mathcal{S}$ be a sequence pattern language and ϕ be a pattern from $\mathcal{L}_\mathcal{S}$. A constraint C is prefix anti-monotonic if and only if for each sequence pattern ϕ satisfying the constraint C, every prefix of ϕ also satisfies C. A constraint is prefix monotonic if and only if for each sequence ϕ satisfying C, every sequence having ϕ as a prefix also satisfies C. A constraint C is prefix-monotone if it is prefix anti-monotonic or prefix monotonic.*

Remark 2.4 *(Anti-)monotonic constraints are trivially prefix (anti-)monotonic.*

In the following we define a legality w.r.t. a regular expression constraint, which is an example of prefix anti-monotonic constraint. The interest of this constraint is that it is a relaxation of a regular expression constraint (see Definition 1.25 in Page 22), and thus enables to exploit the latter using the generalized strategy of [PHW02].

Definition 2.17 (Constraint of Legality w.r.t. a Regular Expression) *Let $\mathcal{L}$ be a string or sequence pattern language and ϕ be a pattern from $\mathcal{L}$. Let re be a regular expression and $\mathcal{A}_{re}$ be a corresponding deterministic finite automaton*[4]*. Define a legality w.r.t. a regular expression constraint $\mathsf{LegalRE}_{\phi}(re)$ to evaluate* **true** *if and only if a state in $\mathcal{A}_{re}$ can be reached following ϕ, or in other words, if a pattern ϕ is legal*[5] *w.r.t. a regular expression re.*

Remark 2.5 *A constraint of legality w.r.t. a regular expression $\mathsf{LegalRE}_{\phi}(re)$ is a relaxation of a regular expression constraint $\mathsf{MatchRE}_{\phi}(re)$, since for each pattern ϕ satisfying $\mathsf{MatchRE}_{\phi}(re)$, every prefix of ϕ must be legal w.r.t. a regular expression re.*

Example 2.15 *Let $I = \{A, B, C, D\}$ be a domain of items and $\mathcal{L}_{\mathcal{S}}$ be a sequence pattern language on I. Consider the regular expression `C(A|D)(A|C)`. An example of a satisfied legality constraint w.r.t. this regular expression is $\mathsf{LegalRE}_{CA}($`C(A|D)(A|C)`$)$. An example of an unsatisfied one is $\mathsf{LegalRE}_{CC}($`C(A|D)(A|C)`$)$.*

Example 2.16 *A constraint of legality w.r.t. a regular expression $\mathsf{LegalRE}_{\phi}(re)$ is prefix anti-monotonic, since for each pattern ϕ that is legal w.r.t. a regular expression re, every prefix of ϕ must also be legal w.r.t. re.*

2.3 Structure of the Solution Set

We analyzed the structure of the pattern space (or, in other words, the pattern language) $\mathcal{L}$. Inductive query answering needs to compute a collection $Th(\mathcal{L}, \mathcal{D}, C)$, i.e., its solution or answer set which is a subset of $\mathcal{L}$. We introduce now a method

[4]Regular expressions have the same expressive power as deterministic finite automata [LP81], thus given any regular expression re one can always build a deterministic finite automaton $\mathcal{A}_{re}$ such that $\mathcal{A}_{re}$ accepts the language generated by re.

[5]A term of a sequence legality w.r.t. a state of a deterministic finite automaton $\mathcal{A}_{re}$ of a regular expression re was introduced in [GRS99]. A term of sequence legality w.r.t. a regular expression re, as defined here, was introduced in [PHW02], and is equivalent to a sequence legality w.r.t. the start state of $\mathcal{A}_{re}$.

to compactly represent $Th(\mathcal{L}, \mathcal{D}, C)$, using the structure of $\mathcal{L}$. First, we examine the case, where the inductive query involve an anti-monotonic constraint $\mathcal{A}$ or a monotonic constraint $\mathcal{M}$, or an arbitrary conjunction $\mathcal{A} \wedge \mathcal{M}$. There, the notion of *version space* is presented. After, we introduce the notion of *generalized version spaces*, which is used to solve the inductive queries that are arbitrary combinations of anti-monotonic and monotonic constraints.

2.3.1 Version Space

Version spaces[Hir91, Hir94, Mit82] (also known as *convex set*) and their border representations [MT97] are traditionally used to characterize the solution space $Th(\mathcal{L}, \mathcal{D}, C)$ in the fields of machine learning [Hir91, Hir94, Mit82] and data mining [MT97, Bay98].

To begin, consider the computation of $Th(\mathcal{L}, \mathcal{D}, C)$, where C is anti-monotonic. According to the property of anti-monotonicity (see Definition 2.10 in Page 37), if a pattern $\phi \in \mathcal{L}$ satisfies the constraint C, then all patterns that are more general than ϕ also satisfy C. In other words, if a pattern ϕ is in a solution set $Th(\mathcal{L}, \mathcal{D}, C)$, then all the patterns that are more general (see Definition2.1 in Page 29) than ϕ also belong to the solution set. This means that ϕ *covers* all its generalisations. Notice that, in general, one pattern does not cover the whole $Th(\mathcal{L}, \mathcal{D}, C)$, and we need several (many) patterns to cover the whole solution set[6]. The set of maximally specific patterns in $Th(\mathcal{L}, \mathcal{D}, C)$ is the minimal covering set[7]. It is called the S-set (set of maximaly *specific* elements).

Definition 2.18 (S-set) *Let $\mathcal{L}$ be a pattern space with a generality relation $\succeq$. Let S be any subset of $\mathcal{L}$. The S-set of S is defined by*

$$\mathsf{S}(S) = \{\phi \in S \mid \nexists\ \psi \in S : \phi \neq \psi \wedge \psi \preceq \phi\}$$

The case, where the constraint C is monotonic is symmetric to the previous case. According the the property of monotonicity (see Definition 2.11 in Page 37), if a pattern $\phi \in \mathcal{L}$ satisfy the constraint C, then all patterns that are more specific than ϕ also satisfy C. Thereby, the pattern ϕ covers all its specialisations. The minimal set of patterns that cover the whole solution $Th(\mathcal{L}, \mathcal{D}, C)$ is called the G-set(set of maximally *general* elements).

Definition 2.19 (G-set) *Let $\mathcal{L}$ be a pattern space with a generality relation $\succeq$. Let S be any subset of $\mathcal{L}$. The G-set of S is defined by*

[6]A set of patterns cover the solution set $Th(\mathcal{L}, \mathcal{D}, C)$, if these patterns and their generalisations form $Th(\mathcal{L}, \mathcal{D}, C)$

[7]The set of covering patterns will be redundant if any of them is a generalisation of another. To minimize the set of covering patterns we need to eliminate such redundancy

$$\mathsf{G}(S) = \{\phi \in S \mid \nexists\, \psi \in S : \phi \neq \psi \wedge \psi \succeq \phi\}$$

Consider now a case, where the constraint C is a conjunction of a anti-monotonic and monotonic constraint $\mathcal{A} \wedge \mathcal{M}$. Such constraint C is neither anti-monotonic, not monotonic, however its solution $Th(\mathcal{L}, \mathcal{D}, C)$ can be easily computed by the set intersection $Th(\mathcal{L}, \mathcal{D}, \mathcal{A}) \cap Th(\mathcal{L}, \mathcal{D}, \mathcal{M})$. Note that the S of $Th(\mathcal{L}, \mathcal{D}, \mathcal{A})$ and the G of $Th(\mathcal{L}, \mathcal{D}, \mathcal{M})$ covers the elements of $Th(\mathcal{L}, \mathcal{D}, \mathcal{A} \wedge \mathcal{M})$, i.e., the solutions of constraint $C = \mathcal{A} \wedge \mathcal{M}$. This means, that given the S-set and the G-set, the solution set $Th(\mathcal{L}, \mathcal{D}, \mathcal{A} \wedge \mathcal{M})$ is in between. The region, sandwiched between the the S-set and G-set is called a *version space*.

Definition 2.20 (Version Space) *Let $\mathcal{L}$ be a pattern space with a generality relation $\succeq$. Let S be the S-set of a subset S of $\mathcal{L}$ and G be the G-set of S. Then S is a version space, defined by:*

$$\{\phi \in \mathcal{L} \mid \exists\, s \in \mathsf{S} \text{ and } \exists\, g \in \mathsf{G} : g \succeq \phi \succeq s\}$$

Take notice that a version space can represent the solution set to a conjunction of any number of anti-monotonic and monotonic constraints, since a conjunction of anti-monotonic constraints is anti-monotonic, and a conjunction of monotonic constraints is monotonic. The level-wise version space algorithm that computes the S-set and G-set set w.r.t. a conjunction of monotonic and anti-monotonic constraints was introduced in [RK01, KRH01].

2.3.2 Generalized Version Space

In the previous section we saw that a version space can be used as a compact representations of the solution set $Th(\mathcal{L}, \mathcal{D}, C)$, where C is of the form $\mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_n \wedge \mathcal{M}_1 \ldots \mathcal{M}_m$. The condition on the constraint C form is however quite restrictive, and there are many constraints, e.g., $\mathcal{A}_1 \vee (\mathcal{A}_2 \wedge \mathcal{M}_1)$, that can not be expressed in this form. [DJDM02, DD03] proposed to to evaluate a constraint C that is an arbitrary Boolean combination of anti-monotonic and monotonic constraints by decomposing C into n subconstraints C_i, such that

- $C \equiv C_1 \vee \ldots \vee C_n$, where n is minimal;
- each subconstraint C_i is the conjunction of an anti-monotonic and monotonic subconstraint, i.e., $C_i = \mathcal{A} \wedge \mathcal{M}$.

Note that the solution set $Th(\mathcal{L}, \mathcal{D}, C_i)$ is a version space, and the solution set $Th(\mathcal{L}, \mathcal{D}, C)$ can be represented by a union of version spaces $\bigcup_{i=1}^{n} Th(\mathcal{L}, \mathcal{D}, C_i)$. The union of version spaces is called a *generalized version space*.

Definition 2.21 (Generalized Version Space) *A subset of a pattern space $\mathcal{L}$ with a generality relation $\succeq$, that can be expressed as a union of version spaces, is called a generalized version space.*

Chapter 3

Generic Solvers

By a *generic solver* we mean a solver capable to evaluate the inductive queries that are arbitrary Boolean compositions of constraints, i.e., arbitrary expressions over constraints using Boolean algebra operations: conjunction, disjunction and negation. On the contrary, an *ad-hoc* solver is a solver, designed to evaluate a particular composition of constraints (e.g., a minimum frequency constraint, a conjunction of minimum frequency constraint with other application specific constraints) on a particular pattern space $\mathcal{L}$ (e.g., itemset pattern language $\mathcal{L}_I$, string pattern language $\mathcal{L}_\Sigma$).

The inductive databases framework provides a basis to study the properties of different pattern languages $\mathcal{L}$, constraints C and their different compositions, as well as different types of data sets $\mathcal{D}$. Thereby it provides a framework to design efficient generic algorithms that solve inductive queries. Indeed, a key issue for designing efficient solvers is to exploit the structure of the pattern space $\mathcal{L}$, the structure of the solution set $Th(\mathcal{L}, \mathcal{D}, C)$ and the constraint C properties to compute $Th(\mathcal{L}, \mathcal{D}, C)$ without exploring the whole search space $\mathcal{L}$. Extensive studies of the anti-monotonicity and monotonicity properties have given rise to a general theory [DJDM02, DD03] for mining general patterns (i.e., patterns from any pattern language $\mathcal{L}$), satisfying an inductive query with the constraint C that is a Boolean combination of anti-monotonic and monotonic primitive constraints.

3.1 Theoretical Framework

The theory of solving inductive query with a constraint C that is an arbitrary Boolean composition of anti-monotonic and monotonic primitive constraints, is based on characterizing the solution set $Th(\mathcal{L}, \mathcal{D}, C)$ by means of version spaces [Hir91, Hir94, Mit82] and border representations [MT97] (2.3.1 in Page 42). More specifically, this theory concerns the decomposition of such constraint C into a set of subconstraints C_i, such that $Th(\mathcal{L}, \mathcal{D}, C) = \bigcup_{i=1}^{k} Th(\mathcal{L}, \mathcal{D}, C_i)$, where k is minimal and each $Th(\mathcal{L}, \mathcal{D}, C_i)$ can be represented using a single version space, i.e., $C_i = \mathcal{A} \wedge \mathcal{M}$[1], where $\mathcal{A}$ denotes an anti-monotonic constraint and $\mathcal{M}$ denotes a monotonic constraint. Such solution set $Th(\mathcal{L}, \mathcal{D}, C)$ that is a union of version spaces $Th(\mathcal{L}, \mathcal{D}, C_i)$ is known as generalized version space [DD03] (see Section 2.3.2 in Page 43). This results in an operational procedure for solving arbitrary Boolean inductive queries, since once each C_i is computed, the resulting solution sets $Th(\mathcal{L}, \mathcal{D}, C_i)$ can be combined using the set manipulation operations to obtain the solution $Th(\mathcal{L}, \mathcal{D}, C)$ [DD03]. While the latter can be done efficiently, the step of computing $C_i = \mathcal{A} \wedge \mathcal{M}$ is not trivial. The general theoretical framework was instantiated to answer inductive queries on patterns from string pattern language $\mathcal{L}_\Sigma$ [DJDM02]. One of the key elements of this instance is the algorithm `VST` to efficiently compute patterns, satisfying an arbitrary conjunction of anti-monotonic and monotonic constraints. The algorithm `VST` and the algorithm `FAVST` [DD04], efficiently solving the same problem in a distributed environment, are presented in the following sections.

Considerable research effort in constraint-based data mining was dedicated for itemset pattern mining. Researchers in itemset mining domain have also considered the issue of extraction under anti-monotonic and monotonic constraints, e.g., [GLW00] presents a method for combining the anti-monotonic and monotonic constraints when mining item sets, and [PH00] discuss the use of `FP-trees` [HPY00] with several anti-monotonic and monotonic constraints. Such approaches should not be confused with the algorithms `VST` and `FAVST`, which are the instances of the general theoretical framework that works *not only* with itemset or string patterns, but with patterns from any pattern language $\mathcal{L}$. Also, the minimum frequency constraint being anti-monotonic and the maximum frequency constraint being monotonic, the algorithms `VST` and `FAVST` certainly solves the conjunction $\mathsf{MinFr}_{\phi}^{\sqsubseteq, S}(minFr, \mathcal{D}1) \wedge \mathsf{MaxFr}_{\phi}^{\sqsubseteq, S}(maxFr, \mathcal{D}2)$ but they should not be confused with the *ad-hoc* approaches dedicated to evaluate this particular conjunction of frequency constraints, such as the algorithm of [FHK05, FHK06].

3.2 Instance for String Patterns

3.2.1 Version Space Tree

A Version Space Tree (VST)[DJDM02] is a data structure, dedicated to represent and index a version space (see Definition 2.20 in Page 43) of string patterns, i.e., a

[1] As noticed in Section 2.3.1 in Page 42, a conjunction of the anti-monotonic constraints is anti-monotonic, and a conjunction of the monotonic constraints is monotonic. Therefore, a constraint $C_i = \mathcal{A}_1 \wedge \ldots \mathcal{A}_n \wedge \mathcal{M}_1 \wedge \ldots \wedge \mathcal{M}_m$ can be rewritten as $C_i = \mathcal{A} \wedge \mathcal{M}$, where $\mathcal{A} = \mathcal{A}_1 \wedge \ldots \mathcal{A}_n$ and $\mathcal{M} = \mathcal{M}_1 \wedge \ldots \wedge \mathcal{M}_m$.

set $Th(\mathcal{L}, \mathcal{D}, C_i)$, where $C_i = \mathcal{A} \wedge \mathcal{M}$. The anti-monotonic constraint $\mathcal{A}$ is required to be non-trivial (i.e., it cannot evaluate true for every pattern ϕ), since otherwise the version space and the corresponding VST will be infinite. There is no such requirement for monotonic constraint $\mathcal{M}$, since the solution set to $\mathcal{A}$ is always bounded by the most general pattern (in our case, an empty string) and the S set of $Th(\mathcal{L}, \mathcal{D}, \mathcal{A})$.

VST was inspired by a well studied suffix tree [Wei73, Ukk95] data structure.

A *trie* is a tree, such that

- each edge is labeled with a symbol from a given alphabet Σ,
- each edge, emerging from a node, must be labeled with an unique label,
- each node $\mathcal{N}$ uniquely represents a string $\sigma = \sigma_1 \ldots \sigma_n$: the path from the root node $\mathcal{R}$ to the node $\mathcal{N}$ spells the string σ. In the following we will denote such string σ as string($\mathcal{N}$). The root node represents an empty string.

A *suffix trie* is a trie, such that

- for each node $\mathcal{N}$ there also exists a node $\mathcal{N}'$, such that string($\mathcal{N}$') is a suffix of string($\mathcal{N}$).
- each node $\mathcal{N}$ has a link to the node $\mathcal{N}'$, called the *suffix link*. The root node is an exception, because it represents an empty string, and its suffix link is defined to be linked to the root node itself.

To design a suffix trie that represents a version space of strings, several modifications are made w.r.t. the classical suffix tree structure:

- such suffix trie is build from all suffixes of a *set of strings*[2], whereas a classical suffix trie is build from all suffixes of a *single string*.
- each node $\mathcal{N}$ is labeled with either $\oplus$ or $\ominus$: the label $\oplus$ indicates that string($\mathcal{N}$) satisfies the constraint and therefore belongs to the version space, and the label $\ominus$ indicates that string($\mathcal{N}$) does not satisfy the constraint and therefore does not belong to the version space,
- the chains of nodes with only one out-going edge are not coalesced into a single edge with one label, since each node represents a string in a string data set, with its proper label (and a count of its occurrences).

[2]This allows to index a *set* of strings.

Take notice that such labeled suffix trie can contain nodes, such that their labels and the labels of their all descendants are $\ominus$. Then such node and the subtrie, composed of its descendants, are uninteresting (the corresponding string patterns do not satisfy a constraint). Therefore, in practise, a labeled suffix trie is often pruned so that all its leaf nodes are labeled with $\oplus$. Such labeled suffix trie is called a *pruned labeled suffix trie.* Take notice that both tries have the same semantics and each fully labeled suffix trie has a unique corresponding pruned labeled suffix trie. In most cases there is no difference which trie is used, and therefore both are often referred as a *labeled suffix trie.*

A Version Space Tree (VST) is a labeled suffix trie that represents a version space of string patterns, i.e., for every string pattern ϕ from a version space, in a VST there is a node $\mathcal{N}$ labeled with $\oplus$, such that $\phi = \mathsf{string}(\mathcal{N})$. The VST data structure enables to compute the sets S and G (see Section 2.3.1 in Page 42) by a simple tree traversal:

- the set S contains the strings represented by the nodes $\mathcal{N}$ labeled with $\oplus$, such that they have no descendant nodes labeled with $\oplus$, and there are no nodes labeled with $\oplus$, whose suffix links points to $\mathcal{N}$;
- the set G contains the strings represented by the nodes $\mathcal{N}$ labeled with $\oplus$, such that their parents are labeled with $\ominus$, and their suffix link point to the node labeled with $\ominus$.

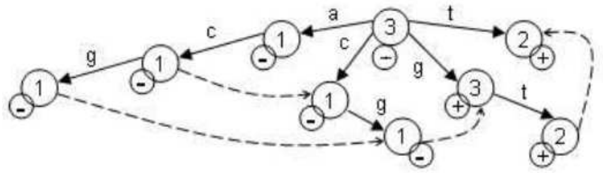

Figure 3.1: Example of a Version Space Tree

Example 3.1 *Let Σ be an alphabet $\{a, c, g, t\}$, $\mathcal{L}_\Sigma$ be a string pattern language on Σ, and $\mathcal{D} = \{\mathsf{acg}, \mathsf{gt}, \mathsf{gt}\}$ be a string data set. Figure 3.1 depicts a VST that indexes the string patterns ϕ present in the data set $\mathcal{D}$. Note that each node contains the frequency value of the corresponding string pattern. The dashed arrows indicate the suffix links (the suffix links of the root and root children are omitted). This VST islabeled for the inductive query $IQ \equiv \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(2, \mathcal{D}) \wedge \mathsf{MinLength}_{\phi}(1)$ (the labels are stuck to the nodes). Here, the nodes labeled with $\ominus$ are left for illustrative reasons, though the branches containing only $\ominus$ nodes are to be pruned. The solution to the IQ is the set $\{\mathsf{gt}, \mathsf{g}, \mathsf{t}\}$ Note that the minimum frequency constraint is anti-monotonic and* *the minimum length constraint is monotonic, and thus the solution set is a version space. The set $S = \{gt\}$ and the set $G = \{g, t\}$.*

3.2.2 Generic Solvers for Strings

In this section we present two algorithms that constructs a version space tree (VST), containing the solution patterns to a constraint of the form $\mathcal{A} \wedge \mathcal{M}$. We remind that the theory of solving the constraint C [DJDM02] that is and arbitrary Boolean composition of anti-monotonic and monotonic constraints consists of decomposing C into the sub-constraints $C_i = \mathcal{A} \wedge \mathcal{M}$, and then combining the solution sets $Th(\mathcal{L}, \mathcal{D}, C_i)$ to these C_i using the set manipulation operations [DD03]. Thus, the algorithm that solves the constraint of the form $\mathcal{A} \wedge \mathcal{M}$ is the essential element of the instance of the generic solver for string patterns.

Both algorithms take the following input elements:

- String data sets $\mathcal{D}_1, \ldots, \mathcal{D}_n$.
- A constraint $C = \mathcal{A} \wedge \mathcal{M}$. The algorithms assume that the VST, and hence the corresponding version space is finite. Therefore, the anti-monotonic constraint $\mathcal{A}$ can not always evaluate `true`, i.e., it must be non-trivial[3]. There is no such restriction for the monotonic constraint $\mathcal{M}$.
- An alphabet Σ_ϕ, which contains the "interesting" symbols, from which the patterns ϕ will be composed. The symbols that do not belong to this Σ_ϕ will be ignored in the data strings.

Two algorithms, `VST` and `FAVST`, were designed to construct a VST and solve the given constraint $C = \mathcal{A} \wedge \mathcal{M}$. One of the contributions of this thesis is build on the `FAVST` algorithm and therefore we will present it in a greater detail.

3.2.2.1 `VST`

The algorithm `VST`, introduced in [DJDM02], is a levelwise algorithm [MT97].

The algorithm consists of two steps:

- `Descend`: top-down growing of the VST, using the anti-monotonic constraint $\mathcal{A}$. This step reuse the idea of the well-known `Apriori` algorithm [AS94].
- `Ascend`: bottom-up marking of the VST, using the monotonic predicate.

[3]Otherwise, the version space, and thus the corresponding VST will grow infinitely

Both steps are designed to minimize the number of data set scans, and, as in the `Apriori` algorithm, one needs at most one scan for each level of the VST. They both function performing the iterations of candidate generation, candidate testing (which involves a data set scan) and pruning.

`Descend` The `Descend` algorithm works in a top-down breath-first manner and exploit the anti-monotonic constraint $\mathcal{A}$. At each VST level k, one disposes the set denoted L_{k-1}, containing the nodes labeled with $\oplus$, found in the previous iteration. These nodes are expanded, labelled with $\oplus$ and put into the set of candidates, denoted C_k. The constraint $\mathcal{A}$ is evaluated for the string patterns $\mathsf{string}(\mathcal{N})$, such that $\mathcal{N} \in C_k$. This evaluation step involves a scan of the string data set $\mathcal{D}$. The nodes in the set C_k, such that $\mathsf{string}(\mathcal{N})$ satisfy the constraint $\mathcal{A}$, are put to the set L_k, and the other nodes are pruned from the VST. This process is continued in the level-wise manner until the set C_k is empty. The suffix links in the VST are set in this phase, during the process.

Take notice that the sets C_k and L_k correspond to the *candidate* and *large* sets in the `Apriori` [AS94] algorithm. To generate the set C_k from the set L_{k-1} (what corresponds to the *join* operation in the `Apriori`), the `VST` algorithm exploits the underlying VST data structure and makes use of the suffix links and the parent-child relationships.

`Ascend` The `Ascend` algorithm handles the monotonic constraint $\mathcal{M}$ and works upwards. It starts with the set F_0 containing the leaf nodes of the VST, constructed by the `Descend` phase (the leaf nodes have $\oplus$ labels). The constraint $\mathcal{M}$ is evaluated for the string patterns $\mathsf{string}(\mathcal{N})$, such that $\mathcal{N} \in F_0$. The nodes $\mathcal{N}$ in the set F_0, such that $\mathsf{string}(\mathcal{N})$ do not satisfy the constraint $\mathcal{M}$, are labeled with $\ominus$. Due to the monotonicity property, all their ancestors are also labeled with $\ominus$ (one can stop the upward labelling with $\ominus$, when the ancestor that is already marked with $\ominus$ is reached: it means that the ancestors from that point upwards was already labeled with $\ominus$ by some other leaf node sharing the ancestors with the current node). The nodes $\mathcal{N}$ in the set F_0, such that $\mathsf{string}(\mathcal{N})$ satisfy the constraint $\mathcal{M}$, guard their $\oplus$ labels. Their parent nodes are put into the set F_1 and treated in the next iteration. The process is continued until the set F_k is empty.

After the steps `Descend` and `Ascend`, both the anti-monotonic constraint $\mathcal{A}$ and the monotonic constraint $\mathcal{M}$ are handled. The VST branches, containing only the nodes labeled with $\ominus$, can be pruned by a simple tree traversal. The resulting pruned labeled suffix trie represents the version space of the string patterns that are solutions to the constraint $\mathcal{A} \wedge \mathcal{M}$.

3.2.2.2 FAVST

As the algorithm `VST`, the algorithm `FAVST`[4], introduced in [DD04], constructs a VST, containing the string patterns that are solutions to the constraint of the form $\mathcal{A} \wedge \mathcal{M}$. Differently from `VST`, `FAVST` is not a level-wise algorithm. Instead, it takes advantage of the well-studied string processing principles, specifically, the suffix tree construction [Ukk95, Wei73]. The particularity of the `FAVST` algorithm w.r.t. the `VST` algorithm, is that it scans the data set only once, instead of at most $2m$ scans, required by `VST`, where m is the length of the longest string pattern, satisfying the anti-monotonic constraint $\mathcal{A}$. Due to this property, `FAVST` is scalable in a distributed environment, where a data set access is slow (e.g. over the Internet or an external disk). On the other hand, `FAVST` is $O(|\mathcal{D}|^2)$ in size, where $|\mathcal{D}|$ is a total number of symbols in the data set $\mathcal{D}$, and thus is less space-efficient than `VST`.

The algorithm `FAVST` is designed to solve the constraints that are conjunctions of frequency constraints and of a maximal length constraint:

$$\mathcal{A} \wedge \mathcal{M} = \wedge_{i=1}^{n} \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr_i, \mathcal{D}_i) \wedge \mathsf{MaxLength}_{\phi}(maxLen) \\ \wedge_{i=1}^{n} \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr_i, \mathcal{D}_i)$$

`FAVST` is shown in Algorithm 1. To simplify the pseudo-code, the `FAVST` is written as if one must specify the minimum and maximum frequency constraints for each string data set $\mathcal{D}_i$. This is not the case, and the minimum frequency constraint $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr_i, \mathcal{D}_i)$ can be eliminated by setting the threshold $minFr_i$ value equal to 0. Similarly, a maximum frequency constraint $\mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr_i, \mathcal{D}_i)$ can be eliminated by setting the threshold $maxFr_i$ value equal to ∞. Concerning other constraints, the evaluation of the syntactic constraints (see Section 2.2.1.2 in Page 35) do not require a data set scanning, and can be performed efficiently by a simple tree traversal and adequate labelling. The evaluation of other data-dependent constraints (see Section 2.2.1.1 in Page 34) would require some constraint-specific modifications.

Algorithm 1 `FAVST`

```
Require: Data sets $\mathcal{D}_1, \ldots, \mathcal{D}_n$, alphabet $\Sigma_\phi$, thresholds $minFr_1, \ldots, minFr_n$,
    $maxFr_1, \ldots, maxFr_n$, $maxLen$
Ensure: VST containing the solution patterns (labelled with $\oplus$) to the constraint $\mathcal{A} \wedge \mathcal{M}$
  VST $\leftarrow$ InitTree( $\mathcal{D}_1$, $\Sigma_\phi$, $minFr_1$, $maxLen$, $maxFr_1$ )
  Prune the VST branches that contain only the nodes labeled with $\ominus$
  for $i = 2$ to $n$ do
    CountAndUnmark( $\mathcal{R}$, $\mathcal{D}_i$, $\Sigma_\phi$, $minFr_i$, $maxFr_i$ )
    Prune the VST branches that contain only the nodes labeled with $\ominus$
  end for
```

[4]`FAVST` stands for Finite Automata-based VST construction. The idea behind the algorithm `FAVST` is that a suffix trie can be treated as a deterministic automata so that one can efficiently visit all the substring patterns, contained in a data string.

`FAVST` The algorithm `FAVST` starts by calling the `InitTree` algorithm to construct an initial VST. `InitTree` handles the frequency and maximum length constraints during the scan of the $\mathcal{D}_1$. Then, to reduce the VST, it prunes the unnecessary branches, i.e., the branches that contain only the nodes labeled with $\ominus$. After, the `CountAndUnmark` algorithm is invoked to process the frequency constraints on the remaining string data sets. Note that `CountAndUnmark` do not grow the VST[5], it only counts the frequency of the patterns in the string data set $\mathcal{D}_i$ and label with $\ominus$ the nodes, representing the patterns that do not satisfy the corresponding frequency constraints. Again, the branches containing only the nodes labeled with $\ominus$ are pruned after scanning each $\mathcal{D}_i$, in order to reduce the number of patterns that need to be processed in the subsequent iteration.

The maximum length constraint threshold $maxLen$ specifies an upper bound of the length of the substring patterns, and thus of the VST depth. Therefore the value of $maxLen$ highly affects the `FAVST` time and space efficiency.

Each algorithm, `InitTree` and `CountAndUnmark`, scans the corresponding string data set $\mathcal{D}_i$ only once. Thus, if the data sets $\mathcal{D}_i$ are disjoint, the algorithm `FAVST` scans the whole data set $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$ only once, and is a *single-scan* algorithm.

`InitTree` The algorithm `InitTree` builds a VST data structure. It is given in Algorithm 2. `InitTree` scans a string data set symbol by symbol in consecutive order and constructs a VST by puting these symbols on the edges of nodes. Processing symbol by symbol we go down the trie thereby increasing the length of a string pattern, represented by the current node $\mathcal{N}$. To push the maximum length constraint $\mathsf{MaxLength}_{\phi}(maxLen)$, the algorithm, before growing down a trie, checks whether the depth of the current node $\mathcal{N}$, i.e., the length of the $\mathsf{string}(\mathcal{N})$, does not exceed the threshold $maxLen$. If it does, the algorithm backs up by following the suffix link. If a current node $\mathcal{N}$ does not have a child with an edge, containing the current symbol, such child node c is created with a algorithm `CreateChild`, so that the current symbol can be added to VST. Then the algorithm `IncreaseFrequency`, given in Algorithm 4, increments the frequency count for the $\mathsf{string}(c)$, and for all its suffixes (i.e., its substrings). Then, continuing from that child node c, which becomes the current node $\mathcal{N}$, the new iteration process the next symbol read in a string data set. When `InitTree` reads a symbol that do not belong to the alphabet Σ_{ϕ} (including a newline symbol, indicating the end of a current data string), the following trie traversal restarts from the root node $\mathcal{R}$.

Take notice that `InitTree` is a classic algorithm to construct a suffix tree, with

[5]The string pattern ϕ that do not belong to the VST constructed by `InitTree`, do not satisfy the constraint $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr_1, \mathcal{D}_1) \wedge \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr_1, \mathcal{D}_1) \wedge \mathsf{MaxLength}_{\phi}(maxLen)$ can not satisfy the constraint $\mathcal{A} \wedge \mathcal{M} = \wedge_{i=1}^{n} \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr_i, \mathcal{D}_i) \wedge \mathsf{MaxLength}_{\phi}(maxLen) \wedge_{i=1}^{n} \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr_i, \mathcal{D}_i)$ and needs not to be considered any further.

the following modifications:

- the frequency of indexed substrings is counted during the trie construction;
- the maximum length constraint $\mathsf{MaxLength}_{\phi}(maxLen)$ is pushed, which means that the depth of the trie is bounded by the threshold $maxLen$ value;
- the trie construction restarts from the root $\mathcal{R}$ when uninteresting symbols (i.e., those that do not belong to the alphabet Σ_{ϕ}) are encountered;
- the algorithm handles a set of data strings instead of a concatenation of these strings. The approach of concatenating multiple strings into one by inserting the delimiter symbols is convenient for the theoretical analysis of complexity, however, in practice, it multiples the depth of a tree, and thus the memory consumption.

Once the VST is constructed, the algorithm `UnlabelVSTForFr`, given in Algorithm 5, is invoked to unlabel with $\ominus$ the nodes, corresponding to $\mathsf{string}(\mathcal{N})$ that do not satisfy the frequency constraints in the data set $\mathcal{D}_1$.

`CreateChild` The algorithm `CreateChild`, given in Algorithm 3, creates and initializes a new node, denoted c. To initialize a suffix link of the node c, one have to find in a VST a node that represents a suffix of $\mathsf{string(c)}$. Take notice that such suffix node can be found through a parent node. Let aw, where a is a symbol and w is a substring, be a string represented by a parent node. Then, the new child node that is to be created represents the string $aw\xi_i$. The suffix of $aw\xi_i$ is $w\xi_i$, which is represented by a child with edge ξ_i of the node representing the string w. As w is a suffix of aw, the node representing the string w is pointed by a suffix link of the c parent. Remark, that a node, representing the suffix of a newly created node c, does not necessarily already exists in a VST. In that case, it is created by recursively invoking `CreateChild`.

`IncreaseFrequency` The algorithm `IncreaseFrequency` increments the frequency count for the $\mathsf{string}(c)$, and for all its suffixes (i.e., all its substrings). In order to avoid a multiple count of the string pattern, occurring several time in the same data string[6], the data string identifier ID is recorded in a field "last-id" and checked before incrementing the frequency.

`UnlabelVSTForFr` The algorithm `UnlabelVSTForFr` traverses the VST to mark the nodes $\mathcal{N}$, such that $\mathsf{string}(\mathcal{N})$ do not satisfy the frequency constraints, with labels $\ominus$.

[6]Consider, e.g., a data string $S = \mathsf{agtgt}$ and the node representing the string pattern $\phi = \mathsf{gt}$.

Algorithm 2 `InitTree`

```
Require: Data set $\mathcal{D}_1$, alphabet $\Sigma_\phi$, thresholds $minFr_1$, $maxFr_1$, $maxLen$
Ensure: VST containing the solution patterns (labelled with $\oplus$) to the constraint
  $\mathsf{MinFr}_\phi^{\sqsubseteq,S}(minFr_1,\mathcal{D}_1) \wedge \mathsf{MaxFr}_\phi^{\sqsubseteq,S}(maxFr_1,\mathcal{D}_1) \wedge \mathsf{MaxLength}_\phi(maxLen)$
  Create VST with the root node $\mathcal{R}$
  $\mathcal{R}$.suffix ← $\top$, $\mathcal{R}$.label ← $\oplus$, $\mathcal{R}$.count ← 0, $\mathcal{R}$.last-id ← undefined
  for all string $S \in \mathcal{D}_1$ with unique id ID do
    $\mathcal{N} \leftarrow \mathcal{R}$
    for all symbol $\xi_i \in S$ do
      if $\xi_i \in \Sigma_\phi$ then {$\xi_i$ is an interesting symbol}
        if depth of $\mathcal{N} \geq maxLen$ then {check for the $\mathsf{MaxLength}_\phi(maxLen)$ }
          $\mathcal{N} \leftarrow \mathcal{N}$.suffix
        end if
        if $\mathcal{N}$ has a child with edge $\xi_i$ then
          $c \leftarrow \mathcal{N}$.child with edge $\xi_i$
        else
          $c \leftarrow$ CreateChild($\mathcal{N}$,$\xi_i$)
          $c$.edge ← $\xi_i$ {set edge of a new child node}
          Add $c$ to $\mathcal{N}$ children
        end if
        IncreaseFrequency($c$, ID) {increase frequency}
        $\mathcal{N} \leftarrow c$ {current node gets one level deeper}
      else {$\xi_i$ is an uninteresting symbol}
        $\mathcal{N} \leftarrow \mathcal{R}$ {break input string, continue from the root}
      end if
    end for
  end for
  UnlabelVSTForFr($\mathcal{R}$, $minFr_1$, $maxFr_1$ )
```

Algorithm 3 `CreateChild`

```
Require: Parent node $\mathcal{N}$, edge symbol $\xi_i$ for the child node
Ensure: The newly created node $c$ with the symbol $\xi_i$ on its edge
  Create node $c$
  $c$.label ← $\oplus$, $c$.count ← 0, $c$.edge ← undefined, $c$.last-id ← undefined
  $sn \leftarrow \mathcal{N}$.suffix
  if $\mathcal{N}$ is a root node $\mathcal{R}$ then
    $c$.suffix ← $\mathcal{N}$
  else
    if $sn$ has a child $sc$ with edge $\xi_i$ then
      $c$.suffix ← $sc$
    else {recursively create the necessary suffix nodes}
      $sc \leftarrow$ CreateChild($sn$,$\xi_i$)
      $sc$.edge ← $\xi_i$ {set edge of a new child node}
      Add $sc$ to $sn$ children
      $c$.suffix ← $sc$
    end if
  end if
```

Algorithm 4 `IncreaseFrequency`

```
Require: Node $c$, ID
Ensure: Frequency for string(c) and all its substrings is increased by 1
  $x \leftarrow c$
  while ($x$ is not a root node $\mathcal{R}$) $\wedge$ ($x$.last-id $\neq$ ID) do
    {if it is not root suffix and its frequency not increased yet}
    $x$.count $\leftarrow x$.count $+ 1$
    $x$.last-id $\leftarrow$ ID
    $x \leftarrow x$.suffix {also count all $c$ suffixes}
  end while
```

Also it prepares the VST for the subsequent calls for `CountAndUnmark` by resetting the data set specific information, contained in the nodes.

Algorithm 5 `UnlabelVSTForFr`

```
Require: VST root node $\mathcal{R}$, $minFr$, $maxFr$
Ensure: Nodes $\mathcal{N}$, such that string($\mathcal{N}$) does not satisfy $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, \mathcal{D}_i) \wedge$
  $\mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr, \mathcal{D}_i)$, arelabeled with $\ominus$
  for all $\mathcal{N}$ in VST do {traverse the VST and unlabel}
    if $\mathcal{N}$.count $< minFr \vee \mathcal{N}$.count $> maxFr$ then
      $\mathcal{N}$.label $\leftarrow \ominus$
    end if
    {reset VST for the subsequent CountAndUnmark traversal}
    $\mathcal{N}$.count $\leftarrow 0$
    $\mathcal{N}$.last-id $\leftarrow$ undefined
  end for
```

`CountAndUnmark` The algorithm `CountAndUnmark`, given in Algorithm 6, takes as input the VST whose nodes have the unset values of "count" and "last-id". It counts the frequency in a data set $\mathcal{D}_i$ of the string patterns, present in that given VST, and set the label $\ominus$ to the nodes, such that string($\mathcal{N}$) does not satisfy the given frequency constraints. `CountAndUnmark` is similar to the the algorithm `InitTree`, with the difference that `CountAndUnmark` does not create any nodes. If a substring pattern, present in a data set $\mathcal{D}_i$, is not in the VST, it means that this pattern does not satisfy the constraints on the data sets processed previously, and thus it will not satisfy the whole conjunctive constraint $C = \mathcal{A} \wedge \mathcal{M}$ and does not belong to the solution set. Therefore, there is no need to create a node, representing such string pattern. Take notice that the maximum length constraint $\mathsf{MaxLength}_{\phi}(maxLen)$ was pushed by the `InitTree` sub-algorithm, and thus it does not need to be considered anymore. As in `InitTree`, the frequency counting and, afterwards, the node unlabelling are performed by `IncreaseFrequency` and `UnlabelVSTForFr` algorithms.

We discussed about generic solvers only, and we do not address the design of ad-hoc solvers (for specific constraints) at purpose.

Algorithm 6 `CountAndUnmark`

```
Require: VST root node $\mathcal{R}$, data set $\mathcal{D}_i$, alphabet $\Sigma_\phi$, thresholds $minFr_i$, $maxFr_i$
Ensure: VST in which the patterns $\phi$ that are not solutions to the constraint
  $\mathsf{MinFr}_\phi^{\sqsubseteq,S}(minFr_i, \mathcal{D}_i) \wedge \mathsf{MaxFr}_\phi^{\sqsubseteq,S}(maxFr_i, \mathcal{D}_i)$ arelabeled with $\ominus$
  for all string $S \in \mathcal{D}_i$ with unique id $\mathsf{ID}$ do
    $\mathcal{N} \leftarrow \mathcal{R}$
    for all symbol $\xi_i \in S$ do
      if $\xi_i \in \Sigma_\phi$ then {$\xi_i$ is an interesting symbol}
        while ($\mathcal{N}$ is not a root node $\mathcal{R}$) $\wedge$ ($\mathcal{N}$ has no child with edge $\xi_i$) do
          {current substring pattern is not in VST}
          $\mathcal{N} \leftarrow \mathcal{N}$.suffix {try a suffix (a shorter string)}
        end while
        if $\mathcal{N}$ is not a root node $\mathcal{R}$ then {if a good child (possibly in a suffix) was found}
          $c \leftarrow \mathcal{N}$.child with edge $\xi_i$
          $x \leftarrow c$
          IncreaseFrequency($c$) {increase frequency}
          $\mathcal{N} \leftarrow c$ {current node gets one level deeper}
        end if
      else {$\sigma_i$ is an uninteresting symbol}
        $\mathcal{N} \leftarrow \mathcal{R}$ {break input string, continue from the root}
      end if
    end for
  end for
  UnlabelVSTForFr($\mathcal{R}$, $minFr_i$, $maxFr_i$ )
```

Chapter 4

Fault-Tolerance Expressed by Constraints

Motivation

The approaches to needed fault-tolerance when searching for regularities in strings comes from different research domains, such as approximate string matching, bioinformatics and data mining. They are formulated and presented in a domain specific manner and terms. This presents a serious impediment to knowledge transfer and exchange, and prohibits the cross-fertilisation among these domains. Our contribution consists of formalizing the different approaches to fault-tolerance when mining strings in the unified constraint-based mining terms thereby putting them into the inductive databases framework.

4.1 Introduction

It is common that real life data contains errors due to technological issues concerning data collection, storage and transmission. In some application domains they may be also due to somewhat exploratory alphabet design. For example, data available as numerical time series can be analyzed by means of algorithms for substrings, provided that the data is discretized and thus encoded as a sequence of events in a "computed" alphabet. Also, data representing real world phenomenon is often intrinsically degenerated[1], e.g., many variants of string patterns in DNA sequences

[1]A term of degeneracy has different meanings in mathematics, physics, medicine. In this manuscript, we employ a term of degeneracy as it is used in biological systems context, i.e., to express the ability of elements that are structurally different to perform the same function or yield the same output [EG01]

can be binding sites for the same transcription factor or many web site browsing sequences can lead to the accomplishment of the same task. To capture knowledge when working with such data, a fault-tolerance is needed.

A motivation for fault-tolerance came from many application domains. One of the classic application areas is *signal processing.* For example, in speech recognition, the parts of the signal may be compressed in time and/or not pronounced, and to discern a word, a perfect matching approach is practically inoperative. In *error correction* domain, to ensure the correct transmission over a channel, it is necessary to be able to recover the correct message after a possible error introduced during the transmission. Another one application domain is *text and information retrieval.* Approximate matching is one of the basic tools for correcting and recognizing the mistyped words. One of the largest application areas remains *computational biology.* Biological sequences can be considered as strings over specific alphabets, e.g., DNA sequences can be seen as strings over a four letter nucleotide alphabet and protein sequences can be seen as strings over a twenty letter amino acids alphabet. Exact matching is rarely convenient when analyzing biological sequences, since they typically contains errors due to sequencing technologies and, in addition to this, they are known to be intrinsically degenerated. A recent and rapid development of computational biology and considerable effort put in bioinformatics[2] have resulted in a number of interesting approaches to tackle the needed tolerance for errors. The "search allowing errors" is a fundamental operator for numerous problems, e.g., DNA sequence reconstruction by aligning and merging its fragments, looking for functional parts in DNA sequences, establishing how different two sequences are in order to reconstruct the phylogenetic trees[3]. A major part of approaches reviewed in this chapter were inspired when trying to answer a particular class of biological problems, but their are not tied to any particular problem and can be directly applied in a variety of application domains.

We concentrate on *deterministic* on the contrary to *probabilistic* approaches[4] that employ correct and complete strategies to extract fault-tolerant patterns in string data. Thus we do not cover non-deterministic approaches, such as Hidden Markov Models (HMMs), Bayesian networks, weight matrices, profiles, etc.

From constraint-based data mining point of view, the notion of fault-tolerance when mining string data implicitly embraces two distinct problems. The first one, which will be presented in Section 4.2, is to find patterns ϕ that are *similar* to a given

[2]Bioinformatics refers to the conception and advancement of algorithms, computational and statistical techniques and theory to solve formal and practical problems arising from the management and analysis of biological data.

[3]A phylogenetic tree, also called an evolutionary tree, is a tree showing the evolutionary relationships among various biological species or other entities that are believed to have a common ancestor.

[4] Deterministic patterns are patterns that either match or do not match a given object, i.e., their *match* functions return true or false values. Probabilistic patterns assigns a probability to the match between a pattern and a given object [BJEG98a].

entity of reference. The second problem, which will be presented in Section 4.3, is to find unknown patterns ϕ that capture *soft regularities*, i.e., not exactly repeating regularities. This problem can be formulated as a problem of extracting patterns satisfying a minimum frequency constraint, when a *match* function $match(\phi, X)$ evaluates true for a set of objects X that are *similar* to ϕ. Note that a *relation of similarity*, either between two patterns, or between a pattern and an object, is present in the formulation of both problems and is central when handling fault-tolerance.

4.2 Similarity Constraint

The problem of revealing elements in data that are somehow similar to a given entity of reference, in constraint-based data mining terms can be formulated as a task of extraction under a similarity constraint.

Definition 4.1 (Similarity Constraint) *Let ϕ be a pattern from pattern language $\mathcal{L}$ and ε be some entity. Define a similarity constraint $\mathsf{Sim}_\phi(\varepsilon)$ to evaluate true if and only if ϕ and ε are in a similarity relation.*

Remark 4.1 *Note, that the similarity constraint is a syntactic constraint.*

The focus of this section is the similarity constraint on patterns in string data and the various relations of similarity, on which such constraint is based.

We reformulate various approaches, including those coming from approximate string matching domain, in the terms of constraint-based data mining. These approaches can be generalized to the pattern extraction under a conjunction of the similarity constraint $\mathsf{Sim}_\phi(\varepsilon)$ and the minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ (see Definition 1.20 in Page 21). These approaches differ in a way they instantiate a pattern language $\mathcal{L}$, an entity ε and a similarity relation.

4.2.1 Approximate String Matching

A problem of approximate string matching is to locate in a data string $\mathcal{D}$[5] all soft-occurrences ϕ of a given string ψ[6],i.e., allowing a limited number of *errors* in the match. Put in constrained-based data mining terms, this means to extract the patterns ϕ that satisfy a conjunction of constraints $\mathsf{Sim}_\phi(\psi) \wedge \mathsf{MinFr}_\phi(1, \mathcal{D})$, where ϕ and ψ are from the same string pattern language $\mathcal{L}_\Sigma$.

[5]In approximate string matching domain it is often referred as *sequence* or *text*.

[6]In approximate string matching domain it is often referred as *pattern*.

A classical way to measure a similarity between two strings ϕ and ψ is to compute their *distance*. The distance between strings is intended to be small when one of the strings is likely to be an erroneous variant of the other. In approximate string matching domain a notion of edit *operation* on string is central when defining distance. The most commonly used operations of *insertion*, *deletion*, *substitution* of a symbol and a *transposition* of two symbols were distinguished in [Dam64]. Then the distance is a minimal number of such operations necessary to perform on a string ϕ in order to obtain a string ψ. Once an operation has converted a substring σ into σ', no further operations can be done on this σ'. This restriction forbids editing the same symbol or substring many times, and thus renders the distance computable.

In many applications, where certain errors on data are more likely to occur and/or are not equally penalizing, it is preferable to assign *weights* to operations. These weights are typically referred to as *costs*, or in other words, penalties, of performing the corresponding operations. Then the distance between two strings ϕ and ψ is the minimal cost of a sequence of operations necessary to perform on a string ϕ in order to obtain a string ψ, where the cost of a sequence of operations is the sum of the costs of the individual operations.

The weights associated to the substitutions can be represented in a form of a matrix, called the *substitution matrix*. Substitution matrices are commonly used in the context of biological sequences (amino acid or nucleotide sequences) alignments, where weights describe the rate at which one symbol in a sequence changes to another symbol over time. Note that the weight of substituting a symbol ξ_i by a symbol ξ_j can measure their dissimilarity, as well as similarity. In the first case a substitution matrix is referred to as a *dissimilarity matrix*, and in the latter case it is referred to as a *similarity matrix*. When substitution weights measure a similarity, they are often referred to as *scores*. Then, having assigned so called *gap penalties* for insertions/deletions, instead of distance we rather compute a similarity or, in other words, a proximity between two strings ϕ and ψ, that is defined as the maximal score of a sequence of operations necessary to perform on a string ϕ in order to obtain a string ψ. Well known examples of similarity matrices for amino acids are `PAM250` [DSO78] and `BLOSUM62` [HH92]. `BLOSUM62` is used as a default scoring matrix in `BLAST 2.0` program from the famous `BLAST` [AGM+90] algorithms family.

For a survey on approximate string matching see, e.g., [Nav01]. In the following sections we overview several well-known symbol-based distances that are used to establish a similarity relation, i.e., given a threshold $k \in \mathbb{N}$, two strings σ^1 and σ^2 are *similar* if and only if their distance is less or equal to k. When computing these distances, except the episode distance (described in Section 4.2.1.3), one can assume that operations are made on either of the strings. There are many approaches to model the needed similarity through a notion of distances, and the overview, presented in this manuscript, is not extensive. Among the not covered distances are inversion distance [KS95], which allows to reverse substrings, block distance [Tic84, EH88, Ukk92, DLT94], which allows to rearrange and permute substrings, q-gram distance [Ukk92], which is based on common q-length substrings and distance allowing swaps [AAL$^+$00, LKPC97].

4.2.1.1 Hamming Distance Similarity Constraint

Hamming distance was introduced by Richard Hamming in his paper on Hamming codes [Ham50].

Definition 4.2 (Hamming distance) *Hamming distance of two strings σ^1 and σ^2, denoted $HammingDist(\sigma^1, \sigma^2)$, is the minimal number of substitutions, necessary to transform a string σ^1 into a string σ^2.*

Remark 4.2 *Note that the Hamming distance of two strings σ^1 and σ^2 is finite if and only if $|\sigma^1| = |\sigma^2|$. In this case it is metric, and it holds $0 \leq HammingDist(\sigma^1, \sigma^2) \leq |\sigma^1| = |\sigma^2|$.*

The Hamming distance between two strings of equal length can be reformulated as the number of positions for which the corresponding symbols are different or as the number of mismatches between two strings.

Two strings are said to be in a Hamming distance similarity relation, if their Hamming distance does not exceed a given threshold.

Definition 4.3 (Hamming Distance Similarity Relation) *Let σ^1 and σ^2 be two strings over a given alphabet Σ, and $k \in \mathbb{N}$ be a threshold. The strings σ^1 and σ^2 are in a Hamming distance similarity relation, denoted $sim_{Hdist}(\sigma^1, \sigma^2, k)$, if and only if $HammingDist(\sigma^1, \sigma^2) \leq k$.*

Hamming distance similarity constraint is defined for string patterns and is based on the Hamming distance similarity relation.

Definition 4.4 (Hamming Distance Similarity Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a string, and $k \in \mathbb{N}$ be a threshold. Define a Hamming distance similarity constraint, denoted $\mathsf{HammingDistSim}_\phi(\sigma, k)$ to evaluate true if and only if ϕ and σ are in the Hamming distance similarity relation $sim_{Hdist}(\phi, \sigma, k)$.*

Given a data string $\mathcal{D}$, a string σ and a threshold $k \in \mathbb{N}$, the problem of extracting string patterns ϕ satisfying the conjunction of constraints $\mathsf{HammingDistSim}_\phi(\sigma, k) \wedge \mathsf{MinFr}_{\overline{\phi}}^{\sqsubseteq, s}(1, \mathcal{D})$, in the approximate string matching domain is known as *string matching with k mismatches.*

4.2.1.2 Edit Distance Similarity Constraint

Edit distance was introduced by Vladimir Levenshtein in [Lev65]. After his name, it is also known as Levenshtein distance.

Definition 4.5 (Edit or Levenshtein Distance) *Edit or Levenshtein distance of two strings σ^1 and σ^2, denoted $EditDist(\sigma^1, \sigma^2)$, is the minimal number of insertions, deletions and substitutions, necessary to transform a string σ^1 into a string σ^2.*

Remark 4.3 *The edit distance is a metric, and we have $0 \leq EditDist(\sigma^1, \sigma^2) \leq max(|\sigma^1|, |\sigma^2|)$.*

Then, two strings are said to be in a edit distance similarity relation, if their edit distance does not exceed a given threshold.

Definition 4.6 (Edit Distance Similarity Relation) *Let σ^1 and σ^2 be two strings over a given alphabet Σ, and $k \in \mathbb{N}$ be a threshold. The strings σ^1 and σ^2 are in an edit distance similarity relation, denoted $sim_{Edist}(\sigma^1, \sigma^2, k)$, if and only if $EditDist(\sigma^1, \sigma^2) \leq k$.*

Edit distance similarity constraint is defined for string patterns and is based on the edit distance similarity relation.

Definition 4.7 (Edit Distance Similarity Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a string, and $k \in \mathbb{N}$ be a threshold. Define an edit distance similarity constraint, denoted $\mathsf{EditDistSim}_\phi(\sigma, k)$ to evaluate true if and only if ϕ and σ are in the edit distance similarity relation $sim_{Edist}(\phi, \sigma, k)$.*

Edit distances have been studied extensively. Given a data string $\mathcal{D}$, a string σ and a threshold $k \in \mathbb{N}$, the problem of extracting string patterns ϕ, satisfying a conjunction of constraints $\mathsf{EditDistSim}_\phi(\sigma, k) \wedge \mathsf{MinFr}_\phi^{\sqsubseteq, s}(1, \mathcal{D})$, in the approximate string matching domain is known as *string matching with k differences.*

If the different operations have different costs or if the costs depend on the involved symbols, then we talk of a *generalized edit distance.*

Definition 4.8 (Generalized Edit Distance) *The generalized edit distance of two strings σ^1 and σ^2, denoted $GenEditDist(\sigma^1, \sigma^2)$, is the minimal cost of a sequence of insertions, deletions and substitutions that are necessary to perform on σ^1 in order to transform it into σ^2, where the cost of a sequence of operations is the sum of the costs of individual operations.*

Remark 4.4 *If the cost of all operation is equal to* 1*, then generalized edit distance is equivalent to edit distance.*

Edit distance has gained a lot of popularity since its generalized version is powerful enough to implement the needed fault-tolerance in wide range of applications. The choice of pertinent costs or scores of operations is a problem of its own, and learning it is one of the possible approaches (see, e.g., [OS06] for a recent development in this area). Despite of the fact that most approximate string matching algorithms focus on the edit distance, many of them can be easily extended to the generalized edit distance. Also, many algorithms that are designed for edit distance can be adapted to other distances that use either a subset of operations allowed by edit distance, such as Hamming distance (see the previous Section 4.2.1.1) or longest common subsequence distance (see Section 4.2.1.4), or a superset of operations allowed by edit distance, such as Damerau-Levenshtein distance that allows all four edit operations of insertion, deletion, substitution and transposition.

4.2.1.3 Episode Distance Similarity Constraint

Episode distance [DFG+97] was designed for finding sequences of events, all of them occurring within a short period of time in a sequence of events.

Definition 4.9 (Episode distance) *Episode distance of two strings σ^1 and σ^2, denoted $EpisodeDist(\sigma^1, \sigma^2)$, is the minimal number of insertions necessary to perform on a string σ^2 in order to transform it into a string σ^1.*

Remark 4.5 *Note that episode distance between two strings σ^1 and σ^2 can be finite if and only if $|\sigma^2| \leq |\sigma^1|$. Also note that even if $|\sigma^2| \leq |\sigma^1|$ it might be not possible to convert σ^2 into σ^1 by performing only the insertions in σ^2. Therefore it holds that either $EpisodeDist(\sigma^1, \sigma^2) = |\sigma^1| - |\sigma^2|$ or $EpisodeDist(\sigma^1, \sigma^2) = \infty$. Episode distance is not a metric.*

Two strings are said to be in an episode distance similarity relation, if their episode distance does not exceed a given threshold.

Definition 4.10 (Episode Distance Similarity Relation) *Let σ^1 and σ^2 be two strings over a given alphabet Σ, and $k \in \mathbb{N}$ be a threshold. The strings σ^1 and σ^2 are in an episode distance similarity relation, denoted $sim_{Epdist}(\sigma^1, \sigma^2, k)$, if and only if $EpisodeDist(\sigma^1, \sigma^2) \leq k$.*

Episode distance similarity constraint is defined for string patterns and is based on the episode distance similarity relation.

Definition 4.11 (Episode Distance Similarity Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a string, and $k \in \mathbb{N}$ be a threshold. Define an episode distance similarity constraint, denoted $\mathsf{EpisodeDistSim}_\phi(\sigma, k)$ to evaluate* true *if and only if ϕ and σ in the episode distance similarity relation $sim_{Epdist}(\phi, \sigma, k)$.*

Given a data string $\mathcal{D}$ and a string σ, the problem of extracting patterns ϕ, satisfying the conjunction of constraints $\mathsf{EpisodeDistSim}_\phi(\sigma, k) \wedge \mathsf{MinFr}_\phi^{\sqsubseteq, s}(1, \mathcal{D})$, where $k \in \mathbb{N}$ is either a given threshold or a minimum possible value, in the approximate string matching domain is known as *episode matching.*

4.2.1.4 Longest Common Subsequence (LCS) Distance

The problem of longest common subsequence is a classical computer science problem, which consists of finding a maximum length subsequence of several (usually two) strings. A famous application of this problem is the `diff` utility to compare two files, developed in the early 1970s on the `Unix` operating system. It is also applied in the bioinformatics domain, see, e.g., [NW70b]. A study concerning longest common subsequences can be found in, e.g., [Apo97].

Definition 4.12 (Longest Common Subsequence (LCS)) *In the following, we provide two definitions of the longest common subsequence (LCS) of two strings σ^1 et σ^2, denoted $LCS(\sigma^1, \sigma^2)$. The second definition is a constructive one.*

1. *A subsequence of a string σ^1 is any sequence Φ that can be obtained by deleting zero or more (not necessarily consecutive) symbols from σ^1 (see Definition 1.11 in Page 18 for a formal definition of sequence and subsequence). Φ is a LCS of the strings σ^1 and σ^2, if it is a subsequence of σ^1, a subsequence of σ^2, and its length is maximal.*

2. *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 than 2, then an optimal sequence of edit operations will always avoid substitutions and produce σ^2 from σ^1 only by means of insertions and deletions. Then the pairs of matching symbols (i.e., symbols on which neither insertion nor deletion was applied) in σ^1 and σ^2 is the longest pairing of symbols that can be made between both strings, preserving the symbol order. It is called the LCS of strings σ^1 and σ^2.*

Remark 4.6 *Given two strings σ^1 and σ^2, their LCS is in general not unique.*

Now we can define the LCS distance.

Definition 4.13 (**LCS Distance**) *Consider the minimal sequence of insertions and deletions that are necessary to produce on a string σ^1 in order to obtain a string σ^2. Then the number of unpaired symbols, i.e. the symbols on which either insertion or deletion was applied, is the longest common subsequence distance between σ^1 and σ^2, denoted $LCSDist(\sigma^1, \sigma^2)$.*

Remark 4.7 *Given an $LCS(\sigma^1, \sigma^2)$, the LCS distance $LCSDist(\sigma^1, \sigma^2) = |\sigma^1| + |\sigma^2| - 2 * |LCS(\sigma^1, \sigma^2)|$.*

Remark 4.8 *The LCS distance is a metric, and we have $0 \leq LCSDist(\sigma^1, \sigma^2) \leq |\sigma^1| + |\sigma^2|$.*

Two strings are said to be in a similarity relation, if their LCS distance does not exceed a given threshold.

Definition 4.14 (**LCS Distance Similarity Relation**) *Let σ^1 and σ^2 be strings over a given alphabet Σ, and $k \in \mathbb{N}$ be a threshold. The strings σ^1 and σ^2 are in a LCS distance similarity relation, denoted $sim_{LCSdist}(\sigma^1, \sigma^2, k)$, if and only if $LCSDist(\sigma^1, \sigma^2) \leq k$.*

The LCS distance similarity constraint is defined for string patterns and is based on the LCS distance similarity relation.

Definition 4.15 (**LCS Distance Similarity Constraint**) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a string, and $k \in \mathbb{N}$ be a threshold. Define a LCS distance similarity constraint, denoted $\mathsf{LCSDistSim}_\phi(\sigma, k)$ to evaluate true if and only if they are in the LCS distance similarity relation $sim_{LCSdist}(\phi, \sigma, k)$.*

One of the earliest usage of the LCS distance similarity constraint was to assess whether a significant homology[7] exists between two proteins [NW70b]. The idea is that a number of matches between two sequences, allowing all possible interruptions in either of them, must be large enough. Further algorithmic improvements concerning the computation of the LCS were proposed in [AG87].

4.2.2 Regular Expression Constraint

Regular expression constraint $\mathsf{MatchRE}_\phi(re)$ (see Definition 1.25 in Page 22) provides a powerful way to specify a set of searched patterns through concatenations, unions,

[7]In biology, homology generally signifies a similarity between subjects that is due to their shared ancestry.

and repetitions of simple strings and other subexpressions. Thus, it is convenient to specify a similarity constraint, when the nature of the needed fault-tolerance is well known and can be specified by a regular expression. Pattern extraction under a regular expression constraint was considered by at least two research domains: regular expression matching and sequential data mining.

The problem of regular expression matching is to find substrings in a data string $\mathcal{D}$ that match a given regular expression re. Put in constraint-based mining terms, this means to extract string patterns ϕ that satisfy a conjunction of constraints $\mathsf{MatchRE}_\phi(re) \wedge \mathsf{MinFr}_\phi(1, \mathcal{D})$. There are at least two fundamentally different families of algorithms that perform a regular expression matching in a string. The fastest algorithms ([MY60, Tho68] being among the first ones) relies on a result in a formal language theory that allows every nondeterministic finite automaton to be transformed into a deterministic finite automaton [RS59]. The other family of algorithms performs a regular expression matching using backtracking. The worst case time complexity of these algorithms is exponential, but they are simple to implement and allow the use of backreferences[8]. Despite of the lower time efficiency, many programming languages, e.g., `Perl`, `Python` and `Java`, have regular expression implementations based on the recursive backtracking.

Sequential data mining community has considered the problem of finding sequence (or string) patterns ϕ that satisfy a conjunction of constraints $\mathsf{MatchRE}_\phi(re) \wedge \mathsf{MinFr}_\phi(minFr, \mathcal{D})$ in a set of sequences (or strings) $\mathcal{D}$. The motivation is to allow users to express the family of sequential patterns they are interested in, and thereby to provide a user-controlled focus when mining frequent pattern in data sets of sequences, e.g. WWW logs, market-basket, telecommunications data. A regular expression constraint $\mathsf{MatchRE}_\phi(re)$ being, in general, neither anti-monotonic not monotonic, the `SPIRIT` algorithms [GRS99] exploit it to prune the search space by pushing deeply into the extraction phase various relaxed versions of $\mathsf{MatchRE}_\phi(re)$. The algorithm `RE-Hackle` [AB03] takes one step further by evaluating the trade-off between pruning by a minimum frequency constraint $\mathsf{MinFr}_\phi(\alpha, \mathcal{D})$ and a relaxed version of a regular expression constraint $\mathsf{MatchRE}_\phi(re)$, and thereby choosing the appropriate pruning strategy dynamically. Although both algorithms are intended for sequence pattern extraction, they can be equally used to extract string patterns in string data sets.

4.2.3 Similarity Constraint Based On Edit Score

An original approach to exploit a similarity constraint based on the edit operations scores is introduced in [CBM02b] and detailed in [Mas05]. The idea behind the proposed similarity relation resemble the one of the generalized edit distance (see

[8]A backreference stores the part of the string matched by the part of the regular expression so that it can be reused.

Definition 4.8 in Page 62), except that with the operations one associates rewarding scores and not penalizing costs, which are multiplied and not summed.

Definition 4.16 (Galibot Similarity Score) *Let the operations of insertion, deletion and substitution be associated with scores that depend on the position and the symbol on which they are applied. Let these scores be from the interval* $[0, 1]$ *and reward the similarity, i.e., the less penalizing an operation, the bigger an associated score. The Galibot similarity score*[9] *between strings* σ^1 *and* σ^2 *is denoted* $GalibotScore(\sigma^1, \sigma^2)$*. It is the maximal score of a sequence of operations necessary to perform on* σ^2 *in order to obtain* σ^1[10]*, where the score of the sequence of operations is the product of the scores of the operations.*

The similarity relation is established on the basis of this score.

Definition 4.17 (Galibot Similarity Relation) *Let* σ^1 *and* σ^2 *be two strings over a given alphabet* Σ*, and* $t \in \mathbb{N}$ *be a threshold. Strings* σ^1 *and* σ^2 *are in a Galibot similarity relation, denoted* $sim_{Galibot}(\sigma^1, \sigma^2, t)$*, iff* $GalibotScore(\sigma^1, \sigma^2) \geq t$*.*

Galibot similarity constraint is defined for string patterns and is based on the Galibot similarity relation.

Definition 4.18 (Galibot Similarity Constraint) *Let* $\mathcal{L}_\Sigma$ *be a string pattern language,* ϕ *be a pattern from* $\mathcal{L}_\Sigma$*,* σ *be a string, and* $t \in \mathbb{N}$ *be a threshold. Define a Galibot similarity constraint, denoted* $\mathsf{GalibotSim}_\phi(\sigma, t)$ *to evaluate* **true** *iff* ϕ *and* σ *are in the Galibot similarity relation* $sim_{Galibot}(\phi, \sigma, t)$*.*

The similarity constraint $\mathsf{GalibotSim}_\phi(\sigma, t)$, because its relaxation is based on a potential similarity of immediate prefix of ϕ, is a convertible anti-monotonic constraint, what allows to push it deeply into an extraction phase. This property is exploited by the `Galibot` solver [CBM02b], which extracts sequence or string patterns ϕ satisfying the constraint $\mathsf{GalibotSim}_\phi(\sigma, t) \wedge \mathsf{MinFr}_\phi(minFr, \mathcal{D})$ in a sequence or string set $\mathcal{D}$.

4.3 Fault-tolerant Patterns

In the previous sections, we presented various approaches to handle the problem of finding patterns in string data that are, according to some definition, similar to

[9]Named after the associated algorithm `Galibot`.

[10]Note that in this measure we cannot think that operations can be performed either on σ^1 or σ^2, since their scores depend on the positions on which they occur.

an entity of reference. However, in many situations, that entity of reference is not available. In this section we address the approaches, which tackle the problem of extracting the unknown *soft-regularities* in string data. By soft-regularities we mean regularities that are not exactly repeating, but approximately similar ones. The difficulty is how to formally specify a set of similar elements, present in a string data. We represent that set as a *fault-tolerant pattern*. In constraint-based data mining terms, we formulate the problem of fault-tolerant pattern discovery as the task of extracting the fault-tolerant patterns ϕ that satisfy a minimum frequency constraint with the *match* function that evaluates true for a set of objects, i.e., in our case, string or substring objects, that, correspondingly, contain or are the elements similar to ϕ[11]. From this property one may induce a relation of similarity between these elements themselves[12]. Therefore, in this section we focus on different languages of patterns allowing to express the approximate regularities and on similarity relations that implements the fault-tolerance through the corresponding *match* functions.

Even if Definition 1.20 for a pattern frequency (see Page 21) does not change for fault-tolerant patterns, i.e., it is the number of objects that pattern ϕ matches, we will call it a *soft-frequency* and denote it $\mathsf{SoftFr}(\phi, \mathcal{D})$, whenever the associated *match* function evaluates true for a set of objects, similar to ϕ. Subsequently, we call a minimum frequency constraint a *minimum soft-frequency constraint* and denote it $\mathsf{MinSoftFr}_\phi(minFr, \mathcal{D})$, whenever it concerns a soft frequency of a pattern ϕ. Similarly, we call a maximum frequency constraint a *maximum soft-frequency constraint* and denote it $\mathsf{MaxSoftFr}_\phi(maxFr, \mathcal{D}))$, when it concerns a soft frequency of a pattern ϕ. It is important to distinguish between the frequency and soft-frequency constraints, because they do not share the same properties, e.g., the soft-frequency constraints are not guaranteed to have the (anti-)monotonicity properties.

Correct and complete approaches to extract deterministic fault-tolerant patterns in string data have been seldom studied by the data mining community, while they constitute one of the core algorithmic issues in bioinformatics. The approaches, which we introduce in this section, come from bioinformatics research community and were conceived while trying to answer molecular biology problems. We reformulate them in constraint-based data mining terms so that they can be put in the inductive databases framework. In the bioinformatics literature it is common to use the term *motif* both to denote an object, present in string data, and the representation of the extracted regularity. Remind that we make a distinction between these two entities and denote the first one a (soft-)occurrence and the latter one a pattern.

[11] Except for clique patterns, presented in Section 4.3.2 in Page 72, where a similarity relation is used not to define a match function, but to construct a pattern.

[12]Remark that these elements are (soft-)occurrences of a pattern.

4.3.1 Fault-tolerant String Patterns

Fault-tolerance can be implemented through string patterns ϕ by acknowledging the substrings, which are not equal to ϕ, as its occurrences, called *soft-occurrences.* Thus the key difference between string patterns and fault-tolerant string patterns is the way in which a *match* function, enabling soft-matching, is defined.

4.3.1.1 Soft-matching through Hamming Distance Similarity Relation

The idea of discovering approximately similar regularities in a data string by means of soft-matching, defined through the Hamming distance similarity relation (see Definition 4.3 in Page 61), was introduced in [SEVS95a].

Definition 4.19 (Hamming Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, s be a substring object from $\mathcal{U}_\Sigma$, and $k \in \mathbb{N}$ be a threshold. Let ϕ be a pattern from a string pattern language $\mathcal{L}_\Sigma$.*

1. *Define a Hamming match function $match_{H,s}(\phi, s, k)$ to evaluate* true*, if and only if ϕ and s are in the Hamming distance similarity relation $sim_{Hdist}(\phi, s, k)$.*
2. *Define a Hamming match function $match_{H,S}(\phi, S, k)$ to evaluate* true*, if and only if here exists a string σ, such that $\sigma \sqsubseteq S$ and σ and ϕ are in the Hamming distance similarity relation $sim_{Hdist}(\phi, \sigma, k)$.*

Remark 4.9 *Note that the Hamming distance is a metric, and therefore, because of the triangular inequality, the Hamming distance similarity relation between the pattern ϕ and its soft-occurrences induces a similarity relation between the soft-occurrences themselves.*

Definition 4.20 (Hamming Soft-Frequency) *Frequency $\mathsf{Fr}(\phi, \mathcal{D})$ that is evaluated using the Hamming match function $match_{H,\{s,S\}}(\phi, \{s, S\}, k)$, is called an Hamming soft-frequency and is denoted $\mathsf{SoftFr}_{H,\{s,S\}}(\phi, \mathcal{D}, k)$.*

Definition 4.21 (Minimum/Maximum Hamming Soft-Frequency) *The minimum (resp. maximum) frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the Hamming match function $match_{H,s}(\phi, s, k)$ is called the minimum (resp. maximum) Hamming soft-frequency constraint. It is denoted $\mathsf{MinSoftFr}_\phi^{H,s}(minFr, \mathcal{D}, k)$ (resp. $\mathsf{MinSoftFr}_\phi^{H,s}(minFr, \mathcal{D}, k)$).*

The associated algorithm `H-Moivre` [SEVS95a], which is an extension of the `KMR` algorithm [KMR72, LAC89], solves the following problems:

1. extract all string patterns $\phi \in \mathcal{L}_\Sigma$ in a data string $\mathcal{D}$ that satisfy the constraint $\mathsf{MinSoftFr}_\phi^{H,s}(minFr, \mathcal{D}, k) \wedge \mathsf{MinLength}_\phi(minLen) \wedge \mathsf{MaxLength}_\phi(maxLen)$, where $minLen = maxLen$;
2. find the greatest length l_{max}, for which there exists at least one l_{max}-length string pattern ϕ that satisfies $\mathsf{MinSoftFr}_\phi^{H,s}(minFr, \mathcal{D}, k))$, i.e., the minimum Hamming soft-frequency constraint, and solve the problem 1 for $minLen = maxLen = l_{max}$.

4.3.1.2 Soft-matching through Edit Distance Similarity Relation

Similarly to the previously introduced approach that employs a Hamming distance similarity relation, the article [SEVS95a] also elaborates a soft-matching through the edit distance similarity relation (see Definition 4.6 in Page 62).

Definition 4.22 (Edit Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, s be a substring object from $\mathcal{U}_\Sigma$, and $k \in \mathbb{N}$ be a threshold. Let ϕ be a pattern from a string pattern language $\mathcal{L}_\Sigma$. Define an edit match function $match_{E,s}(\phi, s, k)$ to evaluate* true*, iff ϕ and s are in the edit distance similarity relation $sim_{Edist}(\phi, s, k)$.*

Remark 4.10 *The edit distance is a metric, and therefore, because of the triangular inequality, the edit distance similarity relation between the pattern ϕ and its soft-occurrences induces a similarity relation between the soft-occurrences themselves.*

Definition 4.23 (Minimum Edit Soft-Frequency Constraint) *The minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the edit match function $match_{E,s}(\phi, s, k)$ is called the minimum edit soft-frequency constraint, denoted $\mathsf{MinSoftFr}_\phi^{E,s}(minFr, \mathcal{D}, k)$.*

The corresponding correct and complete algorithm `L-Moivre` [SEVS95a], is also an extension of the `KMR` algorithm and solves the following problems:

1. extract all string patterns $\phi \in \mathcal{L}_\Sigma$ in a data string $\mathcal{D}$ that satisfy the constraint $\mathsf{MinSoftFr}_\phi^{E,s}(minFr, \mathcal{D}, k) \wedge \mathsf{MinLength}_\phi(minLen) \wedge \mathsf{MaxLength}_\phi(maxLen)$, where $minLen = maxLen$;
2. find the greatest length l_{max}, for which there exists at least one l_{max}-length string pattern ϕ that satisfies $\mathsf{MinSoftFr}_\phi^{E,s}(minFr, \mathcal{D}, k)$, i.e., the minimum edit soft-frequency constraint, and solve the problem 1 for $minLen = maxLen = l_{max}$.

4.3.1.3 Soft-matching through Word-based Similarity Relation

An alternative to symbol-based similarity between two strings is to use their substrings as units of comparison [SVS95]. In bioinformatics literature, the substrings used for the comparison are known as *words*, hence the name *word-based similarity*. The motivation behind a similarity defined by comparing words is that it allows to achieve greater sensitivity[13] when analyzing in biological sequences. The idea of using words as units of string comparison is also employed in the implementation of famous BLAST [AGM+90] program to perform pairwise comparisons between a query string and the strings in the database.

The relation between two words w^1 and w^2 is established by computing their score using a similarity matrix (see Page 60).

Definition 4.24 (Word Similarity Score) *Let $\Sigma = \{\xi_1, \ldots, \xi_n\}$ be an alphabet of symbols. Let M denote a similarity matrix $|\Sigma| \times |\Sigma|$ s.t. $M(\xi_i, \xi_j)$ is the score of similarity between symbols $\xi_i, \xi_j \in \Sigma$. Let $w^1 = w^1_1 \ldots w^1_{l_w}$ and $w^2 = w^2_1 \ldots w^2_{l_w}$ be two words of length l_w over the alphabet Σ. A similarity score of w^1 and w^2, denoted $SimScore(w^1, w^2)$, is the sum of the similarity scores of their corresponding symbols, i.e.,*

$$SimScore(w^1, w^2) = \sum_{i=1}^{l_w} M(w^1_i, w^2_i)$$

Then, two strings are considered similar, if the similarity score of their every l_w-length word is greater than a given threshold.

Definition 4.25 (Word-based Similarity Relation) *Let t be a similarity threshold and l_w be a word length. Let Σ be an alphabet on which a similarity matrix M is defined. Let σ^1 and σ^2 be two strings of equal length over Σ. The strings σ^1 and σ^2 are in a word-based similarity relation $sim_W(\sigma^1, \sigma^2, l_w, t)$, if and only if the word similarity score of their every l_w-length word is greater than t, i.e., if*

$$SimScore(\sigma^1_i \ldots \sigma^1_{i+m-1}, \sigma^2_i \ldots \sigma^2_{i+m-1}) \geq t \textit{ for all } i \in 1, \ldots, |\sigma^1| - m + 1$$

The following *match* function qualifies the substrings of string objects, that are similar to string pattern ϕ according to the word-based similarity relation, as the soft-occurrences of ϕ.

[13]Sensitivity is a fitness measure defined as fraction of true positives among true positives and false positives [LWS+93]. In our context, true positives are the strings that are known to be similar and are similar according to a similarity relation, whereas false positives are strings that are known to be not similar, but are recognized as similar according to a similarity relation.

Definition 4.26 (Word-based Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, on which a similarity matrix M is defined, S be a string object from $\mathcal{U}_\Sigma$, and t be a similarity threshold. Let ϕ be a pattern from a string pattern language $\mathcal{L}_\Sigma$. Define a word-based match function $match_W(\phi, S, l_w, t)$ to evaluate* true *if and only if there exists a string σ, such that $\sigma \sqsubseteq S$ and ϕ and σ are in the word-based similarity relation $sim_W(\phi, S, l_w, t)$.*

Remark 4.11 *Note that when the scores in the similarity matrix M obeys a metric, the word-based similarity relation between the pattern ϕ and its soft-occurrences induces a similarity relation between the soft-occurrences because of the triangular inequality.*

Definition 4.27 (Minimum Word-based Soft-Frequency Constraint) *A minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the word-based match function $match_W(\phi, S, l_w, t)$ is called the minimum word-based soft-frequency constraint. It is denoted $\mathsf{MinSoftFr}_\phi^{W,S}(minFr, \mathcal{D}, l_w, t)$.*

The associated algorithm [SVS95] solves the following problems:

1. extract all string patterns $\phi \in \mathcal{L}_\Sigma$ in $\mathcal{D}$ that satisfy the constraint:
   $\mathsf{MinSoftFr}_\phi^{W,S}(minFr, \mathcal{D}, l_w, t) \wedge \mathsf{MinLength}_\phi(minLen) \wedge \mathsf{MaxLength}_\phi(maxLen)$
   where $minLen = maxLen$;
2. find the greatest length l_{max}, for which there exists at least one l_{max}-length string pattern ϕ that satisfies the minimum word-based soft-frequency constraint $\mathsf{MinSoftFr}_\phi^{W,S}(minFr, \mathcal{D}, l_w, t)$, and solve the problem 1 for $minLen = maxLen = l_{max}$.

The authors also discuss a variant of this algorithm that allows the introduction of gaps between the pattern and its occurrences.

4.3.2 Clique Patterns

One of the classic approaches to handle fault-tolerance is to introduce it in the level of the alphabet of symbols. The symbols of the alphabet can be grouped into equivalence classes by defining an equivalence relation over an alphabet. Then, each class is considered as a new symbol of the reduced alphabet. This idea was introduced for the amino acid sequence analysis in [KG85a]. Such approach can be extended by considering, instead of an equivalence relation, a reflexive, symmetric, but not transitive[14] similarity relation. This is convenient, e.g., for protein sequences that can

[14]A meaningful relation similarity between two entities can not be propagated too far

be seen as strings on alphabet of amino acids. Two or more amino acids usually share some physico-chemical properties, such as electrical charge, polarity, size, different levels of acidity, etc., but rarely all of them. Then, two amino acids are similar, if they share enough physico-chemical properties to be considered related.

Definition 4.28 (Similarity Relation on Alphabet) *Let $\Sigma = \{\xi_{1,\dots,\xi_n}\}$, $n \in \mathbb{N}$ be an alphabet of symbols. Similarity relation on alphabet Σ, denoted $sim_{alph}(\xi_i, \xi_i)$, is a reflexive, symmetric, but not transitive binary relation between symbols from Σ.*

The relation of similarity on alphabet sim_{alph} can be extended to a relation of similarity between strings σ^1 and σ^2 of equal length as follows: σ^1 and σ^2 are similar if and only if in each position their corresponding symbols are related by sim_{alph}. The idea of such relation over amino acid alphabet for pairwise string comparison was independently introduced in [Cob94] and in [BDMR90]. The paper [SVC95] takes one step further and use this relation to construct fault-tolerant patterns.

Definition 4.29 (Alphabet-based Similarity Relation) *Let Σ be alphabet of symbols, on which a similarity relation sim_{alph} is defined. Let σ^1, σ^2 be be two strings of length l on alphabet Σ. Strings σ^1 and σ^2 are in a alphabet-based similarity relation $sim_{alph,l}(\sigma^1, \sigma^2)$, if and only if, for all $1 \leq i \leq l$, σ^1_i and σ^2_i are in $sim_{alph}(\sigma^1_i, \sigma^2_i)$ relation.*

The fault-tolerant patterns are then defined as the maximal cliques of the alphabet-based similarity relation.

Definition 4.30 ((Maximal) Clique of Alphabet-based Similarity Relation) *Let Σ be alphabet of symbols, on which a similarity relation sim_{alph} is defined. A multi-set of strings of length l over Σ, denoted $\mathcal{C}_l$, is a clique of the alphabet-based similarity relation $sim_{alph,l}(\sigma^1, \sigma^2)$, if and only if for all $\sigma^i, \sigma^j \in \mathcal{C}_k$ we have $sim_{alph,l}(\sigma^i, \sigma^j) = \textsf{true}$. A clique $\mathcal{C}_l$ is maximal if and only if by adding any other l-length substring over alphabet Σ, the resulting set is no more a clique. The length of clique $\mathcal{C}_l$, denoted $|\mathcal{C}_l|$, is l.*

Definition 4.31 (Clique Pattern Language) *Consider the alphabet Σ and the relation sim_{alph} from Definition 4.30. The clique pattern language, denoted $\mathcal{L}_C$, is a set of all possible maximal cliques $\mathcal{C}_l$ of alphabet-based similarity relation.*

The particularity of a clique pattern match function is that it associates a pattern ϕ not to one object, but to a set of objects at once.

Definition 4.32 (Clique Match Function) *Consider the alphabet Σ and the relation sim_{alph} from Definition 4.30. Let $\mathcal{U}_\Sigma$ be a universe of strings over Σ, and α be a set of l-length substring objects from $\mathcal{U}_\Sigma$. Let ϕ be a pattern from a clique pattern language $\mathcal{L}_C$. Define a clique match function $match_C(\phi, \alpha)$ to evaluate* true*, if and only if ϕ is equal to the set α.*

Remark 4.12 *Observe that the frequency of a pattern $\phi \in \mathcal{L}_C$, i.e., the number of objects that it matches, is also the number of string elements that it contains.*

Definition 4.33 (Clique Minimum Frequency) *A minimum frequency constraint that uses the clique match function $match_C(\phi, \alpha)$ is called a clique minimum frequency constraint, denoted $\mathsf{MinFr}_\phi^{C,\alpha}(minFr, \mathcal{D})$.*

The corresponding maximal clique of alphabet-based similarity relation extraction algorithm [SVC95] is an extension of the `KMR` algorithm [KMR72, LAC89]. It solves the following problems:

1. extract all clique patterns $\phi \in \mathcal{L}_C$ in a data string $\mathcal{D}$ that satisfy the constraint:
   $\mathsf{MinFr}_\phi^{C,\alpha}(2, \mathcal{D}) \wedge \mathsf{MinLength}_\phi(minLen) \wedge \mathsf{MaxLength}_\phi(minLen)$
   where $minLen = maxLen$;
2. find the greatest length l_{max}, for which there exists at least one l_{max}-length clique pattern ϕ that satisfies a clique minimum frequency constraint denoted $\mathsf{MinFr}_\phi^{C,\alpha}(2, \mathcal{D})$, and solve the problem 1 for $minLen = maxLen = l_{max}$.

Algorithm proposed in [SVPS95] is a generalized version of [SVC95]. It solves its performance bottleneck, which arises when searching common degenerated features in linear coding of protein $3D$ structures, because of large alphabet (typically over 100 symbols), significant degeneracy of a similarity relation defined on alphabet, and a symbol repeating consecutively in data a great number of times.

4.3.3 Patterns that are Strings over Alphabet Subsets

In the previous section we presented an approach to fault-tolerant patterns, which is convenient when the similarity can be expressed by a single relation on the alphabet. However, the problem becomes more difficult when the relations existing on alphabet are overlapping and/or of different levels, what is the case, e.g., if one need to compare the protein sequences at various levels of structure (primary, secondary and tertiary) simultaneously. In this section we present two approaches to model these complex relationships by patterns that are strings over subsets of alphabet.

4.3.3.1 String over Alphabet Cover Patterns

The idea of modeling the relations holding on an alphabet by string patterns on an alphabet cover[15] was introduced in [SVS97].

Definition 4.34 (String over Alphabet Cover) *Let Σ be an alphabet, and $c = \{\alpha_1, \ldots, \alpha_n\}$ where $n \in \mathbb{N}$, be a cover of Σ. A finite sequence $\sigma^c = \alpha_p \ldots \alpha_q$ of elements from a cover c is called a string over alphabet cover. The length of a string over alphabet cover is the number of subsets α_i it contains.*

Definition 4.35 (String over Alphabet Cover Pattern Language) *Let Σ be an alphabet and c be its cover. String over alphabet cover pattern language, denoted $\mathcal{L}_c$, is the set of all possible strings over c.*

String over alphabet cover patterns implement fault-tolerance through their intrinsic capability of soft matching. A l-length string over alphabet cover σ^c match l-length string σ, such that every string σ symbol σ_i is an element of the corresponding string over alphabet cover σ^c subset σ^c_i. Such strings σ are *instances* of the string over alphabet cover σ^c.

Definition 4.36 (Instance of String over Alphabet Cover) *Let Σ be an alphabet and $c = \{S_1, \ldots, S_n\}$, $n \in \mathbb{N}$ be its cover. Consider a l-length string $\sigma = \sigma_1 \ldots \sigma_l$ over Σ and a l-length string $\sigma^c = \sigma^c_1 \ldots \sigma^c_l$ over c. σ is an instance of σ^c if and only if $\sigma_i \in \sigma^c_i$, for all $i = 1, \ldots, l$.*

Further flexibility can be achieved by allowing up to k, $k \in \mathbb{N}$ errors in such σ^c and σ matching. We can then compute the corresponding distance and use it to establish a similarity relation between string over alphabet cover σ^c and string σ.

Definition 4.37 (Set-edit Distance) *Let Σ be an alphabet and c be its cover. Let σ be a string over Σ and σ^c be a string over alphabet cover. The minimum number of deletions, insertions and substitutions, necessary to convert σ into an instance of σ^c, is called the Set-edit distance, denoted $SEDist(\sigma^c, \sigma)$.*

A string over alphabet cover σ^c and a string $\sigma \in \Sigma^*$ are in a set-edit distance similarity relation if their set-edit distance is smaller than a given threshold.

Definition 4.38 (Set-edit Distance Similarity Relation) *Let Σ be an alphabet, c be its cover, $k \in \mathbb{N}$ be a threshold, σ^c be a string over alphabet cover and σ be a string over Σ. σ^c and σ are in the set-edit distance similarity relation $sim_{SEdist}(\sigma^c, \sigma, k)$ if and only if their set-edit distance $SEDist(\sigma^c, \sigma)$ is less or equal to k.*

[15] Alphabet Σ cover is a set of its subsets $\{\alpha_1, \ldots, \alpha_n\}$, $n \in \mathbb{N}$, such that $\sum_{i=1}^{n} \alpha_i = \Sigma$.

The following *match* function associates string over alphabet cover patterns ϕ to string objects.

Definition 4.39 (Set-edit Match Function) *Let Σ be an alphabet, c be its cover, $\mathcal{U}_\Sigma$ be a universe of strings over an Σ, S be a string object from $\mathcal{U}_\Sigma$, and k be a threshold. Let ϕ be a pattern from a string over alphabet cover pattern language $\mathcal{L}_c$. Define a match function $match_{SEdist}(\phi, S, k)$ to evaluate* true *if and only if there exists a string σ such that $\sigma \sqsubseteq S$ and $sim_{SEdist}(\phi, \sigma, k) =$* true.

Remark 4.13 *The set-edit distance is a metric, and therefore, because of the triangular inequality, the set-edit distance similarity relation between the pattern and its occurrences induces a similarity relation between the occurrences.*

Definition 4.40 (Minimum Set-edit Soft-Frequency) *The minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ based on the set-edit match function $match_{SEdist}(\phi, S, k)$ is denoted $\mathsf{MinSoftFr}_\phi^{SEdist,S}(minFr, \mathcal{D}, k)$. It is called the minimum set-edit soft-frequency constraint.*

Two algorithms to extract strings over alphabet cover, `Poivre` and `LePoivre`, are presented in [SVS97]. `Poivre` tackles the special case, when no errors between pattern $\phi \in \mathcal{L}_c$ and its occurrences are allowed, i.e., the occurrences of ϕ are its instances. Thus, the algorithm `Poivre`, uses the match function $match_{SEdist}(\phi, S, 0)$. `LePoivre` tackles the general case, when the threshold k of set-edit distance between a pattern ϕ and its occurrences is allowed to be greater than 0. Both algorithms were inspired by the `KMR` algorithm [KMR72, LAC89]. They solve the following problems:

1. extract all string patterns $\phi \in \mathcal{L}_c$ in a string data set $\mathcal{D}$ that satisfy the constraint
   $\mathsf{MinSoftFr}_\phi^{SEdist,S}(minFr, \mathcal{D}, k) \wedge \mathsf{MinLength}_\phi(minLen) \wedge \mathsf{MaxLength}_\phi(maxLen)$,
   where $minLen = maxLen$;

2. find the greatest length l_{max}, for which there exists at least one l_{max}-length string over alphabet cover pattern ϕ that satisfies a minimum set-edit soft-frequency constraint $\mathsf{MinSoftFr}_\phi^{SEdist,S}(minFr, \mathcal{D}, k)$, and solve the problem 1 for $minLen = maxLen = l_{max}$.

4.3.3.2 String over Weighted Combinatorial Cover Patterns

String over alphabet cover patterns, presented in the previous section, requires to specify in advance the cover, i.e., the alphabet of subsets, over which such patterns will be constructed. However, in many cases one does not know, what alphabet is pertinent to reveal the regularities of interest (concerning biological sequence analysis, see, e.g., [GEW85, KG85b]). To answer this problem, [SV96a] considers fault-tolerant patterns, which are strings over the powerset of the symbols alphabet Σ. This means that the subset alphabet (e.g., a cover) need not be specified in advance[16]. Note that among alphabet Σ subsets, used to construct patterns, there is also a set Σ, consisting of all symbols. This means that the pattern can contain wildcard symbols[17]. Wildcard symbol in a pattern correspond to positions, which are not significant for the captured regularity. This is of interest, e.g., when searching for putative functional elements in biological sequences, since they often contain the non-conserved positions. Observe that employing the concept of wildcard in a pattern is different from working with distance between pattern and its occurrences that counts errors[18]. Consider a l-length pattern that is composed only of wildcard symbols. It is obviously uninteresting, since it match any l-length string over Σ. To avoid such situations, one can associate weights with the elements of Σ powerset, which restrict the number of appearance of these subsets in a pattern (if there no restriction is desired, the corresponding weight is assigned to ∞). Such structure is called a *weighted combinatorial cover.*

Definition 4.41 (Weighted Combinatorial Cover) *Let Σ be an alphabet, and 2^{Σ} be a powerset of Σ. Let $2^{\Sigma+}$ denote the set $2^{\Sigma} \setminus \emptyset$. A weighted combinatorial cover of the alphabet Σ, denoted wcc, is the set of pairs (α_i, h_s), where α_i is an element of $2^{\Sigma+}$, $i \in \mathbb{N}$, and h_s is a non negative integer or ∞.*

The considered patterns are strings over weighted combinatorial cover *wcc.*

Definition 4.42 (String over *wcc*) *Let Σ be an alphabet, and wcc be its weighted combinatorial cover. A finite sequence $\sigma^{wcc} = \alpha_p \ldots \alpha_q$ of elements from $2^{\Sigma+}$, such that the number of α_i appearances in σ^{wcc} does not exceed its h_s, specified in wcc, is called a string over weighted combinatorial cover. The length of a string over weighted combinatorial cover is a number of subsets α_i it contains.*

Definition 4.43 (String over *wcc* Pattern Language) *Let Σ be an alphabet and wcc be its weighted combinatorial cover. String over wcc pattern language, denoted $\mathcal{L}_{wcc}$, is a set of all possible strings over wcc.*

String σ^{wcc} over *wcc* and string σ over alphabet Σ are in a similarity relation, if σ is an instance of σ^{wcc}.

[16]Any possible alphabet Σ subset belongs to the powerset of Σ

[17]Wildcard symbol is a symbol that can be substituted for any other symbol from the alphabet

[18] There is no notion of an error here, instead some positions are considered as non-significant for a regularity.

Definition 4.44 (wcc similarity relation) *Let Σ be an alphabet and wcc be its weighted combinatorial cover. Let $\sigma = \sigma_1 \dots \sigma_l$ be a l-length string over Σ, and $\sigma^{wcc} = \sigma_1^{wcc} \dots \sigma_l^{wcc}$ be a l-length string over wcc. σ^{wcc} and σ are in a wcc similarity relation $sim_{wcc}(\sigma^{wcc}, \sigma)$, if and only if $\sigma_i \in \sigma_i^{wcc}$, for all $i = 1, \dots, l$.*

The following *match* function associates string over *wcc* patterns to string objects.

Definition 4.45 (Match Function for wcc similarity) *Let Σ be an alphabet, wcc be its weighted combinatorial cover, $\mathcal{U}_\Sigma$ be a universe of strings over an Σ, and S be a string object from $\mathcal{U}_\Sigma$. Let ϕ be a pattern from string over wcc pattern language $\mathcal{L}_{wcc}$. Define a match function $match_{wcc}(\phi, S)$ to evaluate true if and only if there exists a string σ such that $\sigma \sqsubseteq S$ and $sim_{wcc}(\phi, \sigma) = \mathsf{true}$.*

Definition 4.46 (Minimum wcc Soft-Frequency) *The Minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the wcc match function $match_{wcc}(\phi, S)$ is called a minimum wcc soft-frequency constraint. It is denoted $\mathsf{MinSoftFr}_\phi^{wcc,S}(minFr, \mathcal{D})$.*

Algorithm, proposed in [SV96a] solves the following problems:

1. extract all string patterns $\phi \in \mathcal{L}_{wcc}$ in a string data set $\mathcal{D}$ that satisfy the constraint
   $\mathsf{MinSoftFr}_\phi^{wcc,S}(minFr, \mathcal{D}) \wedge \mathsf{MinLength}_\phi(minLen) \wedge \mathsf{MaxLength}_\phi(maxLen)$,
   where $minLen = maxLen$;

2. find the greatest length l_{max}, for which there exists at least one l_{max}-length string over wcc pattern ϕ that satisfies a minimum wcc soft-frequency constraint $\mathsf{MinSoftFr}_\phi^{wcc,S}(minFr, \mathcal{D})$, and solve the problem 1 for $minLen = maxLen = l_{max}$.

To locate very degenerated patterns, one can go further and allow errors in string σ^{wcc} over wcc and string σ over alphabet Σ matching. Then, a similarity between σ^{wcc} and σ is evaluated based on their distance, defined analogically to set-edit distance (see Definition 4.37 in Page 75). The idea of the algorithm to extract $\phi \in \mathcal{L}_{wcc}$ patterns with the *match* function that allows up to k distance between ϕ and its occurrence, is also introduced in [SV96a].

4.3.4 Regular Expression Patterns

Fault-tolerance is intrinsic for regular expression patterns, since they match the strings belonging to the language generated by the pattern (see Definition 1.17 in Page 20).

4.3.4.1 String with Wildcards Patterns

We now consider strings with wildcards patterns, introduced in [RF98b, RF98c]. Strings with wildcards are strings over an alphabet composed of literal symbols and wildcard symbols.

Definition 4.47 (String with Wildcards) *Let Σ be an alphabet of literal symbols. Let $\textbf{.}$ denote a wildcard symbol, i.e., a position that match any symbol from Σ. String with wildcard, denoted $\sigma^{\cdot}$, is a string that begins and ends with a symbol from Σ and contains an arbitrary combination of symbols from the extended alphabet $\Sigma \cup \{\textbf{.}\}$.*

Definition 4.48 (String with Wildcards Pattern Language) *Let Σ be an alphabet of literal symbols. String with wildcards pattern language, denoted $\mathcal{L}_{\Sigma \bigcup \{.\}}$, is a set of all possible strings with wilcards over Σ.*

A string with wildcards pattern $\phi \in \mathcal{L}_{\Sigma \bigcup \{.\}}$ is a regular expression, and thus it implements fault-tolerance through its intrinsic capability to match the strings that belongs to the language generated by ϕ.

Definition 4.49 (Language Generated by String with Wildcards) *Let us denote by $L(\xi_i)$ the language generated by a literal symbol $\xi_i \in \Sigma$. It is the set containing one element $\{\xi_i\}$. The language generated by a wildcard symbol $\textbf{.}$, denoted $L(\textbf{.})$, is the set containing all symbols in Σ, i.e., $L(\textbf{.}) = \Sigma$. The language generated by a string with wildcard $\sigma^{\cdot} = \sigma^{\cdot}_1 \ldots \sigma^{\cdot}_n$, denoted $L(\sigma^{\cdot})$, is a set of strings $\sigma = \sigma_1 \ldots \sigma_n$, such that $\sigma_i \in L(\sigma^{\cdot}_i)$, $i = 1, \ldots, n$.*

Definition 4.50 (String with Wildcards Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over literal symbols alphabet Σ, and S be a string object from $\mathcal{U}_\Sigma$. Let ϕ be a pattern from a string with wildcards language $\mathcal{L}_{\Sigma \bigcup \{.\}}$. Define a match function $match_{\Sigma \bigcup \{.\}}(\phi, S)$ to evaluate* true *if and only if there exists a string σ such that $\sigma \sqsubseteq S$ and σ belongs to the language $L(\phi)$.*

Definition 4.51 (Minimum Soft-Frequency for Strings with Wildcards) *The minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the string with wildcards match function $match_{\Sigma \bigcup \{.\}}(\phi, S)$ is called a minimum soft-frequency constraint for strings with wildcards. It is denoted $\mathsf{MinSoftFr}_\phi^{\Sigma \bigcup \{.\}, S}(minFr, \mathcal{D})$.*

Consider, for example, a string with wildcards ac.g occurring in a known number of position in a string data set $\mathcal{D}$. Then, if another string with wildcards a..g occurs at the same positions and nowhere else in $\mathcal{D}$, it is not interesting, since it contain no new knowledge and can be deduced from ac.g. Therefore, in most cases, it is desirable to extract only the most *specific* or, in other words, *maximal* strings with wildcards. We first define the relation of specificity, and then we will employ it to define the constraint of maximality.

Definition 4.52 (Specificity Relation on Strings with Wildcards) *Given an alphabet Σ, let $\sigma^{\cdot 1}$ and $\sigma^{\cdot 2}$ be strings with wildcards. $\sigma^{\cdot 2}$ is said to be more specific than $\sigma^{\cdot 1}$ if $\sigma^{\cdot 2}$ can be obtained from $\sigma^{\cdot 1}$ by changing its one or more wildcard symbols to symbols from Σ, or by appending a string over $\Sigma \bigcup \{.\}$ to the left or/and right of $\sigma^{\cdot 1}$.*

Definition 4.53 (Maximality Constraint) *Let $\phi \in \mathcal{L}_{\Sigma \bigcup \{.\}}$ be a string with wildcards pattern, and $\mathcal{D}$ be a string data set. A maximality constraint $\mathsf{Maximality}_\phi(\mathcal{D})$ is satisfied iff there exists no pattern ψ that is more specific than ϕ and have the same number of occurrences in $\mathcal{D}$.*

Observe that maximality constraint is a data-dependent constraint. In addition to this, syntactic constraints can be put on a number of wildcards in a pattern. This allows to focus on the extraction of more or less degenerated regularities. Such constraint applies on subpatterns of a pattern ϕ.

Definition 4.54 (Subpattern of pattern with wildcards) *Let $\phi \in \mathcal{L}_{\Sigma \bigcup \{.\}}$ be a string with wildcards pattern. Any ϕ substring that also belong to pattern language $\mathcal{L}_{\Sigma \bigcup \{.\}}$ is called a subpattern of ϕ.*

Definition 4.55 (Rigidity Constraint) *Let $\phi \in \mathcal{L}_{\Sigma \bigcup \{.\}}$ be a string with wildcards pattern, and $L, W \in \mathbb{N}$, $L \leq W$, be thresholds. Define a rigidity constraint $\mathsf{Rigidity}_\phi(L, W)$ to evaluate* true *if and only if every subpattern of ϕ of length W contains at least L symbols from alphabet Σ, i.e., not wildcard symbols. Note that if $\mathsf{Rigidity}_\phi(L, W)$ is satisfied, than it is also satisfied with any threshold $W' > W$.*

`Teiresias` algorithm [RF98b, RF98c] extract patterns $\phi \in \mathcal{L}_{\Sigma \bigcup \{.\}}$, satisfying the constraints $\mathsf{MinSoftFr}_\phi^{\Sigma \bigcup \{.\}, S}(minFr, \mathcal{D}) \wedge \mathsf{Maximality}_\phi(\mathcal{D}) \wedge \mathsf{Rigidity}_\phi(L, W)$ in a string data set $\mathcal{D}$.

To avoid a combinatorial explosion in the number of string with wildcard patterns, recently a promising idea was proposed in [PRF+00], which consists of extracting the generator sets of the patterns, called the *bases of motifs*. It is a well-known and precisely defined mathematical object, guaranteeing the property of completeness. Informally, a basis of motifs is a subset of all patterns that satisfy the given constraints. From such subset it is possible to construct all other patterns also satisfying these constraints[19]. The argument behind the basis of [PRF$^+$00] is that a compact enough basis can be always found. Despite of the interesting features of basis defined in [PRF$^+$00], they were revealed to have some drawbacks. An upper bound of a number of basis, satisfying a minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ in data string $\mathcal{D}$, with $minFr \geq 2$, does not hold. Since the associated polynomial time (in number of symbols in a data string) extraction algorithm relies on that bound, the problem of efficiently discovering such bases remains open [PCGS05]. Subsequently, a refinement of the definition of basis and the corresponding incremental extraction algorithm was presented in [AP04]. Recently, a basis defined with stronger conditions than that of [PRF$^+$00] (thus included in the basis of [PRF$^+$00], and therefore smaller), but able to generate the *same* set of patterns, was introduced in [PCGS05]. The preliminary ideas of this work were described in [PCGS02] and published in [PCGS03]. Among few other nice properties, the number of patterns in such basis and the total number of their occurrences, satisfying a minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ in data string $\mathcal{D}$, with $minFr = 2$, have provable upper bounds that are linear in the number of symbols in a data string $\mathcal{D}$. In addition to this, an exponential dependency on $minFr$ of the number of patterns in the basis defined in [PCGS05, PRF$^+$00, PAA03] was revealed. Consequently, there can not exist a polynomial-time algorithm to extract these bases satisfying $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ constraint, with arbitrary values of $minFr \geq 2$.

4.3.4.2 Generalized Regular Patterns

Generalized regular patterns are strings over the alphabet composed of literal symbols, *groups of literal symbols*, *restricted* and *unrestricted length wildcards*. An extensive study of generalized regular pattern extraction and application to biological sequence analysis can be found in [Vil02].

We first define three kinds of symbols, which, altogether with literal symbols, form the alphabet of generalized regular patterns.

Definition 4.56 (Group Symbols) *Let $\Sigma = \{\xi_1, \ldots, \xi_n\}$ be an alphabet of literal symbols. Let $g_1, \ldots, g_n$ be subsets of Σ, such that each subset contains more than one element. To denote such subsets $g_i = \{\xi_p, \ldots, \xi_q\}$ we use bracketed lists of all symbols in g_i, called group symbols and noted $[\xi_p \ldots \xi_q]$. A set of group symbols forms a group alphabet, denoted Γ.*

Definition 4.57 (Wildcard of Unrestricted Length) *Let Σ be a literal symbols alphabet. A wildcard of unrestricted length, denoted $*$, is a symbol that can be substituted for any element of the set Σ^*, i.e., for any string over Σ.*

[19] The patterns that are not in the basis are the combinations of patterns in the basis.

Definition 4.58 (Wildcard of Restricted Length) *Let Σ be a literal symbols alphabet. A wildcard of restricted length, denoted $*(p,q)$, $p,q \in \mathbb{N}$, $p \leq q$, is a symbol that can be substituted for any string of length between p and q over literal symbols alphabet Σ. A set of restricted length wildcards $\{*(p,q) \mid 0 \leq p \leq q < \infty\}$ is denoted W.*

Then, we define generalized regular pattern.

Definition 4.59 (Generalized Regular Pattern) *Let Σ be a literal symbols alphabet, Γ be group symbols alphabet, $*$ be the unrestricted length wildcard and W be a set of restricted length wildcards. Generalized regular pattern r is a string over $\Sigma \cup \Gamma \cup * \cup W$.*

Definition 4.60 (Generalized Regular Pattern Language) *Let Σ be a literal symbols alphabet, Γ be group symbols alphabet, $*$ be the unrestricted length wildcard and W be a set of restricted length wildcards. Generalized regular pattern language, denoted $\mathcal{L}_r$, is a set of all generalized patterns over $\Sigma \cup \Gamma \cup * \cup W$.*

A generalized regular pattern $\phi \in \mathcal{L}_r$ is a regular expression, and, as string with wildcards patterns (presented in the previous section), it implements fault-tolerance by matching the strings that belongs to the language generated by ϕ.

Definition 4.61 (Language defined by a generalized regular pattern) *Let ξ_i be a symbol from literal symbols alphabet Σ. A language defined by ξ_i, denoted $L(\xi_i)$, is a one-element set $\{\xi_i\}$. Let $[\xi_p \dots \xi_q]$ be group symbol from group alphabet Γ. A language defined by $[\xi_p \dots \xi_q]$, denoted $L([\xi_p \dots \xi_q])$, is a set $\{\xi_p, \dots, \xi_q\}$. A language defined by a wildcard of unrestricted length $*$, denoted $L(*)$, is a set of all strings over the literal symbols alphabet Σ, noted Σ^*. A language defined by a wildcard of restricted length $*(p,q)$, $p,q \in \mathbb{N}$, $p \leq q$, denoted $L(*(p,q))$, is a set of all strings of length between p and q over the literal symbols alphabet Σ. Let r be a generalized regular pattern. A language defined by $r = r_1 \dots r_n$, denoted $L(r)$, is a set of strings $\sigma = \zeta_1 \dots \zeta_n$, where ζ_i is a substring of σ[20], such that $\zeta_i \in L(\phi_i)$, $i = 1, \dots, n$.*

Definition 4.62 (Generalized Regular Patterns Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over a literal symbols alphabet Σ, and S be a string object from $\mathcal{U}_\Sigma$. Let ϕ be a regular pattern from pattern language $\mathcal{L}_r$. Define a match function $match_r(\phi, S)$ to evaluate* true *if and only if there exists a string σ, such that $\sigma \sqsubseteq S$ and σ belongs to the language $L(\phi)$.*

[20] ζ_1 is a substring, and not a symbol, of σ, because the languages of $*$ and $*(p,q)$ contain not only symbols, but also substrings

Definition 4.63 (Minimum Soft-Frequency on Generalized Regular Patterns) *A minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the generalized regular patterns match function $match_r(\phi, S)$ is called a minimum soft-frequency constraint for generalized regular patterns. It is denoted $\mathsf{MinSoftFr}_\phi^{r,S}(minFr, \mathcal{D})$.*

SPEXS [Vil98, Vil02] is generalized framework of algorithms that extract generalized regular patterns $\phi \in \mathcal{L}_r$ that satisfy $\mathsf{MinSoftFr}_\phi^{r,S}(minFr, \mathcal{D})$ constraint in one or more data string(s) or a string data set(s) $\mathcal{D}$. Combinations of syntactic constraints on maximum allowed number of group symbols, given the groups alphabet Γ, maximum allowed number of unrestricted wildcards and maximum allowed number of restricted wildcards, given their length, allows to delimit a subset of regular pattern language that is of interest. When the strings in data are related, there may exist many patterns satisfying the minimum frequency constraint even with high frequency thresholds. Also, the more complex is the pattern language, the more different patterns contains the solution set. It is challenging to decide which of these patterns are pertinent. To assist this task, pattern *fitness measures*, which evaluate pattern interestingness and relevance, can be computed. Then, these measures are used to rank and compare the extracted patterns. SPEXS computes the pattern fitness measures, based on pattern improbability [VBJ$^+$00] or on Minimum Description Length (MDL) principle [BJUV96, BUV96]. Then the constraint on pattern *minimum fitness* can be formulated and evaluated by post-processing (see discussion in Page 21). SPEXS also allows to extract patterns overrepresented in a positive string data set $\mathcal{D}_+$ with respect to the negative or random one $\mathcal{D}_-$ by pushing a minimum frequency constraint only on a positive string data set and by post-processing a *minimum ratio* constraint, which restricts how much more frequent pattern is in $\mathcal{D}_+$ than in $\mathcal{D}_-$ [BJVU98a].

4.3.5 Structured Patterns

In the previous sections we considered various languages of fault-tolerant patterns that allow to express regularities, composed of one element. However, there are cases where a regularity, one is trying to capture, potentially contains several elements separated by gaps. A typical example is the transcription factor binding sites, which often work in groups, and the relative positions of such multiple sites, participating in a biological process, are in general not random. Structured patterns, introduced in [MS00a, MS00b], allow to express such characteristics.

Definition 4.64 (Structured Pattern) *Let Σ be an alphabet. Structured pattern, denoted $\mathcal{B}$, is a pair (b, d), where:*

- *b is a p-tuple $(b_1, \ldots, b_p)$, $p \in \mathbb{N}$, of strings b_i over alphabet Σ, representing p structural pattern parts, which are called boxes,*

- *d is a $(p-1)$-tuple of triplets $((d_{min_1}, d_{max_1}, \delta_1), \ldots, (d_{min_{p-1}}, d_{max_{p-1}}, \delta_{p-1}))$, $d_{min_i}, d_{max_i}, \delta_i \in \mathbb{N}$, $d_{max_i} \geq d_{min_i}$, representing the $(p-1)$ intervals of distance between two successive boxes b_i, b_{i+1}.*

The length of a structured pattern is a number of boxes b_i it contains.

Definition 4.65 (Structured Pattern Language) *Let Σ be an alphabet. The structured pattern language, denoted $\mathcal{L}_\mathcal{B}$, is the set of all possible structured patterns (b, d).*

A structured pattern implements the fault-tolerance by allowing up to k mismatches between each its box and a corresponding substring, as well as variations of gap length between two adjacent boxes.

Definition 4.66 (Box-Hamming Similarity Relation) *Let Σ be an alphabet, $\mathcal{B}$ be a structured l-length pattern, σ be a string on Σ, and $k \in \mathbb{N}$ be a threshold. The pattern $\mathcal{B}$ and the string σ are in a box-Hamming similarity relation $sim_{\mathcal{B}H}(\mathcal{B}, \sigma, k)$ iff the positions in a string σ are such that*

- *l substrings $\sigma^1, \ldots, \sigma^i, \ldots, \sigma^l$ of string σ, starting at positions σ_1, …, σ_j, …, $\sigma_{|\sigma|-|b_l|+1}$ are such that Hamming distance between each box b_i and σ^i, $i = \{1, \ldots, l\}$, is less or equal to k,*
- *there exists a d_i from the interval $[d_{min_i} + \delta_i, d_{max_i} - \delta_i]$, such that the distance between the end position of σ^i and the start position of σ^{i+1} is $d_i \pm \delta_i$[21].*

Observe, that when $\delta_i = (d_{max_i} - d_{min_i} + 1)/2$, then δ_i can be omitted, since the allowed distance d_i between two substrings σ^i and σ^{i+1} is in the interval $[d_{min_i}, d_{max_i}]$.

The following match function associates structured patterns to string objects.

Definition 4.67 (Structured Pattern Box-Hamming Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, S be a string object from $\mathcal{U}_\Sigma$, and k be a threshold. Let ϕ be a structured pattern from $\mathcal{L}_\mathcal{B}$. We define a match function $match_{\mathcal{B}H}(\phi, S, k)$ to evaluate* true *iff there exists a string σ such that $\sigma \sqsubseteq S$ and $sim_{\mathcal{B}H}(\phi, \sigma, k) =$* true.

Definition 4.68 (Minimum Box-Hamming Soft-Frequency) *A minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the structured pattern box-Hamming match function $match_{\mathcal{B}H}(\phi, S, k)$ is called a minimum box-Hamming soft-frequency constraint. It is denoted $\mathsf{MinSoftFr}^{\phi,S}_{\mathcal{B}H}(minFr, \mathcal{D}, k)$.*

[21] $\pm\delta_i$ is an allowed variation of the distance d_i.

Note that the frequency of a structured pattern is not simply the number of objects that is matches, but the number of objects that it matches with the same distance d_i, used to establish the box-Hamming similarity relation.

Various on suffix trees based algorithms to extract structured patterns $\phi \in \mathcal{L}_{\mathcal{B}}$ that satisfy a minimum frequency constraint $\mathsf{MinSoftFr}^{\phi,S}_{\mathcal{B}H}(minFr, \mathcal{D}, k)$ in a string data set $\mathcal{D}$ are introduced in [MS00a, MS00b]. Different versions allows to extract 2-boxes length patterns and arbitrary l-boxes length patterns. In addition to this, further syntactic constraints can be added on a structured pattern $\mathcal{B}$ and string σ similarity, e.g., a maximum allowed rate of errors, a maximum or minimal symbol frequency in a box (or in all boxes of $\mathcal{B}$).

Part II

Contribution

Chapter 5

Similarity and Soft-Frequency Constraints

5.1 Problem Setting

Motivation

We study the similarity and soft-frequency constraints that enable the fault-tolerance, indispensable in many application domains (genomic data analysis, seismic data analysis, WWW usage mining being only few examples). To motivate the need for *complete* and *generic* solvers we propose to consider a typical situation, when a domain expert uses data mining techniques in real-life applications, especially when the question he/she is trying to answer is somewhat exploratory. For example, the biologist researcher aims to find the putative transcription factor binding sites (patterns) in a set of promoter sequences (a set of character strings). There two ways to proceed, that is, to use either statistical motif discovery tools that employ *heuristic* strategies or to turn to *correct and complete* pattern extraction. The fact that different heuristic motif discovery tools gives different results on the same data, and that typically these results depend on the ordering the input data sequences, can be not acceptable for some biologist researchers. The alternative way then is the correct and complete pattern extraction. A number of good exhaustive algorithms have been developed to search for biological motifs on sequences. The problem is that these are ad-hoc approaches, that is, they are solvers for one tight inductive query composed of one or several constraints. In an exploratory data mining task, a typical situation is that the results of the current pattern extraction gives rise to the new questions and thus the new inductive queries. If we rely on the ad-hoc solvers, once a question (an inductive query) changes, a new algorithm has to be devised. Though from inductive querying point of view, the goal is to provide a solver capable to evaluate arbitrary Boolean compositions of constraints. It is like to be able to evaluate SQL queries on patterns.

Research Context

Similarity between two strings is the core algorithmic issue in approximate string matching domain, while it is also of great importance in bioinformatics domain. Several approaches to similarity constraint was studied by the data mining community. Similarity relation allows to identify the soft-occurrences and therefore is fundamental to define the fault-tolerant patterns. Fault-tolerant pattern extraction in strings or string data sets through soft-frequency constraints have been extensively studied in bioinformatics area. A state of art on these existing approaches to tackle the similarity constraint and the fault-tolerant patterns extraction when mining strings (presented in Chapter 4) is that efficient solvers are available for specific (ad-hoc) conjunctions of primitive constraints.

An alternative promising approach, proposed by [DJDM02, DD03], is to consider a generic theory, which handles arbitrary Boolean compositions of constraints that are either anti-monotonic or monotonic (see Section 3.1 in Page 45). Such theory was implemented to mine string patterns in string data sets, and resulted in the generic solvers `VST` and `FAVST`. A key issue for designing efficient generic solver is to exploit the opportunities for search space pruning, associated to constraint properties (like anti-monotonicity and its dual monotonicity property).

The research in string pattern mining have recently resulted in a solver that performs the extraction under the minimum frequency constraint, the conjunction of minimum and maximum frequency constraints and some statistic constraints [FHK05, FHK06]. The particularity of this solver is that it is optimal in time, i.e., time is linear in the input and the output size. The key of this approach resides in the use of array-based data structures, in the concrete, suffix-array and lcp-array. This very interesting research work should not be however confused with the generic solvers. The solver of [FHK05, FHK06] is tuned for exploiting the exact-frequency constraints, and cannot solve the constraints having only the (anti-)monotonicity property. Therefore it cannnot be employed as a framework for fault-tolerant string pattern mining through the similarity and soft-frequency constraints.

In most of the application domains, the notion of similarity between two entities ε_1 and ε_2 informally means a "small difference" between ε_1 and ε_2. Obviously, the property "small difference" should not be propagated too far, i.e., the relation of similarity should not be transitive. A similarity constraint (see Definition 4.1 in Page 59), establishing a non-transitive similarity relation between two entities, is fundamentally neither monotonic nor anti-monotonic, since a non-transitive similarity relation can not be isomorphic to a generalisation relation. Due to this property, a fault-tolerant pattern extraction can not benefit from recent algorithmic breakthrough in generic solver design.

Problem Statement

The objective is to formulate the similarity and soft-frequency constraints so that they can be solved efficiently, i.e., as Boolean combinations of anti-monotonic and monotonic primitive constraints. This will enable not only to solve these constraints efficiently, but also to design the generic solver `Marguerite-{Sim,SoftFr}` that employs the efficient generic strategies [DJDM02, DD03, DD04] for solving arbitrary combinations of similarity and/or soft-frequency constraints with other (anti)-monotonic constraints.

5.2 Similarity Constraint

In this section we present our contribution on handling the similarity constraint by exploiting the associated (anti-)monotonicity properties, published in [MB06].

The meaningful similarity relation can not be transitive and therefore the associated similarity constraint is neither anti-monotonic nor monotonic.

Example 5.1 *Let $\Sigma = \{a, c, g, t\}$ be an alphabet, and $\mathcal{L}_\Sigma$ be a string pattern language on Σ. Let $\sigma =$ aactcgc be a reference string on that alphabet. The edit similarity constraint $\mathsf{EditDistSim}_\phi(\sigma, 2)$ (see Definition 4.7 in Page 62) evaluates* true *for example the patterns $\phi_1 =$ aactc, $\phi_2 =$ actcg, $\phi_3 =$ taactcgcc, and* false *for example the patterns $\phi_4 =$ actc, $\phi_5 =$ ct, $\phi_6 =$ ataactcgcc. Note that the constraint $\mathsf{EditDistSim}_\phi(\sigma, 2)$ is neither anti-monotonic (ϕ_4 is more general than ϕ_1), nor monotonic (ϕ_6 is more specific than ϕ_1).*

Recent advance in data mining research have resulted in the theory to efficiently evaluate the arbitrary Boolean compositions of (anti-)monotonic constraints(see Section 3.1 in Page 45). Take notice that, thanks to these results, a non-(anti-)monotonic constraint can profit from efficient evaluation strategies, if only it can be reformulated as a Boolean combination of (anti-)monotonic constraints. Therefore we search how to express a similarity constraint as a Boolean composition of such constraints.

5.2.1 *LCS* Similarity Constraint

It makes sense to evaluate the similarity between two strings, based on their longest common subsequence (LCS), which is the longest pairing of their matching symbols, allowing all possible interruptions in either of the strings (see Section 4.2.1.4). Interestingly, given two strings σ^1 and σ^2, there is a stable relation between the length of $LCS(\sigma^1, \sigma^2)$ and substrings of string σ^1.

Lemma 5.1 *Let σ^1, σ^2 be strings over alphabet Σ. Let $\sigma^{1\prime}$ denote a substring of σ^1. Then, $|LCS(\sigma^1, \sigma^2)| \geq |LCS(\sigma^{1\prime}, \sigma^2)|$.*

Proof 5.1 (Lemma 5.1) *A proof by contradiction is as follows. Assume Φ be a $LCS(\sigma^1, \sigma^2)$, and Φ' be a $LCS(\sigma^{1\prime}, \sigma^2)$. Say that $|\Phi'| = |LCS(\sigma^{1\prime}, \sigma^2)| > |LCS(\sigma^1, \sigma^2)| = |\Phi|$. According to the definition of subsequence (see Definition 1.11 in Page 18), Φ' is also a subsequence of σ^1 and σ^2. If $|\Phi'| > |\Phi|$, then the $LCS(\sigma^1, \sigma^2)$ is Φ' and not Φ. This is a contradiction to the assumption that Φ is the $LCS(\sigma^1, \sigma^2)$.*

Therefore, we propose to search for the (anti-)monotonicity properties of a similarity constraint between two strings by studying their LCS. We first observe that similar strings are expected to have a large enough LCS, and thus formulate a minimum LCS constraint.

Definition 5.1 (Minimum LCS Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a reference string, and $minLCS \in \mathbb{N}$ be a threshold. Define a minimum LCS constraint, denoted $\mathsf{MinLCS}_\phi(\sigma, minLCS)$, to evaluate true if and only if $|LCS(\phi, \sigma)| \geq minLCS$.*

Theorem 5.1 *The minimum LCS constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ is monotonic.*

Proof 5.2 (Theorem 5.1) *Consider substring patterns ϕ and ϕ', such that $\phi' \sqsubseteq \phi$. A consequence of Lemma 5.1 is that if $|LCS(\phi', \sigma)| \geq minLCS$, then $|LCS(\phi, \sigma)| \geq minLCS$.*

Consider the following example.

Example 5.2 *Let $\sigma = \mathsf{tctggga}$ be a reference string over the alphabet $\Sigma = \{a, c, g, t\}$. The string patterns $\phi_1 = \mathsf{gcggga}$ and $\phi_2 = \mathsf{ctggaga}$ from $\mathcal{L}_\Sigma$ satisfy $\mathsf{MinLCS}_\phi(\sigma, 5)$ constraint, since $|LCS(\phi_1, \sigma)| = |\mathsf{cggga}| = 5$ and $|LCS(\phi_2, \sigma)| = |\mathsf{ctggga}| = 6$. Note, that a string pattern $\phi_3 = \mathsf{attagtgttttgggg}$ also satisfies $\mathsf{MinLCS}_\phi(\sigma, 5)$ constraint, since $|LCS(\phi_3, \sigma)| = |\mathsf{ttggg}| = 5$.*

This example illustrates that the minimum LCS constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ enables to specify a minimum number of matching symbols, however, as we can see in the case of string ϕ_3, it does not restrict the number of non-matching symbols and thus is not sufficient to specify a requested degree of similarity. We remind that a subsequence of a string can be obtained by deleting some symbols from that string. By limiting the number of deletions, necessary to perform on a string in order to get its LCS with a reference string, we obtain the needed complementary constraint that bound the number of "errors".

Definition 5.2 (Maximum Deletions Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a reference string, and $maxDels \in \mathbb{N}$ be a threshold. Let fix any $LCS(\phi, \sigma)$, and denote the symbols of ϕ that to not belong to this LCS as* `dels`[1]*. The number of* `dels`*, denoted $DelsLCS(\phi, \sigma)$, is equal to $|\phi| - |LCS(\phi, \sigma)|$. Define a maximum deletions constraint, denoted $\mathsf{MaxDels}_\phi(\sigma, maxDels)$, to evaluate* true*, if and only if $DelsLCS(\phi, \sigma) \leq maxDels$.*

Theorem 5.2 *The maximum deletions constraint $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ is anti-monotonic.*

Proof 5.3 (Theorem 5.2) *Assume that the constraint $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ is satisfied. Let take any LCS of ϕ and σ. Label the symbols of ϕ that belong to the $LCS(\phi, \sigma)$ with labels match, and the symbols of ϕ that do not belong to the $LCS(\phi, \sigma)$ with labels del. The number of symbols labelled with dels is at most $maxDels$. Let us now consider (fixe) any substring ϕ' of ϕ while preserving the labels assigned for ϕ. Obviously, the number of ϕ' symbols labelled with del is less or equal to $maxDels$. If the symbols of ϕ', labelled with match, constitutes the $LCS(\phi', \sigma)$, then we have $\mathsf{MaxDels}_\phi(\sigma, maxDels) \Rightarrow \mathsf{MaxDels}_{\phi'}(\sigma, maxDels)$. If not, a LCS of ϕ' and σ gives rise to a larger number of symbols of ϕ' labelled with match, and thus, necessarily, a smaller number of symbols labelled with del. Thus, we also have $\mathsf{MaxDels}_\phi(\sigma, maxDels) \Rightarrow \mathsf{MaxDels}_{\phi'}(\sigma, maxDels)$.*

Then, we say that two strings are in the LCS similarity relation, if they satisfy the minimum LCS constraint and the maximum deletions constraint.

Definition 5.3 (LCS Similarity Relation) *Let σ^1, σ^2 be two strings over a given alphabet Σ, and $minLCS, maxDels \in \mathbb{N}$ be two thresholds. The strings σ^1 and σ^2 are in LCS similarity relation, denoted $sim_{LCS}(\sigma^1, \sigma^2, minLCS, maxDels)$, if and only if $\mathsf{MinLCS}_{\sigma^1}(\sigma^2, minLCS) \wedge \mathsf{MaxDels}_{\sigma^1}(\sigma^2, maxDels) = \textsf{true}$.*

Based on the LCS similarity relation, we define a LCS similarity constraint.

[1]Notice, that a $LCS(\phi, \sigma)$ can be obtained from ϕ by deleting the `dels`.

Definition 5.4 (*LCS* **Similarity Constraint)** *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a reference string, and $minLCS, maxDels \in \mathbb{N}$ be thresholds. Define a LCS similarity constraint, denoted $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ to evaluate* true, *if and only if a pattern ϕ and a reference string σ are in the LCS similarity relation $sim_{LCS}(\phi, \sigma, minLCS, maxDels)$.*

Example 5.3 *Continuing from Example 5.2, the patterns ϕ_1 and ϕ_2 satisfy the constraint $\mathsf{LCSSim}_\phi(\sigma, 5, 1)$. Pattern $\phi_4 =$ gcgggta satisfies the constraint $\mathsf{LCSSim}_\phi(\sigma, 5, 2)$, since $LCS(\phi_4, \sigma) = |\mathsf{cggga}| = 5$. Pattern ϕ_3 does not satisfy neither constraint $\mathsf{LCSSim}_\phi(\sigma, 5, 1)$ nor constraint $\mathsf{LCSSim}_\phi(\sigma, 5, 2)$.*

Remark 5.1 *The length of a pattern ϕ satisfying $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ constraint, is at least $minLCS$ and at most $|\sigma| + maxDels$. Note that even though the maximal length of a pattern ϕ, satisfying $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$, can be inferred from $|\sigma|$ and $maxDels$, the fact that ϕ satisfies the constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS) \wedge \mathsf{MaxLength}_\phi(|\sigma| + maxDels)$ does not imply that it also satisfies the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$.*

Example 5.4 *Consider a reference string $\sigma =$ agcgac over the alphabet $\Sigma = \{a, c, g, t\}$, a string pattern $\phi =$ gagataga, and the thresholds $minLCS = 4$, $maxDels = 2$. The pattern ϕ satisfies the constraint $\mathsf{MinLCS}_\phi(\sigma, 4) \wedge \mathsf{MaxLength}_\phi(6 + 2)$, but it does not satisfy the constraint $\mathsf{LCSSim}_\phi(\sigma, 4, 2)$, since, even if $LCS(\phi, \sigma) = |agga| = 4$, we have $DelsLCS(\phi, \sigma) = 4 > 2$.*

5.2.2 `Marguerite-Sim` Generic Solver

The *LCS* similarity constraint, which is expressed as a conjunction of a monotonic and an anti-monotonic constraint, can be resolved using the generic constraint exploitation strategies (see Section 3.1 in Page 45). Based on these strategies we developed a generic solver `Marguerite-Sim`, which solves the *LCS* similarity constraint and its arbitrary conjunctions with other (anti-)monotonic constraints on an arbitrary number of string data sets $\mathcal{D}_i$. `Marguerite-Sim` is built on the `FAVST` generic solver [DD04] (see Section 3.2.2.2 in Page 51), and therefore can be seen as its extension[2].

`Marguerite-Sim` generic solver is given in Algorithm 7. It is very similar to the `FAVST` generic solver and the only difference is that instead of Algorithm 2 `InitTree`, presented in Page 54, a new algorithm 8 `InitTreeWithSim` is invoked.

[2]`FAVST` is pronounced `[foust]`, as Faust, a principal character of the *Johann Wolfgang von Goethe's* tragic play *Faust*. *Margaret* (also known as *Gretchen*) is a girl, Faust falls in love with. *Marguerite* is a French version of *Margaret*.

`InitTreeWithSim`, similarly to `InitTree`, constructs a VST on the data set $\mathcal{D}_1$, by simultaneously pushing the LCS similarity constraint. As `FAVST`, `Marguerite-Sim` is written as if one must specify the minimum and maximum frequency constraints for each string data set $\mathcal{D}_i$. If this is not a case then the minimum frequency constraints $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr_i, \mathcal{D}_i)$ can be eliminated by setting the threshold $minFr_i$ value equal to 0. Similarly, the maximum frequency constraints $\mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr_i, \mathcal{D}_i)$ can be eliminated by setting the threshold $maxFr_i$ value equal to ∞. Notice, however, that due to the reused `FAVST` architecture, the minimum frequency constraint with the threshold $minFr$ value 1 is always implicitly enforced, since only the patterns that occur at least once in the (first) string data set $\mathcal{D}_1$ are considered (by putting them into the VST). Also remark, that a maximal length constraint is no longer required, since the length of the solution patterns is implicitly restricted by the LCS similarity constraint.

As in `FAVST`, the evaluation of the other syntactic constraints (see Section 2.2.1.2 in Page 35) does not require a data set scanning, and can be performed efficiently by a simple tree traversal and adequate labelling. The evaluation of the other data-dependent constraints (see Section 2.2.1.1 in Page 34) would require some constraint-specific modifications. Also, as `FAVST`, `Marguerite-Sim` is a *single-scan* algorithm.

Algorithm 7 `Marguerite-Sim`

```
Require: Data sets $\mathcal{D}_1, \ldots, \mathcal{D}_n$, alphabet $\Sigma_\phi$, thresholds $minFr_1, \ldots, minFr_n$,
    $maxFr_1, \ldots, maxFr_n$, $maxLen$, $minLCS$, $maxDels$
Ensure: VST containing the solution patterns (labelled with $\oplus$) to the constraint $\mathcal{A} \wedge \mathcal{M}$
  VST $\leftarrow$ InitTreeWithSim( $\mathcal{D}_1$, $\Sigma_\phi$, $minFr_1$, $maxFr_1$, $minLCS$, $maxDels$ )
  Prune the VST branches that contain only the nodes labelled with $\ominus$
  for $i = 2$ to $n$ do
    CountAndUnmark( $\mathcal{R}$, $\mathcal{D}_i$, $\Sigma_\phi$, $minFr_i$, $maxFr_i$ )
    Prune the VST branches that contain only the nodes labelled with $\ominus$
  end for
```

`InitTreeWithSim` Based on Algorithm 2 `InitTree`, presented in Page 54, we designed the algorithm `InitTreeWithSim`, given in Algorithm 8, that exploits the LCS similarity constraint when constructing a VST. The anti-monotonic sub-constraint $\mathsf{MaxDels}_{\phi}(\sigma, maxDels)$ is pushed during the VST grow-up phase. It is handled similarly to the $\mathsf{MaxLength}_{\phi}(maxLen)$ constraint in the `InitTree` algorithm. Minor algorithmic changes comes to the fact that, if the current node represents a pattern $\phi = \xi_1\xi_2\ldots\xi_n$, satisfying a constraint $\mathsf{MaxDels}_{\phi}(\sigma, maxDels)$, and if a pattern's ϕ extension $\tilde{\phi} = \xi_1\xi_2\ldots\xi_n\xi_{n+1}$ does not satisfy that constraint, its immediate suffix $\xi_2\ldots\xi_n\xi_{n+1}$ does not necessarily satisfy $\mathsf{MaxDels}_{\phi}(\sigma, maxDels)$ (though any prefix of $\tilde{\phi}$ does). The monotonic $\mathsf{MinLCS}_{\phi}(\sigma, minLCS)$ is pushed during the VST unlabelling phase, by the Algorithm 9 `UnlabelVSTForSimAndFr`, presented in Page 97.

Algorithm 8 `InitTreeWithSim`

```
Require: Data set $\mathcal{D}$, alphabet $\Sigma_\phi$, ref. string $\sigma$, thresholds $minFr$, $maxFr$, $minLCS$,
  $maxDels$
Ensure: VST in which the patterns $\phi$ satisfying $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels) \wedge$
  $\mathsf{MinFr}_\phi^{\sqsubseteq,S}(minFr, \mathcal{D})$ are labelled with $\oplus$
  Create VST with the root node $\mathcal{R}$
  $\mathcal{R}$.suffix $\leftarrow \top$, $\mathcal{R}$.label $\leftarrow \oplus$, $\mathcal{R}$.count $\leftarrow 0$, $\mathcal{R}$.last-id $\leftarrow$ undefined
  for all string $S \in \mathcal{D}_1$ with unique id ID do
    $\mathcal{N} \leftarrow \mathcal{R}$
    for all symbol $\xi_i \in S$ do
      if $\xi_i \in \Sigma_\phi$ then {$\xi_i$ is an interesting symbol}
        {check for the $\mathsf{MaxLength}_\phi$ and if $\mathsf{LCSSim}_\phi$ can be satisfied}
        if (depth of $\mathcal{N} \geq maxLen$) $\vee$ (depth of $\mathcal{N} \geq |\sigma| + maxDels$) then
          $\mathcal{N} \leftarrow \mathcal{N}$.suffix
        end if
        if $\mathcal{N}$ has a child with edge $\xi_i$ then
          $c \leftarrow \mathcal{N}$.child with edge $\xi_i$
        else {check if $\phi$ = concat(string($\mathcal{N}$),$\xi_i$) would satisfy $\mathsf{MaxDels}_\phi$}
          $cand \leftarrow$ concat(string($\mathcal{N}$),$\xi_i$)
          $lcs \leftarrow$ computeLCS(cand,$\sigma$)
          $dels \leftarrow |cand| - lcs$
          $z \leftarrow \mathcal{N}$
          $childFound \leftarrow$ false
          while ($dels > maxDels$) $\wedge$ ($z \neq \mathcal{R}$) do {search for other parent for $\xi_i$ }
            $z \leftarrow z$.suffix
            if $z$ has a child with edge $\xi_i$ then
              $c \leftarrow z$ child with edge $\xi_i$
              $childFound \leftarrow$ true
              break
            end if
            $cand \leftarrow$ concat(string(z),$\xi_i$)
            $lcs \leftarrow$ computeLCS(cand,$\sigma$)
            $dels \leftarrow |cand| - lcs$
          end while
          if ($childFound$ = false) $\wedge$ ($z \neq \top$) then
            $\mathcal{N} \leftarrow z$
            $c \leftarrow$ CreateChild($\mathcal{N}$,$\xi_i$) {Algorithm 3 CreateChild is given in p. 54}
            $c$.edge $\leftarrow \xi_i$ {set edge of a new child node}
            $c$.lcs $\leftarrow lcs$
            Add $c$ to $\mathcal{N}$ children
          end if
        end if
        IncreaseFrequency($c$, ID) {Algorithm 4 IncreaseFrequency is given in p. 55}
        $\mathcal{N} \leftarrow c$ {current node gets one level deeper}
      else {$\xi_i$ is an uninteresting symbol}
        $\mathcal{N} \leftarrow \mathcal{R}$ {break input string, continue from the root}
      end if
    end for
  end for
  UnlabelVSTForSimAndFr($\mathcal{R}$, $minLCS$, $minFr$, $maxFr$ )
```

`UnlabelVSTForSimAndFr` Algorithm 9 `UnlabelVSTForSimAndFr` push the monotonic similarity sub-constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ by unlabelling the nodes, such that $\mathsf{string}(\mathcal{N})$ does not satisfy minLCS, with $\ominus$. In addition to this, as its counterpart Algorithm 5 `UnlabelVSTForFr`, given in Page 55, `UnlabelVSTForSimAndFr` unlabels the corresponding VST nodes with $\ominus$ to solve the frequency constraints, and prepares the VST for the subsequent calls to Algorithm 6 `CountAndUnmark`, given in Page 56.

Algorithm 9 `UnlabelVSTForSimAndFr`

```
Require: VST root node $\mathcal{R}$, $minLCS$, $minFr$, $maxFr$
Ensure: Nodes $\mathcal{N}$, such that $\mathsf{string}(\mathcal{N})$ does not satisfy $\mathsf{MinLCS}_\phi(\sigma, minLCS) \wedge$
  $\mathsf{MinFr}_\phi^{\sqsubseteq,S}(minFr, \mathcal{D})$, are labelled with $\ominus$
  for all $\mathcal{N}$ in VST do {traverse the VST and unlabel}
    if ($\mathcal{N}$.lcs $< minLCS$) $\vee$ ($\mathcal{N}$.count $< minFr$) $\vee$ ($\mathcal{N}$.count $> maxFr$) then
      $\mathcal{N}$.label $\leftarrow \ominus$
    end if
    {reset VST for CountAndUnmark traversal}
    $\mathcal{N}$.count $\leftarrow 0$
    $\mathcal{N}$.lcs $\leftarrow 0$
    $\mathcal{N}$.last-id $\leftarrow$ undefined
  end for
```

5.2.3 Experimental Validation

To provide an experimental validation of the defined LCS similarity constraint, we extracted the patterns satisfying the inductive query $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels) \wedge \mathsf{MinFr}_\phi^{\sqsubseteq,S}(1, \mathcal{D})$ in the sets of human promoter sequences[3]. For the inductive query evaluation we implemented the solver `Marguerite-Sim`, presented in the previous section, in `C` programming language. We ran the experiments on a Pentium(R) 4CPU 3.00GHz processor and 1GB main memory.

5.2.3.1 Added Value of Pushing the LCS Similarity Constraint

The interest of the LCS similarity constraint, expressed as a conjunction of anti-monotonic and monotonic sub-constraints, is that it is possible to push it deeply into the extraction phase. The objective of this section is to assess the added value of that possibility to push a similarity constraint and thus to prune the search space efficiently w.r.t. the approach, where a similarity constraint is not pushed, but instead computed by post-processing a set of patterns, calculated beforehand. For the exper-

[3]Available from UCSC Genome Browser website on
http://hgdownload.cse.ucsc.edu/goldenPath/hg17/bigZips/

iments we used the string data set, composed of the upstream promoter sequences of the human genome (20647 sequences of length 5000, 101MB).

To assess the approach of constraint pushing, we extracted the patterns that satisfy the LCS similarity constraint $\mathsf{LCSSim}_{\phi}(\sigma, minLCS, maxDels)$ with the reference string σ of different lengths and with more or less selective parameters $minLCS$ and $maxDels$[4]. We started with a reasonable length of the reference string σ, equal to 6, and augmented that length until the extraction became unfeasible. We did not investigate the entire space of the parameters $minLCS$ and $maxDels$ values, because of the following semantic issues:

- A similarity constraint is intended to enforce enough similarity, a large parameter $minLCS$ and the small parameter $maxDels$ values in the LCS similarity constraint $\mathsf{LCSSim}_{\phi}(\sigma, minLCS, maxDels)$ do not imply a similarity anymore;
- The parameters $minLCS$ and $maxDels$ are related (see Definition 5.2 in Page 93). Enforcing $DelsLCS(\phi, \sigma) \leq maxDels$ also means that at least $(|\phi| - maxDels)$ symbols of ϕ constitute the $LCS(\phi, \sigma)$. Thus, the value of the parameter $maxDels$ specifies a lower bound for the length of $LCS(\phi, \sigma)$, which can be raised by a larger $minLCS$ parameter value in the $\mathsf{MinLCS}_{\phi}(\sigma, minLCS)$ subconstraint.
- The meaning of “large” and “small” parameter value is dependent on the reference string length $|\sigma|$.

Concerning the post-processing approach, we focus on the cost to extract the set of patterns, which then can be filtered to solve the LCS similarity constraint. We do not take into account the cost of filtering, since it is negligible compared with the cost of the pattern extraction. Given that the constraint $\mathsf{LCSSim}_{\phi}(\sigma, minLCS, maxDels)$ implies $|\phi| \leq |\sigma| + maxDels$ (see Remark 5.1 in Page 94), to evaluate a post-processing approach we focus on the extraction of patterns that satisfy the constraint $\mathsf{MaxLength}_{\phi}(|\sigma| + maxDels)$.

To evaluate the added value of the pruning by the LCS similarity constraint, we focus on the phase of the VST[5] construction, since it is the most expensive and crucial for the solver `FAVST` (and thus, also for the solver `Marguerite-Sim`): once a VST is available, it can always be further pruned using any (anti)-monotonic constraint by a simple tree traversal. The memory and time consumption scalability studies of both, constraint pushing and constraint post-processing, approaches are presented, correspondingly, in the Table 5.1 and Table 5.2. The first four columns in these

[4]The smaller the parameter $minLCS$ and the larger the parameter $maxDels$, the less selective is the LCS similarity constraint.

[5]Version Space Tree (VST) is a data structure used by `FAVST` and `Marguerite`. For more details on VST, see Section 3.2.1 in Page 46.

Table 5.1: Scalability of Memory Consumption

LCSSim_ϕ	$\|\sigma\|$	$minLCS$	$maxDels$	Number of nodes in VST		$maxLen$
				Pushing LCSSim_ϕ	Post-proc. LCSSim_ϕ	
LCSSim_ϕ^1	6	4	1	527	21 844	7
LCSSim_ϕ^2	7	5	1	642	87 380	8
LCSSim_ϕ^3	7	5	2	5 280	349 524	9
LCSSim_ϕ^4	10	8	1	5 836	5 386 756	11
LCSSim_ϕ^5	10	8	2	54 524	18 143 975	12
LCSSim_ϕ^6	10	8	3	406 623	not feasible	13
LCSSim_ϕ^7	15	13	1	106 455	not feasible	16
LCSSim_ϕ^8	15	12	2	1 024 215	not feasible	17
LCSSim_ϕ^9	20	18	1	1 861 901	not feasible	21
$\mathsf{LCSSim}_\phi^{10}$	25	23	1	1 027 140	not feasible	26
$\mathsf{LCSSim}_\phi^{11}$	30	28	1	not feasible	not feasible	31

tables give the parameter values for the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$. The fifth column "Pushing LCSSim_ϕ" gives the results, when the corresponding LCS similarity constraint is pushed[6]. The seventh column "$maxLen$" gives the threshold of the $\mathsf{MaxLength}_\phi(maxLen = |\sigma| + maxDels)$ constraint, used for post-processing. The sixth column "Post-proc. LCSSim_ϕ" gives the results, when the corresponding LCS similarity constraint is post-processed.

The experimental results on memory consumption, presented in Table 5.1, reveal that the power of the anti-monotonic LCS similarity sub-constraint $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ is promising. The approach of pushing the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ scales much better on the size of the VST, when $|\sigma|$ and $maxDels$ increase. Moreover, it enables to go far away beyond the limits of the post-processing approach. Starting from the LCSSim_ϕ^6, a VST construction, while pushing only $\mathsf{MaxLength}_\phi(maxLen)$, is no longer possible since the memory consumption memory exceeds 1GB, whereas exploiting $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ reduces a size of VST to 406623 nodes, what requires approximately 20MB of memory.

The VST construction, when pushing the LCS similarity constraint, takes more time than the post-processing approach, because of the LCS computation (see Ta-

[6]We remind, that when pushing the $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ constraint, it is the anti-monotonic sub-constraint $\mathsf{MaxDels}_\phi(\sigma, maxDels)$, that is exploited to prune the nodes during VST construction. The monotonic sub-constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ is employed once the VST is constructed in order to get every pattern ϕ, satisfying $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$. Therefore, in the concrete, this study concerns the pruning by $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ capacity.

Table 5.2: Scalability of Time Consumption

LCSSim_ϕ	$\lvert\sigma\rvert$	$minLCS$	$maxDels$	Time		$maxLen$
				Pushing LCSSim_ϕ	Post-proc. LCSSim_ϕ	
LCSSim^1_ϕ	6	4	1	1min 40s	23s	7
LCSSim^2_ϕ	7	5	1	1min 49s	53s	8
LCSSim^3_ϕ	7	5	2	2min 17s	1min 17s	9
LCSSim^4_ϕ	10	8	1	2min 50s	1min 59s	11
LCSSim^5_ϕ	10	8	2	3min 42s	2min 36s	12
LCSSim^6_ϕ	10	8	3	4min 47s	not feasible	13
LCSSim^7_ϕ	15	13	1	5min 07s	not feasible	16
LCSSim^8_ϕ	15	12	2	6min 48s	not feasible	17
LCSSim^9_ϕ	20	18	1	8min 16s	not feasible	21
$\mathsf{LCSSim}^{10}_\phi$	25	23	1	12min 55s	not feasible	26
$\mathsf{LCSSim}^{11}_\phi$	30	28	1	not feasible	not feasible	31

ble 5.2). To compute the LCS, we have implemented a classical dynamic programming approach of time complexity $O(nm)$ [Hir75]. There is a room for improvements on such a computation (see, e.g., [Apo97] for a survey). Nevertheless, the time cost of pushing the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ is acceptable, since it enables to perform extractions that would not have been possible otherwise.

Even if we consider only several points in the space of the parameters of the $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ constraint, the behaviour for other parameter sets can be induced from the presented results: large $\lvert\sigma\rvert$ and small $minLCS$ parameter values would result in a huge (unless pattern length is limited by a rather small $maxDels$) set of solutions (and thus significant memory and time consumption), since almost every pattern becomes similar to a reference string σ, especially when the alphabet is small.

5.2.3.2 Selectivity of the LCS Similarity Constraint

The constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ intrinsically specifies the lower and the upper bounds of the length of the pattern satisfying it (see Remark 5.1 in Page 94). We thus investigate the selectivity of the LCS similarity constraint by comparing it with the selectivity of the constraint on the pattern length $\mathsf{MinLength}_\phi(minLCS) \wedge \mathsf{MaxLength}_\phi(\lvert\sigma\rvert + maxDels)$.

As in the previous section, for the experiments we used 20647 upstream promoter

Table 5.3: Selectivity of the *LCS* Similarity Constraint

LCSSim_ϕ	Number of patterns satisfying the constraint	
	LCSSim_ϕ	$\mathsf{MinLength}_\phi(minLCS) \wedge$ $\mathsf{MaxLength}_\phi(\lvert\sigma\rvert + maxDels)$
LCSSim^3_ϕ	3 196	349 184
LCSSim^4_ϕ	1 132	5 364 912
LCSSim^5_ϕ	15 965	18 122 131

sequences of length 5000. The results are presented in Table 5.3. The first column represents the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ (it is explicitly given in the first four columns of Table 5.1 or Table 5.2). The second column gives the number of patterns that satisfy the corresponding *LCS* similarity constraint. The third column gives the number of patterns that satisfy the length constraint, derived from the corresponding *LCS* similarity constraint. The results reveal that the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ is quite selective, and much more powerful than the length constraint it induces. Observe that the selectivity of the *LCS* similarity constraint is not linearly related to its pruning capacity, i.e., a larger solution set can be stored in a smaller VST (compare the column "Number of patterns satisfying the constraint LCSSim_ϕ" in Table 5.3 and the column "Pushing LCSSim_ϕ" in Table 5.1 for the constraints LCSSim^3_ϕ and LCSSim^4_ϕ). The reason is that the large values of $|\sigma|$ and the parameter *maxDels* can require many nodes to represent a quite restricted solution set (keep in mind the combinatorial issues, the number and the length of branches in a VST).

5.2.3.3 Empirical Validation

We have defined a similarity constraint as the 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 the generic solver `Marguerite`. Such a definition is, however, only valuable if it captures a useful and intuitive similarity measure. To verify this empirically, we assume 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 the experiments we used the promoter sequences of chromosomes X and Y (1004 sequences of 1000 nucleotides). To obtain noised set of sequences, we perturbed that data by a noise. Introducing $z\%$ of noise means that each symbol undergoes an error event with a probability $z/100$. In case of the error event, we assume that the deletion of a symbol, its substitution by a different one, or an insertion of a supplementary symbol are equally possible. Also, in the case of an insertion or a substitution, any other symbol[7] has an equal probability to be chosen.

In the constraint $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ we used a 10-length string of reference σ, which is present on 25 data sequences, once on each of them. Then, we introduced a 5% noise on the promoter sequences. In the noised sequences, the reference string σ occurs exactly on the 15 same sequences, as in the initial promoter sequences, and on 1 sequence, where it is not present in the initial data[8]. Among the string patterns that satisfy the $\mathsf{LCSSim}_\phi(\sigma, 9, 1)$ constraint in the noised data, there are all 25 (15 exact and 10 approximate) occurrences of σ, which were present (possibly shifted) in the initial data, and 57 other string patterns. It is encouraging that LCSSim_ϕ constraint enables to recover all occurrences. Yet, the number of false positives is quite large. Notice however, that the problem of discriminating between

- pattern that are perturbed occurrences of the reference pattern,
- patterns that are perturbed or not perturbed occurrences of those that were similar to a reference string in the original data, and
- patterns that have become similar to the reference pattern due to the noise,

is different from our current goal, i.e., designing a constraint that captures a similarity.

5.3 Soft-Frequency Constraint

In this section we present our contribution on handling the soft-frequency constraint by exploiting the assiociated (anti-)monotonicity properties, published in [MB07].

With several exceptions[9], the fault-tolerant pattern extraction task from string data, in constraint-based data mining terms, can be formulated as the task of extracting all the patterns ϕ that satisfy a minimum soft-frequency constraint, i.e., a minimum frequency constraint when the associated *match* function evaluates true for a set of objects, similar to the pattern ϕ (see Section 4.3 in Page 68).

The evaluation of the minimum soft-frequency constraint $\mathsf{MinSoftFr}_\phi(minFr, \mathcal{D})$ or the maximum soft-frequency constraint $\mathsf{MaxSoftFr}_\phi(maxFr, \mathcal{D})$ requires to compute a soft-frequency $\mathsf{SoftFr}(\phi, \mathcal{D})$ of a pattern ϕ. For this, one needs to find all

[7]I.e., not equal to the one that is concerned by an error event.

[8]The reference string is considered to occur exactly in the noised data, if it is present on the same sequence as in the initial data, allowing a small shift in its position of occurrence (because of the random insertions and deletions produced by noise).

[9]See, e.g., [SVC95], presented in Section 4.3.2

objects X in data $\mathcal{D}$ that are similar to ϕ, i.e., to evaluate the constraint $\mathsf{Sim}_X(\phi) \wedge \mathsf{MinFr}_X(1, \mathcal{D})$. As explained in Research Context in Page 90, a similarity constraint $\mathsf{Sim}_X(\phi)$ can not have the (anti-)monotonicity properties. The step of computing the soft-frequency $\mathsf{SoftFr}(\phi, \mathcal{D})$ for each pattern ϕ is thus a non-trivial extraction task, where the efficient (anti-)monotonicity-based pruning strategies can not be applied. In addition to this, the minimum soft-frequency constraint $\mathsf{MinSoftFr}_\phi(minFr, \mathcal{D})$ (resp. maximum soft-frequency constraint $\mathsf{MaxSoftFr}_\phi(maxFr, \mathcal{D})$) is not guaranteed to be anti-monotonic (resp. monotonic), since there is no guaranteed stable relation between the string pattern ϕ substrings and superstrings and the cardinality of the sets of their soft-occurrences (and thus the satisfaction of the soft-frequency constraint).

The LCS similarity constraint, introduced in Section 5.2.1, is expressed by the conjunction of two subconstraints that have the appealing monotonicity and anti-monotonicity properties. Therefore, differently from the other similarity constraints, the LCS similarity constraint take advantage of these properties and thus can be solved efficiently by the generic solver `Marguerite-SoftFr`. Hence, we propose to use this similarity constraint to find the soft-occurrences of a pattern ϕ, necessary to evaluate its soft-frequency.

The LCS similarity constraint $\mathsf{LCSSim}_X(\phi, minLCS, maxDels)$ is parametrized by $minLCS$, which is the minimum required length of $LCS(X, \phi)$, and by $maxDels$, which is the maximum allowed number of deletions $DelsLCS(X, \phi)$ (see Definition 5.2 in Page 93). Note, that the value of $minLCS$ threshold, enforcing a minimal number of matches, is directly dependent on the pattern length $|\phi|$ (whereas the value of $maxDels$ threshold is dependent on the $|LCS(X, \phi)|$). When using the LCS similarity constraint to find the soft-occurrences of a pattern ϕ in order to evaluate the soft-frequency constraints, the lengths of the putatively interesting patterns ϕ are various. Therefore, the threshold $minLCS$ in the sub-constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ can not be fixed. Thus, we will reformulate the subconstraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ so that the threshold of the new constraint will dependent on the $|LCS(X, \phi)|$, instead of the length of ϕ.

Interestingly, when the match function uses the newly defined similarity relation to acknowledge a substring σ as a soft-occurrences of a pattern ϕ, the associated soft-frequency constraints are guaranteed to be (anti-)monotonic (see Theorem 5.4 in Page 106). These properties are proved for the *match* function that matches a pattern ϕ against a string object S, but take notice that they are also valid when a pattern ϕ is matched against a substring object s.

5.3.1 *InsDels* Similarity Relation

The LCS similarity constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$, enforcing a large enough LCS between the strings σ and ϕ, can be reformulated so that it is no longer directly dependent on the pattern length $|\phi|$. It is expressed in terms of *insertions* and is called a maximum insertions constraint.

Definition 5.5 (Maximum Insertions Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a reference string, and $maxIns \in \mathbb{N}$ be a threshold. Let fix any $LCS(\phi, \sigma)$, and denote the symbols of σ that do not belong to this LCS as* `ins`*. The number of* `ins`*, denoted $InsLCS(\phi, \sigma)$, is equal to $|\sigma| - |LCS(\phi, \sigma)|$. Define a maximum insertions constraint, denoted $\mathsf{MaxIns}_\phi(\sigma, maxIns)$, to evaluate* **true***, if and only if $InsLCS(\phi, \sigma) \leq maxIns$.*

Remark 5.2 *Notice, that σ can be produced from ϕ by inserting into ϕ the symbols* `ins` *and deleting from ϕ the symbols* `dels` *(see Definition 5.2 in Page 93).*

Theorem 5.3 *Maximum insertions constraint $\mathsf{MaxIns}_\phi(\sigma, maxIns)$ is monotonic.*

Proof 5.4 (Theorem 5.3) *Consider the substring patterns ϕ and ϕ', such that $\phi' \sqsubseteq \phi$. Then, $LCS(\phi', \sigma) \leq LCS(\phi, \sigma)$ (Lemma 5.1, proved in Page 92), so $InsLCS(\phi', \sigma) \leq InsLCS(\phi, \sigma)$. Consequently, if $InsLCS(\phi', \sigma) \leq maxIns$, then $InsLCS(\phi, \sigma) \leq maxIns$.*

Example 5.5 *Consider a reference string $\sigma = \mathsf{tctggga}$ over the alphabet $\Sigma = \{a, c, g, t\}$. The string patterns $\phi_1 = \mathsf{gcggga}$ and $\phi_2 = \mathsf{ctggaga}$ satisfy $\mathsf{MaxIns}_\phi(\sigma, 2)$ constraint, since $InsLCS(\phi_1, \sigma) = |\sigma| - |\mathsf{cggga}| = 2$ and $InsLCS(\phi_2, \sigma) = |\sigma| - |\mathsf{ctggga}| = 1$. Note, that a string pattern $\phi_3 = \mathsf{attagtgttttgggg}$ also satisfies the $\mathsf{MaxIns}_\phi(\sigma, 2)$ constraint, since $InsLCS(\phi_3, \sigma) = |\sigma| - |\mathsf{ttggg}| = 2$.*

This example illustrates that the maximum insertions constraint $\mathsf{MaxIns}_\phi(\sigma, maxIns)$, as the minimum LCS constraint $\mathsf{MinLCS}_\phi(\sigma, minLCS)$, enables to specify the minimum number of matching symbols, but, as we can see in the case of string pattern ϕ_3, it does not restrict the number of non-matching symbols, and thus is not sufficient to enforce a similarity. To bound the number of "errors", i.e., mismatches we will use the maximum deletions constraint $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ (see Definition 5.2 in Page 93).

Definition 5.6 (*InsDels* Similarity Relation) *Let σ^1, σ^2 be two strings over a given alphabet Σ, and $maxIns, maxDels \in \mathbb{N}$ be two thresholds. The strings σ^1 and*

σ^2 are in the insertions-deletions (InsDels) similarity relation, denoted $sim_{InsDels}(\sigma^1, \sigma^2, maxIns, maxDels)$, if and only if $\mathsf{MaxIns}_\phi(\sigma, maxIns) \wedge$ $\mathsf{MaxDels}_{\sigma^1}(\sigma^2, maxDels) = \mathsf{true}$.

Based on the $InsDels$ similarity relation, we can define the $InsDels$ similarity constraint.

Definition 5.7 ($InsDels$ Similarity Constraint) *Let $\mathcal{L}_\Sigma$ be a string pattern language, ϕ be a pattern from $\mathcal{L}_\Sigma$, σ be a reference string, and $maxIns, maxDels \in \mathbb{N}$ be two thresholds. Define an InsDels similarity constraint, denoted $\mathsf{InsDelsSim}_\phi(\sigma, maxIns, maxDels)$ to evaluate true, if and only if a pattern ϕ and a reference string σ are in the insertions-deletions similarity relation $sim_{InsDels}(\phi, \sigma, maxIns, maxDels)$.*

5.3.2 Soft-matching through $InsDels$ Similarity Relation

We use the $InsDels$ similarity relation, which is a conjunction of the monotonic and anti-monotonic subconstraints, and thus can be evaluated efficiently, to find the soft-occurrences of a string pattern ϕ, i.e., the objects X that softly match the pattern ϕ.

Definition 5.8 ($InsDels$ Match Function) *Let $\mathcal{U}_\Sigma$ be a universe of strings over an alphabet Σ, S be a string object from $\mathcal{U}_\Sigma$, and $maxIns, maxDels \in \mathbb{N}$ be the thresholds. Let ϕ be a pattern from a string pattern language $\mathcal{L}_\Sigma$. Define an InsDels match function $match_{InsDels}(\phi, S, maxIns, maxDels)$ to evaluate true, if and only if there exists a string σ such that $\sigma \sqsubseteq S$ and $sim_{InsDels}(\phi, \sigma, maxIns, maxDels) =$ true.*

Definition 5.9 ($InsDels$ Soft-Frequency) *The frequency constraint $\mathsf{Fr}(\phi, \mathcal{D})$ that is evaluated using the InsDels match function $match_{InsDels}(\phi, S, maxIns, maxDels)$, is called an InsDels soft-frequency and is denoted $\mathsf{SoftFr}_{InsDels,S}(\phi, \mathcal{D}, maxIns, maxDels)$.*

Definition 5.10 ($InsDels$ Soft-Frequency Constraints) *Minimum frequency constraint $\mathsf{MinFr}_\phi(minFr, \mathcal{D})$ that uses the InsDels match function is called a minimum InsDels soft-frequency constraint and is denoted $\mathsf{MinSoftFr}_\phi^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$. Maximum frequency constraint $\mathsf{MaxFr}_\phi(maxFr, \mathcal{D})$ that uses the InsDels match function is called a maximum InsDels soft-frequency constraint and is denoted $\mathsf{MaxSoftFr}_\phi^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels)$.*

Example 5.6 *Let Σ be the alphabet $\{a, c, g, t\}$, $\mathcal{L}_\Sigma$ be a string pattern language on Σ and $\mathcal{D} = \{\mathsf{acg}, \mathsf{act}, \mathsf{gt}, \mathsf{t}, \mathsf{gt}\}$ be a string data set. There are five string objects in $\mathcal{D}$: $S_1 = \mathsf{acg}$, $S_2 = \mathsf{act}$, $S_3 = \mathsf{gt}$, $S_4 = \mathsf{t}$ and $S_5 = \mathsf{gt}$. $\mathsf{SoftFr}_{InsDels,S}(\mathsf{acg}, \mathcal{D}, 1, 1)$ is equal to 2, since there are two string objects (S_1 and S_2) that contain pattern's acg soft-occurrences (cg, act, acg, ac). $\mathsf{SoftFr}_{InsDels,S}(\mathsf{gt}, \mathcal{D}, 1, 1)$ is equal to 5, since there are five string objects (S_1, S_2, S_3, S_4 and S_5) that contain the pattern's gt soft-occurrences (t, gt, g, ct,cg). $\mathsf{MinSoftFr}_{\mathsf{gt}}^{InsDels,S}(4, \mathcal{D}, 1, 1)$ and $\mathsf{MaxSoftFr}_{\mathsf{acg}}^{InsDels,S}(2, \mathcal{D}, 1, 1)$ are examples of satisfied soft-frequency constraints.*

The use of the *InsDels match* function to find the soft-occurrences of a pattern ϕ enables not only to efficiently compute the pattern's ϕ soft-frequency, but also, what is equally important, it guarantees the anti-monotonicity (resp. monotonicity) properties for the minimum (resp. maximum) soft-frequency constraints.

Theorem 5.4 *The minimum InsDels soft-frequency constraint $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$ is anti-monotonic (dually, the maximum InsDels soft-frequency constraint $\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels)$ is monotonic), if $maxDels \geq maxIns$.*

Remark 5.3 *The InsDels similarity relation between ϕ and its soft-occurrence σ, induced by the match function $match_{InsDels}(\phi, S, maxIns, maxDels)$, is symmetric*[10]*, if and only if $maxIns = maxDels$.*

Remark 5.4 *In most application domains, it makes sense to use a similarity relation that is symmetric. When the InsDels similarity relation $sim_{InsDels}(\phi, \sigma, maxIns, maxDels)$ is symmetric (i.e., $maxIns = maxDels$), the associated InsDels soft-frequency constraints have the desired (anti-)monotonicity properties.*

Proof 5.5 (Theorem 5.4) *The minimum InsDels soft-frequency constraint $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$ is anti-monotonic, if given a string σ that is a soft-occurrence of a string pattern ϕ, i.e., such that $sim_{InsDels}(\phi, \sigma, maxIns, maxDels) = \mathsf{true}$, for every pattern $\phi' \sqsubseteq \phi$, there exists a string $\sigma' \sqsubseteq \sigma$, such that it is a soft-occurrence of the pattern ϕ', i.e., $sim_{InsDels}(\phi', \sigma', maxIns, maxDels) = \mathsf{true}$. Let take any $LCS(\phi, \sigma)$ and label the corresponding ϕ and σ symbols as `ins` and `dels` (see Definition 5.5 in Page 104 and Definition 5.2 in Page 93). Let fix any pattern's $\phi = \phi_1 \ldots \phi_n$ substring $\phi' = \phi_k \ldots \phi_l$. Note, that the pattern $\phi' = \phi_k \ldots \phi_l$ contains at most maxIns insertions `ins`. There are four possible cases:*

[10]I.e., if ϕ is similar to σ, then σ is similar to ϕ.

1. *The symbols ϕ_k and ϕ_l belong to the $LCS(\phi, \sigma)$, and thus they were not labelled as* `ins`*. Now, fix string's $\sigma = \sigma^1 \ldots \sigma^m$ substring $\sigma' = \sigma^h \ldots \sigma^j$, so that σ^h is paired with ϕ_k and σ^j is paired with ϕ_l in the $LCS(\phi, \sigma)$. The pattern's ϕ substring $\phi' = \phi_k \ldots \phi_l$ contains at most $maxIns$ symbols labelled as* `ins`*, and the string's σ substring $\sigma' = \sigma^h \ldots \sigma^j$ contains at most $maxDels$ symbols labelled as* `dels`*. Notice, that by computing the $LCS(\phi', \sigma')$ and relabelling ϕ' and σ' symbols as* `ins` *and* `dels`*, we can only increase the $|LCS(\phi', \sigma')|$, and thus decrease the actual number of* `ins` *and* `dels`*. Therefore, σ' and ϕ' are necessarily in the relation $sim_{InsDels}(\phi', \sigma', maxIns, maxDels)$, what means that the string σ' is pattern's ϕ' soft-occurrence.*

2. *The pattern $\phi' = \phi_k \ldots \phi_l$ contains only symbols that were labelled as* `ins`*. Note, that $|\phi_k \ldots \phi_l| \leq maxIns$. Fix any string's $\sigma = \sigma^1 \ldots \sigma^m$ substring $\sigma' = \sigma^h \ldots \sigma^j$, such that $|\sigma'| = |\phi'|$. Compute the $LCS(\phi', \sigma')$. In the worst case, when $|LCS(\phi', \sigma')| = 0$, the pattern ϕ' contains $|\phi'|$ symbols labelled* `ins` *and the string σ' contains $|\sigma'| = |\phi'|$ symbols labelled* `dels`*. Given that $|\phi_k \ldots \phi_l| \leq maxIns$ and that $maxDels \geq maxIns$, σ' and ϕ' are in the relation $sim_{InsDels}(\phi', \sigma', maxIns, maxDels)$, what means that the string σ' is a pattern's ϕ' soft-occurrence.*

3. *The symbol ϕ_k was labelled as* `ins`*, and the symbol ϕ_l was not labelled as* `ins`*, i.e., ϕ_l belongs to the $LCS(\phi, \sigma)$. Fix the first symbol ϕ_w that is not labelled as* `ins`*, so that $k < w \leq l$. Now fix the string's $\sigma = \sigma^1 \ldots \sigma^m$ substring $\sigma' = \sigma^h \ldots \sigma^j$, such that in the $LCS(\phi, \sigma)$ σ^h is paired with ϕ_w and σ^j is paired with ϕ_l. Recall that the pattern $\phi' = \phi_k \ldots \phi_l$ contains at most $maxIns$ symbols that were labelled as* `ins` *and that the symbols of $\phi_k \ldots \phi_{w-1}$ were labelled as* `ins`*, and apply Case 1 for $\phi_w \ldots \phi_k$ and $\sigma^h \ldots \sigma^j$ to prove that the number of symbols labelled as* `dels` *in $\sigma^h \ldots \sigma^j$ is at most $maxDels$. By computing the $LCS(\phi', \sigma')$ and relabelling ϕ' and σ' symbols as* `ins` *and* `dels`*, we can only increase the $|LCS(\phi', \sigma')|$, and thus decrease the actual number of* `ins` *and* `dels`*. Therefore, σ' and ϕ' are necessarily in the relation $sim_{InsDels}(\phi', \sigma', maxIns, maxDels)$, what means that the string σ' is pattern's ϕ' soft-occurrence.*

4. *The symbol ϕ_k was not labelled as* `ins`*, i.e., ϕ_k belongs to the $LCS(\phi, \sigma)$, and the symbol ϕ_l was labelled as* `ins`*. This case is analogous to Case 3.*

Example 5.7 (Counter-example) *This example shows that minimum InsDels soft-frequency constraint $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$ is not anti-monotonic, when $maxDels < maxIns$.*
Let Σ be the alphabet $\{a, c, g, t\}$, $\mathcal{L}_{\Sigma}$ be a string pattern language on Σ and $\mathcal{D} = \{\mathsf{aagc}, \mathsf{gc}, \mathsf{gc}\}$ be a string data set. The $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(2, \mathcal{D}, 2, 1)$ constraint (i.e., with fault-tolerant matching parameters $maxIns = 2$ and $maxDels = 1$) is satisfied for the pattern $\phi = \mathsf{aagc}$, but not satisfied for its substring $\phi' = \mathsf{aa}$, since $\mathsf{SoftFr}_{InsDels,S}(\mathsf{aagc}, \mathcal{D}, 2, 1) = 3$ and $\mathsf{SoftFr}_{InsDels,S}(\mathsf{aa}, \mathcal{D}, 2, 1) = 1$.

5.3.3 `Marguerite-SoftFr` Generic Solver

The `Marguerite-SoftFr` generic solver is designed to answer an inductive query that is a conjunction of anti-monotonic and monotonic minimum and maximum *InsDels* soft-frequency constraints[11] (see Definition 5.10 in Page 105) with other (anti-)monotonic on an arbitrary number of string data sets $\mathcal{D}_i$. `Marguerite-SoftFr`[12] was built on the `FAVST` generic solver [DD04], presented in Section 3.2.2.2 in Page 51.

To ease the comprehension, we made an effort to keep the structure of `Marguerite-SoftFr` as close as possible to the structure of `FAVST` (see Algorithm 1 in Page 51). As `FAVST`, to simplify the pseudo-code, the `Marguerite-SoftFr` is written as if one must specify the minimum and maximum *InsDels* soft-frequency constraints for each string data set $\mathcal{D}_i$. This is not a case, and a minimum soft-frequency constraint $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr_i, \mathcal{D}_i, maxIns_i, maxDels_i)$ can be eliminated by setting the threshold $minFr_i$ value equal to 0. Similarly, a maximum frequency constraint $\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr_i, \mathcal{D}_i, maxIns_i, maxDels_i)$ can be eliminated by setting the threshold $maxFr_i$ value equal to ∞. Concerning other constraints, the evaluation of the syntactic constraints (see Section 2.2.1.2 in Page 35) do not require a data set scanning, and can be performed efficiently by a simple tree traversal and adequate labelling. The evaluation of other data-dependent (see Section 2.2.1.1 in Page 34) constraints would require some constraint-specific modifications.

Algorithm 10 `Marguerite-SoftFr`

```
Require: Data sets $\mathcal{D}_1, \dots, \mathcal{D}_n$, alphabet $\Sigma_\phi$, thresholds $minFr_1, \dots, minFr_n$,
  $maxFr_1, \dots, maxFr_n$, $maxDels_1, \dots, maxDels_n$, $maxIns_1, \dots, maxIns_n$, $maxLen$
Ensure: VST containing the solution patterns (labelled with $\oplus$) to the constraint $\mathcal{A} \wedge \mathcal{M}$
  VST ← InitTreeForSoft( $\mathcal{D}_1$, $\Sigma_\phi$, $minFr_1$, $maxFr_1$, $maxDels_1$, $maxIns_1$, $maxLen + maxDels_1$ )
  Prune the VST branches that contain only the nodes labelled with $\ominus$
  for $i = 2$ to $n$ do
    CountAndUnmarkForSoftFr($\mathcal{R}$, $\mathcal{D}_i$, $\Sigma_\phi$, $minFr_i$, $maxFr_i$, $maxDels_i$, $maxIns_i$, $maxLen + maxDels_i$ )
    Prune the VST branches that contain only the nodes labelled with $\ominus$
  end for
```

`Marguerite-SoftFr` The algorithm 10 `Marguerite-SoftFr` starts by calling the `InitTreeForSoft` sub-algorithm to construct an initial VST. `InitTreeForSoft` handles the *InsDels* soft-frequency during the scan of the $\mathcal{D}_1$. Note that, `InitTreeForSoft` algorithm, differently from its counterpart Algorithm 2, given in Page 54, do not prune by the $\mathsf{MaxLength}_\phi(maxLen)$ constraint during the phase of VST[13] grow-

[11] I.e., with the parameters $maxDels \geq maxIns$ (see Theorem 5.4 in Page 106)

[12] For the etymology of the solver name, see Footnote 2 in Page 94.

[13] Version Space Tree (VST) is a data structure used by `FAVST` and `Marguerite`. For more details on VST, see Section 3.2.1 in Page 46.

ing. Indeed, in order to evaluate the soft-frequency of $\mathsf{string}(\mathcal{N})$, one needs to keep in a VST all the strings that are putative soft-occurrences of $\mathsf{string}(\mathcal{N})$. The soft-occurrence can be of length at most $maxLen$ (maximum pattern's length) + $maxDels$, and therefore VST stores the strings till that length. The $\mathsf{MaxLength}_\phi(maxLen)$ constraint, together with others, is verified in sub-Algorithm 19 `UnlabelVSTForSoftFr` (Page 115). Then, to reduce the VST, `Marguerite-SoftFr` prunes the unnecessary branches, i.e., the branches that contain only the nodes labelled with $\ominus$.

After, the `CountAndUnmarkForSoft` algorithm is invoked to process the soft-frequency constraints on the remaining string data sets. Note that, differently from its counterpart Algorithm 6, given in Page 56, `CountAndUnmarkForSoft` do grow the VST. The patterns that satisfy the constraints in the previous data sets can have new soft-occurrences in the current data set $\mathcal{D}_i$, which must be stored in the VST. Again, the branches containing only the nodes labelled with $\ominus$ are pruned after scanning each $\mathcal{D}_i$, in order to reduce the number of patterns that need to be processed in the subsequent iteration.

As in `FAVST`, each algorithm of `Marguerite-SoftFr`, `InitTreeForSoftFr` and `CountAndUnmarkForSoftFr`, scans the corresponding string data set $\mathcal{D}_i$ only once. Thus, if the data sets $\mathcal{D}_i$ are disjoint, the algorithm `Marguerite` scans the whole data set $\mathcal{D} = \bigcup_{i=1}^{n} \mathcal{D}_i$ only once, and is a *single-scan* algorithm.

`InitTreeForSoftFr` The algorithm `InitTreeForSoftFr`, given in Algorithm 11, builds a VST data structure so that one can compute $InsDels$ soft-frequency for the string patterns it contains. This algorithm is indeed very similar to its counterpart Algorithm 2 `InitTree`, given in Page 54. The only differences between `InitTreeForSoftFr` and `InitTree`, are that it calls the adapted `CreateChildForSoftFr`, `IncreaseFrequencyForSoft` and `UnlabelVSTForSoftFr` algorithms, and that the new Algorithm 14 `ComputeSoftFr` is invoked to compute the soft-frequency.

`CreateChildForSoftFr` The algorithm `CreateChildForSoftFr`, given in Algorithm 12, like its counterpart Algorithm 3 `CreateChild` (see Page 54), creates and initializes a new node. To evaluate the soft-frequencies, we make the following change to the VST structure: each node $\mathcal{N}$ is complemented with a soft-frequency of $\mathsf{string}(\mathcal{N})$, a set of identifiers of data strings, on which $\mathsf{string}(\mathcal{N})$ occur, and a soft-frequency label. The only difference between `CreateChildForSoftFr` and `CreateChild` algorithm is that it initializes these new fields of a VST node.

`IncreaseFrequencyForSoftFr` The algorithm `IncreaseFrequencyForSoftFr`, given in Algorithm 13, as its counterpart Algorithm 4 (see in Page 55) `IncreaseFrequency`, increments the frequency count for the $\mathsf{string}(c)$, represented by a child node c, and for all its suffixes (i.e., all its substrings). Thanks to this, as in the `FAVST` algorithm,

Algorithm 11 `InitTreeForSoftFr`

```
Require: Data set $\mathcal{D}_1$, alphabet $\Sigma_\phi$, thresholds $minFr_1$, $maxFr_1$, $maxIns_1$, $maxDels_1$,
  $maxLen$
Ensure: VST containing the solution patterns (labelled with $\oplus$) to the constraint
  $\mathsf{MinSoftFr}_\phi^{InsDels,S}(minFr, \mathcal{D}, maxIns_1, maxDels_1) \wedge$
  $\mathsf{MaxSoftFr}_\phi^{InsDels,S}(maxFr, \mathcal{D}, maxIns_1, maxDels_1) \wedge \mathsf{MaxLength}_\phi(maxLen)$
  Create VST with the root node $\mathcal{R}$
  $\mathcal{R}$.suffix $\leftarrow \top$, $\mathcal{R}$.label $\leftarrow \oplus$, $\mathcal{R}$.count $\leftarrow 0$, $\mathcal{R}$.last-id $\leftarrow$ undefined
  for all string $S \in \mathcal{D}_1$ with unique id ID do
    $\mathcal{N} \leftarrow \mathcal{R}$
    for all symbol $\xi_i \in S$ do
      if $\xi_i \in \Sigma_\phi$ then {$\xi_i$ is an interesting symbol}
        if depth of $\mathcal{N} \geq maxLen$ then {check for the $\mathsf{MaxLength}_\phi(maxLen)$ }
          $\mathcal{N} \leftarrow \mathcal{N}$.suffix
        end if
        if $\mathcal{N}$ has a child with edge $\xi_i$ then
          $c \leftarrow \mathcal{N}$.child with edge $\xi_i$
        else
          $c \leftarrow$ CreateChildForSoftFr($\mathcal{N}$,$\xi_i$)
          $c$.edge $\leftarrow \xi_i$ {set edge of a new child node}
          Add $c$ to $\mathcal{N}$ children
        end if
        IncreaseFrequencyForSoftFr($c$) {increase frequency}
        $\mathcal{N} \leftarrow c$ {current node gets one level deeper}
      else {$\xi_i$ is an uninteresting symbol}
        $\mathcal{N} \leftarrow \mathcal{R}$ {break input string, continue from the root}
      end if
    end for
  end for
  ComputeSoftFr($\mathcal{R}$, $\mathcal{R}$, $maxFr_1$, $maxIns_1$, $maxDels_1$ )
  UnlabelVSTForSoftFr($\mathcal{R}$, $minFr_1$, $maxFr_1$, $maxLen$ )
```

Algorithm 12 `CreateChildForSoftFr`

```
Require: Parent node $\mathcal{N}$, edge symbol $\xi_i$ for the child node
Ensure: The newly created node $c$ with the symbol $\xi_i$ on its edge
  Create node $c$
  $c$.label ← $\oplus$, $c$.count ← 0, $c$.edge ← undefined, $c$.last-id ← undefined
  $c$.softFr ← 0, $c$.occs ← $\emptyset$, $c$.softFrLabel ← $\oplus$
  $sn$ ← $\mathcal{N}$.suffix
  if $\mathcal{N}$ is a root node $\mathcal{R}$ then
    $c$.suffix ← $\mathcal{N}$
  else
    if $sn$ has a child $sc$ with edge $\xi_i$ then
      $c$.suffix ← $sc$
    else {recursively create the necessary suffix nodes}
      $sc$ ← CreateChild($sn, \xi_i$)
      $sc$.edge ← $\xi_i$ {set edge of a new child node}
      Add $sc$ to $sn$ children
      $c$.suffix ← $sc$
    end if
  end if
```

the exact-frequency is counted for every string($\mathcal{N}$) in a VST. However, the goal of `Marguerite-SoftFr` is to compute the *InsDels* soft-frequency of string($\mathcal{N}$) and thus to evaluate the *InsDels* soft-frequency constraint. We remind that a soft-frequency of a pattern is the number of data strings, where a pattern and its soft-occurrences are present. In other words, it is the cardinality of the set, containing the data strings (or rather their unique identifiers), where a pattern and its soft-occurrences are present. Such set, named "occs“, is associated with each $\mathcal{N}$. The algorithm `IncreaseFrequencyForSoftFr` fills-in that set with the unique identifiers of data strings, where the corresponding pattern string($\mathcal{N}$) occurs exactly. Afterwards, we will see that Algorithms {16,18} `FindSoftOccs-Push`{`M,A`} continues to fill-in that set with the unique identifiers of data strings, where the pattern string($\mathcal{N}$) occurs softly. Then, the soft-frequency of a pattern string($\mathcal{N}$) is simply the cardinality of its associated "occs“ set.

Algorithm 13 `IncreaseFrequencyForSoftFr`

```
Require: Node $c$, ID
Ensure: Frequency for string(c) and all its substrings is increased by 1
  $x$ ← $c$
  while ($x$ is not a root node $\mathcal{R}$) $\wedge$ ($x$.last-id $\neq$ ID) do
    {if it is not a root suffix and its frequency not increased yet}
    $x$.count ← $x$.count + 1
    Add ID to $x$.occs {remember the string of occurrence}
    $x$.last-id ← ID
    $x$ ← $x$.suffix {also count all $c$ suffixes}
  end while
```

`ComputeSoftFr` The algorithm `ComputeSoftFr`, given in Algorithm 14, goes down the VST by visiting each node $\mathcal{N}$. For each node $\mathcal{N}$ it invokes Algorithm {16,18} `FindSoftOccs-Push{M,A}`. Both algorithms perform the same task: they fill-in the $\mathcal{N}$.occs set with the identifiers of the data strings where the soft-occurrences of string($\mathcal{N}$) occur. They differ in whether the monotonic or anti-monotonic similarity sub-constraint is exploited to prune the search space. Depending on the context, the first strategy or the second one is more efficient. The automatic and dynamic choice of a better strategy is an open question from the adaptive constraint pushing strategies domain. Once the soft-occurrences of a node $\mathcal{N}$ are found, and the set "occs" is completed, the soft-frequency of string($\mathcal{N}$) is given by the cardinality of that $\mathcal{N}$.occs. The Algorithm 15 `ResetVSTForSoftFr` clears the flag "softFrLabel" used when searching for the soft-occurrences of the current node $\mathcal{N}$, and thus prepares the VST for the search for the soft-occurrences of the subsequent node. The phase of finding the soft-occurrences, necessary to compute the soft-frequency, is computationally expensive. Algorithm `ComputeSoftFr` exploits the anti-monotonicity property of the *InsDels* minimum soft-frequency constraint and stops descending in a VST as soon it encounters a node $\mathcal{N}$, such that the pattern string($\mathcal{N}$) does not satisfy the given MinSoftFr constraint - indeed any super-string of string($\mathcal{N}$) will not satisfy it neither.

Algorithm 14 `ComputeSoftFr`

```
Require: $\mathcal{N}$, VST root node $\mathcal{R}$, thresholds $minFr$, $maxIns$, $maxDels$
Ensure: For each $\mathcal{N}$ that can satisfy MinSoftFr, its soft-frequency is computed
  {if $\mathcal{N}$ is not a root and if string($\mathcal{N}$) satisfy constraints in the previous data sets}
  if ($\mathcal{N}$ is not a root node $\mathcal{R}$) $\wedge$ ($\mathcal{N}$.label $= \oplus$) then
    {we will search soft-occurrences for the current pattern string($\mathcal{N}$) }
    FindSoftOccs-Push{M,A}($\mathcal{R}$, string($\mathcal{N}$), $\mathcal{N}$, $maxDels$, $maxIns$ )
    $\mathcal{N}$.softFr $\leftarrow$ cardinality($\mathcal{N}$.occs)
    maxLengthOfSoftOcc $\leftarrow$ |string($\mathcal{N}$)| $+ maxDels$
    ResetVSTForSoftFr($\mathcal{R}$, maxLengthOfSoftOcc)
  end if
  {if string($\mathcal{N}$) satisfies MinSoftFr or if $\mathcal{N}$ is a root}
  if ($\mathcal{N}$.softFr) $> minFr \vee$ ($\mathcal{N}$ is a root node $\mathcal{R}$) then
    for all child $c$ of $\mathcal{N}$ do
      ComputeSoftFr($c, minFr, maxIns, maxDels$)
    end for
  end if
```

`FindSoftOccs-Push{M,A}` Both, Algorithm 16 `FindSoftOccs-PushM` and Algorithm 18 `FindSoftOccs-PushA`, traverse the VST to find the soft-occurrences of a pattern ϕ in question. We remind that a string in a VST string($\mathcal{N}$) is a soft-occurrence of ϕ if and only if the conjunction of the monotonic MaxIns and the anti-monotonic MaxDels (see Definition 5.6 in Page 104) constraints are evaluated to true. The two algorithms differ in whether they actively exploit the monotonic constraints (thus MaxIns) or the anti-monotonic constraints (thus MaxDels).

```
Algorithm 15 ResetVSTForSoftFr
Require: $\mathcal{N}$, $depth$
Ensure: VST is reset and thus prepared to find soft-occurrences of the subsequent string($\mathcal{N}$)
  if depth of $\mathcal{N} > depth$ then
    return {used part of VST is already reset}
  end if
  $\mathcal{N}$.softFrLabel $\leftarrow \oplus$ {reset this VST $\mathcal{N}$ }
  for all child $c$ of $\mathcal{N}$ do
    ResetVSTForSoftFr($c$, $depth$)
  end for
```

By ascending the VST, `FindSoftOccs-PushM` pushes the monotonic MinLength constraint[14], and then the monotonic MaxIns constraint. The reason of verifying first the MinLength constraint is that it allows to reject the patterns that can not satisfy the MaxIns constraint[15]. Note that the evaluation of MinLength is straightforward, while the evaluation of MaxIns is expensive. If the pattern string($\mathcal{N}$) does not satisfy the MinLength or MaxIns constraint, the algorithm `FindSoftOccs-PushM` stops ascending the VST, invokes Algorithm 17 `UnsetAncestorSoftFrLabel`, and thereby prunes the search space. `UnsetAncestorSoftFrLabel` marks with labels $\ominus$ the $\mathcal{N}$ ancestor nodes, which no longer need to be considered due to the MinLength and MaxIns motonicity property. Remark, that the algorithm `FindSoftOccs-PushM` also exploit the anti-monotonic MaxLength constraint, what assures that one does not descends in a VST too deep, where the soft-occurrences no longer can be found.

By ascending the VST, `FindSoftOccs-PushA` pushes the anti-mononotonic MaxLength constraint, and then the anti-monotonic MaxDels constraint. Similarly to the case of `FindSoftOccs-PushM`, it is clever to evaluate first the MaxLength constraint, since the evaluation of the MaxDels constraint is expensive. The algorithm `FindSoftOccs-PushA` stops descending in a VST, and thereby prune the search space, when it encounters a node $\mathcal{N}$, such that the corresponding pattern string($\mathcal{N}$), does not satisfy one of the anti-monotonic MaxLength or MaxDels constraints.

`UnlabelVSTForSoftFr` Algorithm 19 `UnlabelVSTForSoftFr` traverses the VST to mark the nodes $\mathcal{N}$, such that their string($\mathcal{N}$) do not satisfy the soft-frequency or maximum length constraints, with labels $\ominus$. Also the VST is prepared for the subsequent calls for Algorithm 20 `CountAndUnmarkForSoftFr` by resetting the data set specific information stored in the nodes.

[14]Which is an approximation of the MaxIns constraint.

[15]If the constraint $\mathsf{MinLength}_\phi(|\sigma| - maxIns)$ is not satisfied, the constraint $\mathsf{MaxIns}_\phi(\sigma, maxIns)$ can not be satisfied neither.

Algorithm 16 `FindSoftOccs-PushM`

```
Require: $\mathcal{N}$, ref. string $\sigma$, ref. node $r\mathcal{N}$, $maxDels$, $maxIns$
Ensure: $\mathcal{N}$.occs contain all data string IDs, where string($\mathcal{N}$) has soft-occurrences
  maxLengthOfSoftOcc $\leftarrow |\sigma| + maxDels$ {maximum length of a putative soft-occurrence}
  if depth of $\mathcal{N}$ > maxLengthOfSoftOcc then
    return {string($\mathcal{N}$) can not be soft-occurrence of $\sigma$ }
  end if
  if $\mathcal{N}$ has children then
    for all child $c$ of $\mathcal{N}$ do
      FindSoftOccs-PushM($c, \sigma$,r$\mathcal{N}$ , $maxDels, maxIns$)
    end for
  end if
  {if $\mathcal{N}$ is not a root and if monotonic MinLength or MaxIns is not already known to be
  unsatisfied, then treat this $\mathcal{N}$ }
  if ($\mathcal{N}$ is not a root node $\mathcal{N}$) $\wedge$ ($\mathcal{N}$.softFrLabel $\neq \ominus$) then
    if depth of $\mathcal{N} < (|\sigma| - maxIns)$ then {string($\mathcal{N}$) can not be soft-occurrence of $\sigma$ }
      UnsetAncestorSoftFrLabel($\mathcal{N}$ ) {because of MinLength monotonicity}
    else
      $lcs \leftarrow$ computeLCS(string($\mathcal{N}$), $\sigma$)
      $dels \leftarrow |$string($\mathcal{N}$)$| - lcs$
      $ins \leftarrow |\sigma| - lcs$
      if $(dels \leq maxDels) \wedge (ins \leq maxIns)$ then {string($\mathcal{N}$ is soft-occurrence of $\sigma$) }
        if $(dels \neq 0) \wedge$ (ins $\neq 0$) then {if it is not an exact occurrence}
          $r\mathcal{N}$.occs $\leftarrow r\mathcal{N}$.occs $\cup\, \mathcal{N}$.occs {union of occurrences sets}
        end if
      else if $ins > maxIns$ then
        UnsetAncestorSoftFrLabel($\mathcal{N}$ ) {because of MaxIns monotonicity}
      end if
    end if
  end if
```

Algorithm 17 `UnsetAncestorSoftFrLabel`

```
Require: $\mathcal{N}$
Ensure: softFrLabel is unset for $\mathcal{N}$ and all its ancestor nodes
  $\mathcal{N}$.softFrLabel $\leftarrow \ominus$ {unset softFrLabel}
  {if $\mathcal{N}$ is not $\mathcal{R}$, and if its parent softFrLabel is not already unset}
  if ($\mathcal{N}$ is not a root node $\mathcal{R}$) $\wedge$ (softFrLabel of $\mathcal{N}$parent is not $\ominus$) then
    UnsetAncestorSoftFrLabel(parent of $\mathcal{N}$ )
  end if
```

Algorithm 18 `FindSoftOccs-PushA`

```
Require: $\mathcal{N}$, ref. string $\sigma$, ref. node $r\mathcal{N}$, $maxDels$, $maxIns$
Ensure: $\mathcal{N}$.occs contain all data string IDs, where string($\mathcal{N}$) has soft-occurrences
  maxLengthOfSoftOcc $\leftarrow |\sigma| + maxDels$ {max. length of a putative soft-occurrence}
  if depth of $\mathcal{N}$ > maxLengthOfSoftOcc then
    return {string($\mathcal{N}$) can not be soft-occurrence of $\sigma$ }
  end if
  $lcs \leftarrow$ computeLCS(string($\mathcal{N}$), $\sigma$)
  $dels \leftarrow |\text{string}(\mathcal{N})| - lcs$
  $ins \leftarrow |\sigma| - lcs$
  if $(dels \leq maxDels) \wedge (ins \leq maxIns)$ then {string($\mathcal{N}$ is soft-occurrence of $\sigma$) }
    if $(dels \neq 0) \wedge (\texttt{ins} \neq 0)$ then {it is not an exact occurrence}
      $r\mathcal{N}$.occs $\leftarrow r\mathcal{N}$.occs $\cup\ \mathcal{N}$.occs {union of occurrences sets}
    end if
  end if
  if $dels \leq maxDels$ then {$\mathcal{N}$ satisfied anti-monotonic MaxDels}
    for all child $c$ of $\mathcal{N}$ do {descend in VST}
      FindSoftOccs-PushA($c$, $\sigma$, $r\mathcal{N}$, $maxDels$, $maxIns$ )
    end for
  end if
```

Algorithm 19 `UnlabelVSTForSoftFr`

```
Require: $\mathcal{R}$, $minFr$, $maxFr$, $maxLen$
Ensure: Nodes $\mathcal{N}$, such that string($\mathcal{N}$) does not satisfy
  $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$
  $\wedge \mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels) \wedge \mathsf{MaxLength}_{\phi}(maxLen)$, are labelled
  with $\ominus$
  for all $\mathcal{N}$ in VST do {traverse the VST and unlabel}
    if ($\mathcal{N}$.softFr $< minFr$) $\vee$ ($\mathcal{N}$.softFr $> maxFr$) $\vee$ (depth of $\mathcal{N} > maxLen$) then
      $\mathcal{N}$.label $\leftarrow \ominus$
    end if
    {reset VST for the subsequent CountAndUnmarkForSoftFr traversal}
    $\mathcal{N}$.count $\leftarrow 0$
    $\mathcal{N}$.occs $\leftarrow \emptyset$
    $\mathcal{N}$.softFr $\leftarrow 0$
    $\mathcal{N}$.last-id $\leftarrow$ undefined
  end for
```

`CountAndUnmarkForSoftFr` Algorithm 20 `CountAndUnmarkForSoftFr` as its counterpart Algorithm 6 `CountAndUnmark`, given in Page 56, takes as input the VST, constructed beforehand, and for the patterns $\mathsf{string}(\mathcal{N})$ that satisfy the constraints in the previous data set $\mathcal{D}_{i-1}$, evaluates the given constraints in the current data set $\mathcal{D}_i$. Differently from `CountAndUnmark`, the algorithm `CountAndUnmarkForSoftFr` does increase the VST by adding the substrings that are putative soft-occurrences in $\mathcal{D}_i$ of the patterns that satisfy the constraints on the data sets processed previously. Indeed, the algorithm `CountAndUnmarkForSoftFr` is very similar to Algorithm 11 `InitTreeForSoftFr`. The only difference is that in `CountAndUnmarkForSoftFr` the VST root node $\mathcal{R}$ is not created to build a new VST, but instead is given by a parameter, and thereby the existing VST is grown-up, if needed.

Algorithm 20 `CountAndUnmarkForSoftFr`

```
Require: $\mathcal{R}$, data set $\mathcal{D}_i$, alphabet $\Sigma_\phi$, thresholds $minFr_i$, $maxFr_i$, $maxIns_i$, $maxDels_i$,
    $maxLen$
Ensure: VST, in which the patterns $\phi$ that are not solutions to the constraint
    $\mathsf{MinSoftFr}_\phi^{InsDels,S}(minFr_i, \mathcal{D}_i, maxIns, maxDels)$
    $\wedge \mathsf{MaxSoftFr}_\phi^{InsDels,S}(maxFr_i, \mathcal{D}_i, maxIns, maxDels)$
    for all string $S \in \mathcal{D}_1$ with unique id ID do
      $\mathcal{N} \leftarrow \mathcal{R}$
      for all symbol $\xi_i \in S$ do
        if $\xi_i \in \Sigma_\phi$ then {$\xi_i$ is an interesting symbol}
          if depth of $\mathcal{N} \geq maxLen$ then {check for the $\mathsf{MaxLength}_\phi(maxLen)$ }
            $\mathcal{N} \leftarrow \mathcal{N}$.suffix
          end if
          if $\mathcal{N}$ has a child with edge $\xi_i$ then
            $c \leftarrow \mathcal{N}$.child with edge $\xi_i$
          else
            $c \leftarrow$ CreateChildForSoftFr($\mathcal{N}$,$\xi_i$)
            $c$.edge $\leftarrow \xi_i$ {set edge of a new child node}
            Add $c$ to $\mathcal{N}$ children
          end if
          IncreaseFrequencyForSoftFr($c$) {increase frequency}
          $\mathcal{N} \leftarrow c$ {current node gets one level deeper}
        else {$\xi_i$ is an uninteresting symbol}
          $\mathcal{N} \leftarrow \mathcal{R}$ {break input string, continue from the root}
        end if
      end for
    end for
    ComputeSoftFr($\mathcal{R}$, $\mathcal{R}$, $maxFr_i$, $maxIns_i$, $maxDels_i$ )
    UnlabelVSTForSoftFr($\mathcal{R}$, $minFr_i$, $maxFr_i$, $maxLen$ )
```

5.3.4 Case of a Differential Extraction

The evaluation of the inductive query that is a conjunction of a minimum (soft-)frequency constraint on one data set and a maximum (soft-)frequency on the other data set is known as differential extraction in a positive and negative data sets. Such query, expressed by the following composition of constraints:

$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr_1, \mathcal{D}_1, maxIns_1, maxDels_1) \wedge$
$\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(\infty, \mathcal{D}_1, maxIns_1, maxDels_1) \wedge$
$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(0, \mathcal{D}_2, maxIns_2, maxDels_2) \wedge$
$\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr_2, \mathcal{D}_2, maxIns_2, maxDels_2) \wedge$
$\mathsf{MaxLength}_{\phi}(maxLen)$

can be solved `Marguerite-SOftFr` generic solver (Algorithm 10, Page 108). `Marguerite-SoftFr`, being a generic solver, can not provide the constraint combination specific (i.e., ad-hoc) optimisations. Given that soft-frequency evaluation is computationally expensive, it is practical to provide such optimisations for frequently used constraint combinations.

In this section we will consider the case of differential extraction. Take notice, that the minimum soft-frequency constraint is not pushed in the negative (second) data set $\mathcal{D}_2$. In `Marguerite-SoftFr`, the constraints on $\mathcal{D}_2$ are handled by Algorithm 20 `CountAndUnmarkForSoftFr`. The soft-frequency is counted by invoking Algorithm 14 `ComputeSoftFr`. We remind that the algorithm `ComputeSoftFr` reduces the number of nodes, for which soft-frequency is evaluated, by exploiting the anti-monotonic minimum soft-frequency constraint. However, in the differential extraction, there is no such constraint on the negative data set $\mathcal{D}_2$, and thus, there is no pruning at all. This results in a serious decrease of the extraction efficiency.

The situation can be considerably alleviated by exploiting the monotonic maximum soft-frequency constraint. To compute the soft-frequency in a negative data set $\mathcal{D}_2$, we propose to ascend the VST by starting from the nodes that constitute the S set (see Definition 2.18 in Page 42 and discussion in Section 3.2.1 in Page 48) of the solution to the minimum soft-frequency constraint in the positive data set $\mathcal{D}_1$. Then, due to the maximum-soft frequency monotonicity property, one stops ascending the VST and evaluating the soft-frequency when the soft-frequency of the current $\mathsf{string}(\mathcal{N})$ exceeds the $maxFr_2$ threshold. The corresponding algorithms `ComputeSoftFrFromBorder` and `ComputeSoftFr-PushM` are given in Algorithm 21 and Algorithm 22. Thus, to perform an optimized differential extraction with `Marguerite-SoftFr`, the algorithm `ComputeSoftFrFromBorder` have to be invoked in Algorithm 20 `CountAndUnmarkForSoftFr`, instead of Algorithm 14 `ComputeSoftFr`.

Algorithm 21 `ComputeSoftFrFromBorder`

```
Require: VST root node $\mathcal{R}$, thresholds $maxFr$, $maxDels$, $maxIns$
Ensure: For each $\mathcal{N}$ in VST, that can satisfy MaxSoftFr, its soft-frequency is computed
  border ← S border of the solution, stored in VST
  for all $\mathcal{N} \in$ border do
    ComputeSoftFr-PushM($\mathcal{N}$, $\mathcal{R}$, parMaxFr, $maxDels$, $maxIns$ )
  end for
```

Algorithm 22 `ComputeSoftFr-PushM`

```
Require: $\mathcal{N}$, VST root node $\mathcal{R}$, thresholds $maxFr$, $maxDels$, $maxIns$
Ensure: For each $\mathcal{N}$, and its ancestors, that can satisfy MaxSoftFr, their soft-frequency is computed
  if $\mathcal{N}$.softFr = 0 then {$\mathcal{N}$ soft-frequency is not already computed}
    FindSoftOccs-Push{M,A}($\mathcal{R}$, string($\mathcal{N}$), $\mathcal{N}$, $maxDels$, $maxIns$ )
    $\mathcal{N}$.softFr ← cardinality($\mathcal{N}$.occs)
    maxLengthOfSoftOcc ← |string($\mathcal{N}$)| + $maxDels$
    ResetVSTForSoftFr($\mathcal{R}$, maxLengthOfSoftOcc)
    if $\mathcal{N}$.softFr $\leq maxFr$ then {satisfy MaxSoftFr}
      if $\mathcal{N}$. is not a root node $\mathcal{R}$ then {if $\mathcal{N}$ is not already a root}
        ComputeSoftFr-PushM(parent of $\mathcal{N}$, $\mathcal{R}$, $maxFr$, $maxDels$, $maxIns$ )
      end if
    end if
  end if
```

5.3.5 Pattern with Hamming Match Function Extraction

Note that the minimum Hamming soft-frequency constraint $\mathsf{MinSoftFr}_{\phi}^{H,\{s,S\}}(minFr, \mathcal{D}, k)$ (see Definition 4.21 in Page 69) is anti-monotonic, since, given a string σ that is a Hamming soft-occurrence of a string pattern ϕ, i.e., such that the Hamming distance similarity relation $sim_{Hdist}(\phi, \sigma, k) = \mathsf{true}$, for every pattern $\phi' \sqsubseteq \phi$, there exists a string $\sigma' \sqsubseteq \sigma$, such that it is a Hamming soft-occurrence of the pattern ϕ', i.e., $sim_{Hdist}(\phi', \sigma', k) = \mathsf{true}$. Dually, the maximum Hamming soft-frequency constraint $\mathsf{MaxSoftFr}_{\phi}^{H,\{s,S\}}(maxFr, \mathcal{D}, k)$ is monotonic.

We developed an instance of `Marguerite-SoftFr` algorithmic framework, referred as `Marguerite-H`, to solve the Hamming soft-frequency constraints. To find the soft-occurrences of the pattern ϕ we have to evaluate the Hamming distance similarity constraint, where the string of reference is ϕ. Take notice that the Hamming distance similarity constraint is neither anti-monotonic, nor monotonic, but the search space is rather restricted (only the strings σ of the same length as ϕ are its putative soft-occurrences). Therefore the pattern's ϕ soft-occurrences, necessary to evaluate its Hamming soft-frequency, were found by verifying it for every string of length $|\phi|$ in the VST, constructed with Algorithm 11 `InitTreeForSoftFr`[16].

[16]I.e., using a constraint post-processing approach

5.3.6 Experimental Validation

To provide an experimental validation of the soft-frequency constraints that employ the defined *InsDels match* function $match_{InsDels}(\phi, S, maxIns, maxDels)$ and thus benefit from the (anti-)monotonicity properties, we performed the extractions under exact-frequency and *InsDels* soft-frequency constraints. For the extractions under the exact-frequency constraints we the solver `FAVST` (see Section 3.2.2.2 in Page 51) and for the extractions under the *InsDels* soft-frequency constraints we used the solver `Marguerite-SoftFr` (see Section 5.3.3 in Page 108). For the differential extractions we used an optimized ad-hoc instance of `Marguerite-SoftFr` (see Section 5.3.4 in Page 117). In all extractions under the *InsDels* soft-frequency constraints, to find the soft-occurrences we used Algorithm 16 `FindSoftOccs-PushM`, since its constraint pushing strategy was revealed to be more efficient for the used alphabet size and pattern length. All used solvers are implemented in `C` and `C++` programming languages. We processed `KDD Cup 2000` real-world clickstream data sets [KBF+00] on a Intel(R) Pentium(R) M 1.69GHz processor and 1GB main memory.

To produce the time ordered sequences of the templates requested for each session, we extracted the attributes "Session ID", "Request Sequence", "Request Template". There are 137 different request templates, i.e., the analyzed sequences of templates are the strings over an alphabet of 137 symbols. The produced string data set, denoted $\mathcal{D}_{\texttt{S}}$, contains $234,954$ strings. The shortest string is of length 1, and the largest one is of length $5,487$. We also extracted the attributes "Session First Request Day Of Week" and "Session First Request Hour Of Day". According to these attributes the string data set $\mathcal{D}_{\texttt{S}}$ was split into four string data sets:

- $\mathcal{D}_{\texttt{SWE}}$ for sessions requested on Saturday or Sunday ($47,229$ strings),
- $\mathcal{D}_{\texttt{SWD}}$ for sessions requested on workdays ($187,725$ strings),
- $\mathcal{D}_{\texttt{SD}}$ for sessions requested from 8 am till 7 pm ($137,593$ strings),
- $\mathcal{D}_{\texttt{SN}}$ for the sessions requested from 7 pm to 8 am ($97,361$ strings).

5.3.6.1 Impact of the Similarity Parameters in *InsDels* Match Function

The frequency of a pattern ϕ is the number of objects that pattern ϕ matches. Thus the employed *match* function and the chosen parameters for this function are crucial to the resulting frequency values. The *InsDels* soft-frequency constraints use the *InsDels match* function $match_{InsDels}(\phi, S, maxIns, maxDels)$, which has two parameters : *maxIns*, limiting the allowed number of insertions, and *maxDels*, limiting the allowed number of deletions, necessary to produce ϕ from its soft-occurrence σ (see Remark 5.2 in Page 104).

Table 5.4: Frequency w.r.t. soft-frequency and the impact of the different *InsDels match* function parameters

	$\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_{\mathtt{S}})$	$\frac{\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_{\mathtt{S}})\times 100\%}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,1)}$	$\frac{\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_{\mathtt{S}})\times 100\%}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,2)}$	$\frac{\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_{\mathtt{S}})\times 100\%}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},2,1)}$	$\frac{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,1)\times 100\%}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,2)}$	$\frac{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,1)\times 100\%}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},2,1)}$
Mean val	57.89	14.61	12.37	6.9	0.76	0.37
Stand Dev	70.63	21.26	20.6	14.72	0.09	0.18
Min val	23	1.53	1.14	0.54	0.45	0.06
Max val	843	100	100	97.6	1	0.99

To assess the impact of the fault-tolerant matching in general, and the impact of different similarity parameters in the fault-tolerant match function, we computed the exact-frequency $\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_{\mathtt{S}})$ and the *InsDels* soft-frequencies $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,1)$, $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,2)$ and $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},2,1)$ for the 796 patterns ϕ that are the solutions to the inductive query

$$IQ_1 = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(0.01\%,\mathcal{D}_{\mathtt{S}}) \wedge \mathsf{MinLength}_{\phi}(7) \wedge \mathsf{MaxLength}_{\phi}(7),$$

i.e, that occur in at least 0.01% of $\mathcal{D}_{\mathtt{S}}$ strings and are of length 7. We took the patterns of the same length so that the soft-frequency would not be influenced by the variable pattern length, but only by *maxIns* and *maxDels* values. A summary of the results is provided in Table 5.4.

Observe that, in most cases, the exact-frequency of a pattern is quite small w.r.t. its soft-frequency. Also, $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,1)$ tends to be smaller than $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,2)$ and $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},2,1)$. This is explained by the less restrictive fault-tolerant matching parameters used by the *InsDels match* function in the two latter cases. Finally, $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},2,1)$ tends to be greater than $\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,2)$. This is explained by the fact that, when more `ins` symbols are allowed (see Definition 5.5 in Page 104), the shorter strings σ are acknowledged as soft-occurrences of a pattern ϕ, and there are more shorter strings than longer strings.

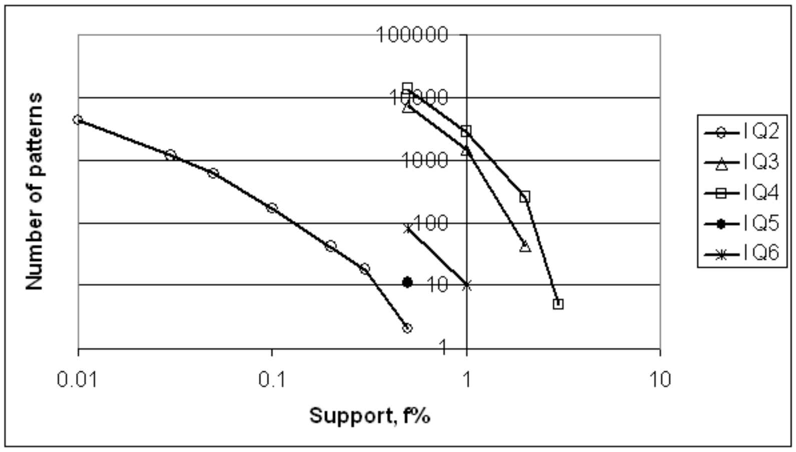

Figure 5.1: Selectivity of the minimum exact-frequency and minimum *InsDels* soft-frequency constraints

5.3.6.2 Selectivity of the Minimum *InsDels* Soft-Frequency Constraint

Minimum soft-frequency constraints, employing the match functions that allow mismatches between a pattern ϕ and its soft-occurrence σ, decrease the selectivity of a classical minimum frequency constraint, which employs the exact match function. To evaluate the selectivity decrease, provided by the *InsDels* minimum soft-frequency constraint, we computed the solutions of the following inductive queries:

- $IQ_2 = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, \mathcal{D}_{\mathsf{S}}) \wedge \mathsf{MinLength}_{\phi}(5) \wedge \mathsf{MaxLength}_{\phi}(10)$
- $IQ_3 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, 1, 1) \wedge \mathsf{MinLength}_{\phi}(5) \wedge \mathsf{MaxLength}_{\phi}(10)$
- $IQ_4 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, 1, 2) \wedge \mathsf{MinLength}_{\phi}(5) \wedge \mathsf{MaxLength}_{\phi}(10)$
- $IQ_5 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, 1, 0) \wedge \mathsf{MinLength}_{\phi}(5) \wedge \mathsf{MaxLength}_{\phi}(10)$
- $IQ_6 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, 1, 2) \wedge \mathsf{MinLength}_{\phi}(4) \wedge \mathsf{MaxLength}_{\phi}(10)$

The plot of the size of the solutions of these inductive queries against different minimum frequency thresholds $minFr$ is given in Figure 5.1

For the minimum exact-frequency constraint $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, \mathcal{D}_{\mathsf{S}})$, which employs the exact *match* function, we started at $minFr$ value equal to 0.01%. This is a pretty small value, and in most applications contexts it is fair to consider that patterns, which do not satisfy this constraint, are not interesting. For the minimum

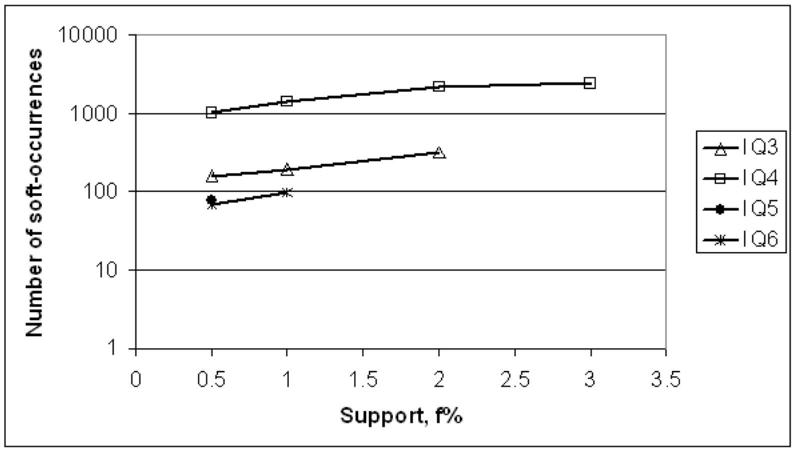

Figure 5.2: Number of soft-occurrences

$InsDels$ soft-frequency constraint $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, maxIns, maxDels)$, we started at $minFr$ value equal to 0.5%, because of the consumed time restrictions (see Section 5.3.6.4). For both constraints we increased the $minFr$ value, until the corresponding solution set became empty. We take notice that in all cases there are 767, 238 patterns ϕ of length $4 \leq |\phi| \leq 10$ and 727, 873 patterns of length $5 \leq |\phi| \leq 10$.

Observe that $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, \mathcal{D}_{\mathsf{S}})$ constraint with even very small frequency threshold $minFr$ values drastically prunes, while the same support values for $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, maxIns, maxDels)$ are not selective at all. This emphasizes the added value of the use of a fault-tolerant match function in frequency counting: one can assume that at least 1% of the sessions share some common features in their requested templates, and the $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(1\%, \mathcal{D}_{\mathsf{S}}, maxIns, maxDels)$ enables to extract these regularities, while $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(1\%, \mathcal{D}_{\mathsf{S}})$ leads to the empty collection.

Figure 5.2 plots the mean values of the number of soft-occurrences for patterns that are solutions to IQ_3, IQ_4, IQ_5 and IQ_6. It reveals that, in general, the greater pattern's soft-frequency, the more soft-occurrences, i.e., similar patterns, it has. This is however not a theorem[17].

[17]To evaluate the $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}_{\mathsf{S}}, maxIns, maxDels)$ constraint we count the string objects S, for which the match function $match_{InsDels}(\phi, S, maxIns, maxDels)$ evaluates `true`. One string object may contain many pattern's ϕ soft-occurrences σ, but this does not influence the frequency value.

Table 5.5: Identification of templates

ID	Template
A	`main/home\.jhtm`
B	`main/departments\.jhtml`
C	`main/search_results\.jhtml`
D	`products/productDetailLegwear\.jhtml`
E	`main/shopping_cart\.jhtml`
F	`main/login2\.jhtml`
G	`main/registration\.jhtml`
H	`main/welcome\.jhtml`
I	`checkout/expressCheckout\.jhtml`
J	`main/boutique\.jhtml`
K	`main/assortment\.jhtml`
L	`main/vendor\.jhtml`
M	`main/leg_news\.jhtml`
N	`products/productDetailLegcare\.jhtml`

5.3.6.3 Empirical Assessment of Soft-Support Constraint

The fault-tolerant matching is intended to enable to find the valid regularities or patterns, when data or the phenomenon, we would like to capture, is somehow noisy. To assess this empirically, we analyzed the extracted patterns, which are the contiguous sequences of requested templates. In the examples below, we denote the template by the Latin alphabet capital letters (see Table 5.5).

In the previous section we noticed that a minimum soft-frequency constraint enables to identify regularities whose exact-frequency is not discriminant (i.e., the relevant pattern is blurred among many other ones). For instance, among the patterns, which belong to the solution to inductive query

$$IQ_7 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(2\%, \mathcal{D}_{\mathtt{S}}, 1, 1) \wedge \mathsf{MinLength}_{\phi}(5) \wedge \mathsf{MaxLength}_{\phi}(10),$$

the highest frequency, equal to 2.9%, there is the pattern `ABCCD`. Its exact-frequency is only 0.18%. Similarly, among the patterns, which belong to the solution to inductive query

$$IQ_8 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(0.5\%, \mathcal{D}_{\mathtt{S}}, 1, 1) \wedge \mathsf{MinLength}_{\phi}(7) \wedge \mathsf{MaxLength}_{\phi}(10),$$

the highest frequency, equal to 0.8%, there is the pattern `DCDDDDD`. Its exact-frequency is 0.06% only. To get some empirical feedback on the ratio of soft-support and its corresponding exact-support, we evaluated the query

$$IQ_9 = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(0.5\%, \mathcal{D}_{\mathtt{S}}, 1, 1) \wedge \mathsf{MinLength}_{\phi}(5) \wedge \mathsf{MaxLength}_{\phi}(10).$$

We retrieved the exact-frequency and the *InsDels* soft-frequency of the patterns satisfying IQ_9 and computed the ratio $\frac{\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_{\mathtt{S}})}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_{\mathtt{S}},1,1)}$. Table 5.6 gives the number of patterns for the different intervals of this ratio. Observe that the value of the exact-frequency is not discriminant for the major part of the solutions to IQ_9.

In a number of cases, the exact-frequency and the soft-frequency of a pattern ϕ do not coincide. When looking for data set characteristic patterns, exact-frequencies can differ significantly, while soft-frequencies are similar. For example, when searching for the patterns that are characteristic to the data set $\mathcal{D}_{\mathtt{SWE}}$, composed of sessions, requested on weekends, w.r.t. the data set $\mathcal{D}_{\mathtt{SWD}}$, composed of sessions, requested on workdays, with the inductive query

$$IQ_{10} = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(0.01\%, \mathcal{D}_{\mathtt{SWE}}) \wedge \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(0.005\%, \mathcal{D}_{\mathtt{SWD}}) \wedge \\ \mathsf{MinLength}_{\phi}(4) \wedge \mathsf{MaxLength}_{\phi}(10),$$

the pattern ADDD is among the extracted ones. However its soft-frequency in both data sets is quite similar: $\mathsf{SoftFr}_{InsDels,S}(\mathtt{ADDD}, \mathcal{D}_{\mathtt{SWE}}, 0, 1) = 0.5\%$ and $\mathsf{SoftFr}_{InsDels,S}(\mathtt{ADDD}, \mathcal{D}_{\mathtt{SWD}}, 0, 1) = 0.4\%$[18]. Similarly, when searching for the patterns that are characteristic to the data set $\mathcal{D}_{\mathtt{SN}}$, composed of sessions, requested from 7 pm to 8 am, w.r.t. the data set $\mathcal{D}_{\mathtt{SD}}$, composed of sessions, requested on from 8 am to 7 pm, with the inductive query

$$IQ_{11} = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(0.1\%, \mathcal{D}_{\mathtt{SN}}) \wedge \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(0.05\%, \mathcal{D}_{\mathtt{SD}}) \wedge \\ \mathsf{MinLength}_{\phi}(4) \wedge \mathsf{MaxLength}_{\phi}(10),$$

the pattern AMALA belongs to the solution, but

$$\mathsf{SoftFr}_{InsDels,S}(\mathtt{AMALA}, \mathcal{D}_{\mathtt{SN}}, 0, 1) = 0.5\%$$

and

$$\mathsf{SoftFr}_{InsDels,S}(\mathtt{AMALA}, \mathcal{D}_{\mathtt{SD}}, 0, 1) = 0.4\%[19].$$

Exact-frequency and soft-frequency values can be even contradictory. For instance, the patterns DBDBN and ABDBDB belong to the solution set to the inductive query

$$IQ_{12} = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(0.1\%, \mathcal{D}_{\mathtt{SN}}) \wedge \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(0.07\%, \mathcal{D}_{\mathtt{SD}}) \wedge \\ \mathsf{MinLength}_{\phi}(4) \wedge \mathsf{MaxLength}_{\phi}(10),$$

but their soft-frequencies are

[18] We set the stricter fault-tolerance parameters setting $maxIns = 0$ and $maxDels = 1$, since the length of the pattern ADDD is only 4.

[19] We allowed the fault-tolerance parameters setting $maxIns = 1$ and $maxDels = 1$, since the length of the pattern ADDD is 5.

Table 5.6: Number of patterns for exact-support/soft-support intervals

$r = \frac{\mathsf{Fr}_{\sqsubseteq,S}(\phi,\mathcal{D}_\mathtt{S})}{\mathsf{SoftFr}_{InsDels,S}(\phi,\mathcal{D}_\mathtt{S},1,1)}$	Nb of patterns
$0.0002 \le r < 0.001$	2131
$0.001 \le r < 0.01$	3159
$0.01 \le r < 0.1$	1720
$0.1 \le r < 0.5$	419
$r \ge 0.5$	5

$$\mathsf{SoftFr}_{InsDels,S}(\mathtt{DBDBN}, \mathcal{D}_{\mathtt{SN}}, 1, 1) = \mathsf{SoftFr}_{\mathtt{InsDels,S}}(\mathtt{ABDBDB}, \mathcal{D}_{\mathtt{SN}}, 1, 1) = 0.3\%$$

and

$$\mathsf{SoftFr}_{InsDels,S}(\mathtt{DBDBN}, \mathcal{D}_{\mathtt{SD}}, 1, 1) = \mathsf{SoftFr}_{\mathtt{InsDels,S}}(\mathtt{ABDBDB}, \mathcal{D}_{\mathtt{SD}}, 1, 1) = 0.8\%$$

These examples illustrate that soft-frequency constraints are needed to avoid misleading hypothesis, when characterizing data set.

5.3.6.4 Time Efficiency

The time needed to solve the inductive queries IQ_2, IQ_3, IQ_4, IQ_5, and IQ_6 with different minimum frequency threshold $minFr$ values is plotted in Figure 5.3. The constraint $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$ evaluation is time-demanding because of the following reasons. The minimum soft-frequency constraint

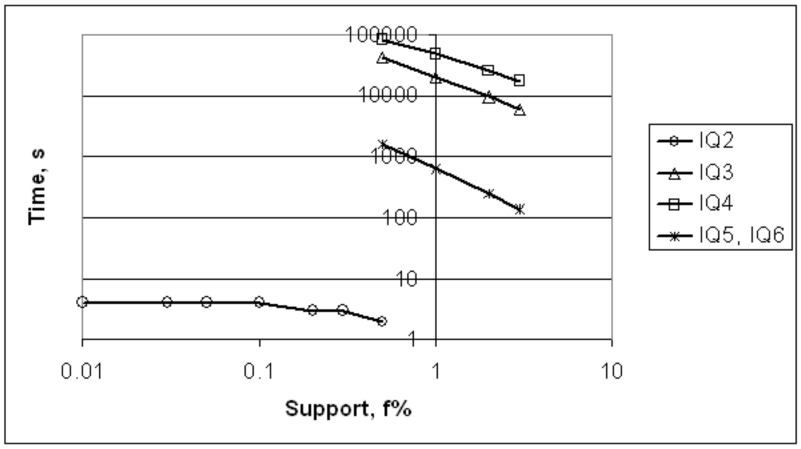

Figure 5.3: Time efficiency

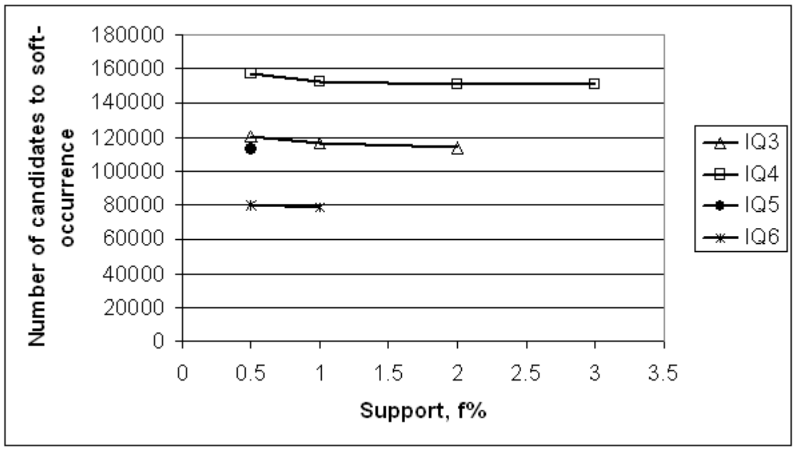

Figure 5.4: Number of candidates to soft-occurrences

is far less selective than the minimum exact-frequency constraint, famous for its pruning capacity (see Figure 5.1). In addition to this, the calculation of the soft-frequency $\mathsf{SoftFr}_{InsDels,S}(\phi, \mathcal{D}, maxIns, maxDels)$ is much more expensive than the exact-frequency counting: it requires to find pattern's ϕ soft-occurrences by evaluating the similarity constraint $\mathsf{InsDelsSim}_{\phi}(\sigma, maxIns, maxDels)$ (see Definition 5.7 in Page 105 and Algorithm 14 in Page 112). Even if we push its (anti)-monotonic conjunctions $\mathsf{MaxDels}_{\phi}(\sigma, maxDels)$ and $\mathsf{MaxIns}_{\phi}(\sigma, maxIns)$ deeply into the extraction phase, the number of candidates for which $\mathsf{InsDelsSim}_{\phi}(\sigma, maxIns, maxDels)$ still has to be evaluated can be huge (e.g., hundreds of thousands, see Figure 5.4).

The evaluation of the similarity relation $sim_{InsDels}(\phi, \sigma, maxIns, maxDels)$ when solving the constraint $InsDels$ similarity constraint $\mathsf{InsDelsSim}_{\phi}(\sigma, maxIns, maxDels)$ is expensive as well: it requires to compute the LCS of two given strings. To compute the LCS, we implemented a classical dynamic programming approach of time complexity O(nm) [Hir75]. However, this step can be optimized (see, e.g., [Apo97] for a survey). Also, when the fault-tolerance parameters are such that $maxIns = maxDels$, one can exploit the symmetric property of the underlying similarity relation. The computations can be also tuned by choosing dynamically the order of constraints to push.

Notice however that, even though soft-frequency counting may take hours (as we see in Figure 5.3), it does not prevent from optimizing the sequences of inductive queries involving the soft-frequency constraints. Indeed, it is possible to evaluate *once* the patterns' soft-frequency with some minimum frequency threshold $minFr$ value α (resp. maximal maximum frequency threshold $maxFr$ value β), before solving the queries which involve the minimum soft-frequency constraints $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$, such that $minFr \geq \alpha$ (resp. max-

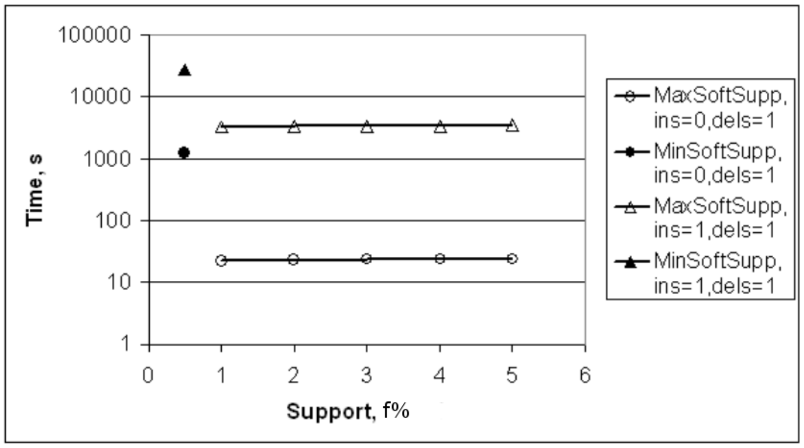

Figure 5.5: Time efficiency of solving maximum *InsDels* soft-frequency constraint in differential extraction

imum soft-frequency constraints $\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels)$, such that $maxFr \leq \beta$).

When evaluating the conjunction of constraints

$$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels) \wedge \mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels),$$

the calculation of the maximum *InsDels* soft-frequency constraint is far less expensive, since the search space is already pruned by the anti-monotonic minimum *InsDels* soft-frequency constraint and the S set is available (see Section 5.3.4 in Page 117). Figure 5.5 plots the time needed to evaluate each constraint of the conjunction

$$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(0.5\%, \mathcal{D}, maxIns, maxDels) \wedge \mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels),$$

with the fault-tolerance parameters settings $maxIns = 0, maxDels = 1$ and $maxIns = 1, maxDels = 1$, and the $maxFr$ ranging from 1% to 5%.

5.4 Discussion

The leading motivation of this contribution was to propose the generic solvers capable to handle the *similarity and the soft-frequency constraints* and their arbitrary conjunctions with other (anti-)monotonic constraints. Thanks to the observed stable relation between the length of the longest common subsequence between the reference and candidate strings and the substrings of the candidate string, we define the *similarity constraint* as a conjunction of an anti-monotonic and monotonic constraints. Such similarity constraint can be solved using the existing generic strategies for string mining. We implemented the corresponding generic solver in `C` programming language. The experimental validation confirms that the algorithm scales well both in time and space, and that the approach of pushing the similarity constraint allows to go far away beyond the limits of the approach of the similarity constraint post-processing. The empirical validation of the defined similarity constraint confirms that it enforces an useful similarity measure, since it allows to recover in data, perturbed by some random noise, all occurrences of the patterns that hold in the data initially. Concerning the *soft-frequency constraint*, is important to note that, when a soft-match function is used and thereby the soft-occurrences are taken into account, the associated minimum frequency constraint is, in general, no longer anti-monotonic. When searching how to solve it efficiently, we observe that the step of finding the pattern's soft-occurrences resumes indeed to an extraction under a similarity to a pattern of reference constraint. Thanks to our previous contribution, we know how to solve that constraint efficiently. We slightly reformulate the previously defined similarity constraint to use it as a soft-match function. Then, interestingly, when this soft-match function is used, the associated soft-frequency constraints are guaranteed to be (anti-)monotonic, provided that a (sensible) condition on similarity constraint parameters is satisfied. We implemented the corresponding generic solver in `C` and `C++` programming languages. The experimental validation confirms that the selectivity of the soft-frequency constraint is significantly decreased w.r.t. the selectivity of the exact frequency constraint, what allows to recover the regularities whose exact frequency is not discriminative. The computations are expensive due to the decreased selectivity and the cost of evaluating the similarity, however, the time efficiency can be improved by optimizing the sequences of inductive queries, e.g., by hashing the intermediate querying results.

Chapter 6

Studying the Twilight Zone

6.1 Problem Setting

Motivation

Numerous local pattern types (e.g., substrings, regular expression patterns, structured patterns, episode rules, subtrees, subgraphs, etc.) and constraints (e.g., similarity and soft-frequency constraints employing various match functions) have been extensively studied so far to suit better with user expectations and the application domain (see Chapter 4 for an example). New pattern and constraint types that support domain driven mining tasks make patterns more actionable and therefore are of crucial interest. Obviously, it does not solve all the problems. Providing the solvers for the pattern extraction is only one step in the process of knowledge discovery. When applying these solvers to answer the questions in the real-life applications, we are faced to the following problems: (1) how to set the extraction parameters in order to obtain an *interesting* collection of patterns, and (2) how to measure the pattern's *interest*, once a collection of patterns is extracted.

Concerning the first problem, an interesting parameter setting is, generally, the one that results in an exploitable collection of patterns, i.e., a collection that contains a reasonable number of patterns. To find these parameters values means to balance the stringency of constraints. A common practise to set the extraction parameters is to try several extractions, and, based on the results, guess (extrapolate) what could be the interesting parameters values. This guess is feasible in simple extraction contexts, where an inductive query is composed of one constraint. The inductive queries that express real-life questions are often expressed as combinations of several constraints. For example, to find the putative transcription factor binding sites that are characteristic to a given biological situation, the biologist want to identify the the patterns that are frequent in a positive data set $\mathcal{D}_+$ and not frequent in the negative data set $\mathcal{D}_-$. This question can be expressed by the following inductive query:

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_+, k) \ \wedge \ \mathsf{MaxSoftFr}_{\phi}^{H,S}(maxFr, \mathcal{D}_-, k) \ \wedge \\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen),$$

where $minLen = maxLen$, and $\mathsf{MinSoftFr}_{\phi}^{H,S}$, $\mathsf{MaxSoftFr}_{\phi}^{H,S}$ are the soft-frequency constraints, with the Hamming match function[1].

Such inductive query gives rise to a multidimensional parameter space

$$\{minFr, maxFr, k, minLen = maxLen\},$$

where it is difficult to guess the number of extracted patterns. Even after having found a parameter setting that results in a collection containing a reasonable number of patterns, all interesting patterns are not necessarily inside. Indeed, when the domain knowledge alone does not allow to derive the *a priori* interesting extraction paremeters values, most probably, an application domain analyst is interested in the first patterns that are extracted with the most stringent parameters of the constraints. Therefore one would seek to identify that border in the parameters space where the first patterns appear. A possible approach to identify that border is to serialize the extractions to probe the parameters space. However the extractions allowing for fault-tolerance are costly, and therefore this approach is rarely acceptable for the routine usage. The alternative approach is *to estimate the expected number of patterns that satisfy the given constraints* with the specified vectors of parameter values.

The second problem concerns measuring the pattern's interest. Even if the parameters for an extraction are sophistically tuned, among the extracted patterns there are many irrelevant ones (see, e.g., [CZ06]), what is the bottleneck of most local pattern extraction methods. In general, a pattern is interesting, if it is extracted not due to noise. Noise can be the random background or already known structure in the data. For example, when mining patterns in data streams without a priori knowledge, all except random patterns are interesting. When mining DNA sequences, nucleotide frequencies are in general known, and one can be interested in all patterns except those that are due only because of the differences in nucleotide frequencies. When mining Web page browsing sequences, the frequencies of the clicks are in general known, and patterns due to first order Markov dependencies between the linked pages can be considered as not interesting.

Such spurious patterns can be discarded using the statistical measures, and, in turn, the statistical constraints. These constraints, generally, cannot be fully/easily described by means of combinations of monotonic and anti-monotonic local constraints and thus cannot be efficiently pushed deeply to an extraction phase. Even if they do not have the nice monotonic properties, it would be of great interest to know whether a given pattern or a given collection of patterns is statistically unexpected

[1]See Section 4.3.1.1 in Page 69

given an input data set and some parameter values. For doing that, we can get inspired of [KP02b, GMMT07], which proposed to assess local patterns (respectively substrings and frequent itemsets) relevancy comparing the number of extracted patterns in the input data set with the number of patterns, extracted in random data sets exhibiting the same features (structural properties of the data set, e.g., size of the data set, number of sequences/items, length of the sequences/itemsets) as the original data set. The issue here is to be able *to estimate the expected number of patterns that satisfy the given constraints* in a random data set, modeling the noise.

Observe, that to solve both problems: the one of setting the extraction parameters and the one of evaluating pattern's interest, we need to know the expected number of patterns that satisfy the given constraints: under a data model, if we are setting the parameters, and under a noise model, if we are evaluating the patterns interest.

Research Context

Estimating the expected number of patterns that satisfy a constraint, is in general much more difficult than estimating the probability that a given pattern satisfies such a constraint. This second problem has received a lot of attention, leading to many statistical measures to assess the interestingness of the patterns. Concerning the first problem, only a few solutions have been proposed. [RMZ03] and [LRS05] analyze the feasible distributions of frequent itemsets (also of closed itemsets for [LRS05] and of maximal itemsets for [RMZ03]). [RMZ03] focuses on the kind of distributions one can expect for various kinds of data sets. They answer the question whether there exists a frequent or maximal frequent itemset collection that has a given number of frequent itemsets of a given length. [LRS05] computes the average number of frequent (closed) itemsets using probabilistic techniques. These authors especially focus on the minimum frequency threshold and how it influences the number of extracted patterns, considering fixed and/or proportional thresholds. Another approach has been proposed by Geerts et al. [GGdB05], providing a tight upper bound on the number of candidate patterns that can arise while mining frequent patterns in a level-wise setting. Given the current level and the current set of frequent patterns, they propose a tight bound of the maximal number of candidate patterns that can be generated on the next level. In the domain of string mining, [KP02b] designs an estimate of the number of patterns due to the random background, and that are likely to be extracted with respect to a frequency constraint and according to the structure of the data set. The so-called Twilight Zone (TZ) was defined as the set of values of the scoring function for which we can expect to have some random patterns exhibiting such score values. We propose to consider the notion of extraction parameters in a broad sense, including structural properties of the data set (e.g., number of sequences, length of the sequences) and the constraints thresholds. Then, the TZ can be seen as a region (or set of regions) in the parameter space, where we are likely to obtain random patterns among the extracted patterns, these random patterns having scores as good (or even better) as the *interesting* patterns. All these proposals uses a global analytical model to compute the estimations. Analytical approach has many advantages, however it becomes particularly difficult when extending it to consider more complicated data sequences model, handle additional constraints or different semantics for pattern occurrences.

Problem Statement

The objective is to estimate the expected number of string patterns with the Hamming match function[2] that satisfy a conjunction of a minimum soft-frequency constraint and a maximum soft-frequency constraint[3] in two data sets of random strings and syntactic constraints. To answer this question, there are three main approaches that can be considered:

1. One can extract and count the patterns that occur in random string data sets, generated with the desired features.
2. One can compute an analytical estimate of the expected number of extracted patterns from the features of the data set.
3. One can compute an estimate of the expected number of extracted patterns from pattern space samples and analytical formulas.

There is a clear trade-off between the time efficiency and the accuracy of the estimate. The first method, while providing a good estimate of the expected number of patterns (as long as we are able to generate random data sets exhibiting the desired properties) is too time consuming, since extractions allowing for fault-tolerance are costly and numerous complete extractions have to be performed, both to probe the parameters space in order to set the parameters sophistically, and to compute a measure of interest of the extracted patterns (since this measure have to be computed for a number of patterns). Therefore we focus on the two remaining alternatives, and develop the formulas that compute that estimate analytically (see Section 6.2) and the approach to evalute that estimate on pattern space samples (see Section 6.3).

6.2 Analytical Estimation

In this section we present our contribution on computing analytically, from the features of the data set, an estimate of the expected number of patterns, published

[2]See Section 4.3.1.1 in Page 69

[3]I.e., a differential extraction

in [MRS$^+$08].

We study the string patterns ϕ with the Hamming match function $match_{H,s}(\phi, s, k)$ (see Definition 4.19 in Page 69). We remind that the Hamming distance is finite, and therefore the Hamming match function is defined, if the pattern ϕ and its soft-occurrence is of the same length. We denote that length by l. In the following we will refer the string patterns ϕ with the Hamming match function as *exact matching patterns* (EMP), if the errors threshold $k = 0$[4], otherwise we will refer them as *soft-matching patterns* (SMP). We consider data strings of the same length L on the alphabet containing n symbols. As in [KP02b], we suppose that the data strings are composed of independent and uniformly distributed symbols, having the same occurrence probability, and that the overlapping of the occurrences of the patterns has a negligible impact on the number of patterns extracted (since $l \ll L$). We want to estimate the number of EMPs and SMPs of length l that will be extracted by the differential extraction, expressed by the following inductive query

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_+, k) \;\wedge\; \mathsf{MaxSoftFr}_{\phi}^{H,S}(maxFr, \mathcal{D}_-, k) \;\wedge\; \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen),$$

$$\text{where } minLen = maxLen,$$

for the given thresholds values $minFr$, $maxFr$ and k.

Additionally, we suppose that the two data sets, the positive one $\mathcal{D}_+$, and the negative one $\mathcal{D}_-$, are independent.

6.2.1 Occurrences at a given position

The hypotheses made on the distribution of the symbols imply that the probability that a pattern ϕ of length l has an exact occurrence starting at a given position in a string[5] is $P(\textit{exact occ. of } \phi \textit{ at one position}) = \frac{1}{n^l}$.

From an exact occurrence of ϕ, one can construct the soft-occurrences of ϕ within an Hamming distance k by placing i substitutions in $\binom{l}{i}$ possible ways, with $i \in \{0, \dots k\}$. Since we have n symbols, then for each position, where we have a substitution, we have $n-1$ different possible substitutions. Thus, for a pattern ϕ, there are $\sum_{i=0}^{k} \binom{l}{i} \times (n-1)^i$ substrings that are soft-occurrences of ϕ. Then, the probability that a pattern has a soft-occurrence starting at a given position in a data string is $P(\textit{soft occ. of } \phi \textit{ at one position}) = \frac{\sum_{i=0}^{k} \binom{l}{i} \times (n-1)^i}{n^l}$. In the following, we

[4]Note, that with the errors threshold $k = 0$, the Hamming match function is equivalent to exact string match function (see Definition 1.21 in Page 21). Therefore, to simplify the notations, where convenient, in this chapter we refer the exact string match function and the exact-frequency constraints as the Hamming match function and the Hamming soft-frequency constraints with errors threshold $k = 0$.

[5]Except the last $l - 1$ positions.

also need the probability that a pattern ϕ has a *strict* soft-occurrence starting at a given position (a *strict* soft-occurrence of ϕ, is a soft-occurrences of ϕ that is not an exact occurrence). In this case we have simply $P(\textit{strict soft occ. of } \phi \textit{ at one position}) = \frac{\sum_{i=1}^{k} \binom{l}{i} \times (n-1)^i}{n^l}$.

6.2.2 Occurrences in a data string

In a data string there are $(L - l + 1)$ possible positions to place the beginning of an occurrence of ϕ. Since $l \ll L$, for the sake of simplicity we approximate the number of possible positions by L. Then, the probability that there is no soft-occurrence of ϕ in a random data string is $P(\textit{no soft occ. of } \phi \textit{ in a string}) = (1 - P(\textit{soft occ. of } \phi \textit{ at one position}))^L$. Thus, the probability that there is at least one soft-occurrence of ϕ in a data string is $P(\textit{exists soft occ. of } \phi \textit{ in a string}) = 1 - (1 - P(\textit{soft occ. of } \phi \textit{ at one position}))^L$. Similarly, the probability that there is at least one strict soft-occurrence of ϕ is $P(\textit{exists strict soft occ. of } \phi \textit{ in a string}) = 1 - (1 - P(\textit{strict soft occ. of } \phi \textit{ at one position}))^L$, and the probability that there is at least one exact occurrence is $P(\textit{exists exact occ. of } \phi \textit{ in a string}) = 1 - (1 - \frac{1}{n^l})^L$.

6.2.3 Minimum Hamming Soft-Frequency Constraint

To determine $P(\phi \textit{ sat. min. H. soft-fr.})$, i.e., the probability of ϕ to satisfy the minimum hamming soft-frequency constraint, let us define X as the number of strings, in the positive data set, that contains at least one exact occurrence of ϕ. The probability $P(\phi \textit{ sat. min. H. soft-fr.})$ can be decomposed using the conditional probability of ϕ *sat. min. H. soft-fr.* given the value of X, as follows:

$$P(\phi \textit{ sat. min. H. soft-fr.}) = \sum_{i=1}^{N^+} (P(X = i) \times P(\phi \textit{ sat. min. H. soft-fr.} | X = i)), \tag{6.1}$$

where N^+ is the number of strings in the positive data set. Notice that the sum starts at $i = 1$, and not at $i = 0$, since the pattern must have at least one exact occurrence in the positive data set.

The variable X follows a binomial distribution $B(N^+, P(\textit{exists exact occ. of } \phi \textit{ in a string}))$, thus we have: $P(X = i) = \binom{N^+}{i} \times P(\textit{exists exact occ. of } \phi \textit{ in a string})^i \times (1 - P(\textit{exists exact occ. of } \phi \textit{ in a string}))^{N^+ - i}$.

$P(\phi \textit{ sat. min. H. soft-fr.} | X = i)$ is the probability that ϕ satisfies the minimum Hamming soft-frequency constraint, given that exactly i strings contain at least one exact occurrence of ϕ. This also means that $(N_+ - i)$ strings do not have any exact occurrence of a pattern. Then according to i there are two cases:

1. if $i \geq minFr$ then $P(\phi$ *sat. min. H. soft-fr.*$|X = i)) = 1$ since the constraint is already satisfied by the i strings that contain each at least one exact occurrence of ϕ.

2. if $i < minFr$ then $P(\phi$ *sat. min. H. soft-fr.*$|X = i)$ is equal to the probability that at least $(minFr - i)$ of the $(N^+ - i)$ remaining strings contain at least one strict soft-occurrence. This number of strings that contain at least one strict soft-occurrence of ϕ also follows a binomial distribution $B(N^+ - i, P($*exists strict soft occ. of* ϕ *in a string*$))$. Then we have: $P(\phi$ *sat. min. H. soft-fr.*$|X = i)) = \sum_{z=minFr_{min}-i}^{N^+-i}(\binom{N^+-i}{z} \times P($*exists strict soft occ. of* ϕ *in a string*$)^z \times (1 - P($*exists strict soft occ. of* ϕ *in a string*$))^{N^+-i-z})$.

Thus, we can obtain $P(\phi$ *sat. min. H. soft-fr.*$)$ by computing the sum in equation 6.1 and $P(\phi$ *sat. min. H. soft-fr.*$|X = i)$ according to the two cases above.

6.2.4 Maximum Hamming Soft-Frequency Constraint

Let Y be the number of data strings that contain the occurrences of ϕ in the negative data set. A pattern ϕ satisfies the maximum Hamming soft-frequency constraint with threshold $maxFr$ if $Y \leq maxFr$. The variable Y follows a binomial distribution $B(N^-,\ P($*exists soft occ. of* ϕ *in a string*$))$, where N^- is the number of data strings in the negative data set. Then the probability that ϕ satisfies the maximum Hamming soft-frequency constraint is $P(\phi$ *sat. max. supp.*$)) = \sum_{z=0}^{maxFr}\binom{N^-}{z} \times P($*exists soft occ. of* ϕ *in a string*$)^z \times (1 - P($*exists soft occ. of* ϕ *in a string*$))^{N^- - z}$.

6.2.5 Frequency Constraints for Differential Extraction

Given our hypothesis that the positive and negative data sets are independent, the probability that a pattern satisfies a conjunction of minimum Hamming soft-frequency and maximum Hamming soft-frequency constraints is $P(\phi$ *sat. min. and max. supp.*$) = P(\phi$ *sat. min. supp.*$) \times P(\phi$ *sat. max. supp.*$)$.

6.2.6 Number of Expected Patterns and Twilight Zone Indicator

Let $ENP(l, minFr, maxFr, k)$ be the Expected Number of Patterns of length l that will be extracted under the thresholds $minFr$, $maxFr$ and k, i.e., the approximate we seek to compute analytically. Since there are n^l possible patterns of length l, and from the hypothesis that the overlapping of the occurrences of the patterns has a negligible impact on the number of patterns extracted, we can approximate $ENP(l, minFr, maxFr, k)$ by $P(\phi$ *sat. min. and max. soft. H. freq.*$) \times n^l$.

Given a pattern ϕ with the Hamming soft-frequencies $\mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_+, k)$ and $\mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_-, k)$ for a given threshold k, its Twilight Zone Indicator, denoted $TZI(\phi)$ is defined as $ENP(|\phi|, \mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_+, k), \mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_-, k), k)$.

6.2.7 Experimental validation

To validate empirically the computation of the TZI, i.e., of the expected number of patterns, we compared it with the number of patterns extracted from random string data sets and gene promoter sequences data sets. The gene promoter sequences data sets, denoted R and A, are described in Section 7.1.2.2 in Page 157, were provided by our biologist collaborator Dr. Olivier Gandrillon and his group BM2A in the Center for Molecular and Cellular Genetics (CNRS UMR 5534). We precise that each sequence is a string of length of 4000 symbols over an alphabet of size 4[6]. Two pairs of random data sets that mimic the biological data sets R and A were constructed using the tool *RanDNA* [PP06]: (1) pair $\langle R^*, A^* \rangle$, where R^* (resp. A^*) have the same number of data strings and the same total number of nucleotides per string as R (resp. A), and are built from independent and uniformly distributed nucleotides with a homogeneous nucleotide composition (i.e., 25% of A, C, G and T); (2) pair $\langle R^{**}, A^{**} \rangle$, where R^{**} and A^{**} are generated using the same constraints as $\langle R^*, A^* \rangle$, except for the relative nucleotide composition. For R^{**} (resp. A^{**}) the relative nucleotide composition is the same as the one of R (resp. A). Moreover, the same sequencing uncertainties (N regions)[7] as in R (resp. A) have been implanted in R^{**} (resp. A^{**}).

The graphs in the left column in Figure 6.1 depict the number of EMPs, satisfying the constraints

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, R, 0) \ \wedge \ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, A, 0) \wedge \\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen)^{8},$$

They were extracted using our implementation of the `FAVST` solver [DD04] (see Section 3.2.2.2 in Page 51).

The graphs in the right column in Figure 6.1 depict the number of SMPs, satisfying the constraints

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, R, k) \ \wedge \ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, A, k) \wedge \\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen),$$

[6]DNA sequences are composed of 4 nucleotides: adenine, guanine, thymine, cytosine. Therefore such sequences and patterns in them can be seen as strings over the alphabet of 4 symbols $\{A, C, G, T\}$.

[7]Notice that `Marguerite` does not require a predefined alphabet, and can therefore handle sequences containing undefined bases, denoted by the symbol N.

[8]Or, equivalently, $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, R) \ \wedge \ \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr, A) \ \wedge \ \mathsf{MinLength}_{\phi}(minLen) \ \wedge \ \mathsf{MaxLength}_{\phi}(maxLen)$.

where $k > 0$.

They were extracted using our generic solver's instance `Marguerite-H` (see Section 5.3.5 in Page 118).

The frequency and pattern length threshold values were chosen to be pertinent w.r.t. the associated biological problem[9] (see Chapter 7). The value of $minFr$ varies from 1 to 29, and $maxFr$ is set to 7, whereas the pattern length is restricted by $minLen = 5$ and $maxLen = 11$. Since we use a correct and complete pattern extraction approach, there is no need to run it several times with the same parameter values (contrarily to approaches based on incomplete heuristics). For each parameter setting, $minFr$, $maxFr$ (and k for SMPs), we compute the number of expected patterns due to the random background using the TZI formula.

The graphs depicted in Figure 6.1 allows to compare the expected number of patterns with the number of patterns extracted in random and biological data sets (the extractions used for the illustration are representative, i.e., the behaviour remains the same also for other $minFr$ values). The number of patterns extracted in data sets $\langle R^*, A^* \rangle$ coincides with the expected number (graphs A vs. B and E vs. F in Figure 6.1).

This argues in favor of the correctness of the hypothesis made concerning the limited impact of the overlapping of the occurrences of the patterns on the number of extracted patterns. The number of patterns extracted in the data sets $\langle R^{**}, A^{**} \rangle$ (with the exceptions of the EMP of length 6 and of the SMP of length 8) is still well modeled by the computed number of expected patterns (graphs A vs. C and E vs. G in Figure 6.1). This confirms that the simplification of considering an equiprobable nucleotide distribution and not taking into account the sequencing uncertainties do not modify the counts significantly. The number of patterns extracted in the biological data sets R, A deviates more from the expected number and is greater than in the random data sets, but the estimations still model well the tendencies, especially in the range of parameters that are interesting to our problem, i.e., when α_{min} is large and α_{max} is small (graphs D and H in Figure 6.1). This indicates that at least a part of the hidden structure of the biological data set pair $\langle R, A \rangle$ (absent from the model of random background and absent from the random data sets) seems to be captured by the extracted patterns.

We used this analytically computed expected number of patterns in our application to promoter sequences analysis (Section 7.1) to guide the extraction parameters tuning (see Section 7.1.2.4 in page 160) and to select the patterns according TZI (see Section 7.1.2.3 in page 159).

[9]These settings are also representative, and other settings lead to a similar global behaviour.

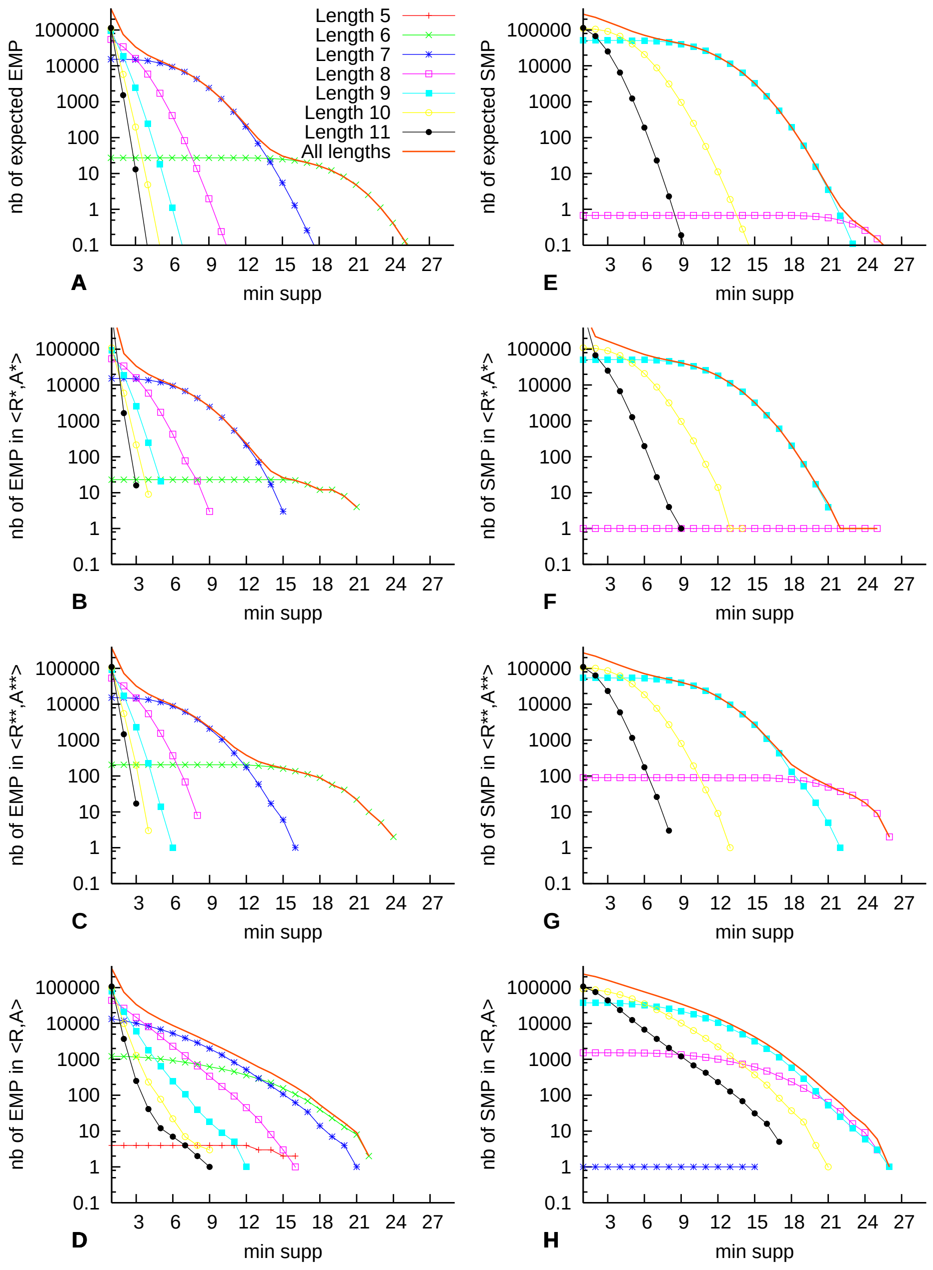

Figure 6.1: In the left column: number of EMPs, satisfying a conjunction of a minimum exact-frequency constraint in a positive data set and a maximum exact-frequency constraint in a negative data set. In the right column: number of SMPs, satisfying a conjunction of a minimum Hamming soft-frequency constraint in a positive data set and a maximum Hamming soft-frequency constraint in a negative data set.

6.3 Estimation through Pattern Sampling

Analytical estimation of the expected number of patterns that satisfy the constraints becomes very difficult when the data model or the inductive query is more complicated, e.g., when considering non-equiprobable symbol distributions or when adding syntactic constraints. Analytical estimation is difficult since we have to compute the probability that an abstract pattern (i.e., any pattern from the pattern language) satisfies the constraints. It is much more easier to compute this probability for a pattern at a hand (i.e., a given concrete pattern). Based on this observation, we propose to compute the expected number of patterns from the samples of the pattern space. This work is published in [BRMB08].

As in the previous section, we study the string patterns ϕ with the Hamming match function $match_{H,s}(\phi, s, k)$ (see Definition 4.19 in Page 69), and we refer such string patterns as *exact matching patterns* (EMP), if the errors threshold $k = 0$[10], otherwise we will refer them as *soft-matching patterns* (SMP). Let S_C be the set of patterns in $\mathcal{L}$ that satisfy the inductive query, expressed through the following constraint conjunction C

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_+, k) \ \wedge \ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_-, k) \ \wedge C_{synt},$$

where C_{synt} is a syntactic constraint[11].

In this section, we present a simple method to estimate the cardinality of the set S_C by sampling the pattern space $\mathcal{L}$ and using a function that gives the probability of a pattern ϕ to satisfy the constraint C, denoted $P(\phi\ sat.\ C)$.

We chose three symbol distributions for the symbols of the input sequences, to show that our method can be used with different models. However, this choice is not central in the contribution, and depending on the application domain, other dedicated models that would be more accurate could be used to provide a better estimate.

The three retained models of distributions here are:

- μ_E: independence of all occurrences of the symbols with equal occurrence frequencies of each symbol;
- μ_D: independence of all occurrences of the symbols with different occurrence frequencies of the symbols;
- μ_M: a first-order Markov chain.

For each of the three models mentioned above, it is easy to compute the probability for a given pattern ϕ to occur in a string, then to obtain the probability to satisfy

[10]See Footnote 4 in Page 133.

[11]See Section 2.2.1.2 in Page 35.

a frequency constraint using a binomial law, and to finally determine $P(\phi\ sat.\ \mathcal{C})$.

6.3.1 Expected Number of Patterns that Satisfy Constraints in a Sample

Let us associate to each pattern ϕ a random variable X_ϕ, such that $X_\phi = 1$ when ϕ satisfies C and $X_\phi = 0$ otherwise. Then $|S_C| = \sum_{\phi \in \mathcal{L}} X_\phi$. Considering the expected value of $|S_C|$, by linearity of the expectation operator we have $E(|S_C|) = \sum_{\phi \in \mathcal{L}} E(X_\phi)$. Since $E(X_\phi) = 1 \times P(X_\phi = 1) + 0 \times P(X_\phi = 0)$, then $E(|S_C|) = \sum_{\phi \in \mathcal{L}} P(\phi\ sat.\ C)$. Let $S_{C_{synt}}$ be the set of patterns in $\mathcal{L}$ that satisfy C_{synt}. As $P(\phi\ sat.\ C) = 0$ for all patterns that do not satisfy C_{synt}, we have $E(|S_C|) = \sum_{\phi \in S_{C_{synt}}} P(\phi\ sat.\ C)$.

Computing this sum over $S_{C_{synt}}$ would be prohibitive, since we want to obtain $E(|S_C|)$ for a large number of points in the parameter space. Thus we estimate $E(|S_C|)$ using only a sample of the patterns in $S_{C_{synt}}$. Let S_{samp} be such a sample, then we use the following value as an estimate of $E(|S_C|)$:

$$\frac{|S_{C_{synt}}|}{|S_{samp}|} \times \sum_{\phi \in S_{samp}} P(\phi\ sat.\ C)$$

In practice, many techniques can be used to compute the sample. In the experiments, presented in the next section, we use the following process:

- Step 1: build an initial sample S_{samp} of C_{synt} (sampling with replacement) of size 5% of $|C_{synt}|$ and compute the estimate of $E(|S_C|)$.

- Step 2: go on sampling with replacement to add 1,000 elements to S_{samp}. Compute the estimate, and if the absolute value of the difference between the new estimate and the previous one is greater than 5% of the previous estimate, then repeat Step 2.

Since the variables X_ϕ are not independent (the occurrence of a pattern has an impact on the possibility of occurrence of other patterns), there is no straightforward analytical confidence bound of the estimate. It is out of the scope of this chapter to further discuss this issue. However, in the next section, we show that the estimate is quite accurate in practice.

6.3.2 Experimental validation

6.3.2.1 Empirical evaluation of the estimate

To empirically assess our method, we have to check both its efficiency in terms of running time and accuracy. Recall that we may need to estimate the expected number of patterns satisfying a user-defined constraint for a large number of values in parameter domains.

We generated three pairs of random strings data sets $\mathcal{D}_+$ and $\mathcal{D}_-$, and on each pair we performed a set of experiments. Each pair is based on a different symbol distribution and/or on a different data set structure. The symbol distributions used for the estimate were the same as the ones used for the generation. For each set of experiments, we present graphics to compare the estimate of the expected number of patterns versus the real number of patterns extracted in the data sets when using the same parameters. In the experiments, we explore different regions in the parameter space, at different scales, and we do not try to focus on parameter ranges that lead to the best estimates.

In the experiments, we extracted all EMPs, satisfying the inductive query

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_+, 0) \ \wedge\ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_-, 0) \ \wedge \\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen)^{12}, \\ \text{where } minLen = maxLen,$$

They were extracted using our implementation of the `FAVST` solver [DD04] (see Section 3.2.2.2 in Page 51).

We also extracted all SMPs, satisfying the inductive query

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_+, k) \ \wedge\ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, \mathcal{D}_-, k) \ \wedge \\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen), \\ \text{where } k = 1 \text{ and } minLen = maxLen.$$

They were extracted using our generic solver's instance `Marguerite-H` (see Section 5.3.5 in Page 118).

In all graphics, the isolated dots represent the estimates, the dots linked by a line represent the real number of extracted patterns. On the horizontal axis we have the minimal frequency thresholds, and on the vertical axis we have the associated number of patterns (for the sake of readability, we use for some of the graphics a log scale axis). The settings used for the three sets of experiments are the following:

[12] Or, equivalently, $\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, \mathcal{D}_-) \ \wedge\ \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr, \mathcal{D}_-) \ \wedge\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen)$.

- First set of experiments (Figure 6.2): 4 symbols with distribution μ_D (frequencies of the symbols are 0.4, 0.1, 0.2 and 0.3), data sets $\mathcal{D}_1$ and $\mathcal{D}_2$ contain 100 strings of length 1,000.

- Second set of experiments (Figure 6.3): 4 symbols with distribution μ_M, data sets $\mathcal{D}_+$ and $\mathcal{D}-$ contain 100 strings of length 1,000. The arbitrary conditional probabilities $Prob(\phi_i = Y|\phi_{i-1} = X)$ used for the Markov chain (with symbols A, B, C and D) are given by the table:

X / Y	A	B	C	D
A	0.2	0.28	0.18	0.34
B	0.04	0.36	0.3	0.3
C	0.32	0.08	0.2	0.4
D	0.2	0.24	0.24	0.32

- Third set of experiments (Figure 6.4): 8 symbols with distribution μ_E, data sets $\mathcal{D}_1$ and $\mathcal{D}_2$ contain 100 strings of length 30,000. In four of the graphics, the estimates are so close to the real values that the corresponding dots are superimposed.

In all experiments, the estimates closely follow the trends of the real extractions, and in most cases, the estimates are sufficiently accurate to give a reasonable picture of the shape of the extraction landscape. For each experiment, only between $4,000$ and $8,000$ sampled patterns were necessary to converge to a stable estimate, i.e., the difference between two successive estimates is smaller than 5% of the first one.

The experiments were run on a Linux platform with an Intel 2Ghz processor and 1Gb of RAM.

The estimate of the number of patterns, based on the pattern space sampling, was implemented in Perl. For each single EMPs extraction, the running time was about a few tens of seconds to a few minutes. In the case of SMPs, an extraction takes from a few tens of minutes to several hours to complete, while for the sampling-based estimate, computing one estimate requires between 1 and 30 seconds.

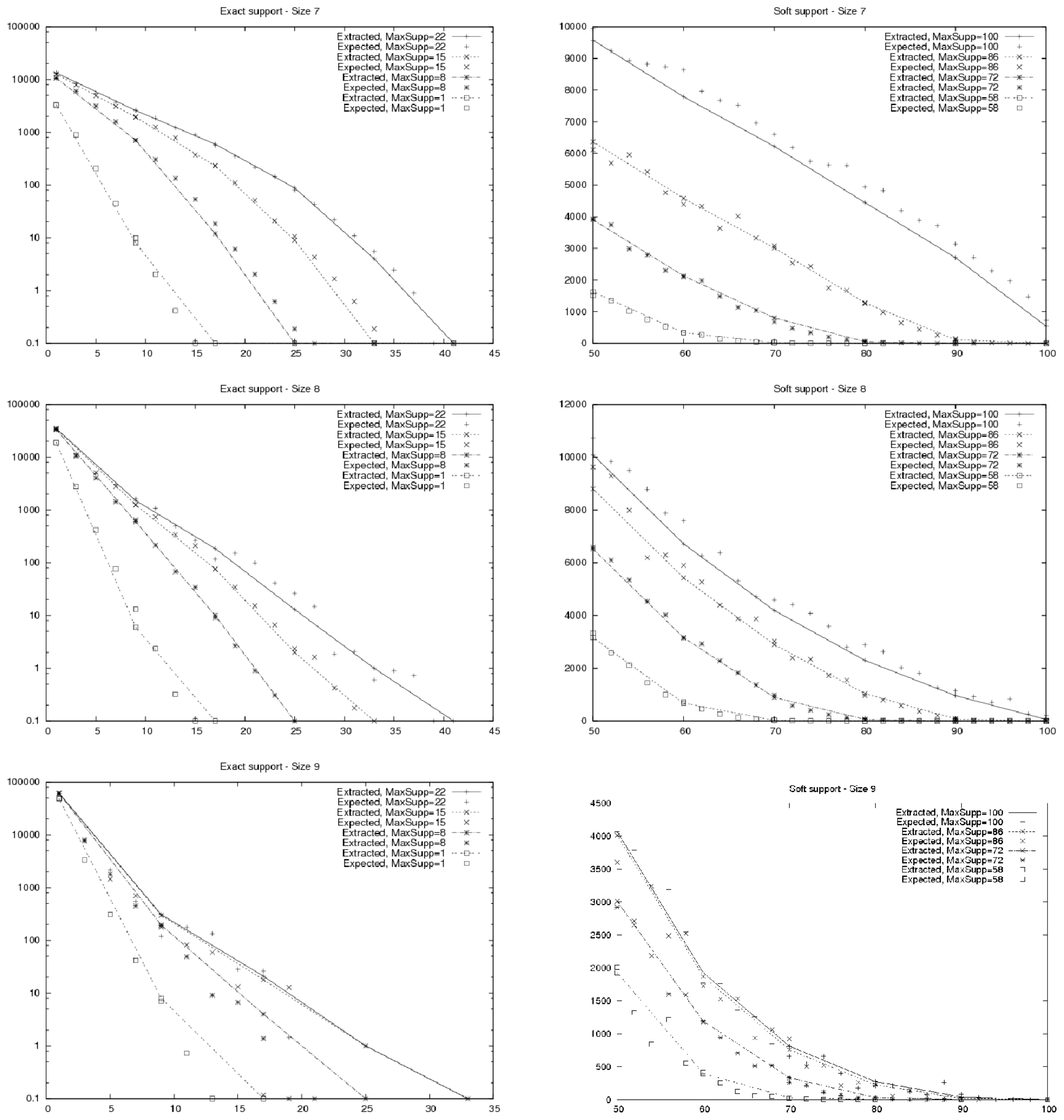

Figure 6.2: Experiments under μ_D distribution (with symbol frequencies 0.4, 0.1, 0.2 and 0.3). In the left column we have the exact-frequency threshold values on the horizontal axis and the number of resulting EMPs. In the right column we have the Hamming soft-frequency threshold values on the horizontal axis and the number of resulting SMPs.

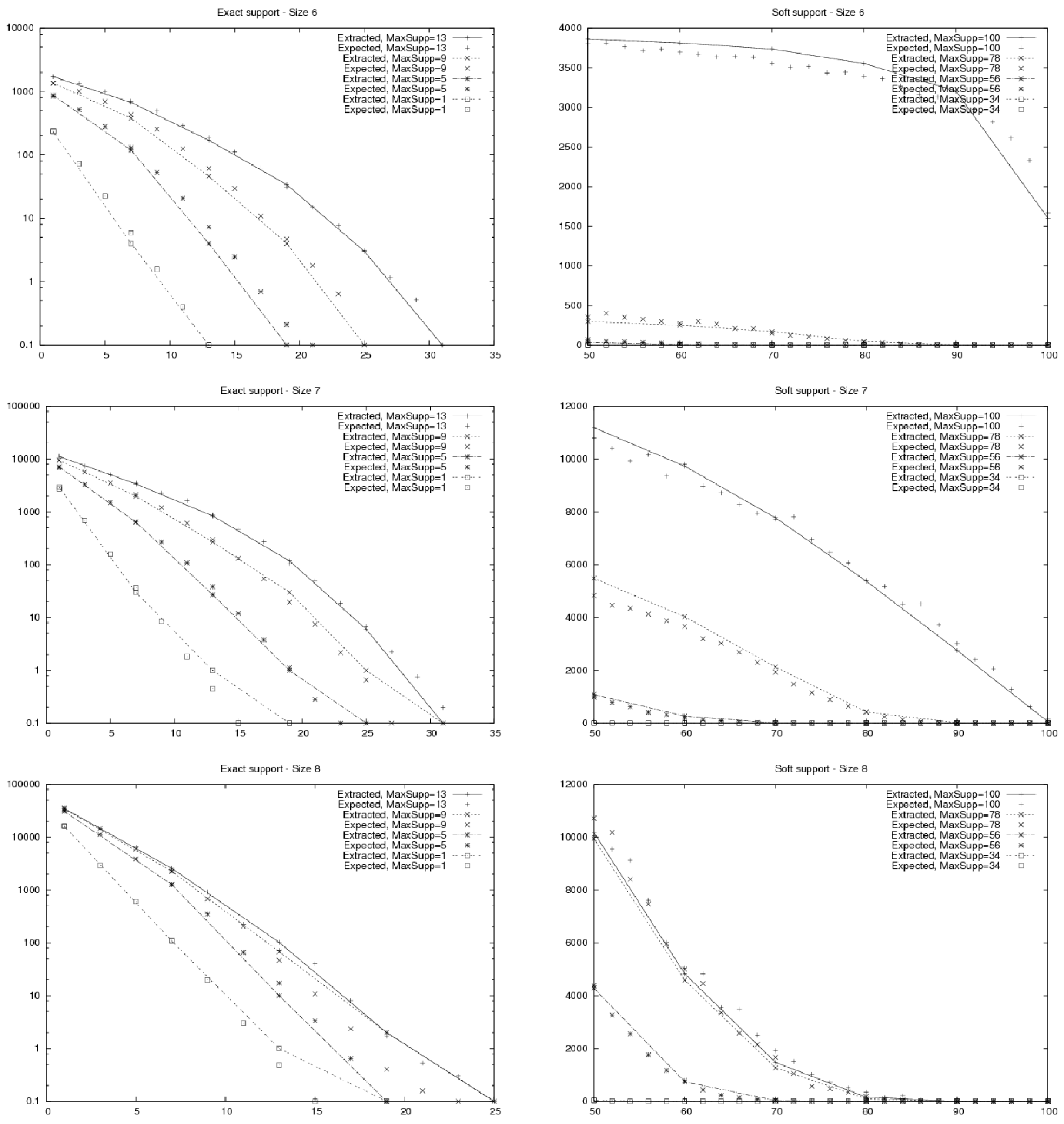

Figure 6.3: Experiments under μ_M distribution (1st-order Markov chain). In the left column we have the exact-frequency threshold values on the horizontal axis and the number of resulting EMPs. In the right column we have the Hamming soft-frequency threshold values on the horizontal axis and the number of resulting SMPs.

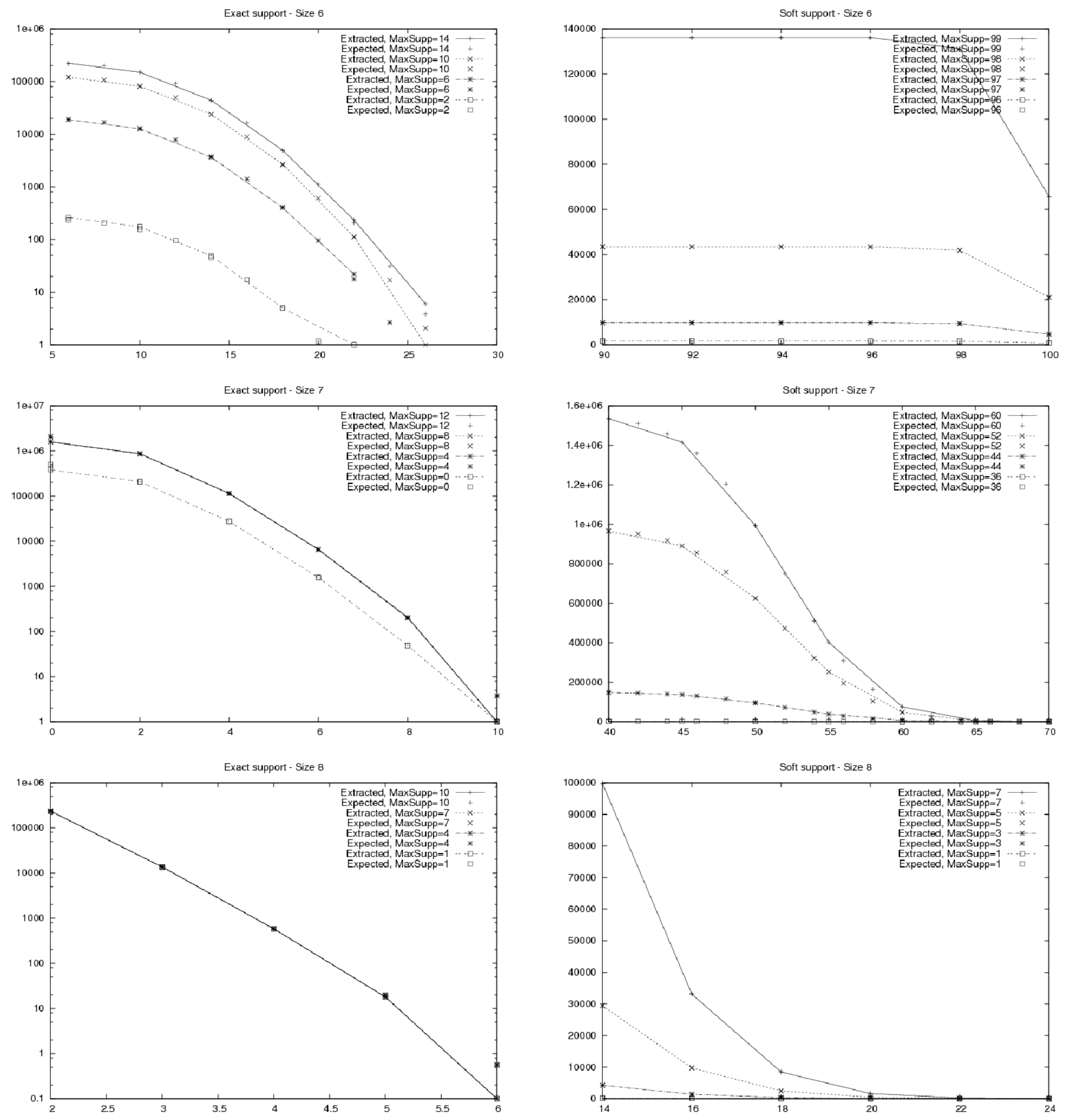

Figure 6.4: Experiments under μ_E distribution (same frequency for symbols) and string size 30,000. In the left column we have the exact-frequency threshold values on the horizontal axis and the number of resulting EMPs. In the right column we have the Hamming soft-frequency threshold values on the horizontal axis and the number of resulting SMPs.

6.3.3 Application to Promoter Sequences Data Sets

We use two data sets of DNA sequences that are promoter sequences of genes. These gene promoter sequences data sets, denoted R and A and described in Section 7.1.2.2 in Page 157, were provided by our biologist collaborator Dr. Olivier Gandrillon and his group BM2A in the Center for Molecular and Cellular Genetics (CNRS UMR 5534). We precise that each sequence is a string of length of 4000 symbols over an alphabet of size 4[13]

We look for SMPs, satisfying the inductive query

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, R, k) \ \wedge\ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, A, k) \ \wedge \\ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen), \\ \text{where } k = 1 \text{ and } minLen = maxLen.$$

Such a conjunction of constraints, with soft-frequency definition based on the Hamming distance, has been shown to be useful in practice for transcription factor binding sites identification (see Chapter 7).

The estimates were computed with distribution μ_D, using as symbol frequencies their respective frequencies in the data (0.23, 0.26, 0.27, 0.24 respectively for A, C, G, and T). In this case, the model is more a description of the random background than a description of the biological organisation along the sequences. Representative graphics obtained using these estimates, and depicting portions of the extraction landscape, are presented in Figure 6.5, on the right. A typical use of such graphics is, for instance, to look for points, in the parameter space, corresponding to a large frequency on R, but a low frequency on A, a large pattern length, and a rather small number of expected patterns (since here the distribution represents the random background). Such a point, that can be used as an initial guess of the parameters to perform real extractions, is for instance: pattern length $l = 10$, minimal Hamming soft-frequency threshold $minFr$ on R of 15 and maximal Hamming soft-frequency $maxFr$ on A of 5 (the graphic in the middle on the right indicates that, for this setting, only about 1 pattern due to the random background is expected).

For the sake of completeness, in Figure 6.5 on the left, we give the real numbers of extracted patterns. In practice, these graphics are not easily accessible to the user, since in these experiments the running time of a single extraction ranges from several tens of minutes to several hours (while for an estimate, only a few tens of seconds is needed). Even though the global trends correspond to the estimates, they are important differences in some portions of the parameter space (these differences could be expected since the distribution used does not incorporate complex biological knowledge). However, the estimates can still be used to help to choose initial parameter

[13]DNA sequences are composed of 4 nucleotides: adenine, guanine, thymine, cytosine. Therefore such sequences and patterns in them can be seen as strings over the alphabet of 4 symbols $\{A, C, G, T\}$.

values in an exploratory mining stage, and moreover, finding such differences, when running the real extractions, can also be a useful piece of information in itself. For example, for the setting $l = 10$, $minFr = 15$ and $maxFr = 5$, we have about 100 patterns really extracted, while we expected only one. If we suppose that the pattern space sampling can provide reasonable estimates when the data satisfy distribution μ_D (as supported by the experiments of Section 6.3.2.1), then the 100 patterns obtained are likely to be due to a particular unknown structure in the data and not to the distribution μ_D only. This suggest that it makes sense to look for patterns in this region of the parameter space, since we can expect to obtain some interesting/useful patterns (not only patterns due to the random background captured by μ_D).

6.4 Discussion

Two answer both problems, the one of setting the extraction parameters and the one of measuring the pattern's interest, we need to estimate the number of patterns that satisfy the given constraints. Concerning the first problem, the expected number of patterns that are extracted with each point (vector) in the parameters space allows to identify the border in the parameters space where the first patterns are extracted. A domain analyst, in general, will want to analyse the collections of these first patterns, extracted with the most stringent values of constraint parameters, when the application domain knowledge is not sufficient to derive the *a priori* interesting parameters for the extraction. Concerning the second problem, the minimum number of patterns that will be extracted due to the noise with a pattern in question (the so-called TZI) can be used as a sensible measure of pattern's interest - the bigger the TZI, the more indistinguishable from the noise the pattern is. Our contribution consists of estimating the number of patterns that satisfy the given constraints in the data sets under a given sequence model: under a data sequences model, if we are setting the parameters, and under a noise model, if we are evaluating the pattern's interest.

We develop an analytical estimation of the number of patterns that satisfy a conjunction of the minimum and maximum soft-frequency constraints, where the Hamming match function is used, and the length constraints. The data sequences model assumes that the sequences are of the same length, the symbols are independent and equiprobable, and the overlapping occurrences are not taken into account. The analytical estimation becomes particularly difficult if the data sequences model is more complicated or if there is a need to take into account additional constraints, e.g., syntactic constraints. The difficulty of the analytical approach resides in the fact that we have to compute the probability that any pattern (i.e., an abstract pattern) satisfies the constraints. It is much more easier to compute this probability for a

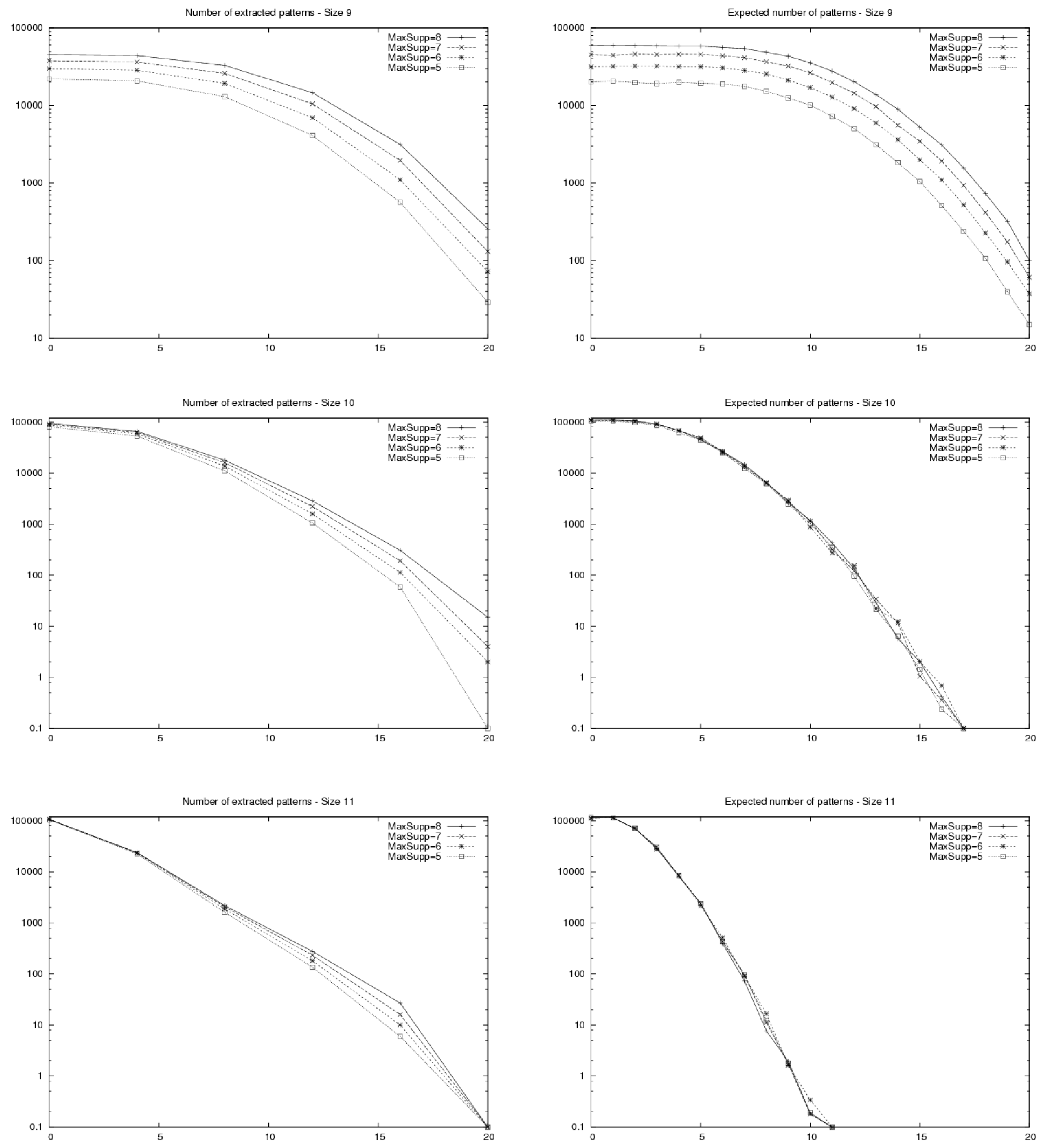

Figure 6.5: Number of expected and extracted SMPS, using two promoter sequence data sets. Horizontal axis: minimum support, vertical axis: number of patterns.

concrete given pattern. Based on this observation we develop the estimation through sampling the pattern space. The different data sequences models and the syntactic constraints can be easily taken into account in this approach of estimation through sampling. We performed the experimental validation to compare the extracted and the estimated number of patterns using three different sequence models. We used both generated and real-world (gene promoter sequences) data sets. The experiments show that the estimations model well the reality, and thus confirm that we sample in the right way.

Part III

Application

Chapter 7

Genomic Sequence Analysis

In this chapter we present our application of the developed solvers and the TZI (Twilight Zone Indicator) measure of pattern interest to the gene promoter sequence analysis, published in [MRS$^+$08]. We also present an opening to the application to comparative genomics.

7.1 Promoter Sequence Analysis

7.1.1 Background

Application Context

We applied

- our implementation of `FAVST` solver [DD04][1] to perform correct and complete differential extractions under exact frequency constraints,
- our generic solver's instance `Marguerite-H`[2] to perform the correct and complete differential extractions under Hamming soft-frequency constraints[3],
- the developed pattern interest measure Twilight Zone Indicator (TZI)[4], enabling to assess the interest of the patterns obtained by such extractions,

[1]See Section 3.2.2.2 in Page 51
[2]See Section 5.3.5 in Page 118
[3]See Section 4.3.1.1 in Page 69
[4]See Section 6.2.6 in Page 135

to find the signature motifs[5] in promoter sets of differentially expressed genes. This work is accomplished in a close collaboration with Dr. Olivier Gandrillon and his group Molecular Basis of Self-Renewal (BM2A) in the Center for Molecular and Cellular Genetics (CNRS UMR 5534).

Self-renewal, which is a characteristic property of stem cells, the molecular basis of which is still elusive, is the research topic of our collaborator BM2A group. Deregulation of this process occurs frequently during cancer generation. The BM2A team investigates this process through the discovery of differentially expressed genes by using the SAGE technique [VEVK95] on the primary model of T2ECs, that are normal chicken erythroid progenitors [GSBS99]. These cells can be induced to self-renew or to differentiate when normal. The expression of the v-erbA oncogene induces transformation by blocking their differentiation process [GJP+89]. Therefore the BM2A team decided to identify the v-erbA target genes responsible for the transformation process induced by v-erbA. For this they compared the transcriptome of T2ECs, expressing an oncogenic form of v-erbA, with the transcriptome of T2ECs, expressing the S61G mutant of v-erbA. This mutant is defective in its ability to inhibit differentiation and to induce erythroid transformation [SP91]. Thus, the comparison between the transcriptome of cells, expressing either the transforming form of v-erbA or the S61G mutant of v-erbA, allowed to generate a list of 110 differentially expressed genes between these two conditions [BKF+07].

We used these v-erbA data sets to extract exact and soft-matching patterns under a differential extraction. We selected the most exceptional patterns based on the developed measure of interest TZI. The biological evaluation of such patterns confirm their putative functional role and thereby exemplify the potential of our motif discovery method. In order to assess the generality of our approach, we also applied our method to a second data set, made from a set of promoters of genes, showing differential expression, as assessed by SAGE, between self-renewing and differentiating erythroid progenitors [DKGG+04].

Research Context

To understand the regulation of the gene expression remains one of the major challenges in molecular biology. One of the elements through which the regulation works is the initiation of the transcription by the interaction between gene promoter elements at the level of the DNA sequence and multiple activator and repressor proteins called Transcription Factors (TFs). This interaction occurs when a TF binds on its binding site on a gene promoter. Numerous efforts have given rise to a variety of computational methods to discover putative Transcription Factors Binding Sites (TFBSs) in sets of promoters of co-regulated genes. Among them two fam-

[5]In this chapter we use the term *motif* to denote a pattern in DNA sequences that is known or supposed to exhibit a biological function.

ilies can be distinguished: statistical or stochastic approaches, and combinatorial approaches [VMS99a].

Concerning the family of statistical and stochastic approaches, a recent review of the most widely used algorithms exhibits rather limited results [TLB$^+$05], and concludes to the necessity to go on exploring alternative methods. There are several reasons for their limited success, but it seems that the difficulty to separate the patterns from the *random background* is among the principal ones. Statistical methods make hypothesis about the distribution models and assumptions for computational as well as statistical reasons, but no one knows the correct stochastic process that nature uses, and what is the *biological randomness*. Moreover, this stochastic process seems to be different from species to species: many tools perform much better on the yeasts data sets than on other species [TLB$^+$05, DD07]. In addition to this, considering the employed measures of interest, statistical significance is very dependent on the choice of the length of the promoter sequences: considering longer promoters would allow to identify regulatory elements located further upstream, but conversely then random motifs become statistically as significant as the regulatory elements [KP02b].

We focused on the family of combinatorial approaches that aims at an exhaustive motif extraction[6] without *a priori* hypothesis on the underlying stochastic process. Exhaustive algorithms enumerate all objects they were built to find. According to [KP02a], probably the best tools for finding consensus based motifs in DNA sequences are the pattern-driven algorithms that test all the 4^l different patterns[7] of length l, score each pattern by the number of approximate occurrences and find the high-scoring patterns. The exhaustive search through all these 4^l patterns becomes impractical for large l, but the length of binding sites in promoter sequences is estimated to be between 5 and 15 base-pairs (bp) [Bul03] and the mean of these lengths in $Transfac^{®}$ [MFG$^+$03] is 14.3 bp with standard deviation 4.7 bp [FWV$^+$05]. These rather reasonable values of l turn the search to be tractable in practice. However, the exhaustive methods are often not selective enough to discriminate true sites from false positives, and thus, because of the large number of patterns obtained, the user has to rank them by different statistical measures of interest computed under different hypothesis. An effort on developing exhaustive and optimal approaches (i.e., with guarantee to find all the patterns having the highest or demanded fitness values) for the discovery of patterns in biosequences has resulted in a number of algorithms to search for putative TFBS, e.g., [QWK82, WAG84, Sta89, SEVS95b, SV96b, BJVU98b, RF98a] (a systematized survey of main algorithmic ideas can be found in [BJEG98b]). In practice, they all require some *fitness measure* used as a ranking and/or a selection criterion to help the user to differentiate the true positive patterns from the false positive ones. Many

[6]I.e., correct and complete extractions (see discussion in Page 26).

[7]DNA sequences are composed of 4 nucleotides: adenine, guanine, thymine, cytosine. Therefore such sequences and patterns in them can be seen as strings over the alphabet of 4 symbols $\{A, C, G, T\}$.

different measures have been proposed, e.g., statistical significance [WAG84, RF98a], information content [Sta89, WFHW96], ratio of the score of a pattern in a positive data set divided by the score of the same pattern in a random data set [BJVU98b]. The approach of [DD07] takes one step further and, after having ranked the extracted patterns according to a measure of fitness, use the most significant ones as the seeds to build the motifs modelling the TFBSs (in the concrete, the position specific scoring matrices (PSSMs)).

Motivation

Having in mind the difficulties to model statistically the biological randomness, we propose to postpone the phase of significant pattern selection, based on a statistical measure, and to use beforehand the supplementary biological information to constraint the search and thereby reduce the number of extracted patterns. This additional information comes in the form of a second data set representing an *opposite biological situation*. To collect this information, the method starts with a classical operation used in molecular biology: the search for differentially expressed genes[8]. This allows to obtain two groups of genes from which one can derive two sets of promoters. To look for putative TFBSs regulating the overexpressed genes, we choose the first set (the promoters of the over-expressed genes) to be used as a positive set, and the second set as a negative one[9]. Then our method consists in finding the patterns occurring on at least $minFr$ promoters from the positive set and on at most $maxFr$ promoters from the negative set, where the parameter $minFr$ (resp. $maxFr$) is supposed to be a large (resp. small) frequency constraint threshold value. The originality of the proposed method w.r.t the other combinatorial algorithms, which allow to extract patterns from several data sets (e.g., SPEXS [BJVU98b] or DRIM [ELYY07]), is that the maximal frequency threshold is set explicitly. This is particularly interesting, when there is a clear semantic cut between a positive and negative data sets, and the negative data set has an opposite biological sense (presence/absence of a mutation; addition or not of a given drug, etc.), and does not just represent random background. Two kinds of patterns are handled by our method: patterns having exact matches in the sequences and patterns having approximate matches (through Hamming match function, see Definition 4.19 in Page 69). Interestingly, in both cases, the enrichment of the pattern discovery context, using a negative data set, reduces the size of the solution set by several orders of magnitude. Even then, the set of the extracted patterns remains large, and thus we developed a set of complementary solutions to help to tune the parameters in order to focus on a manageable and potentially interesting set of patterns. In the next step, we select

[8] It consists in comparing two biological situations, A and B, in order to obtain two groups of genes: one that is up-regulated, and the other one that is down-regulated, when going from A to B.

[9] Notice that to characterize the promoters of the repressed genes, one simply has to choose the repressed genes as the positive set.

the exceptional patterns according to the measure of *subtlety* [KP02b]: a pattern ϕ is considered to be subtle if we expect that some random patterns could occur at least as often as ϕ in the positive data set and at the same time no more often than ϕ in the negative data set. Then, we verify which patterns are known TFBSs. Identification of the TFs that can bind on the patterns specific to the positive data set can help to discover new regulators of the concerned biological process. Patterns that do not correspond to known TFBSs are equally interesting since they can be unknown elements of regulation.

7.1.2 Finding Signature Motifs

7.1.2.1 The Choice of The Solvers

To extract the exact matching patterns (EMP) we used our implementation of the `FAVST` solver [DD04] (see Section 3.2.2.2 in Page 51) and to extract the soft matching patterns (SMP) we used our generic solver's instance `Marguerite-H` (see Section 5.3.5 in Page 118). These solvers enable to evaluate arbitrary Boolean combinations of (anti-)monotonic constraints, including, correspondingly exact-frequency (see Definition 1.23 in Page 22) and Hamming soft-frequency constraints (see Definition 4.21 in Page 69). Among them, through a conjunction of a minimum frequency constraint on a positive data set and of a maximum frequency constraint on a negative data set, they perform differential extractions.

Other approach to perform differential extractions is to use only a minimum frequency constraint on a positive data set $\mathcal{D}_+$ and to filter the patterns according to a score, which is a ratio of data strings, containing a pattern in $\mathcal{D}_+$, divided by a ratio of sequences, containing a pattern in a negative data set $\mathcal{D}_-$ [BJVU98b]. These two approaches are not equivalent. The latter one is convenient, when $\mathcal{D}_-$ is a random data set, since it pick out the patterns that are overrepresented in $\mathcal{D}_+$ w.r.t. $\mathcal{D}_-$. However, when $\mathcal{D}_-$ represents an opposite biological situation, and we want to extract patterns that are not implied in a biological process of that situation, we need to explicitly push an upper bound for the pattern frequency in $\mathcal{D}_-$.

Notice that our solvers we use do not use a predefined alphabet, and can be used, for instance, on sequences containing extra symbols, like the symbol N to indicate undefined bases.

7.1.2.2 Taking into Account Biological Information in Combinatorial Pattern Extraction

This work is based on the hypothesis that the transforming activity of v-erbA arises from the repression of a set of genes and that at least some these genes share some regulating TFs, that are absent from most genes activated by v-erbA. The motivation underlying the development of the method presented here is to help to discover the TF that participate in the v-erbA-induced transformation process. A classical approach would consist of identifying the genes repressed by v-erbA and then extracting the putative TFBS that are the patterns shared by the promoter sequences of these genes. The problem is that a combinatorial pattern extraction in such a context results in a large solution set containing many false positives. It is then very hard to pick out manually true positives in such a plethora of extracted patterns. Our approach is to first refine the pattern extraction task by introducing a negative data set that represents the opposite biological situation and thereby reduce the number of false positives.

In order to find signature motifs for v-erbA target genes, we first created two sets of promoter sequences of differentially expressed genes:

- a data set denoted R, for the genes *repressed* by the v-erbA oncogene, composed of 29 promoters
- a data set denoted A, for the genes *activated* by of v-erbA, composed of 21 promoters.

Promoter sequences were extracted as previously described in [BKF$^+$07]. If the sequences are too short, or if we have a very small number of sequences then many TFBSs will be absent from the data, or poorly represented, and the random background itself will be underrepresented. As this can only degrade the result of differential extractions, we selected the largest data sets available. However, the data sets should not be too large, in the sense that they must remain specific to TFBSs locations and to biological situations. Thus the selected genes (29 in the positive data set and 21 in the negative one, out of the 110 differentially expressed genes) are chosen because there are known to play a role in the corresponding biological situations. Concerning the parts of the genes retained to form the sequences, we should avoid to incorporate in the data sets portions that are not likely to contain binding sites, thus we selected sequences composed of 3000 bp upstream and 1000 bp downstream, a portion known to be rich in TFBSs [WHA$^+$03], and thus are composed of 4000 bp.

Data sets R and A represent two biologically opposite situations of interest, and are used respectively as the positive and the negative data set. A priori interesting patterns are strings that have many (soft-)occurrences in the positive data set but only a few in the negative data set. We focus the search on putative TFBSs that could be used to regulate the transcription of the genes of the positive data set while they are not likely to have an important impact on the regulation of the genes of the other set[10]. To find such putatively interesting patterns we performed differential

[10]However, we keep in mind that a single TF might be both an activator in the positive set and

extractions of the fault-tolerant string patterns with Hamming match function[11], expressed by the following inductive query

$$\mathsf{MinSoftFr}_{\phi}^{H,S}(minFr, R, k) \ \wedge \ \mathsf{MaxSoftFr}_{\phi}^{H,S}(minFr, A, k) \ \wedge \ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen), \quad \text{where } k > 0.$$

We remind that the solutions to this inductive query are referred as soft matching patterns (SMPs). We also performed the differential extractions of exact matching patterns (EMPs) with exact string match function (see Definition 1.21 in Page 21), expressed by the following inductive query

$$\mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(minFr, R) \ \wedge \ \mathsf{MaxFr}_{\phi}^{\sqsubseteq,S}(maxFr, A) \ \mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen).$$

Both SMPs and EMPs are necessary: SMPs allow to gather the degenerated TFBSs while EMPs are dedicated to pick out the conserved ones.

Figure 7.1 shows the reduction in the number of patterns when using two data sets (positive and negative) instead of using the positive one only. Graph A (resp. B) gives the number of EMPs (resp. SMPs) satisfying only a minimum exact (resp. soft) frequency constraint in the positive data set R w.r.t. the number of EMPs (resp. SMPs) satisfying both an exact (resp. soft) frequency constraint in the positive data set R and a maximum exact (resp. soft) frequency constraint in the negative data set A. The plots represent the number of patterns of length from 5 to 11, extracted when the minimum frequency threshold $minFr$ varies from 1 to 29, and the maximum frequency threshold $maxFr$ is set to 7. In the case of SMPs, the allowed Hamming distance k is set to 1.

7.1.2.3 Selecting Patterns by a Measure of Twilight Zone Indicator

Even if refining the extraction context with a negative data set, corresponding to the opposite biological situation, reduces the number of extracted patterns up to several orders of magnitude, there are still too many of them to be verified manually. A classical approach is to associate to the patterns a measure of interest and select those having the most relevant measure value. In order to assess the significance of a pattern we used the notion of Twilight Zone (TZ) [KP02b], presented in Chapter 6, to build a Twilight Zone Indicator (TZI) (see Section 6.2.6 in Page 135). We remind that a TZ is a zone in the parameter space, where we are likely to obtain patterns produced by the random background. Let ϕ be a pattern of length $|\phi|$, occurring

inhibitor in the negative one (the role of a TF can be determined by the molecular context around its binding site), but in this case we can detect its influence only if it has a TFBS in the positive data set different from its TFBS in the negative data set.

[11]See Definition 4.19 in Page 69

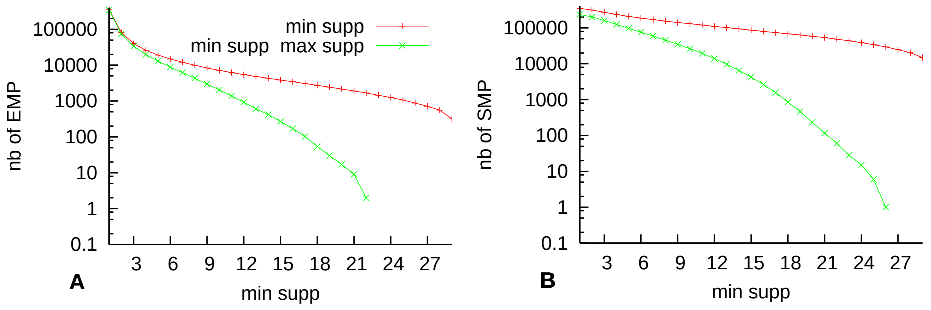

Figure 7.1: Reduction in the number of patterns when using two data sets (positive and negative) instead of using the positive one only

in $\mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_+, k)$ sequences of the positive data set and in $\mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_-, k)$ sequences of the negative data set. Then $TZI(\phi)$ is an estimate of the minimum number of patterns of length $|\phi|$ that will be extracted together with ϕ due to the random background. This minimum value is obtained in the most stringent conditions (i.e., with the most stringent constraint thresholds) that still lead to the extraction of ϕ. These conditions are obtained when we choose $minFr = \mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_+, k)$ and $maxFr = \mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_-, k)$. We remind that the computation of the TZI is based on the same hypothesis made in [KP02b]: the data strings are composed of independent and uniformly distributed nucleotides, and the possible overlapping of the occurrences of the patterns is considered to have a limited impact on the number of extracted patterns. In addition, we suppose that the positive and the negative data sets are independent.

An empirical validation of the TZI on the data sets R and A is given in Section 6.2.7 in Page 136.

7.1.2.4 Rising Patterns

Even if we use some domain knowledge to construct a positive and a negative data sets, a poor constraint setting can lead to the extraction of many patterns likely to be due to the random background. Therefore we want to choose the parameter values that focus on the patterns satisfying the most stringent constraints (i.e., having a large support on the positive data set and a small one on the negative data set). Such patterns, denoted *rising patterns*, are interesting, since they are exceptionally conserved in the context of positive and negative data sets, and thus might be there to accomplish a biological function.

To find these patterns, we use a parameter tuning method based on the following remark. When $minFr$ is very large and $maxFr$ is very small, we are likely to have no pattern satisfying the two constraints. Then if we decrease $minFr$ and/or increase $maxFr$ (i.e., weaken the constraints) we go towards points in the parameter space for which we will start to obtain some patterns. Consistently, if from one of these points we go on decreasing $minFr$ and/or increasing $maxFr$, we reach points in the parameter space for which we obtain more and more patterns satisfying the constraints. The parameter tuning method is based on the notion of *rising pattern* defined as follows. For a given value of $minFr$ we consider the minimal value of $maxFr$ such that we have at least one pattern ϕ, satisfying $\mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_+, k) \geq minFr$ and $\mathsf{SoftFr}_{H,S}(\phi, \mathcal{D}_-, k) \leq maxFr$. The patterns obtained for this $minFr$ and $maxFr$ values are the rising patterns, i.e., there is no pattern for lower $maxFr$ values, and for larger $minFr$ values we will have more patterns (or at least an equal number). Thus the rising patterns are located in the parameter space along a border that corresponds to the most stringent constraints that still lead to the extraction of at least one pattern.

Procedure to Find Rising Patterns To find these rising patterns and the corresponding parameter values we run an automated serialisation of extractions. Let T_{min} (resp. T_{max}) be a set of possible values for $minFr$ (resp. $maxFr$) ordered by increasing values. Finding the rising patterns in the parameter space $T_{min} \times T_{max}$, is performed as follows:

1. Let $minFr$ (resp. $maxFr$) be the first element in T_{min} (resp. T_{max}).
2. Let S_p be the set of patterns obtained when running an extraction with the constraint parameters $minFr$ and $maxFr$.
3. If S_p is empty and $maxFr$ is not the last element in T_{max} then set $maxFr$ to the next value in T_{max}. Goto step (2).
4. Output S_p as a set of rising patterns.
5. If $minFr$ is not the last element in T_{min} then set $minFr$ to the next value in T_{min}. Goto step (2).

It should be noticed that if the set S is empty for a conjunction of minimum and maximum frequency constraints with the given thresholds $minFr$ and $maxFr$, then S is also empty for this conjunctions with thresholds $minFr'$ and $maxFr$, where $minFr' \geq minFr$. The procedure avoids the test of such useless conjunctions.

A more formal way of finding rising patterns, would be to consider it as a multi-objective optimisation problem [Ste85]: maximizing $minFr$, minimizing $maxFr$, under the constraint $N > 0$, where N is the number of patterns satisfying the $minFr$ and $maxFr$ thresholds. However, in practice, a pure maximisation of $minFr$ is too restrictive, because a value of $minFr$, that is slightly smaller than an optimal one,

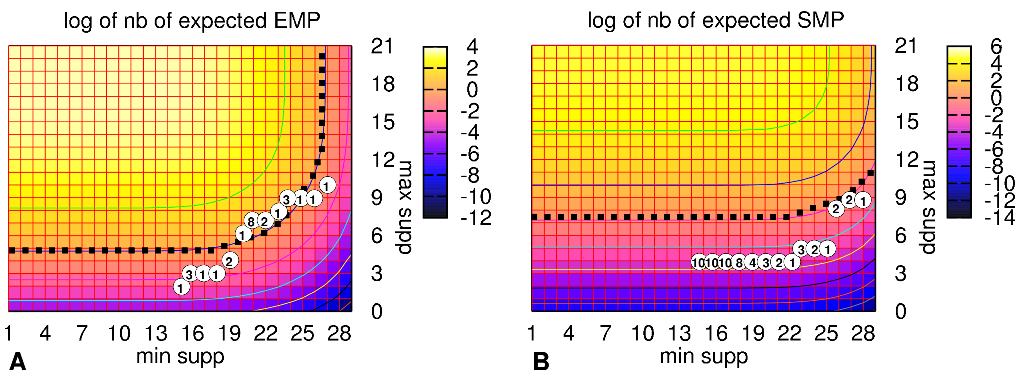

Figure 7.2: Number of rising patterns and number of expected random patterns

can lead to a few more patterns, and can also be interesting. Thus, we consider as rising patterns (in their definition and in the procedure to find them) the points in the parameter space that are the solutions of this optimisation problem (the Pareto optimal set), and also points that are suboptimal solutions (for each $minFr$ value we find the minimal $maxFr$ such that $N > 0$) and extract the patterns for all these points in the parameter space.

Rising Patterns in the Data Sets R and A The graphs A and B in Figure 7.2 give the numbers of rising EMPs and SMPs found in the data sets $\langle R, A\rangle$ (the exploration of the parameter is not made for $minFr$ values smaller than 15, since we are not likely to be interested in patterns occurring in less than 15 strings out of 29). For a point $\langle min\ supp, max\ supp\rangle$ in the parameter space the color in the background corresponds to the log_{10} of the number of expected random patterns (i.e., the log_{10} of the TZI value for $minFr = min\ supp$ and $maxFr = max\ supp$) in the data sets $\langle R, A\rangle$. The values in the white in graph A (resp. B) circles give the number of rising EMPs (resp. SMPs) that were extracted with the minimum exact-frequency (resp. Hamming soft-frequency) constraint threshold $minFr = min\ supp$ and maximum exact-frequency (resp. Hamming soft-frequency) constraint threshold $maxFr = max\ supp$. The dashed line indicates the border of the TZ. We observe that, consistently with the notion of TZI, the rising patterns are situated outside or in the very beginning of the estimated TZ.

The graph A in Figure 7.2 gives the numbers of rising EMPs of length 6 found in the data sets $\langle R, A\rangle$ and the graph B in Figure 7.2 gives the numbers of rising SMPs of length 8 with errors threshold $k = 1$ in the same data sets.

The graphs in Figure 7.2 give a global picture of the parameter space and are used to guide the setting of the parameters. For instance, if we are looking for an EMP with a high frequency in the positive data set (e.g., $minFr = 27$), we have to accept a rather high frequency in the negative data set ($maxFr = 10$). Or, on the contrary, we can use this graph to choose a moderate frequency in the positive data set (e.g., $minFr = 17$), and in this case we know that we can get some patterns having a low support ($maxFr = 4$) on the negative data set. Moreover, since this point ($minFr = 17, maxFr = 4$) in the parameter space is not in the TZ, we can decide to increase a little the $maxFr$ value and to run an extraction to try to get a few more patterns. Of course, in this case we will enter the beginning of the TZ, and thus, among the patterns that will be obtained, several of them are likely to be due to the random background.

The length of the patterns is also a parameter in itself, and in order to ease its setting we used `Marguerite-H` solver (see Section 5.3.5 in Page 118) that performs an exhaustive extraction within a range of length (from $minLen$ to $maxLen$). In practice, no pattern, or no interpretable pattern was found for a length out of the range $minLen = 6$ and $maxLen = 10$. Finally, the last parameter k (used for SMPs) should be kept as low as possible. When k increases, a pattern matches the occurrences that are more degenerated, and then is likely to be less specific to one of the two data sets and/or to be harder to interpret. In this study, the reasonable choice for k is limited to $k = 1$ or 2, and patterns that we could interpret, in a useful way, were found only for $k = 1$.

Locating the Rising Patterns Analytically It should be noticed that for the SMPs the increase of execution time[12] can, in some cases, prevent the possibility to run the extractions in a systematic way to explore the parameter space. In this situation we can still compute very efficiently the expected number of random patterns using the TZI and thus locate the TZ border. Then, the points along this border can be used as initial guess to find the rising patterns and their corresponding parameter settings. Moreover, the rising patterns are located along a rather regular contour, thus if the extraction time for SMPs is really too high, we can compute them only for a few $minFr$ values, and still have an idea of the whole curve.

7.1.2.5 Workflow of the Motif-Discovery Process

Finding signature motifs, characteristic of a given positive promoter set w.r.t. a negative set, is only the first step of the process. The diagram given in Figure 7.3 depicts the complete workflow ultimately designed to find putative TFBSs specific for a given promoter set.

[12]Due to the soft occurrences handling and to the decreased minimum frequency constraint selectivity because of the soft-matching [MB07].

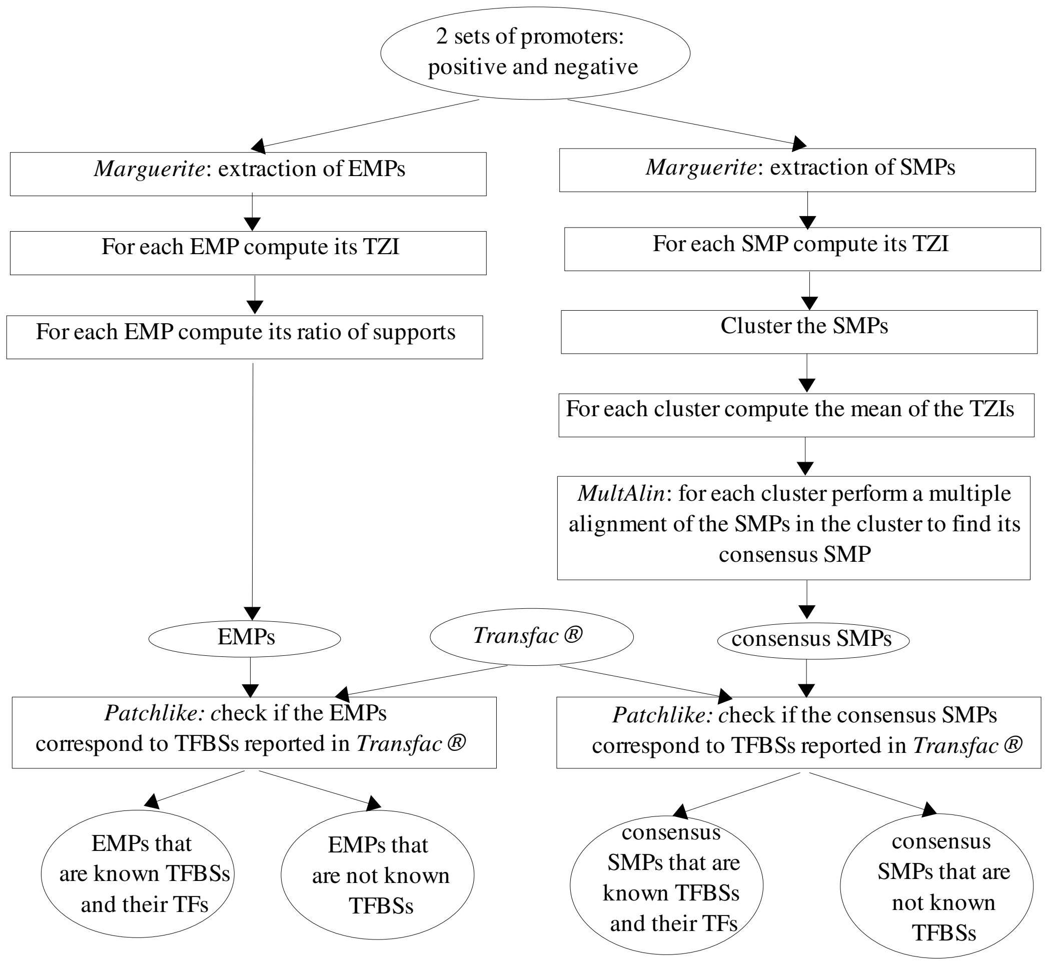
2 sets of promoters: positive and negative
Marguerite: extraction of EMPs
Marguerite: extraction of SMPs
For each EMP compute its TZI
For each SMP compute its TZI
For each EMP compute its ratio of supports
Cluster the SMPs
For each cluster compute the mean of the TZIs
MultAlin: for each cluster perform a multiple alignment of the SMPs in the cluster to find its consensus SMP
EMPs
Transfac ®
consensus SMPs
Patchlike: check if the EMPs correspond to TFBSs reported in Transfac ®
Patchlike: check if the consensus SMPs correspond to TFBSs reported in Transfac ®
EMPs that are known TFBSs and their TFs
EMPs that are not known TFBSs
consensus SMPs that are known TFBSs and their TFs
consensus SMPs that are not known TFBSs

Figure 7.3: Diagram depicting the steps of the whole motif-discovery process

EMPs and SMPs Extraction Using the solver `Marguerite-H` we extract EMPs (resp. SMPs), specific to the positive data set, i.e., all patterns satisfying a minimum exact-frequency (resp. Hamming soft-frequency) constraint in the positive data set and maximum exact-frequency (resp. Hamming soft-frequency) constraint in the negative one.

Measures of Interest for EMPs The TZI measure (see Section 6.2.6 in Page 135) is computed for every extracted EMP. For EMPs, we also compute as an additional measure the following ratio: the pattern frequency in the positive data set divided by the pattern frequency in the negative data set. The higher is the value of this ratio, the more specific to the positive data set is the pattern.

SMPs Clusters The TZI measure is also computed for every extracted SMPs. However, the number of extracted SMPs is much larger than the number of EMPs (see graphs H and D in Figure 6.1), and many SMPs are similar to other SMPs obtained at the extraction (due to the fault-tolerant matching). Thus, we grouped the similar SMPs by performing a hierarchical clustering of the patterns. The hierarchical clustering of the SMPs patterns is computed by the *hclust* function of the package *stat* of the *R* environment [r]. The proximity between clusters is computed using the complete linkage method. To construct a distance matrix we estimate the dissimilarity of each pair of SMPs as follows. For each pair $\langle \phi_1, \phi_2 \rangle$ we compute its optimal pairwise global alignment [NW70a] with the following parameters: the score for a mismatch is 1, the score for a match is 0, the insertions and deletions inside an alignment are not allowed, the terminal gaps are not penalized, and the length of an alignment (terminal gaps are not included in the alignment length) must be at least a half of the shortest pattern in the pair (i.e., must be of size greater or equal to $min(|\phi_1|, |\phi_1|)/2$). Then, the dissimilarity of the pair of SMPs is computed as the score of the best alignment divided by its length. To ameliorate the quality and efficiency of the clustering we process the SMPs by groups of patterns having the same length.

Measure of Interest of SMPs Clusters The measure of interest of each SMPs cluster is the average of the TZI of the patterns in the cluster.

Consensus of the SMPs Cluster To find the representative (consensus) pattern of each cluster of SMPs we align the patterns in each cluster using the multiple alignment tool *MultAlin* [Cor88]. We use the following alignment scoring parameters: gap creation and extension penalty is -5, terminal gaps are not penalized, score for a match is 2, and score for any mismatch is 0. Once a consensus SMP is computed we can use *Patchlike* or consult $Transfac^{®}$ to check whether it is a known TFBS. Figure 7.4 gives an example of a cluster whose consensus SMP is a binding site of the TF c-Myb-isomorf1.

EMPs and Consensus SMPs that are the known TFBSs Finally, for both EMPs and consensus SMPs, we use the tool `Patchlike` to check with respect to the $Transfac^{®}$ database, which patterns are known TFBSs. `Patchlike` is a tool, developed in our biologist collaborator team BM2A in the Center for Molecular and Cellular Genetics. It not only mimics[13] but also serializes the search that can be

[13]To analyse the biological sequences the BM2A team uses a *Macintosh* platform. The programs coming with $Transfac^{®}$ are platform dependent, and unfortunately there is neither compiled distribution for the *Macintosh* platform nor source code available. Therefore, to obtain the `Patch`-like functionality, the `Patchlike` tool was developed (it is written in *Perl* language).

```
                Alignment          TZI

                .CGGCCGTT...      23.94
                .GCGCCGTT...       0.68
                ...GCCGTTAT.       4.4
                ....CCGTTCGT       4.4
                ...GCCGTTCG.      23.75
                ....CCGTTAGG       0.68
                TTGGCCGT....      23.75
                ...GCCGTAAC.     107.37
                ..TGCCGTAA..       0.58
Consensus       ...gCCGTt...
Transfac:      c-Myb-isoform1
Mean of TZI:    21.06
```

Figure 7.4: Example of a cluster of SMPs and its consensus computed by a multiple alignment. The concensus corresponds to a binding site of the TF c-Myb-isomorf1. A base is weakly conserved if it is shared by at least 50% of the patterns and highly conserved if it is shared by at least 90% of the patterns. The bases that are highly conserved appear in red in the patterns and as uppercase letters in the consensus. Bases that are weakly conserved appear in green in the patterns and as lowercase letters in the consensus. Not conserved bases appear in black in the patterns. The positions with no conserved bases are indicated by dots in the consensus.

made with the tool `Patch`$^{\text{TM}}$[14] : `Patchlike` takes a collection of strings as input (in our case, they are the extracted patterns) and searches in these strings for the TFBSs, listed in the $Transfac^{®}$ data files. For a given string (pattern ϕ), `Patchlike` retains the TFBSs that are contained in ϕ (i.e., equal to ϕ or substrings of ϕ). In `Patchlike` we only vertebrate data files are used. The mismatches between an input string (pattern) and a TFBS are not allowed. In spite of that, the searches can result in a quite large number of TFBSs that might burden the analysis. Therefore we retrieve only the longest TFBS that are contained in ϕ. `Patchlike` considers an input sequence and its reverse complement to look for the TFBSs in forward and reverse direction.

We are particularly interested in patterns that are the binding sites of the TFs, involved in the v-erbA transforming activity. Untill this point, the extraction process does not rely upon any collection of known motifs. It is therefore obvious that some of the extracted motifs will not correspond to any known TFBS, either in $Transfac^{®}$ or in any other database. Those motifs can be qualified as putative unknown binding sites, and are candidates for experimental analysis.

7.1.3 Results

7.1.3.1 Patterns that are Putative Binding Sites of TFs Involved in v-erbA Transforming Activity

The particularity of our method to use both a positive and a negative data set, representing opposite biological situation, is a major reason why it would not be relevant to test our algorithm on a classical benchmark, such as [TLB+05] or [HK05], which uses only one set of sequences. Furthermore, as pointed by [DD07], we lack an absolute standard against which to measure the correctness of any motif-finding tool. Therefore, to assess our approach, we concentrated on the biological interpretation of some of the extracted patterns.

EMPs We first extracted rising EMPs (see Section 7.1.2.4in Page 160) of length from 5 to 10 within an interval of thresholds for minimum frequency from 15 to 29 (corresponding to a relative frequency ranging from 51.7% to 100% of the sequences) in data set R and an interval of thresholds for maximum frequency from 0 to 11 (corresponding to a relative frequency ranging from 0% to 52.4% of the sequences) in data set A. These intervals are rather large, and the worst case ($minFr = 15$ and $maxFr = 11$) is likely to lead to uninteresting patterns (not biologically founded).

[14]`Patch`$^{\text{TM}}$ is a tool integrated in $Transfac^{®}$ which identifies known TFBSs in a given string. One can verify whether the extracted patterns are known TFBSs by supplying them as input to $Patch^{\text{TM}}$. The database $Transfac^{®}$ is distributed in plain text files altogether with a graphic interface written in *Perl CGI* and *C* programs that implements various functionalities, including `Patch`$^{\text{TM}}$.

EMP	Supp in R	Ratio $\frac{\text{Supp in } R}{\text{Supp in } A}$	TZI	TFBS	TF
`GGAAACA`	18	6	0.02	`GGAAAC` (+)	Net
				`TGTTTC` (–)	AR, GR-alpha
`CGCTGCG`	17	5.67	0.09	`GCTGC` (+)	CTCF
`TGCAAAC`	17	5.67	0.09	`GTTTG` (–)	ZEB (1124 AA)
`CAGTTA`	19	4.75	0.1	`CAGTTA` (+)	c-Myb, c-Myb-isoform1
				`TAACTG` (–)	c-Myb
				`TAACT` (–)	RXR-alpha
`AGATAT`	17	4.25	0.2	`AGATAT` (+)	GATA-3/3 isoform 1/ 4/5A/5B/6A/6B
				`ATATCT` (–)	GATA-1/1 isoform 1/ 3/3 isoform-1
				`AGATA` (+)	GATA-1/3/4/6
				`TATCT` (–)	GATA-1

Table 7.1: Rising EMPs that are putative TFBSs bound by TFs involved in the v-erbA transforming activity. The sign indicates the direction of the match: (+) forward and (–) reverse complement.

However on data sets, containing an underlying structure, the procedure to find rising patterns does not reach such extreme cases, since rising patterns are obtained before, for more interesting $minFr$ and $maxFr$ values. This is actually the case for the data sets R and A, where, for the rising patterns, $minFr$ was always greater than $2.4 \times maxFr$. In these extractions we obtained 33 rising EMPs, for each of them we computed its TZI measure and its frequency ratio (i.e., frequency in data set R divided by its frequency in data set A), and looked at putative binding TFs with `Patchlike`. After visual inspection of these information, the biologist collaborators from the BM2A group selected five rising EMPs as candidates for further biological explorations, because they had a high frequency in the positive data sets R, a high frequency ratio, an interesting TZI value (i.e., low value) and meaningful putative binding TFs (Table 7.1).

These patterns were extracted for the following $(minFr, maxFr)$ pairs: (17,3), (18,3) and (19,4). Some other EMPs have quite high frequency ratio, high frequency in data set R, and low TZI value, but are not known TFBS in $Transfac^{®}$ (not shown). This is one of the benefit of such unsupervised approach to allow such unknown motif discovery. Since our knowledge of TFs-TFBSs relationship is still very incomplete, the best rated of those motifs could be used for functional assay using reporter gene transfection, and may lead to the discovery of new TFs, relevant for v-erbA-induced transformation.

Among the EMPs, displayed in Table 7.1, one of the most interesting for our biologist collaborators is $CAGTTA$, which is a known binding site for the transcription factor c-Myb. This pattern has a quite high frequency ratio (4.75), a high frequency in data set R (19 out of 29 promoters), and an interesting TZI value (0.1). Since this pattern appeared in a previous exploration of the same set of promoters, we had the opportunity to assess its putative relevance for the v-erbA-induced transformation process. The biologists indeed could demonstrate the existence of a functional interaction between v-erbA and c-myb [BKF+07], thereby demonstrating the biological relevance of this approach. Other patterns, given in Table 7.1, were also reported as interesting ones, since they can be expected knowing the molecular action of v-erbA (for detailed analysis see our paper [MRS+08]).

Take notice that a similar search can of course be performed using the A set of promoter sequences as the positive set. In this case, one should note that, although patterns are, by construction, specific of a given promoter set, TFs binding those patterns can appear on both data sets (of course, in this case, the *same* TF will bind *different* patterns in the two sets). The user can perform both searches using sequentially both promoter sets as the positive set, and then either focus only on set-specific TFs or on TFs shared by the two sets.

SMPs In the case of SMP, we estimated analytically (as described in Section 7.1.2.4 in Page 163) that with minimum Hamming soft-frequency threshold equal to 17 (it corresponds to 58.6% in relative frequency) on a data set R^* and maximum Hamming soft-frequency threshold equal to 10 (it corresponds to 47.6% in relative frequency) on a data set A^* the SMP of length between 7 and 11 are outside or in the beginning of the TZ. Thus we extracted the SMPs satisfying the length and Hamming soft-frequency constraints with these parameters in data sets R and A.

Table 7.2 gives five consensus SMPs that have the best mean of TZI and are known TFBSs. These consensus SMPs are issued from hierarchical clustering of the SMPs of length 8 using the complete linkage method and a cut-off level of 50% (see Paragraph *SMPs Clusters* in Page 165).

One can note an extensive similarity between the TFs binding to the SMPs and those binding to the EMPs. This concerns both the sites bound by nuclear receptors (RXR, RAR and GR) and those bound by GATA-1. Among the unexpected factors, the biologist collaborators found the HoxA9 homeogene. For the BM2A group it opens some several interesting hypothesis to examine (see our paper [MRS+08] for details).

Consensus SMP	Nb of SMP in cluster	Mean of TZI	TFBS	TF
`ta.cTaTg`	9	16.7	`TAACT` (+)	RXR-alpha
			`TATCT` (+)	GATA-1
			`AGATA` (–)	GATA-1/4/6
`TaGttag`	11	24	`TAACT` (–)	RXR-alpha
`aTagTg.t`	13	34.9	`AGTGGT` (+)	GR, GR-alpha
`t.TCAACt`	6	35.8	`TCAACT` (+)	CAR2:RXR-alpha CAR/PXR:RXR
			`AGTTGA` (–)	RAR-alpha1, RXR-alpha
			`CAACT` (+)	c-Myb-isoform1
`aCgTt.a`	17	36.9	`TGAACG` (–)	HOXA9
			`TAACG` (–)	c-Myb-isoform1
			`GTTCA` (+)	RAR-alpha1, T3R-alpha

Table 7.2: Consensus SMPs that are putative TFBSs bound by TFs involved in the v-erbA transforming activity. The sign indicates the direction of the match: (+) forward and (–) reverse complement.

7.1.3.2 Patterns that are Putative Binding Sites of TFs Involved in the Self-Renewal of Eryhroid Progenitors

In order to assess the generality of our approach, the biologist collaborator group BM2A also applied the presented method to a second data set, made from the promoters of genes showing differential expression, as assessed by SAGE, between self-renewing and differentiating erythroid progenitors [DKGG+04]. In this case we only applied the EMP-based strategy, in order to isolate TFBSs specific for the promoters of genes significantly more expressed in the self-renewal condition than after inducing differentiation. The data set was made of 28 promoter sequences in the positive set, denoted *AR*, and of 16 promoter sequences in the negative set, denoted *Diff* (promoters of genes whose the expression is up-regulated during the first 24 hours of differentiation [DKGG+04]). An exhaustive search for rising EMPs returned 55 different motifs. As previously described, patterns were selected for further analysis based upon : 1) their TZI value, 2) their frequency ratio in a positive and negative data sets, and 3) their *Transfac*® identification. This left us with 4 motifs (Table 7.3), the biological significance of them was further assessed by the BM2A group (see our paper [MRS+08]).

EMP	Supp in AR	Ratio $\frac{\text{Supp in } AR}{\text{Supp in } Diff}$	TZI	TF
CAGTTCT	16	5.3	0.41	c-Myb
CTGCTGG	21	3.5	.000042	c-Maf (long form)
ATGCAGC	17	5.7	0.078	CTCF
CACCCAC	15	7.5	1.05	EKLF

Table 7.3: Rising EMPs that are putative TFBSs bound by TFs involved in the self-renewal of normal erythroid progenitors.

7.1.4 Discussion

An important research effort has been dedicated to the extraction of motifs to find putative TFBSs, but even the best today techniques report limited results in practice [TLB+05, HK05, DD07]. These techniques, based on combinations of efficient extraction strategies together with dedicated statistical measures, often still suffer from high false positive rates and/or from the difficulty to select appropriated parameters. We have used a new method, that incorporates as a central aspect the use of background knowledge, in the extraction algorithm itself, to reduce the number of false positives, and that makes use of an effective parameter tuning strategy. To help the user to pick up the most promising patterns among the ones extracted, the whole process also includes the two following additional steps:

1. the computation of an interestingness measure based on the notion of the Twilight Zone [KP02b] (see Chapter 6 and in particular Section 6.2.6 in Page 135),
2. the automated retrieval of the TFBSs known in $Transfac^{®}$ (together with the corresponding TFs), for the TFBSs that are similar to the extracted patterns.

The current knowledge about TFs and TFBSs does not permit to determine the set of true positive (the set of TFBSs in a gene promoter sequence involved in the regulation of this gene), and thus we do not provide an estimate of the number of true positive TBFSs that may be missed by the method.

The major strength of the whole approach is that it does not only rely on a model of the random background to assess the interestingness of the patterns, but uses one data set in which the patterns are searched, and incorporates as background knowledge a second data set in which the patterns of interest are not likely to appear. Although it seems simple, this strategy is not supported by the state of the art techniques to find putative TFBSs. In fact, putting at work this strategy is not straightforward in practice, in particular we have to face a large parameter spaces (one frequency threshold for each of the two data sets, the size of the patterns, the degree of approximated matching allowed) and selecting appropriated parameter values is a difficult task that may even turn to be prohibitive. Thus, the second main point of the proposed method is to provide an explicit parameter tuning technique, that turns out to be effective in practice.

This approach was assessed by using two pairs of sets of promoters, each pair consisting in positive examples, and negative ones, derived from differential gene expression experiments. The results of this study provided our biologist collaborators with new hypothesis and insights in the self-regulation mechanism (they are detailed in our paper [MRS$^+$08]).

7.2 Starting an Application to Comparative Genomics

7.2.1 Background

The domain of comparative genomics concerns the relationship of genome structure and function across different biological species or strains. Informally, comparative genomics can be seen as an attempt to take advantage of the information provided by the signatures of selection to understand the function and evolutionary processes that act on genomes. One of the comparative genomics tools is to identify the common parts of genomes of the different biological species.

In this section we present an *opening* to the application of correct and complete fault-tolerant pattern extraction to the comparative genomic analysis. In the concrete, we considered the extraction of frequent $InsDels$ fault-tolerant patterns[15] that are common to the four species of yeast: *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus*. To extract such patterns we applied our generic solver `Marguerite-SoftFr`[16]. This work was carried out in a collaboration with the IQ (FP6-516169) project collaborator, biologist and computer scientist Dr. Ross D. King (Department of Computer Science, University of Aberystwyth).

7.2.2 Mining Yeasts Genomes

The task of finding frequent $InsDels$ fault-tolerant patterns that are common to the four yeats genomes *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus* can be expressed by the following inductive query:

$$IQ_{yeasts} = \mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, S.\ cerevisiae, maxIns, maxDels) \wedge$$
$$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, S.\ bayanus, maxIns, maxDels) \wedge$$

[15]See Section 5.3.2 in Page 105.
[16]See Section 5.3.3 in Page 108

Table 7.4: *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus* genomes data description

	Nb of seqs	Min Length	Max Length
S. cerevisiae	5306	360	23820
S. bayanus	4492	360	29400
S. mikatae	4525	480	39900
S. paradoxus	4788	420	99360

$$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, S.\ mikatae, maxIns, maxDels)\wedge$$
$$\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, S.\ paradoxus, maxIns, maxDels)\wedge$$
$$\mathsf{MinLength}_{\phi}(minLen) \wedge \mathsf{MaxLength}_{\phi}(maxLen)$$

We performed the extraction for various $minFr$ values. We extracted the patterns of length 6 and 7 (i.e., $minLen = 6$ and $maxLen = 7$): the shorter patterns tend to be an artifact and the extraction of longer patterns becomes costly in time efficiency. We estimated that for this pattern length the adequate fault-tolerance thresholds, establishing a symmetric similarity relation, are $maxIns = 1$ and $maxDels = 1$.

7.2.2.1 Data

We downloaded the genome sequences, composed of the Open Reading Frames[17] (ORFs) and the intergenic regions, of *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus* from Broad Institute of MIT and Harvard [18]. These genomes have sufficient sequence similarity and enough sequence divergence to recognize conserved functional elements [KPE$^+$03].

Table 7.4 gives the number of sequences, minimal and maximal sequence length for each such genome.

7.2.2.2 Number of Patterns in ORFs and Intergenic Regions

The graph in Figure 7.5 depicts the number of patterns satisfying the inductive query IQ_{yeasts} (see in Section 7.2.2 in Page 172), i.e., a minimum $InsDels$ soft-frequency constraint *in all four genomes* of *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus*, with varying minimum frequency threshold $minFr$. We separately extracted patterns in the ORFs and intergenic regions.

[17] An open reading frame is a portion of an organism's genome which contains a sequence of bases that could potentially encode a protein.

[18] http://www.broad.mit.edu/annotation/fungi/comp_yeasts/downloads.html

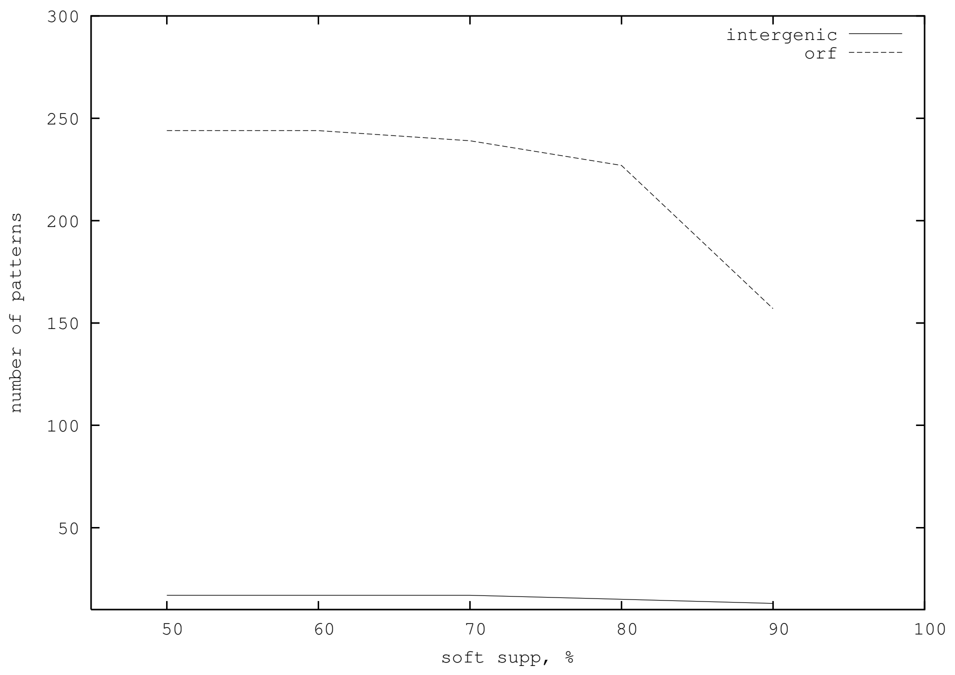

Figure 7.5: Number of patterns common to all four yeasts genomes with different soft-frequency thresholds

We observe that there are far less patterns in the intergenic regions than in the ORFs. This is biologically coherent since ORFs tend to be highly conserved, and intergenic regions are prone to mutations.

7.2.2.3 Distribution of Locations of Patterns

We refer the position in a genomic sequence, where pattern's soft-occurrence (i.e. a string that is in a *InsDels* similarity relation[19] with a pattern) as the *location* of a pattern. We recall that an occurrence that is equal to a pattern, called the exact-occurrence, is also its soft-occurrence (it is certainly similar enough to a pattern).

By a distribution of locations of patterns we mean the distribution of soft-occurrences of patterns in sequences. To study such distribution, we divide sequences into intervals of 20 nucleotides (symbols), and we count how many soft-occurrences of extracted patterns are placed in each interval. Take notice that if several soft-occurrences of the same pattern are placed in the same interval on the same sequence, we increment a count for that interval by one (and not by a number of soft-occurrences in question).

[19]See Definition 5.6 in Page 104

We remind that we extracted patterns that satisfy a minimum $InsDels$ soft-frequency constraint in all four yeasts genomes, and we want to estimate the distribution of locations of these patterns in sequences of *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus*.

Distribution of Locations in ORFs We divide each ORF into virtual intervals of 20 nucleotides, and we count the number of locations of patterns' soft-occurrences in such an interval.

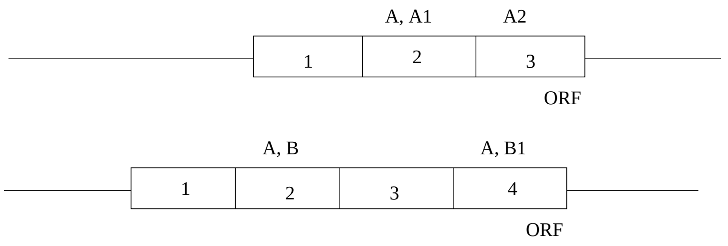

Figure 7.6: Distribution of locations of patterns in ORF

Example 7.1 *Assume that ϕ_1 and ϕ_2 are extracted patterns. Let A, A_1 and A_2 denote the soft-occurrences of ϕ_i (note that exact-occurrences are also soft-occurrences), and B, B_1 denote the soft-occurrences of ϕ_2. Consider Figure 7.6, which depicts the locations of ϕ_1 and ϕ_2 in the ORFs of two DNA sequences, divided into intervals. According to our approach to examine the distribution of locations of extracted patterns, one can see that in the first sequence interval numbered 1 has no locations, intervals 2, 3 have one location of pattern ϕ_1, and in the second sequence intervals 1, 3 have no locations, and intervals 2, 4 contain one location of ϕ_1 and one of ϕ_2. When counting a number of these locations for intervals, we get that interval 1 has no locations, interval 2 has three locations, interval 3 has one location, and interval 4 has two locations.*

The length of ORFs in DNA sequences of studied genomes is variable, and consequently the number of intervals of 20 nucleotides in each DNA sequence is also variable. Some sequences that have very long ORFs and thus many intervals, but there are very few of them (a number of sequences for a number of intervals is plotted in Figure 7.7).

To estimate the distribution of locations of extracted patterns we decided to look at intervals that are present in (supported by) at least 70% of sequences in corresponding genomes. The number of intervals and the number of sequences containing them is depicted in Figure 7.8.

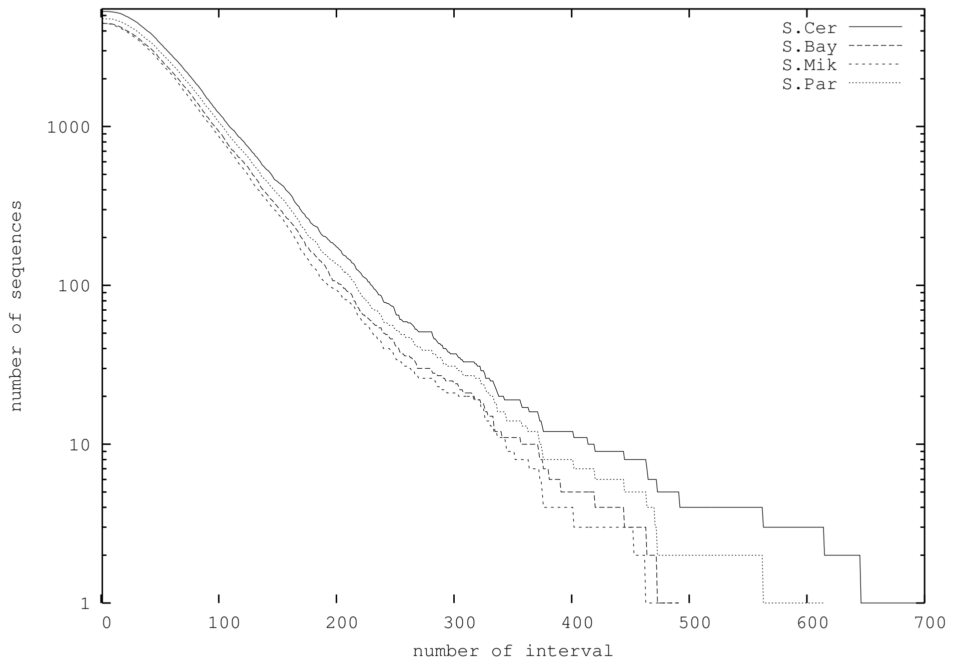

Figure 7.7: Number of sequences that contain a given number of ORF intervals

If the distribution of patterns' locations is strongly correlated with the corresponding curves given in Figure 7.8, then the number of patterns' locations in ORF intervals could be explained only by the number of sequences having these intervals in their ORFs. This would mean that there is any evidence whether the extracted patterns capture the regularities, common to the four genomes due to their evolutional relationship. To investigate this, we took the patterns, satisfying the minimum *InsDels* soft-frequency constraint in the four genomes with the *minFr* threshold of 70%, and computed the distribution of their locations. There are 239 such patterns of length $6 \leq l \leq 7$.

The number of patterns' locations present in the ORF intervals is plotted in Figure 7.9. We observe that the number of locations decrease constantly but not linearly when the interval goes away from the start of ORF. It is interesting that the peak of the number of locations is not in the very begining of the ORF, and after that peak we observe a quite deep decrease, while the number of sequences containing these parts of ORFs (Figure 7.8) decreases in a very slow and stable manner. The peaks alter the decreases during all the way of the curves of locations' number, and they all follow the same pattern of movement. One can also see that it is the curve of *S. paradoxus* that reproduces the most closely the behaviour of the curve of *S. cerevisiae*. This is encouraging since phylogenetically *S. cerevisiae* and *S. paradoxus*

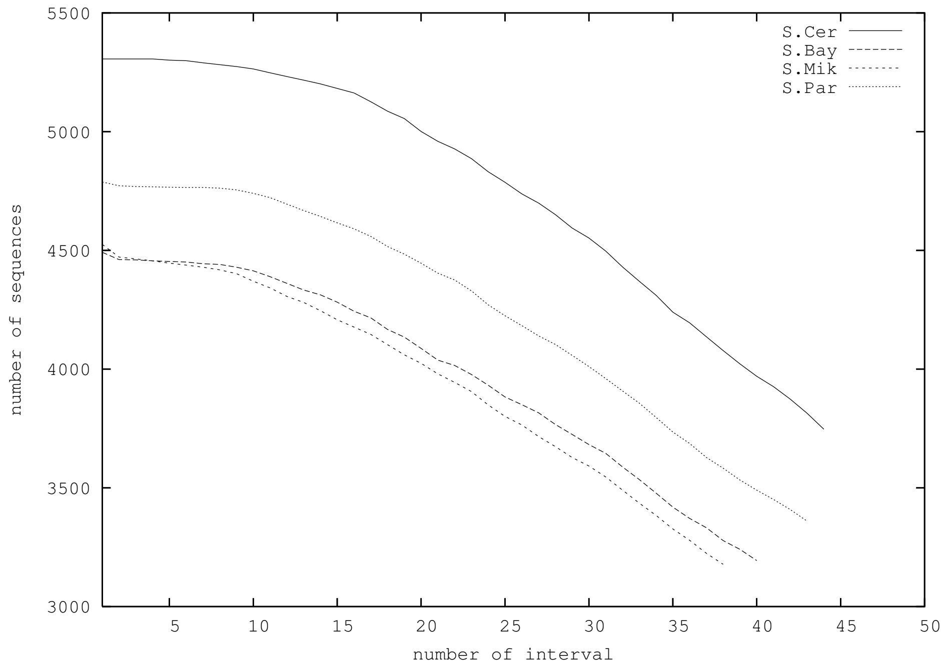

Figure 7.8: Number of sequences that contain a given number of ORF intervals that are present in at least 70% sequences

are most closely related [KPE+03]. Another most close relative of *S. cerevisiae* is *S. mikatae*. All these observations make part of arguments that distribution of extracted patterns is not uniform but reflects some biological reality.

Distribution of Locations in Intergenic Regions To estimate the distribution of locations of soft-occurrences of patterns in the intergenic regions we proceeded in a manner analogous to the one used for ORFs.

Intergenic regions were divided into intervals of 20 nucleotides. As intergenic regions are DNA parts that precedes and follows an ORF, to number them we consider that an ORF is a zero point, first 20 nucleotides preceeding an ORF constitutes an interval nb. -1, second 20 nucleotides preceeding an ORF constitutes an interval nb. -2, etc. ; first 20 nucleotides following an ORF constitutes an interval nb. 1, second 20 nucleotides following an ORF constitutes an interval nb. 2, etc. We count a number of locations of extracted patterns in these intervals.

Example 7.2 *Assume that ϕ_1 and ϕ_2 are extracted patterns. Let A, A_1 and A_2 denote the soft-occurrences of ϕ_i (note that exact-occurrences are also soft-occurrences),*

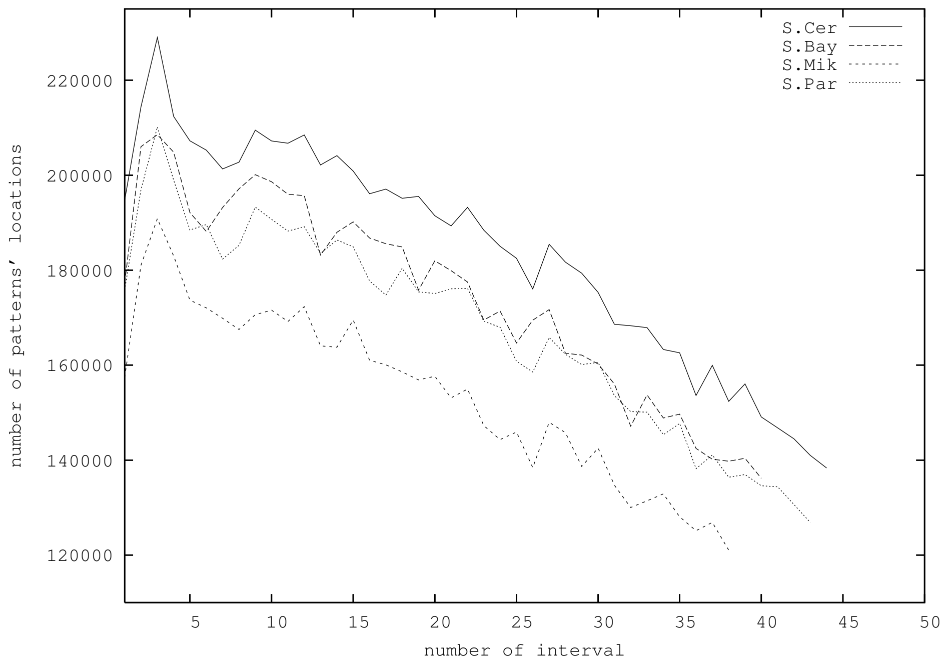

Figure 7.9: Number of locations present in a corresponding interval in ORF

and B, B_1 denote the soft-occurrences of ϕ_2. Consider Figure 7.10, which depicts the locations of ϕ_1 and ϕ_2 in the intergenic regions of two DNA sequences, divided into intervals. According to our approach to examine the distribution of locations of extracted patterns, one can see that in first sequence intervals numbered $3, -1, -3, -4$ *have no pattern locations, interval* 1 *has one location of ϕ_1 and one of ϕ_2, interval* 2 *has one location of ϕ_2, and interval* -2 *has one location of ϕ_1; and in the second sequence intervals numbered* $1, 3, -1$ *have no locations, interval* 2 *has one location of ϕ_1 and one of ϕ_2, and interval* -2 *has a location of ϕ_2. When counting a number of these locations for intervals, we get that intervals* $3, -1, -3, -4$ *have no locations, intervals* $1, -2$ *has two locations, and interval* 2 *has three locations.*

Again, as the length of ORFs in DNA sequences of studied genomes varies, the number of intervals of 20 nucleotides in the intergenic regions of different DNA sequences is also variable. Some sequences have very long intergenic regions and thus many intervals, but there are quite few of them (a number of sequences for a number of intervals is plotted in Figure 7.11).

To estimate the distribution of locations of extracted patterns we looked at the intervals that are present in (supported by) at least 70% of sequences in corresponding genomes. The numbered intervals and the number of sequences containing them is

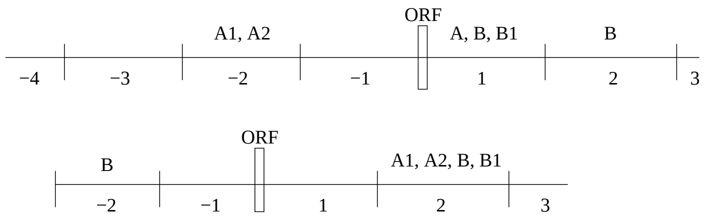

Figure 7.10: Distribution of locations of patterns in intergenic regions

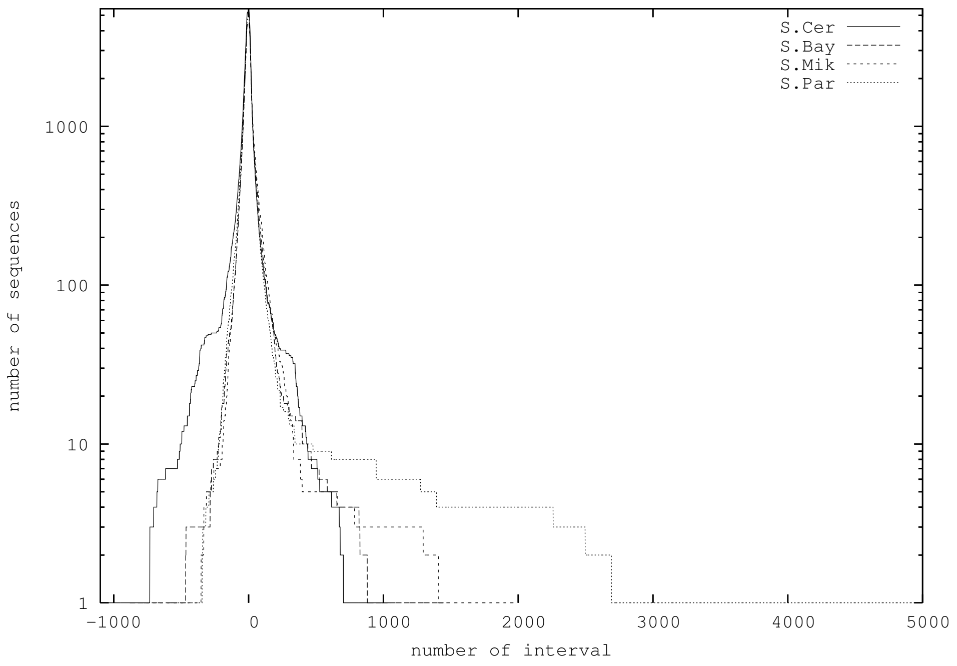

Figure 7.11: Number of sequences that contain a given number of intergenic intervals

depicted in Figure 7.12.

We took the patterns satisfying the minimum $InsDels$ soft-frequency constraint in the four genomes with the $minFr$ threshold of 70% to compute the distribution of their locations. There are 17 such patterns of length $6 \leq l \leq 7$. A number of patterns' locations present in intervals of intergenic region is plotted in Figure 7.13.

We observe that the distribution of patterns' locations in the intervals of intergenic regions can not be explained only by the number of sequences containing these intervals (see Figure 7.12). The largest peak of patterns's locations is at small distance from the start of an ORF, followed by the largest decrease just before an ORF

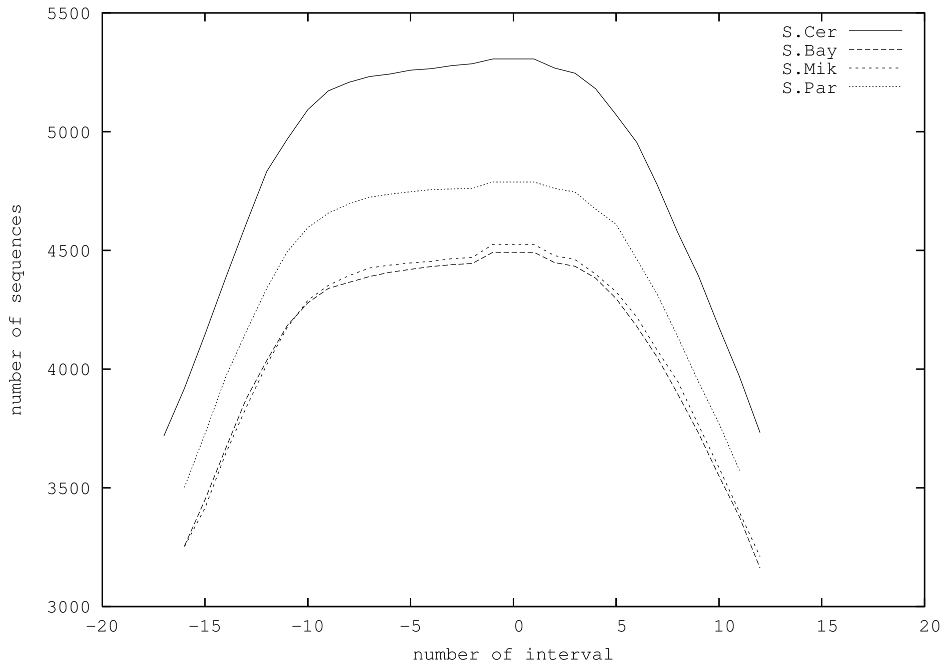

Figure 7.12: Number of sequences that contains a number of intergenic intervals that are present in at least 70% sequences

(note that the number of sequences increases in that part of intervals). The curves for different genomes follows the same pattern of movement. These observations suggest that the distribution of the extracted patterns is not uniform but reflects some biological reality.

7.2.3 Discussion

The obtained results suggest that the extracted patterns that are common and frequent in all four genomes of *S. cerevisiae*, *S. bayanus*, *S. mikatae* and *S. paradoxus* captures and reflects their conserved genome structure. Note that to assess this suggestion one should validate that these results is not the artifact of having divided the sequences into the intervals of 20 nucleotides.

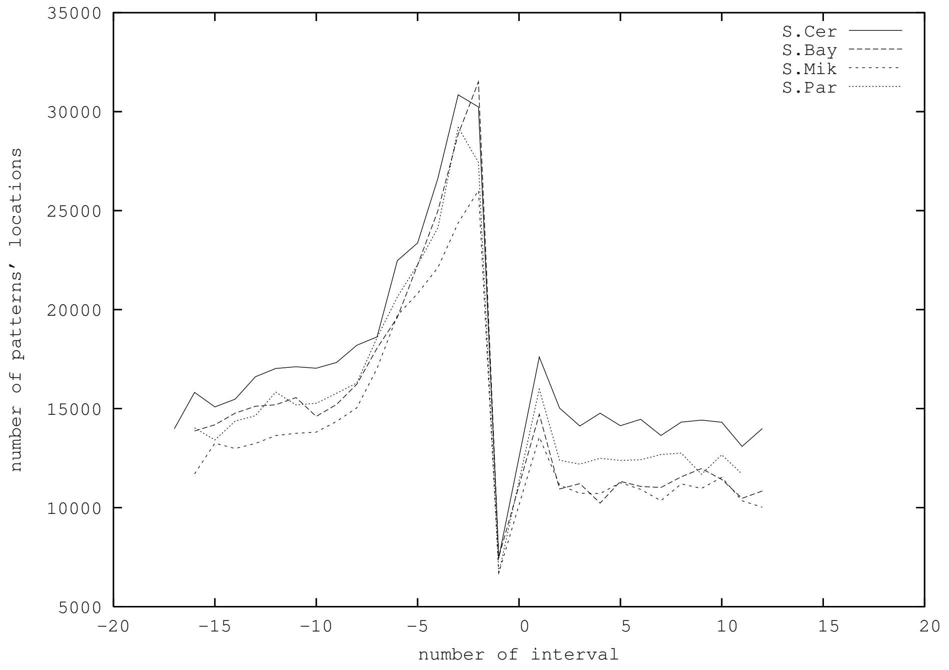

Figure 7.13: Number of locations present in a corresponding interval in the intergenic regions

Part IV

Conclusions and Perspectives

Conclusions

In this doctoral research work, we focused on Knowledge Data Discovery in string data with an application on promoter sequence analysis. The state-of-art in the beginning of our work could be resumed as

(a) recent advantages in inductive databases research have given rise to a generic solver capable to solve the inductive queries that are arbitrary Boolean combinations of anti-monotonic and monotonic constraints to mine generic patterns (i.e., the patterns from any pattern language) [DJDM02, DD03]; the instance of this generic solver is available to mine string patterns from string data sets [DJDM02, DD04];

(b) the research work in the domains of signal processing, error correction, text and information retrieval, and especially bioinformatics have given a number of approaches to represent and extract the fault-tolerant patterns in string data; the resulting solvers are *ad-hoc* solvers, i.e., they are dedicated to evaluated one tight combination of constraints;

(c) the generic solvers exploit the search space pruning strategies that are based on the property of (anti-)monotonicity; the fault-tolerant patterns rely on the similarity constraint, which is fundamentally neither anti-monotonic, nor monotonic, because of the non-transitivity of the similarity relation.

We decided to study the perspective of the generic solver to mining fault-tolerant patterns in string data. Subsequently we faced the problem of evaluating the interestingness of the extracted (fault-tolerant) string patters - the local pattern discovery should not overwhelm the domain experts with the huge collections of patterns containing predominantly irrelevant patterns. Irrelevancy must be understood in terms of both objective and subjective interestingness. We propose to evaluate it by estimating the expected number of (fault-tolerant) string patterns that will be extracted with the given extraction parameters. The main results of this doctoral research work concern these two research topics and can be summarized as following:

- Similarity and Soft-Frequency Constraints [MB06, MB07]

It is common that real life data contains errors due to technological issues concerning data collection, storage and transmission. In some application domains they may be also due to somewhat exploratory alphabet design. Also, data representing real world phenomenon is often intrinsically degenerated. To capture knowledge when working with such data, a fault-tolerance is needed.
The objective was to formulate similarity and soft-frequency constraints so that they are Boolean combinations of monotonic and anti-monotonic primitive constraints. We present two alternative ways of tackling fault-tolerance by means of (anti)-monotonic constraints using the notion of similarity.

We designed the generic solver `Marguerite-{Sim,SoftFr}` that employs the efficient generic strategies [DJDM02, DD03, DD04] for solving arbitrary combinations of similarity and/or soft-frequency constraints with other (anti)-monotonic constraints.

- Twilight Zone [BRMB08, MRS$^+$08]
Within the Inductive DataBase (IDB) framework, the constraints being an interesting medium to define the kind of local patterns we are looking for, we do not have much information about how to fix the constraint parameters. In other words, if the constraint thresholds setting is too lax then a huge collection of potentially irrelevant patterns is provided to the domain expert, and if the constraint thresholds setting is too stringent then no pattern is extracted. A common practice is to count the number of patterns obtained for a few different parameter settings to guess what could be the interesting parameter values for a deeper investigation. When considering a conjunction of primitive constraints giving rise to a large multidimensional parameter space, we cannot afford to run hundreds or thousands of experiments to probe such a space.
The objective is to estimate the expected number of string patterns that satisfy a conjunction of minimum and maximum (soft-)frequency constraints.
We proposed two approaches to estimate the expected number of patterns: (1) computing an analytical estimate of the expected number of extracted patterns from the features of the data set and (2) computing an analytical estimate of the expected number of extracted patterns from pattern space samples.

- Application to promoter sequence analysis [MRS$^+$08]
To understand the regulation of gene expression remains one of the major challenges in molecular biology. We collaborated with biologist of the BM2A group working on "Molecular Basis of Self-Renewal". Self-renewal is a characteristic property of stem cells. Therefore the BM2A team decided to identify v-erbA target genes responsible for the transformation process induced by v-erbA. We used these v-erbA data sets to extract the exact and soft-matching patterns under a differential extraction.
We selected the most exceptional patterns based on the developed measure of interest TZI. The biological evaluation of such patterns confirm their puta-

tive functional role and thereby exemplify the potential of our motif discovery method. The results of this work also provided our biologist collaborators with the new hypothesis and insights in the self-regulation mechanism).

Perspectives

We now provide some perspectives of these doctoral work.

Generic Solver to Mine the Structured Patterns

In this thesis work we considered the generic solver to mine fault-tolerant patterns, expressing the regularities composed of one element. In some cases the regularities can be compound of several such elements, separated by gaps (also referred as *non-conserved* regions). This is typically a case for the transcription factors, which often work in groups and thereby binds a number of sites to launch the expression of a gene. *Marie-France Sagot et al.* considered the structured fault-tolerant patterns language, allowing to capture such compound regularities and proposed an algorithm to extract such structured patterns under the minimum frequency constraint[20] [MS00a, MS00b]. The proposed algorithm being an ad-hoc solver, one can not use it solve a different combination of constraints, e.g., to mine the structured patterns satisfying the minimum frequency constraint in a positive data set and the maximum frequency constraint in a negative data set. One of the perspectives of this thesis is to search for the generic solver instances for other fault-tolerant pattern languages, e.g., the structured fault-tolerant patterns. The algorithm of [MS00a, MS00b] exploits the underlying suffix tree data structure, which is a close cousin of the VST data structure[21], employed by the generic solver instances `FAVST` and `Marguerite`. This holds a promise of the presumed success on this research perspective.

Tune for the Promoter Sequence Analysis

The close collaboration between the computer scientists TURING group in LIRIS and biologists group BM2A in CMCG opened a possibility to apply the obtained results in the domain of data mining to the biological sequence analysis, in the concrete,

[20]See Section 4.3.5 in Page 83

[21]See Section 3.2.1 in Page 46

to search the gene promoter sequences for the putative transcription factor binding sites (TFBS). The fault-tolerance offered by the developed *InsDels* pattern matching function, which concerns the operations of a symbol insertion and deletion, does not coincide well with the widely accepted and assessed model to represent the TFBS variability. The use of the Hamming match function was a good trade-off between the need to express the TFBS variability (where the symbol substitutions are much more common then the insertions and deletions) and the difficulty to handle the resulting fault-tolerance by a generic solver. We successfully applied the corresponding solver `Marguerite-H` to mine the promoter sequences and among the extracted fault-tolerant patterns found the binding sites of the factors that are known or expected to regulate the associated biological process.

Despite of this success, the generic solver that resulted from our contribution is a general-purpose extractor, and not a *tool* for the promoter sequence analysis. To tune these contributions for the identification of the regulatory elements in the promoter sequences is a perspective in its-own.

In bioinformatics, the concept of *consensus sequence* (also known as *consensus motif*) has been widely used to represent the specificity of the transcription factors [Sto00]. It is however arbitrary how a consensus sequence is defined. In general it refers to a sequence that matches all of the known (or supposed) binding sites closely, but not necessarily exactly. A consensus sequence that represents the binding sites of a transcription factor must account a trade-off between the number of allowed mismatches, the ambiguity in the consensus sequence, the sensitivity and precision of the representation [Sto00]. The so called *weight matrices* is an alternative way to represent the TFBS. In such a matrix there is a score for each nucleotide at every position of a binding site. Thereby, each particular binding site is given a score, which is the sum of the matrix values for each position in that site. Any binding site that differ from the consensus, implicitly represented by the weight matrix, will have a lower score, but the decrease depends on the differences in question. As remarks [Sto00], a consensus sequence can always be converted into a weight matrix, such that the same set of examples will be matched, but the converse is not true.

Take notice, that the *InsDels* fault-tolerant patterns that we proposed and the Hamming fault-tolerant patterns *are* consensus sequences. The difficulty to use them for the TFBS extraction resides in that fact the match function used in these consensus sequences

- does not correspond well the state of art hypothesis in biology how a transcription factor matches (or recognizes) its binding site;
- is different from the implicit match function used to represent (specify) the known binding sites by weight matrices in *Transfac*$^{®}$ database.

The match function that would better account the binding of transcription factors would be the one that uses the generalized edit distance[22], i.e., an edit distance with the weighted costs of substitutions, insertions and deletions. Neither the similarity nor the soft-frequency constraints based on such match function is neither anti-monotonic not monotonic. We did not find an equivalent expression of these constraints through a Boolean combination of (anti-)monotonic constraints. However one can search for the Boolean combination of (anti-)monotonic constraints that are the relaxations C'[23] of the constraints in question. Such relaxation can be soundly solved by the generic solver `Marguerite`. Our optimism regarding this research perspective is also motivated by the fact that the Galibot similarity constraint[24], based on a version of the generalized edit distance, where the operations are associated with rewarding scores instead of penalizing costs, which are multiplied instead of being summed, is convertible anti-monotonic [CBM02b].

Last but not least, the tool for the TFBS extraction must consider the question of the visualisation/representation of the extracted fault-tolerant patterns, or, so called, consensus sequences. For a biologist, assessing the interest of the fault-tolerant, it is indispensable to see what are the soft-occurrences of this pattern, if speaking in data mining terms, or what are the binding sites that match this consensus sequence, if speaking in bioinformatics terms. Among such representations possibilities are, for example the use of the standard set of ambiguity codes, each representing one character from their respective nucleotide subset (e.g., R stands for $\{A, G\}$), or converting the consensus sequence to weight matrix.

TZI to Evaluate the Cost of a Query Plan

The solution set $Th(\mathcal{L}, \mathcal{D}, C)$ to an inductive query with the constraint C that is a Boolean expression over the primitive constraints can be obtain by applying the set operations on the solution sets of the subconstraints of C. When the constraint C is an Boolean expression over the anti-monotonic and monotonic primitive constraints, the solution set $Th(\mathcal{L}, \mathcal{D}, C)$ is a generalized version space [DJDM02]. The generalized version spaces are closed under the set operations [DD03]. The resulting algebraic framework can be used for inductive query plan optimisation. Take notice that a single query can be solved using different execution plans. Consider, for example the inductive query of the form

$$IQ = (\ \mathcal{A}_1 \vee \mathcal{A}_2\ )\ \wedge \mathcal{M},$$

where $\mathcal{A}_1 = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(50\%, \mathcal{D}_1)$, $\mathcal{A}_2 = \mathsf{MinFr}_{\phi}^{\sqsubseteq,S}(80\%, \mathcal{D}_2)$ and $\mathcal{M} = \mathsf{MinLength}_{\phi}(6)$.

Among the possible execution plans are the following:

[22]See Definition 4.8 in Page 62

[23]See Paragraph Constraint Relaxation in Page 25

[24]See Section 4.2.3 in Page 66

- Using one call to a solver, compute the solution set $Th(\mathcal{L}_\Sigma, (\mathcal{D}_1, \mathcal{D}_2), IQ)$;
- Compute the sets $Th(\mathcal{L}_\Sigma, \mathcal{D}_1, \mathcal{A}_1)$, $Th(\mathcal{L}_\Sigma, \mathcal{D}_1, \mathcal{A}_2)$ and $Th(\mathcal{L}_\Sigma, -, \mathcal{M})$. Compute the union of $Th(\mathcal{L}_\Sigma, \mathcal{D}_1, \mathcal{A}_1)$ and $Th(\mathcal{L}_\Sigma, \mathcal{D}_1, \mathcal{A}_2)$, and intersect the obtained result with $Th(\mathcal{L}_\Sigma, -, \mathcal{M})$.
- Compute the set $Th(\mathcal{L}_\Sigma, (\mathcal{D}_1, \mathcal{D}_2), \mathcal{A}_1 \vee \mathcal{A}_2)$ and the set $Th(\mathcal{L}_\Sigma, -, \mathcal{M})$, and intersect them.
- Note that the inductive query IQ can be equivalently written as $(\mathcal{A}_1 \wedge \mathcal{M}) \vee (\mathcal{A}_2 \wedge \mathcal{M})$. Compute the set $Th(\mathcal{L}_\Sigma, \mathcal{D}_1, \mathcal{A}_1 \wedge \mathcal{M})$ and the set $Th(\mathcal{L}_\Sigma, \mathcal{D}_2, \mathcal{A}_2 \wedge \mathcal{M})$. Produce their union.

Generally, the different execution plans exhibit different efficiencies to solve the same inductive query. An inductive query optimizer is intended to generate different execution plans, estimate the cost of each such plan and choose the one with the minimal cost. This problem is similar to the query optimisation problem in relational databases.

This optimisation problem involves two research problems: (1) how to find a suitable execution plan in a reasonable amount of time, knowing that the number or semantically equivalent but syntactically different inductive queries (Boolean expressions over constraints[25]) increases exponentially with the number of the involved primitive constraints; and (2) how to estimate the cost of an execution plan. Concerning the latter one can, as observed in [DD03], an example of a cost function can be the product of the expected number of data scans and of the size of the data set. Another example of a cost function is a number of times, where a match function is evaluated. One can equally assume that a call to a conjunctive query $\mathcal{A} \wedge \mathcal{M}$[26] solver is unit and the only allowed set operation is union. Then the cost of an execution plan is the number of times, where a solver is invoked. The query plan optimisation problem under such assumption was solved in [DJDM02], which proposes the strategy of decomposing an the constraint of an inductive query into n disjuncts of the form $\mathcal{A} \wedge \mathcal{M}$, so that n is minimal.

The challenging question what is the best execution plan cost function (and subsequently what is the best optimisation strategy) is open. One must take into account that the queries can be related (especially in the interactive querying sessions [BP99]), and that the subqueries of a single query can be also related.

The latter case is extensively exploited by the SQL queries optimizers, which uses a mathematical model of query execution costs that relies heavily on estimates of the cardinality, or number of tuples, flowing through each edge in a query plan. The use of the expected number of patterns, satisfying the parts of an inductive query, can be

[25]A constraint is a synonym of predicate

[26]$\mathcal{A}$ denotes an anti-monotonic constraint and $\mathcal{M}$ denotes an monotonic constraint

also exploited to optimize an inductive query. For example, one can anticipate that, given an inductive query $C_1 \wedge C_2$, where the solution set $Th(\mathcal{L}, \mathcal{D}, C_1)$ is expected to contain 1000 patterns and the solution set $Th(\mathcal{L}, \mathcal{D}, C_2)$ is expected to contain 5 patterns, the execution plans of

- evaluate first C_1 and then, exploiting the obtained results, evaluate C_2; and
- evaluate first C_2 and then, exploiting the obtained results, evaluate C_1;

are not equivalent (and we expect the second one to be more efficient). The Twilight Zone Indicator, developed during this thesis research work[27], estimates the expected number of patterns, satisfying the subqueries, and can be exploited to evaluate the query plan cost and thereby to optimize the inductive queries.

[27]See Chapter 6

Bibliography

[AAL$^+$00] A. Amir, Y. Aumann, G. M. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. J. Algorithms, 37(2):247–266, 2000.

[AB03] H. Albert-Lorincz and J.-F. Boulicaut. Mining frequent sequential patterns under regular expressions: a highly adaptative strategy for pushing constraints. In Proceedings 3rd SIAM SDM'03, pages 316–320, San Francisco, CA, 2003.

[AG87] A. Apostolico and C. Guerra. The longest common subsequence problem revisited. Algorithmica, 2:315–336, 1987.

[AGM$^+$90] J. F. Altschul, W. Gish, W. Miller, E. W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. J. Mol. Biol., 215:403–410, 1990.

[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In SIGMOD, pages 207–216, Washington, D.C., USA, 1993. ACM Press.

[ALB03] H. Albert-Lorincz and J-F. Boulicaut. Mining frequent sequential patterns under regular expressions: a highly adaptative strategy for pushing constraints. In Proc. SIAM DM'03, pages 316–320, 2003.

[AP04] A. Apostolico and L. Parida. Incremental paradigms of motif discovery. J. Comp. Biol., 11(1):15–25, 2004.

[Apo97] 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.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, pages 487–499. Morgan Kaufmann, 1994.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In 11th International Conference on Data Engineering, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. In SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data, pages 85–93, Seattle, Washington, United States, 1998. ACM.

[Bay02] R. Bayardo. Constraints in data mining. Special issue of SIGKDD Explorations, 4(1):1–15, 2002.

[BBMM04] M. Botta, J-F. Boulicaut, C. Masson, and R. Meo. Query languages supporting descriptive rule mining: a comparative study. In Database Technologies for Data Mining - Discovering Knowledge with Inductive Queries, volume 2682 of LNCS, pages 27–56. Springer Verlag, 2004.

[BCF$^+$08] H. Blockeel, T. Calders, E. Fromont, B. Goethals, A. Prado, and C. Robardet. An inductive database prototype based on virtual mining views. In Proceedings ACM SIGKDD'08, pages 1061–1064. ACM Press, 2008.

[BCFY05] D. Burdick, M. Calimlim, J. Flannick, and T. Yiu. Mafia: A maximal frequent itemset algorithm. IEEE Trans. on Knowl. and Data Eng., 17(11):1490–1504, 2005. Member-Johannes Gehrke.

[BCKL02] D. Braga, A. Campi, M. Klemettinen, and P. Lanzi. Mining association rules from xml data. In Proceedings of the 4th International Conference on Data Warehousing and Knowlege Discovery, volume 2454 of LNCS, pages 133–156. Springer, 2002.

[BDM05] Jean-François Boulicaut, Luc De Raedt, and Heikki Mannila, editors. Constraint-Based Mining and Inductive Databases, volume 3848 of LNCS. Springer, 2005.

[BDM06] J-F. Boulicaut, L. De Raedt, and H. Mannila, editors. Constraint-based mining and inductive databases, volume 3848 of LNCS. Springer, 2006.

[BDMR90] D. L. Brutlag, J.-P. Dautricourt, S. Maulik, and J. Relph. Improved sensitivity of biological sequence database searches. Comput Appl Biosci., 6(3):237–245, 1990.

[BH03] M. R. Berthold and D. J. Hand, editors. Intelligent Data Analysis. Springer, 2003.

[BJ00] J-F. Boulicaut and B. Jeudy. Using constraint for itemset mining: should we prune or not? In Proceedings BDA'00, pages 221–237, Blois, F, 2000.

[BJEG98a] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. J. Comp. Biol., 5(2):277–304, 1998.

[BJEG98b] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. J. Comp. Biol., 5(2):277–304, 1998.

[BJUV96] A. Brazma, I. Jonassen, E. Ukkonen, and J. Vilo. Discovering patterns and subfamilies in biosequences. In Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology, pages 34–43. AAAI Press, 1996.

[BJVU98a] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. Genome Res., 8(11):1202–1215, 1998.

[BJVU98b] A. Brazma, Inge Jonassen, Jaak Vilo, and Esko Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. Genome Res., 8(11):1202–1215, 1998.

[BKF$^+$07] C. Bresson, C. Keime, C. Faure, Y. Letrillard, M. Barbado, S. Sanfilippo, N. Benhra, O. Gandrillon, and S. Gonin-Giraud. Large-scale analysis by SAGE revealed new mechanisms of v-erba oncogene action. BMC Genomics, 8(390), 2007.

[BKM98] J-F. Boulicaut, M. Klemettinen, and H. Mannila. Querying inductive databases: a case study on the mine rule operator. In Principles of Data Mining and Knowledge Discovery, volume 1510 of LNCS, pages 194–202, Nantes, France, 1998. Springer-Verlag.

[BKM99] J-F. Boulicaut, M. Klemettinen, and H. Mannila. Modeling kdd processes within the inductive database framework. In Data Warehousing and Knowledge Discovery (DaWak), volume 1676 of LNCS, pages 293–302, 1999.

[BM05] J-F. Boulicaut and C. Masson. Data mining query languages. In O. Maimon and L. Rokach, editors, The Data Mining and Knowledge Discovery Handbook, pages 715–727. Springer, 2005.

[BMS07] M. R. Berthold, K. Morik, and A. Siebes, editors. Parallel Universes and Local Patterns, volume 07181 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[BP99] E. Baralis and G. Psaila. Incremental refinement of mining queries. In DaWaK '99: Proceedings of the First International Conference on

Data Warehousing and Knowledge Discovery, pages 173–182. Springer-Verlag, 1999.

[BPB$^+$07] S. Blachon, R. Pensa, J. Besson, C. Robardet, J.-F. Boulicaut, and O. Gandrillon. Clustering formal concepts to discover biologically relevant knowledge from gene expression data. In Silico Biology, 7(0033):1–15, 2007.

[BRBR05] J. Besson, C. Robardet, J-F. Boulicaut, and S. Rome. Constraint-based concept mining and its application to microarray data analysis. Intelligent Data Analysis, 9(1):59–82, 2005.

[BRMB08] J. Besson, C. Rigotti, I. Mitasiunaite, and J.-F. Boulicaut. Parameter tuning for differential mining of string patterns. In ICDMW '08: Proceedings of the 2008 IEEE International Conference on Data Mining Workshops, pages 77–86. IEEE Computer Society, 2008.

[BS07] R. Bathoorn and A. Siebes. Finding composite episodes. In Mining Complex Data MCD'07 Revised Selected Papers, volume 4944 of LNCS, pages 157–168, 2007.

[Bul03] M. L. Bulyk. Computational prediction of transcription-factor binding site locations. Genome Biol., 5(201), 2003.

[BUV96] A. Brazma, E. Ukkonen, and J. Vilo. Discovering unbounded unions of regular pattern languages from positive examples (extended abstract). In ISAAC '96: Proceedings of the 7th International Symposium on Algorithms and Computation, pages 95–104, London, UK, 1996. Springer-Verlag.

[CBM02a] M. Capelle, J-F. Boulicaut, and C. Masson. Extraction de motifs séquentiels sous contrainte de similarité. In Actes EGC'02, pages 65–76, Montpellier (F), janvier 2002.

[CBM02b] 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.

[Cob94] A. L. Cobbs. Fast identification of approximately matching substrings. In CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, pages 64–74. Springer-Verlag, 1994.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377–387, 1970.

[Cor88] F. Corpet. Multiple sequence alignment with hierarchical clustering. Nucl. Acids Res., 16(22):10881–10890, 1988.

[CS04] M. Crochemore and M.-F. Sagot. Motifs in sequences: Localization and extraction. In A. K. Konopka and M. J. C. Crabbe, editors, Handbook of Computational Chemistry, pages 47–97. Marcel Dekker, New York, 2004.

[CZ06] Longbing Cao and Chengqi Zhang. Domain-driven actionable knowledge discovery in the real world. In Proceedings PAKDD'06, volume 3918 of LNCS, pages 821–830. Springer, 2006.

[Dam64] F. J. Damerau. A technique for computer detection and correction of spelling errors. Commun. ACM, 7(3):171–176, 1964.

[DD03] S. Dan Lee and L. De Raedt. An algebra for inductive query evaluation. In Proceedings IEEE ICDM'03, pages 147–154, 2003.

[DD04] 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.

[DD07] M. K. Das and H.-K. Dai. A survey of dna motif finding algorithms. BMC Bioinformatics, 8(Suppl 7), 2007.

[DFG$^+$97] G. Das, Ru. Fleischer, L. Gasieniec, D. Gunopulos, and J. Kärkkäinen. Episode matching. In CPM '97: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching, pages 12–27. Springer-Verlag, 1997.

[DJDM02] 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.

[DKGG$^+$04] F. Damiola, C. Keime, S. Gonin-Giraud, S. Dazy, and O. Gandrillon. Global transcription analysis of immature avian erythrocytic progenitors: from self-renewal to differentiation. Oncogene, 23:7628–7643, 2004.

[DLT94] D. D. Lopresti and A. Tomkins. On the searchability of electronic ink. In Proceedings of the 4th International Workshop on Frontiers in Handwriting Recognition, pages 156–165, Taipei, Taiwan, 1994.

[DSO78] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in protein. Atlas of Protein Sequences and Structure, 5:345–352, 1978.

[EG01] G. M. Edelman and J. A. Gally. Degeneracy and complexity in biological systems. Proc Natl Acad Sci USA, 98(24):13763–13768, 2001.

[EH88] A. Ehrenfeucht and D. Haussler. A new distance metric on strings computable in linear time. Discrete Appl. Math., 20(3):191–203, 1988.

[ELYY07] E. Eden, D. Lipson, S. Yogev, and Z. Yakhini. Discovering motifs in ranked lists of DNA sequences. PLOS Computational Biology, 3(3):508–522, 2007.

[FHK05] J. Fischer, V. Heun, and S. Kramer. Fast frequent string mining using suffix arrays. In ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining, pages 609–612. IEEE Computer Society, 2005.

[FHK06] J. Fischer, V. Heun, and S. Kramer. Optimal string mining under frequency constraints. In PKDD '2006, 10th European Conference on Principles and Practice of Knowledge Discovery, pages 139–150, 2006.

[FWV$^+$05] G. B. Fogel, D. G. Weekes, G. Varga, E. R. Dow, A. M. Craven, H. B. Harlow, E. W. Su, J. E. Onyia, and C. Su. A statistical analysis of the TRANSFAC database. Biosystems, 81(2):137–154, 2005.

[GEW85] D. Galas, M. Eggert, and M. Waterman. Rigorous pattern-recognition methods for dna sequences. analysis of promoter sequences from escherichia coli. J. Mol. Biol., 186(1):117–128, 1985.

[GGdB05] F. Geerts, B. Goethals, and J. Van den Bussche. Tight upper bounds on the number of candidate patterns. ACM Trans. on Database Systems, 30(2):333–363, 2005.

[GJP$^+$89] O. Gandrillon, P. Jurdic, B. Pain, C. Desbois, J. J. Madjar, M. G. Moscovici, C. Moscovoco, and J. Samarut. Expression of the v-erba product, an altered nuclear hormone receptor, is sufficient to transform erythrocytic cells in vitro. Cell, 58(1):115–121, 1989.

[GLW00] G. Grahne, L. V. S. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In ICDE '00: Proceedings of the 16th International Conference on Data Engineering, pages 512–521. IEEE Computer Society, 2000.

[GMMT07] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas. Assessing data mining results via swap randomization. ACM Trans. KDD, 1(3), 2007.

[GRS99] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. In VLDB'99: Proceedings of the 25th International Conference on Very Large Data Bases, pages 223–234. Morgan Kaufmann Publishers Inc., 1999.

[GSBS99] O. Gandrillon, U. Schmidt, H. Beug, and J. Samarut. Tgf-beta cooperates with tgf-alpha to induce the self-renewal of normal erythrocytic progenitors: evidence for an autocrine mechanism. EMBO J., 18(10):2764–2781, 1999.

[Ham50] R. Hamming. Error-detecting and error-correcting codes. Bell System Technical Journal, 29(2):147–160, 1950.

[Han02] D. J. Hand. Pattern detection and discovery. In D. J. Hand, N. M. Adams, and R. J. Bolton, editors, Pattern Detection and Discovery, volume 2447 of LNCS, pages 1–12. Springer, 2002.

[HFW$^+$96] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A data mining query lmeopsailaceri96vldbanguage for relational databases. In SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96), Montreal, Canada, 1996.

[HH92] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. Proc Natl Acad Sci USA, 89(22):10915–10919, 1992.

[Hir75] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. CACM, 18(6):341–343, 1975.

[Hir91] H. Hirsh. Theoretical underpinnings of version spaces. In Proc. of the 12th IJCAI, pages 665–670, Sidney, Australia, 1991.

[Hir94] H. Hirsh. Generalizing version spaces. Mach. Learn., 17(1):5–46, 1994.

[HK05] J. Hu and D. Kihara. Limitations and potentials of current motif discovery algorithms. Nucleis Acids Res, 33:4899–4913, 2005.

[HMS01a] D. Hand, H. Mannila, and P. Smyth, editors. Principles of Data Mining. MIT Press, 2001.

[HMS01b] D. J. Hand, H. Mannila, and P. Smyth. Principles of Data Mining, chapter Models and Patterns. MIT Press, 2001.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. SIGMOD Rec., 29(2):1–12, 2000.

[HWLT02] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top.k frequent closed patterns without minimum support. In ICDM '02: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02), page 211. IEEE Computer Society, 2002.

[IM96] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. Communications of the ACM, 39(11):58–64, November 1996.

[IV99] T. Imieliński and A. Virmani. MSQL: A query language for database mining. Data Mining and Knowledge Discovery, 3(4):373–408, 1999.

[Jeu02] B. Jeudy. Optimisation de requêtes inductives: Application à l'extraction sous contraintes de règles d'association. PhD thesis, Institut National des Sciences Appliquèes de Lyon, 2002.

[KBF$^+$00] R. Kohavi, C. Brodley, B. Frasca, L. Mason, and Z. Zheng. KDD-Cup 2000 organizers' report: Peeling the onion. SIGKDD Explorations, 2(2):86–98, 2000. http://www.ecn.purdue.edu/KDDCUP.

[KG85a] S. Karlin and G. Ghandour. Multiple-alphabet amino acid sequence comparisons of the immunoglobulin kappa-chain constant domain. Proc. Natl. Acad. Sci. USA, 82(24):8597–8601, 1985.

[KG85b] S. Karlin and G. Ghandour. The use of multiple alphabets in kappa-gene immunoglobulin dna sequence comparisons. EMBO. J., 4(5):1217–1223, 1985.

[KMR72] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing, pages 125–136. ACM, 1972.

[KP02a] U. Keich and P. A. Pevzner. Finding motifs in the twilight zone. Bioinformatics, 18(10):1374–1381, 2002.

[KP02b] U. Keich and P. A. Pevzner. Subtle motifs: defining the limits of motif finding algorithms. Bioinformatics, 18(10):1382–1390, 2002.

[KPE$^+$03] M. Kellis, N. Patterson, M. Endrizzi, B. Birren, and E. S. Lander. Sequencing and comparison of yeast species to identify genes and regulatory elements. Nature, 423:241–254, 2003.

[KRH01] S. Kramer, L. De Raedt, and C. Helma. Molecular feature mining in hiv data. In KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pages 136–143, San Francisco, California, 2001. ACM.

[KS95] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for the inversion distance between two permutations. Algorithmica, 13:180–210, 1995.

[LAC89] A. M. Landraud, J. F. Avril, and P. Chretienne. An algorithm for finding a common structure shared by a family of strings. IEEE Trans. Pattern Anal. Mach. Intell., 11(8):890–895, 1989.

[Lev65] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii Nauk SSSR, 163(4):845–848, 1965. English translation in Soviet Physics Doklady, 10(8):707-710, 1966. Doklady is Russian for "Report", sometimes transliterated in English as Doclady or Dokladi.

[LKPC97] J.-S. Lee, D. K. K., K. Park, and Y. Cho. Efficient algorithms for approximate string matching with swaps (extended abstract). In CPM '97: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching, pages 28–39. Springer-Verlag, 1997.

[LP81] H. R. Lewis and C. H. Papadimitriou. Elements of the Theory of Computation. Prentice Hall., Inc., 1981.

[LRS05] L. Lhote, F. Rioult, and A. Soulet. Average number of frequent (closed) patterns in bernouilli and markovian databases. In Proceedings IEEE ICDM'05, pages 713–716, 2005.

[LWS$^+$93] R. Lathrop, T. Webster, R. Smith, P. Winston, and T. Smith. Integrating AI with sequence analysis, pages 210–258. American Association for Artificial Intelligence, 1993.

[Mac67] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1, pages 281–297. University of California Press, Berkeley, CA, USA, 1967.

[Mas05] C. Masson. Contribution au cadre des bases de donnèes inductives : formalisation et èvaluation de scènarios d'extraction de connaissances. PhD thesis, Institut National des Sciences Appliquèes de Lyon, 2005.

[MB06] I. Mitasiunaite and J.-F. Boulicaut. Looking for monotonicity properties of a similarity constraint on sequences. In Proceedings of ACM SAC'06, Special Track on Data Mining, pages 546–552. ACM Press, 2006.

[MB07] I. Mitasiunaite and J.-F. Boulicaut. Introducing softness into inductive queries on string databases. In O. Vasilecas, J. Eder, and A. Caplinskas, editors, Databases and Information Systems IV, volume 155 of Frontiers in Artificial Intelligence and Applications, pages 117–132. IOS Press, 2007.

[MBS05] K. Morik, J-F. Boulicaut, and A. Siebes, editors. Local Pattern Detection, International Seminar Dagstuhl Castle Revised Selected Papers, volume 3539 of LNCS. Springer, 2005.

[MFG$^+$03] V. Matys, E. Fricke, R. Geffers, E. Gössling, M. Haubrock, R. Hehl, K. Hornischer, D. Karas, A. E. Kel, O. V. Kel-Margoulis, D.-U. Kloos, S. Land, B. Lewicki-Potapov, H. Michael, R. Münch, I. Reuter, S. Rotert, H. Saxel, M. Scheer, S. Thiele, E., and Wingender. Transfac : transcriptional regulation, from patterns to profiles. Nucl. Acids Res., 31(1):374–378, 2003.

[Mit82] T. M. Mitchell. Generalization as search. Artif. Intell., 18(2):203–226, 1982.

[MLK04] R. Meo, P. L. Lanzi, and M. Klemettinen, editors. Database Support for Data Mining Applications: Discovering Knowledge with Inductive Queries, volume 2682 of LNCS. Springer, 2004.

[MPC96] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases, pages 122–133, Bombay, India, 1996. Morgan Kaufmann Publishers Inc.

[MPC98] R. Meo, G. Psaila, and S. Ceri. An extension to SQL for mining association rules. Data Mining and Knowledge Discovery, 2(2):195–224, 1998.

[MR04] N. Méger and C. Rigotti. Constraint-based mining of episode rules and optimal window sizes. In Proceedings PKDD'04, volume 3202 of LNCS, pages 313–324. Springer, 2004.

[MRS+08] I. Mitasiunaite, C. Rigotti, S. Schicklin, L. Meyniel, J.-F. Boulicaut, and O. Gandrillon. Extracting signature motifs from promoter sets of differentially expressed genes. In Silico Biology, 8(43), 2008. Bioinformation Systems e.V.

[MS00a] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification. J. Comput. Biol., 7(3/4):345–360, 2000.

[MS00b] L. Marsan and M.-F. Sagot. Extracting structured motifs using a suffix tree—algorithms and application to promoter consensus identification. In RECOMB '00: Proceedings of the fourth annual international conference on Computational molecular biology, pages 210–219. ACM, 2000.

[MT97] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery, 1(3):241–258, 1997.

[MTP04] Florent Masseglia, Maguelonne Teisseire, and Pascal Poncelet. Extraction de motifs séquentiels. problèmes et méthodes. Ingénierie des Systèmes d'Information, 9(3-4):183–210, 2004.

[MTV97] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov., 1(3):259–289, 1997.

[MY60] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. IRE Transactions on Electronic Computers, 9(1):39–47, 1960.

[Nav01] G. Navarro. A guided tour to approximate string matching. ACM Computing Surveys, 33(1):31–88, 2001.

[NLHP98] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. SIGMOD Rec., 27(2):13–24, 1998.

[NR07] S. Nijssen and L. De Raedt. Iql: A proposal for an inductive query language. In Proceedings of the 5th International Workshop on Knowledge Discovery in Inductive Databases, volume 4747 of LNCS, pages 189–207. Springer, 2007.

[NW70a] S. Needleman and C. 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, 1970.

[NW70b] 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, 1970.

[OS06] J. Oncina and M. Sebban. Learning stochastic edit distance: application in handwritten character recognition. Pattern Recognition, 39(9):1555–1812, 2006.

[PAA03] J. Pelfrene, S. Abdeddaim, and J. Alexandre. Extracting approximate patterns (extended abstract). In Combinatorial Pattern Matching: 14th Annual Symposium, CPM 2003, pages 328–347. Springer-Verlag, 2003.

[PCGS02] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot. A basis for repeated motifs in pattern discovery and text mining. Technical report, IGM 2002-10, Institut Gaspard-Monge, Univ. of Marne-la-Vallée, 2002.

[PCGS03] N. Pisanti, M. Crochemore, R. Grossi, and M. Sagot. A basis of tiling motifs for generating repeated patterns and its complexity for higher quorum. In Math. Foundations of Computer Science (MFCS), pages 662–631. Springer-Verlag, 2003.

[PCGS05] N. Pisanti, M. Crochemore, R. Grossi, and M.-F. Sagot. Bases of motifs for generating repeated patterns with wild cards. IEEE/ACM Trans. Comput. Biol. Bioinformatics, 2(1):40–50, 2005.

[PH00] J. Pei and J. Han. Can we push more constraints into frequent pattern mining? In KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 350–354, Boston, Massachusetts, United States, 2000. ACM.

[PHL01] J. Pei, J. Han, and L. V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In Proceedings of the 17th International Conference on Data Engineering, pages 433–442. IEEE Computer Society, 2001.

[PHW02] J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management, pages 18–25, McLean, Virginia, USA, 2002. ACM.

[PP06] F. Piva and G. Principato. RanDNA : a random dna sequence generator. In Silico Biol., 6(3):253 – 258, 2006.

[PRF$^+$00] L. Parida, I. Rigoutsos, A. Floratos, D. Platt, and Y. Gao. Pattern discovery on character sets and real-valued data: linear bound on irredundant motifs and an efficient polynomial time algorithm. In SIAM Symposium on Discrete Algorithms (SODA), pages 297–308, 2000.

[PSF91] G. Piatetsky-Shapiro and W. J. Frawley, editors. Knowledge Discovery in Databases. AAAI/MIT Press, 1991.

[QWK82] C. Queen, M. N. Wegman, and L. J. Korn. Improvements to a program for DNA analysis: a procedure to find homologies among many sequences. Nucl. Acids Res., 1(10):449–456, 1982.

[r] The R project for statistical computing, http://www.r-project.org/.

[Rae02] L. De Raedt. A perspective on inductive databases. SIGKDD Explorations Newsletter, 4(2):69–77, 2002.

[RF98a] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: The teiresias algorithm. Bioinformatics, 14(1):55–67, 1998.

[RF98b] I. Rigoutsos and A. Floratos. Combinatorial pattern discovery in biological sequences: The teiresias algorithm. Bioinformatics, 14(1):55–67, 1998.

[RF98c] I. Rigoutsos and A. Floratos. Motif discovery without alignment or enumeration (extended abstract). In RECOMB '98: Proceedings of the second annual international conference on Computational molecular biology, pages 221–227. ACM, 1998.

[RFP$^+$00] I. Rigoutsos, A. Floratos, L. Parida, Y. Gao, and D.Platt. The emergence of pattern discovery techniques in computational biology. Metabolic Engineering, 2(3):159–177, 2000.

[RJBA99] Jr. R. J. Bayardo and R. Agrawal. Mining the most interesting rules. In KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 145–154, San Diego, California, United States, 1999. ACM.

[RK01] L. De Raedt and S. Kramer. The levelwise version space algorithm and its application to molecular fragment finding. In IJCAI, pages 853–862, 2001.

[RMB$^+$08] C. Rigotti, I. Mitasiunaité, J. Besson, L. Meyniel, J-F. Boulicaut, and O. Gandrillon. Using a solver over the string pattern domain to analyze gene promoter sequences. Technical report, LIRIS CNRS UMR 5205, F-69621 Villeurbanne, France, 2008. 20 pages. Chapter proposal for the IQ book. In Press.

[RMZ03] G. Ramesh, W. Maniatty, and M. J. Zaki. Feasible itemset distributions in data mining: theory and application. In Proceedings ACM PODS'03, pages 284–295, 2003.

[RS59] M. Rabin and D. Scott. Finite automata and their decision problems. IBM Journal of Research and Development, 3:114–125, 1959.

[SA96] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology, pages 3–17. Springer-Verlag, 1996.

[Sag00] M.-F. Sagot. Hope is the thing with feathers - combinatorial algorithms and molecular biology. Habilitation thesis, 2000.

[SEVS95a] M.-F. Sagot, V. Escalier, A. Viari, and H. Soldano. Searching for repeated words in a text allowing for mismatches and gaps. In Proceedings 2nd South American Workshop on String Processing, pages 87–100, Vinas del Mar, Chile, 1995.

[SEVS95b] M.-F. Sagot, V. Escalier, A. Viari, and H. Soldano. Searching for repeated words in a text allowing for mismatches and gaps. In Proceedings 2nd South American Workshop on String Processing, pages 87–100, Vinas del Mar, Chile, 1995.

[SP91] M. Sharif and M.L. Privalsky. v-erba oncogene function in neoplasia correlates with its ability to repress retinoic acid receptor action. Cell, 66(5):885–893, 1991.

[Sta89] R. Staden. Methods for discovering novel motifs in nucleic acid sequences. Comput. Applic. Biosci., 5(4):293–298, 1989.

[Ste85] R.E. Steuer. Multiple Criteria Optimization: Theory, Computation and Application. John Wiley & Sons, New York, NY, 1985.

[Sto00] G. D. Stormo. Dna binding sites: representation and discovery. Bioinformatics, 16(1):16–23, 2000.

[SV96a] M.-F. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. In CPM '96: Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching, pages 186–208. Springer-Verlag, 1996.

[SV96b] M-F. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. In Proceedings Combinatorial Pattern Matching '96, pages 186–208. Springer-Verlag, 1996.

[SVC95] H. Soldano, A. Viari, and M. Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. Pattern Recogn. Lett., 16(3):233–246, 1995.

[SVPS95] M.-F. Sagot, A. Viairi, J. Pothier, and H. Soldano. Finding flexible patterns in a text - an application to 3d molecular matching. Comput. Appl. Biosci., 11:59–70, 1995.

[SVS95] M.-F. Sagot, A. Viari, and H. Soldano. A distance-based block searching algorithm. In ISMB, pages 322–331, 1995.

[SVS97] M.-F. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison - a peptide matching approach. Theor. Comput. Sci., 180(1-2):115–137, 1997. This work was presented at the 6th Annual Symposium of Combinatorial Pattern Matching in 1995.

[SW03] M.-F. Sagot and Y. Wakabayashi. Pattern inference under many guises. In Recent advances in algorithms and combinatorics. Springer Verlag, 2003.

[Tei07] M. Teisseire. Mémoire Habilitation à Diriger des Recherches : Autour et Alentour des motifs séquentiels. PhD thesis, Université Montpellier 2, France, Dec 2007. 114 pages.

[Tho68] K. Thompson. Regular expression search algorithm. Communications of the ACM, 11(6):419–422, 1968.

[Tic84] W. F. Tichy. The string-to-string correction problem with block moves. ACM Trans. Comput. Syst., 2(4):309–321, 1984.

[TLB$^+$05] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transciption factor binding sites. Nat. Biotechnol., 23(1):137–144, 2005.

[TSK06] P-N. Tan, M. Steinbach, and V. Kumar, editors. Introduction to Data Mining. Addison-Wesley, 2006.

[TYH03] P. Tzvetkov, X. Yan, and J. Han. Tsp: Mining top-k closed sequential patterns. In ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining, page 347. IEEE Computer Society, 2003.

[TYH05] P. Tzvetkov, X. Yan, and J. Han. Tsp: Mining top-k closed sequential patterns. Knowl. Inf. Syst., 7(4):438–457, 2005.

[Ukk92] E. Ukkonen. Approximate string-matching with q-grams and maximal matches. Theor. Comput. Sci., 92(1):191–211, 1992.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. Algorithmica, 14(3):249–260, 1995.

[VBJ$^+$00] J. Vilo, A. Brazma, I. Jonassen, A. Robinson, and E. Ukkonen. Mining for putative regulatory elements in the yeast genome using gene expression data. In Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology, pages 384–394. AAAI Press, 2000.

[VEVK95] B. Vogelstein V. E. Velculescu, L. Zhang and K.W. Kinzler. Serial analysis of gene expression. Science, 270(5235):484–487, 1995.

[Vil98] J. Vilo. Discovering frequent patterns from strings (C-1998-9). Technical report, Department of Computer Science, University of Helsinki, 1998.

[Vil02] J. Vilo. Pattern Discovery from Biosequences. PhD thesis, University of Helsinki, 2002.

[VMS99a] A. Vanet, L. Marsan, and M.-F. Sagot. Promoter sequences and algorithmical methods for identifying them. Res. Microbiol., 150(9-10):779–799, 1999.

[VMS99b] A. Vanet, L. Marsan, and M.-F. Sagot. Promoter sequences and algorithmical methods for identifying them. Res. Microbiol., 150(9-10):779–799, 1999.

[WAG84] M. S. Waterman, R. Arratia, and D. J. Galas. Pattern recognition in several sequences: Consensus and alignment. Bull. Math. Biol., 46(4):515–527, 1984.

[Wei73] P. Weiner. Linear pattern matching algorithm. In 14th IEEE Symp. Switching and Automata Theory, pages 1–11, 1973.

[WFHW96] F. Wolferstetter, K. French, G. Herrmann, and T. Werner. Identification of functional elements in unaligned nucleic acid sequences by a novel tuple search algorithm. Comput. Applic. Biosci., 12(1):71–80, 1996.

[WHA$^+$03] G. A. Wray, M. W. Hahn, E. Abouheif, J. P. Balhoff, M. Pizer, M. V. Rockman, and L. A. Romano. The evolution of transcriptional regulation in eukaryotes. Mol. Biol. Evol., 20(9):1377–1419, 2003.

[Wil82] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival, editor, Ordered sets, pages 445–470. Reidel, Dordrecht, 1982.

[ZRS$^+$02] H. Zhang, Y. Ramanathan, P. Soteropoulos, M. Recce, and P. Tolias. Ez-retrieve: e web-server for batch retrieval of cooridnate-specific human dna sequences and underscoring putative transcription factor-binding sites. Nucleic Acids Res., 30(121), 2002.

Appendix A

Résumé en Français

A.1 Contexte de recherche

A.1.1 Bases de données inductives

Ces dernières années, la communauté d'ECD (Extraction de Connaissances à partir de Données) a beaucoup travaillé au développement d'algorithmes efficaces pour extraire différents types de motifs dans différents types de jeux de données. Malgré tout, l'étape d'extraction de motifs dans le processus d'analyse de données ou d'extraction de connaissances (KDD en anglais) [PSF91] ne doit pas se limiter à l'utilisation d'algorithmes qui produisent des collections de motifs ou des modèles globaux, solutions de la tâche d'extraction. Dans les applications réelles, le processus de découverte de connaissances dans les données est composé de nombreuses extractions réalisées de manière itérative et interactive qui traite non seulement de types de données différents (chaine, tableau booléen, graphe, etc.) mais aussi s'intéresse à des motifs et modèles différents. Par exemple, si l'on s'intéresse à la découverte de groupes de gènes sur-exprimés dans certaines situations biologiques, le processus ECD suivant pourra être mis en place:

- Collecte du profil d'expression des gènes dans différentes situations biologiques d'intérêt.
- Discrétisation des profils d'expression des gènes sous la forme d'une matrice booléenne, les colonnes représentant les gènes et les lignes les situations biologiques. Un "1" dans la matrice booléenne d'expression indique que le gène associé à la colonne est sur-exprimé dans la situation biologique associée à la ligne, "0" indiquant le cas contraire.
- Extraction des concepts formels [Wil82], c'est-à-dire les ensembles maximaux de

gènes (colonnes) qui sont sur-exprimés dans un ensemble maximal de situations biologiques (lignes). En d'autres termes, les rectangles maximaux de "1" avec permutations des lignes et des colonnes sont extraits de la matrice booléenne.

- Un clustering hiérarchique est réalisé sur les motifs ainsi obtenus.
- Un cluster particulier est sélectionné (i.e., un nøeud dans la hiérarchie).
- Afin d'évaluer la pertinence des associations ainsi trouvées, une visualisation en deux dimensions des résultats est réalisée.
- D'autres clusters peuvent ensuite être sélectionnés et évalués.

Pour étudier et assister de tels processus de découverte de connaissances, la théorie des Bases de Données Inductives (inductive databases) a été proposée dans [IM96]. Leurs développements ultérieurs sont issus des travaux suivants [BKM98, BKM99, MT97]. Une base de données inductive contient à la fois des données et des motifs. L'idée centrale est que les tâches de sélection, de manipulation et d'interrogation des données et des motifs peuvent être vues comme des requêtes et par conséquence dans ce cadre, un processus d'extraction de connaissances n'est autre qu'une série de requêtes. L'interrogation de données et de motifs peut être formulé par exemple sous la forme d'une requête SQL (Structured Query Language) et être traité par un gestionnaire de bases de données (DBMS). Au niveau des motifs, l'idée sous-jacente est de considérer une tâche d'extraction comme une requête inductive qui spécifie déclarativement les propriétés des motifs recherchés dans un espace de recherche donné (possiblement exprimé en intention, c'est-à-dire non matérialisé). Une spécification déclarative au lieu d'une déclaration constructiviste est très séduisante car elle permet d'envisager un algèbre pour la fouille de données.

La situation actuelle en fouille de données est très comparable à celle des bases de données avant la découverte de l'algèbre relationnelle de Codd en 1970. A cette période, une multitude de solutions existaient, chacune associée à un domaine d'application précis. La découverte d'un langage d'interrogation de bases de données inductives qui intégrerait les mécanismes d'interrogation et les primitives pour traiter simultanément des données et des motifs permettrait sans aucun doute de formaliser les processus d'extraction de connaissances sous la forme de requêtes qui satisfont la propriété de fermeture : chaque requête utilise et renvoie une instance de la base de données inductive. Ce serait un équivalent à la théorie de l'algèbre relationnelle en fouilles de données.

Cependant, la découverte d'un tel langage reste un objectif à long terme. Plusieurs langages de requêtes dédiés ont été proposés et implémentés comme `DMQL` [HFW$^+$96], `MSQL` [IV99], `MINE RULE` [MPC96, MPC98], `XMine` [BCKL02] (voir aussi [BBMM04, BM05, Mas05] pour un état de l'art plus complet). Une analyse plus complète du concept de base de données inductive à été récemment proposée dans [Rae02] conduisant à un nouveau langage de requêtes inductives appelé `IQL` [NR07].

Ces dernières années, des progres significatifs sur les bases de données inductives ont été réalisés au sein du contrat européen FET-IST cInQ (**c**onsortium on knowledge discovery by **In**ductive **Q**ueries). Mon travail de thèse a été réalisé dans le cadre du contrat européen FET-IST IQ (**I**nductive **Q**ueries) qui est la suite du contrat européen cInQ. Le but du contrat IQ est d'arriver à mieux comprendre et appréhender le cadre des requêtes inductives en fouille de données par le biais du développement d'un nombre significatif d'études de cas de requêtes inductives en bio-informatique et en essayant de développer la théorie sous-jacente: le développement de primitives pour l'extraction de motifs et de modèles et leur intégration dans un langage de requête inductive qui permet la découverte de connaissances dans des applications réelles.

A.1.2 Requêtes inductives

La tâche d'extraction de motifs ou de modèles peut être formalisée comme le calcul de l'ensemble $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$, c'est-à-dire l'extraction des motifs $\phi \in \mathcal{L}$ qui satisfont la contrainte C [MT97]. Ainsi, la contrainte C permet de spécifier déclarativement les motifs recherchés.

Le calcul de la collection $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$ est appelé une requête inductive avec la contrainte C. Comme les requêtes inductives sont souvent formulées par l'intermédiaire de contraintes, les requêtes inductives sont très liées au concept de fouille de données sous-contraintes [Bay02]. Le développement d'extracteurs efficaces est particulièrement important, car en général, l'espace des motifs $\mathcal{L}$ qui est souvent l'espace de recherche peut être très grand. De plus, le coût de l'évaluation des contraintes qui nécessite un accès à des données peut être très important.

Dans la pratique, toutes les requêtes inductives ne peuvent être évaluées. Dans la suite, nous présentons les impasses classiques en fouille de données et des solutions possibles pour y remédier.

Restriction des requêtes inductives Si l'on considère un ensemble d'items I_{20000} contenant 20 000 éléments, par exemple la collection des articles en français dans Wikipedia[1] à la fin de l'année 2003. Alors, la requête inductive qui demande tous les motifs $\phi \in \mathcal{L}_{\mathcal{I}_{20000}}$ contenant au moins 15 articles ne peut pas être résolue. Il est simplement inenvisageable de fournir (en extension) une ensemble contenant environ 10^{60} éléments[2]. Une telle requête inductive n'est pas assez sélective pour pouvoir être exploitée. Une solution possible pour rendre l'extraction faisable est d'utiliser une contrainte plus sélective, par exemple dans notre cas en demandant la présence de certains mots peu fréquents dans les pages wikipédia renvoyés ou en ne s'intéressant

[1]Wikipedia: http://www.wikipedia.org/

[2]En terme de comparaison, il y a 10^{50} atomes dans la planète Terre et 10^{57} atomes dans notre système solaire.

qu'aux pages qui ont été regardées au moins un certain nombre de fois (une contrainte de fréquence minimale).

Relaxation de contraintes Si une requête inductive ne peut pas être résolues efficacement car la contrainte associée ne possède pas les propriétés nécessaires qui permettent d'employer les techniques traditionnelles d'élagage d'espace de recherche, une solution peut être de faire une relaxation de contraintes. Cette idée peut paraitre assez surprenante et contre-intuitive et est en effet l'inverse de l'idée présentée dans le paragraphe précédent. Le principe est en ayant relâché la contrainte et trouvé une contrainte C' moins sélective que l'originale (i.e., $Th(\mathcal{L}, \mathcal{D}, C) \subset Th(\mathcal{L}, \mathcal{D}, C')$) mais qui possède par contre les propriétés d'élagage désirées, nous pouvons maintenant calculer la collection $Th(\mathcal{L}, \mathcal{D}, C')$ et réaliser un post-traitement pour obtenir la collection demandée. La famille des algorithmes SPIRIT [AS95, SA96] est un très bon exemple de l'utilisation de cette idée. Ils utilisent un mécanisme de relaxation de contraintes pour pouvoir exploiter des contraintes d'expression régulière pour des motifs séquence (sequence patterns).

Optimisation du plan d'exécution des requêtes L'efficacité pour résoudre une requête inductive, qui est une composition de plusieurs contraintes, dépend généralement de l'ordre dans lequel ces contraintes sont exploitées (pushed) et jusqu'à quelle profondeur dans l'arbre d'énumération elles sont utilisées. Une approche classique est de se baser sur les propriétés des contraintes pour choisir la stratégie appropriée. Par exemple, les différents algorithmes SPIRIT, suivant la contrainte obtenue C' après avoir relaxer C, exploitent prioritairement la contrainte de fréquence minimale ou C'. L'efficacité d'un algorithme SPIRIT particulier dépend de la sélectivité de C', information non-connue à priori. L'algorithme RE-Hackle [AB03] s'attaque à ce problème en mettant en place une stratégie d'élagage dynamique. Au sein d'une même extraction, suivant l'espace de recherche et la contrainte considérée, différentes stratégies d'élagage sont employées. La découverte d'un plan optimal d'évaluation d'une requête inductive est une tâche typique d'un système de gestion de bases de données inductives. C'est typiquement un problème d'optimisation similaire à celui de l'évaluation des requêtes en base de données. Ce problème prend encore plus d'importance lorsque l'on considère des sessions de requêtes inductives. Durant chaque session, l'utilisateur formule des requêtes afin d'obtenir des renseignements sur les motifs contenus dans les données. A partir des résultats obtenus, l'utilisateur va raffiner ces requêtes jusqu'à l'obtention du résultat désiré. Des mécanismes de "cache" pour les résultats intermédiaires pourraient améliorer sensiblement l'efficacité de l'évaluation d'une série de requêtes.

Optimisation locale Soit un ensemble d'items I et $\mathcal{L}_{part}$ un langage de motifs défini comme une partition disjointe de I. La requête inductive qui demande un motif

$\phi \in \mathcal{L}_{part}$ tel que la distance intra-cluster est minimale devient impossible à résoudre quand le nombre d'items atteint 10. Dans ce cas, des techniques d'optimisation locale sont employées comme par exemple avec le K-Means [Mac67] et le clustering hiérarchique, permettent de calculer un motif représentant des partitions qui sont proches de l'optimale. Il faut remarquer qu'il n'est pas possible de spécifier précisément la qualité des motifs extraits en utilisant les techniques d'optimisation locale. Dans la suite, les méthodes qui fournissent des motifs qui ne satisfont pas exactement les contraintes de la requête inductive ou seulement un sous-ensemble de la collection recherchée sont appelées heuristiques. Par opposition, les méthodes qui parviennent à fournir les collections exactes recherchées sont dénommées "correcte et complète". Elles ne peuvent bien évidement s'employer que dans les cas où les contraintes utilisées sont suffisamment sélectives.

L'avantage irréfutable des méthodes correctes et complètes est que les motifs extraits sont formellement définis.

A.1.3 Extraction sous-contraintes

Pour pouvoir calculer la collection $Th(\mathcal{L}, \mathcal{D}, C) = \{\phi \in \mathcal{L} \mid C_\phi(\mathcal{D})\}$, c'est-à-dire extraire l'ensemble des motifs satisfaisants une contrainte donnée, il est inenvisageable dans les cas réels d'imaginer générer l'ensemble des motifs possibles de l'espace des motifs puis de ne conserver que ceux qui satisfont la contrainte donnée. L'extraction de motifs sous-contraintes consiste à rechercher des stratégies efficaces d'évaluation de la contrainte par un parcours malin de l'espace de recherche permettant de n'explorer qu'un sous-espace de l'espace de recherche.

L'idée principale pour concevoir de tels parcours est d'exploiter la structure de l'espace (langage) des motifs, les types de contraintes et leurs propriétés, et la structure de l'espace de solution.

A.1.3.1 Structure de l'espace des motifs

Pour établir une structure dans l'espace des motifs, nous avons besoin de relations de *généralisation* et de *spécialisation* pour toutes les paires de motifs du langage des motifs.

Definition A.1 (Relation de généralisation et de spécialisation) *Soient $\mathcal{L}$ un langage de motifs, $\mathcal{U}$ un autre langage de motifs, ϕ et ψ deux motifs de $\mathcal{L}$, X un objet de $\mathcal{U}$ et $\mathcal{U} \times \mathcal{L} \rightarrow \{\mathsf{true}, \mathsf{false}\}$ une function. Le motif ϕ est dit plus général que le motif ψ (noté $\phi \succeq \psi$) si et seulement si $match(\psi, X) \Rightarrow match(\phi, X)$.*

La relation de généralisation $\succeq$ implique une structure dans l'espace des motifs $\mathcal{L}$. Les méthodes qui parviennent efficacement à calculer $Th(\mathcal{L}, \mathcal{D}, C)$ utilisent fortement cette structure. La relation de généralisation est un pré-ordre (une relation binaire réflexive et transitive). En revanche, cette relation n'est pas forcement anti-symétrique et donc n'est pas nécessairement un ordre partiel. Or, nous avons nécessairement besoin d'une relation d'ordre partiel comme relation de généralisation afin de pouvoir organiser l'espace des motifs de manière à pouvoir l'exploité efficacement. Pour cela, nous pouvons dériver une relation d'équivalence dans l'espace des motifs et utiliser l'ensemble dit "ensemble quotient" comme nouvel espace de motifs. A partir de cet ensemble, une relation de généralisation peut être construite qui est une relation d'ordre partiel.

Definition A.2 (Ensemble quotient) *Soit $\mathcal{L}$ un ensemble avec un pré-ordre $\succeq$. Nous pouvons définir une classe d'équivalence $\sim$ de la manière suivante :*

$$\phi \sim \psi \textit{ si et seulement si } \phi \succeq \psi \textit{ et } \psi \succeq \phi$$

$\phi/\sim$ représente la classe d'équivalence de ϕ et $\mathcal{L}/\sim$ celle de $\mathcal{L}$.

Definition A.3 (Relation de généralisation sur les ensembles quotient) *Nous pouvons définir une relation de généralisation $\succeq^{\sim}$ sur $\mathcal{L}/\sim$ de la manière suivante :*

$$\phi/\sim \succeq^{\sim} \psi/\sim \textit{ si et seulement si } \phi \succeq \psi.$$

La réflexivité et la transitivité de $\succeq^{\sim}$ proviennent de $\succeq$. Il faut noter que si $\phi/\sim \succeq^{\sim} \psi/\sim$ et $\psi/\sim \succeq^{\sim} \phi/\sim$ alors nous avons $\phi/\sim = \psi/\sim$. La relation d'ordre $\succeq^{\sim}$ ainsi obtenue est anti-symetrique. Ce qui veut dire que c'est une relation d'ordre partiel.

Ainsi à partir d'un langage de motifs $\mathcal{L}$ et une relation de généralisation $\succeq$ qui est un pré-ordre, nous pouvons toujours dériver un autre langage de motifs $\mathcal{L}/\sim$ avec la relation d'ordre partiel $\succeq^{\sim}$. Un ordre de généralisation qui est un ordre partiel permet d'arranger l'espace des motifs sous la forme d'un graphe orienté acyclique, structure très commode à parcourir et sur laquelle de nombreux mécanismes d'élagage peuvent être mises en place. Un treillis est un exemple d'une telle structure (voir Figure A.1.3.1).

A.1.3.2 Types de constraintes

Trois types de contraintes différentes peuvent être distingués: dépendante des données, syntactique et d'optimisation.

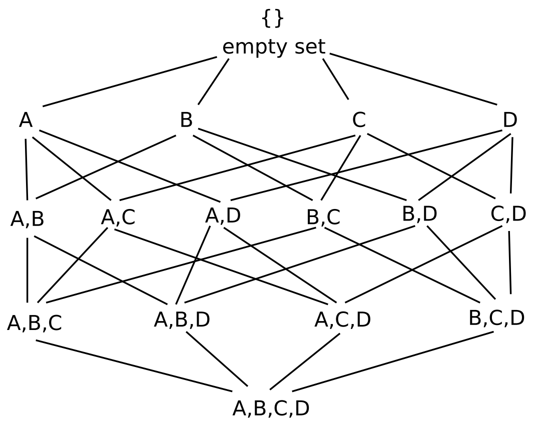

Figure A.1: Graphe partiellement ordonné qui représente un treillis

Contraintes dépendantes des données

Les contraintes qui nécessite un accès aux données sont appelées contraintes dépendantes des données.

Definition A.4 (Contraintes dépendantes des données) *Soient $\mathcal{L}$ un langage de motifs, ϕ un motif de $\mathcal{L}$ et C_ϕ une contrainte sur ϕ. Alors C_ϕ est dite indépendante des données si et seulement si l'évaluation de la contrainte implique l'accès à des données.*

Les contraintes de fréquence minimale et maximale sont des exemples classiques de contraintes dépendantes des données.

Contraintes syntaxiques

Par opposition aux contraintes dépendantes des données, les contraintes syntaxiques sont des contraintes qui nécessitent aucun accès aux données. Ces contraintes contrôlent la forme des motifs recherchés en réduisant le langage des motifs à explorer.

Definition A.5 (Contrainte syntaxique) *Soient $\mathcal{L}$ un langage de motifs, ϕ un motif de $\mathcal{L}$ et C_ϕ une contrainte sur ϕ. Alors C_ϕ est une contrainte syntaxique si et seulement si l'évaluation de la contrainte n'implique pas l'accès à des données.*

Example A.1 *Les contraintes de taille minimale et maximale et les contraintes d'expression régulière sont des exemples de contraintes syntaxiques.*

Contraintes d'optimisation

A chaque motif peut être associé une fonction d'évaluation qui définie la sémantique du motif [BKM99]. Les contraints qui retournent vraie pour les motifs qui ont une fonction d'évaluation optimale sont appelées contraintes d'optimisation.

Definition A.6 (Fonction d'évaluation) *Soient $\mathcal{D}$ un jeu de données, $\mathcal{L}$ un langage de motifs et ϕ un motif de $\mathcal{L}$. Soit r un ensemble valeurs. Une fonction d'évaluation $e(\mathcal{D}, \phi)$ met en relation chaque paire $(\mathcal{D}, \phi)$ avec un élément de r.*

Les mesures statistiques [WAG84, RF98a] et de contenu d'information [Sta89, WFHW96] sont des mesures d'évaluations très utilisées par exemple en biologie.

Definition A.7 (Contrainte d'optimisation) *Soient $\mathcal{L}$ un langage de motifs, ϕ un motif of $\mathcal{L}$, $\mathcal{D}$ un jeu de données et $e(\mathcal{D}, \phi)$ une fonction d'évaluation. Une contrainte C_ϕ est dîte contrainte d'optimisation si et seulement si elle est vraie quand $e(\mathcal{D}, \phi)$ a une valeur optimale.*

De nombreux algorithmes recherchent des modèles globaux qui satisfont une contrainte d'optimisation. Par exemple, une tâche de clustering consiste à trouver des groupes d'éléments (clusters) tels que la distance inter-groupe est maximale et la distance intra-groupe est minimale. Les classifieurs essayent de minimiser le taux d'erreur de prédiction. Dans la plupart des cas, des heuristiques sont employées pour résoudre de tels problèmes. Les contraintes d'optimisation sont difficiles à exploiter en extraction de motifs locaux et sont souvent réalisées en post-traitement. Les règles d'association et leurs différentes fonctions d'évaluation [AIS93], et les itemsets maximaux fréquents [BCFY05] sont des exceptions notables.

A.1.3.3 Propriétés des contraintes

L'idée derrière les stratégies d'élagage d'espaces de recherche est de pouvoir écarter sans perdre de bons candidats de grands espaces de recherche (espace de motifs). Cette élagage pour être réalisée doit être basée sur des propriétés formelles des contraintes assurant que dans telle portion de l'espace de recherche il est assuré qu'aucun bon candidat n'existe.

Contraintes anti-monotones et monotones

La classe des contraintes anti-monotones est l'une des plus facile à prendre en compte, et de nombreux algorithmes comme `A priori` [AIS93] exploitent cette propriété pour élaguer l'espace de recherche et rendre les extractions correctes et complètes, efficace.

Definition A.8 (Anti-monotonicité) *Soit $\mathcal{L}$ un langage de motif. Une contrainte C est anti-monotone si et seulement si pour tous les motifs $\phi, \psi \in \mathcal{L}$, nous avons si $\phi \succeq \psi$, alors $C_\psi \Rightarrow C_\phi$.*

L'élagage de l'espace de recherche est basée sur la contraposée de la définition précédente: Une contrainte C est anti-monotone si et seulement si pour tous les motifs $\phi \succeq \psi$, nous avons si C_ϕ est faux alors C_ψ est faux.

La monotonicité est la propriété duale de l'anti-monotonicité.

Definition A.9 (Anti-monotonicité) *Soit $\mathcal{L}$ un langage de motif. Une contrainte C est monotone si et seulement si pour tous les motifs $\phi \preceq \psi$, nous avons si $\phi \succeq \psi$, alors $C_\psi \Rightarrow C_\phi$.*

Contraintes succinctes

Les contraintes succinctes ont été définies sur les itemsets [NLHP98]. Ce sont des contraintes facilement exploitables pour lesquelles l'élagage peut être réalisé sans génération de candidats et sans accès au jeu de données. Ainsi, les contraintes succinctes sont des contraintes syntaxiques.

Definition A.10 (Succincte) *Soient I un ensemble d'items et $\mathcal{I}$ un itemset sur I. Soient $\mathcal{L}_I$ un langage de motifs et π_p un prédicat de sélection. $\pi_p(I)$ retourne les items qui satisfont le prédicat p. Alors,*

- *un itemset $\mathcal{I} \subseteq I$ est un ensemble succinct s'il peut être exprimé sous la forme de $\pi_p(I)$;*
- *$SP \subseteq 2^I$ est un ensemble des parties succints (succint powerset) s'il y a $n \in \mathbb{N}$ ensembles succints $\mathcal{I}_1, \ldots, \mathcal{I}_n \subseteq I$ tels que SP puisse être exprimé comme un ensemble des parties de $\mathcal{I}_1, \ldots, \mathcal{I}_n$ en utilisant les opérateurs ensemblistes union and moins;*
- *une contrainte C est une contrainte succincte si $Th(\mathcal{L}_I, C)$ est un ensemble des parties succinct.*

Contraintes convertibles

La notion de contraintes convertibles sur les itemsets a été introduite dans [PH00, PHL01]. L'idée est que pour une contrainte qui est ni monotone ni anti-monotone, on puisse ordonner les items tels qu'il existe une relation stable entre la satisfaction de la contrainte et les préfixes de l'itemset.

Definition A.11 (Préfixe) *Soient R un ordre total sur les items I et $m, n \in \mathbb{N}$ tels que $m \leq n$. On considère des itemsets $\mathcal{I}_1 = \{i_1, \dots, i_m\}$ et itemset $\mathcal{I}_2 = \{i_1, \dots, i_n\}$ tels que les items dans $\mathcal{I}_1$ et $\mathcal{I}_2$ sont triés suivant R. Alors, l'itemset $\mathcal{I}_1$ est appelé un préfixe de l'itemset $\mathcal{I}_2$ par rapport à R.*

Definition A.12 (Contrainte convertible anti-monotonique) *Soient $\mathcal{L}_I$ un langage de motifs et $\phi \in \mathcal{L}_I$ un itemset. Une contrainte C est convertible anti-monotonique si et seulement si il existe un ordre R sur les items tel que lorsqu'un itemset ϕ satisfait C alors tous les préfixes de ϕ satisfont aussi C.*

A.1.3.4 Structure de l'ensemble solution : Espace de version

Les espaces de version (Version spaces) [Hir91, Hir94, Mit82] et leurs frontières [MT97] sont traditionnellement utilisés pour caractériser l'espace de solution $Th(\mathcal{L}, \mathcal{D}, C)$ en fouille de données [MT97, Bay98] et en apprentissage [Hir91, Hir94, Mit82].

L'ensemble des motifs les plus spécifiques de $Th(\mathcal{L}, \mathcal{D}, C)$ est l'ensemble minimal couvrant. Il est appelé le S-set.

Definition A.13 (S-set) *Soit $\mathcal{L}$ un espace de motifs avec une relation de généralisation $\succeq$. Soit S un sous-ensemble de $\mathcal{L}$. Le S-set de S est défini par :*

$$\mathsf{S}(S) = \{\phi \in S | \nexists \psi \in \mathcal{L} : \phi \neq \psi \wedge \psi \preceq \phi\}$$

L'ensemble minimal des motifs les plus généraux qui couvre $Th(\mathcal{L}, \mathcal{D}, C)$ est appelé G-set.

Definition A.14 (G-set) *Soit $\mathcal{L}$ un espace de motifs avec une relation de généralisation $\succeq$. Soit S un sous-ensemble de $\mathcal{L}$. Le G-set de S est défini par :*

$$\mathsf{G}(S) = \{\phi \in S | \nexists \psi \in \mathcal{L} : \phi \neq \psi \wedge \psi \succeq \phi\}$$

Definition A.15 (Espace de version) *Soit $\mathcal{L}$ un espace de motifs avec une relation de généralisation $\succeq$. Soient S le S-set de l'ensemble S de $\mathcal{L}$ et G le G-set de S. Alors S est un espace de version définit par :*

$$\{\phi \in \mathcal{L} | \exists s \in \mathsf{S} \text{ et } \exists g \in \mathsf{G} : g \succeq \phi \succeq s\}$$

L'espace de version est employé pour représenté l'ensemble solution lié à une conjonction de contraintes monotones et anti-monotones. Un algorithme par niveau qui calcul les S-set et les G-set a été introduit dans [RK01, KRH01].

Definition A.16 (Espace de version généralisé) *Un sous-ensemble d'un espace de motifs avec une relation de généralisation qui peut être exprimé comme l'union d'espaces de version est appelé un espace de version généralisé.*

A.1.4 Un example de scénario ECD

Un scénario est la série des opérations à réaliser pour répondre à un objectif d'analyse. Nous dérivons maintenant un scénario réaliste pour identifier des sites de fixation putatifs.

1. L'utilisateur (e.g., un biologiste) dispose d'une collection de noms de gènes $\{G_1, \ldots, G_n\}$. Ainsi, appelons G_{1s} et G_{1d} (resp. G_{2s} et G_{2d}) l'ensemble des gènes régulés positivement (resp. négativement) dans deux situations biologiques.
2. L'utilisateur récupère les séquences promotrices de ces gènes dans le génome humain (disponible par exemple à partir du site de *UCSC Genome Bioinformatics* [3]). On nomme cette base de séquences Δ_1. Ensuite, l'utilisateur stocke les séquences de Δ_1 qui correspondent à chaque nom de G_{1s} et G_{1d} (resp. G_{2s} et G_{2d}), pour constituer les bases D_{1s} et D_{1d} (resp. D_{2s} et D_{2d}) des séquences promotrices des gènes régulés positivement (resp. négativement).
3. (a) L'utilisateur fait alors l'hypothèse que les motifs qui sont présents dans beaucoup de séquences de D_{1s} et qui ne sont pas présents dans beaucoup de séquences de D_{2s} (ou l'inverse), sont potentiellement impliqués dans la régulation génique liée au mécanisme biologique étudiée, et donc qu'ils sont a priori intéressants. On peut en effet faire l'hypothèse que de tels motifs représentent les sites de fixation putatifs pour des facteurs de transcription. Pour les extraire et les conserver respectivement dans M_1 et M_2, il/elle peut formuler les requêtes inductives suivantes : $\Re_1$: $M_1 = minfr(\phi, f1, D_{1s}) \wedge maxfr(\phi, f2, D_{2s})$
   $\Re_2$: $M_2 = minfr(\phi, f1, D_{2s}) \wedge maxfr(\phi, f2, D_{1s})$.
   Du point de vue biologique, les motifs extraits représentent des sites de fixation putatifs présents sur un brin noté + (sur les deux).

   (b) En sachant que l'ADN est composé de deux brins et qu'il y a des facteurs de transcription qui peuvent s'accrocher sur le brin $-$, l'utilisateur effectue l'opération du renversement et du complément RC sur ses données D_{1s} et D_{2s}. Il obtient alors deux nouveaux jeux de données : $D'_{1s} = RC(D_{1s})$ $D'_{2s} = RC(D_{2s})$
   Les requêtes inductives qui correspondent à $\Re_1$ et $\Re_2$ permettent ensuite de construire de nouvelles collections de motifs M'_1 et M'_2. On note aussi

[3]http://hgdownload.cse.ucsc.edu/downloads.html#human

que dans le cas où, par exemple, $f1 \neq f2$, il serait impossible de calculer $minfr(\phi, f2, D'_{1s})$ par post-traitement de $minfr(\phi, f1, D_{1s})$.

(c) Ensuite (probablement inspiré par l'analyse des résultats déjà obtenus) l'utilisateur peut vouloir identifier les sites de fixation putatifs pour les facteurs de transcription qui régulent positivement les gènes dans la première situation biologique mais pas dans la seconde.

La requête inductive correspondante est :

$\Re_3 : M_3 = minfr(\phi, f1, D_{1s}) \wedge maxfr(\phi, f2, D_{2s})$
$\wedge(maxfr(\phi, f3, D_{1d}) \vee minfr(\phi, f4, D_{2d}))$.

(d) Imaginons ensuite que l'utilisateur souhaite chercher des sites de fixation putatifs pour les facteurs régulent (positivement ou négativement) les gènes dans la première situation biologique mais pas dans la seconde. La requête inductive correspondante est $\Re_4 : M_4 =$
$(minfr(\phi, f1, D_{1s}) \wedge maxfr(\phi, f2, D_{2s}) \wedge (maxfr(\phi, f3, D_{1d}) \vee minfr(\phi, f4, D_{2d})))$
$\vee(minfr(\phi, f5, D_{2s}) \wedge maxfr(\phi, f6, D_{1s}) \wedge (maxfr(\phi, f7, D_{2d}) \vee minfr(\phi, f8, D_{1d})))$

4. L'utilisateur peut décider de poursuivre l'exploitation des motifs de M_1. Il/elle veut savoir quels sont les motifs de $M1$ qui sont des sites de fixation de facteurs de transcription connus. Pour faire cela, il peut compléter les données en construisant une nouvelle base de données Δ_2 qui contient les facteurs de transcription, les génes qu'ils régulent, les sites de fixation dans les séquences promotrices de ces gènes, les scores attribués à ces sites et leurs positions. Par exemple, Δ_2 peut êre construite en appliquant `EZ-Retrieve` [ZRS+02] à qui l'on fournit en entrée Δ_1. Pour chaque $\phi \in M_1$, on peut formuler des requêtes du type ≪sélectionner à partir de Δ_2 les facteurs de transcription, les gènes, et les positions dans les séquences promotrices qui s'accrochent au site de fixation ϕ avec le score a≫. C'est une requête tout-à-fait classique et qui, par exemple exprimée en SQL, peut être résolue par n'importe quelle Système de Gestion de Bases de Données. Désignons par tf_1 un facteur de transcription qui s'accroche à ϕ_1, considérons un motif ϕ_2 qui n'était pas reconnu comme site de fixation pour un facteur de transcription.

5. En formulant une requête dont l'argument est tf_1 sur la base de données publique des facteurs de transcription eucaryote $Transfac^{®}$ [MFG+03][4], l'utilisateur peut récupérer la matrice et la séquence *consensus* des sites de fixation de tf_1 (clairement, ϕ_1 est en accord avec cette séquence consensus). Puisque ϕ_1 était intéressant du fait de sa présence dans beaucoup de séquences de D_{1s} et dans peu de séquences de D_{2s} et que c'est le facteur de transcription tf_1 qui s'accroche à ϕ_1, cela peut suggérer que tf_1 est impliqué dans la régulation des phénomènes étudiés. Néanmoins, tf_1 peut s'accrocher aussi aux sites de fixation qui sont similaires à ϕ_1. La notion de cette ≪similaire≫ est donnée par la matrice ou

[4]http://www.gene-regulation.com/pub/databases.html#transfac

la séquence consensus que l'utilisateur a obtenu à partir de $Transfac^{®}$. En utilisant la séquence consensus, l'utilisateur peut par exemple formuler une contrainte de similarité à une expression régulière e, et, ainsi utiliser le solveur décrit dans [ALB03] pour évaluer les requêtes inductives :

$\Re_5 : M_5 = minfr(\phi, f1, D_{1s}) \wedge simexpreg(\phi, e)$

$\Re_6 : M_6 = minfr(\phi, f2, D_{2s}) \wedge simexpreg(\phi, e)$.

En fixant les paramètres $f1$, $f2$ et en calculant la somme des fréquences des motifs dans M_5 et la somme des fréquences des motifs dans M_6, l'utilisateur peut estimer si les motifs correspondants aux sites de fixation pour tf_1 sont effectivement présents dans beaucoup de séquences de D_{1s} et dans peu de séquences de D_{2s}. Le même parcours peut-être effectué en exploitant la similarité donnée par la matrice : le solveur décrit dans [CBM02a] prend comme argument une contrainte de similarité *simmat* et des matrices m (définies à partir de celles fournies par $Transfac^{®}$) et ainsi calculer les solutions aux requêtes suivantes :

$\Re_7 : M_7 = minfr(\phi, f1, D_{1s}) \wedge simmat(\phi, m)$

$\Re_8 : M_8 = minfr(\phi, f1, D_{2s}) \wedge simmat(\phi, m)$.

6. L'utilisateur peut maintenant reprendre les analyses décrites à l'étape 4 du scénario et s'intéresser à un motif ϕ_2. Supposons qu'il/elle fasse l'hypothèse qu'il s'agisse d'un site de fixation putatif inconnu. On rappelle que ϕ_2 est intéressant parce qu'il est présent dans beaucoup de séquences de D_{1s} et peu présent dans les séquences de D_{2s}. Comme dans l'étape 5, le but est de savoir si des motifs similaires à ϕ_2 sont également présents dans beaucoup de séquences de D_{1s} et peu présents dans des séquences de D_{2s}. L'utilisateur peut donc calculer les solutions des requêtes suivantes :

   $\Re_9 : M_9 = minfr(\phi, f1, D_{1s}) \wedge sim(\phi, \phi_2, t, h, l)$

   $\Re_{10} : M_{10} = minfr(\phi, f2, D_{1s}) \wedge sim(\phi, \phi_2, t, h, l)$.

   En fixant les paramètres $f1, f2, t, h, l$ et en calculant la somme des fréquences des motifs dans M_9 et la somme des fréquences des motifs dans M_{10}, il/elle peut estimer si la propriété intéressante de ϕ_2 tient pour les motifs similaires à ϕ_2. Ces connaissances peuvent être une fois encore enrichies par l'analyse de résultats de la requête suivante :

   $\Re_{11} : M_{11} = minfr(\phi, f1, D_{1s}) \wedge maxfr(\phi, f1, D_{2s}) \wedge sim(\phi, \phi_2, t, h, l)$.

7. Considérons maintenant que les facteurs de transcription qui ont été identifiés comme intéressants après l'étape 5 sont stockés dans la collection α et que les motifs qui restaient pertinents après l'étape 6 sont stockés dans la collection β. Ces collections α et β peuvent être utilisées pour enrichir les contextes d'extraction de bi-ensembles fréquents (i.e., gènes-situations) comme cela a été présenté dans [BRBR05]. On peut alors formuler une requête inductive pour

trouver des associations entre, e.g., les gènes qui sont régulés positivement ou négativement dans nos deux situations biologiques, et les sites de fixation putatifs dans les séquences promotrices de ces gènes (i.e., les motifs dans la collection β).

Enfin, pour rendre compte de la puissance du cadre des bases de données inductives considérons une dernière situation réaliste pour un chercheur s'intéressant à la régulation génique. On suppose qu'il faut extraire des motifs structurés au sens de [Sag00, CS04, SW03]. Le problème de l'extraction des motifs structurés peut être défini sous la forme de requêtes inductives. On peut spécifier la fonction d'évaluation de manière à ce qu'elle fournisse aussi l''information sur les positions du motif dans les séquences. Ainsi, une requête inductive pour extraire de tels motifs peut ête formulée de manière informelle comme suit: extraire les motifs qui sont présents au moins $f1$ fois dans les séquences différentes, au moins $p1$ fois dans chaque séquence et de telle sorte que la différence entre ces positions soit au plus $d1$. Si l'utilisateur souhaite ensuite que les composants du motif structuré soient similaires à un motif donné il suffit alors d'ajouter une contrainte de similarité.

A.1.5 Les séquences en biologie moléculaire

Les séquences (génomiques et/ou protéiques) font l'objet de manipulations quotidiennes en biologie moléculaire. Les motifs séquentiels et chaines dans de telles données sont intéressants. Donnons quelques exemples d'applications de l'extraction de motifs dans des données biologiques séquentielles ([Sag00], [RFP$^+$00]).

1. *Alignement multiple.* L'un des premiers usages des motifs séquentiels a concerné l'étude de l'alignement multiple de séquences génomiques ou protéiques. L'objectif est de trouver des motifs qui représentent des blocs (i.e., des groupes) de séquences reliées (i.e., ayant des similarités). Cela permet d'identifier des caractéristiques conservées dans les séquences d'ADN ou de protéines. D'autres applications importantes sont la représentation de familles de protéines, la déduction des mécanismes d'évolution à partir des séquences biologiques, l'assemblage des séquences ≪shotgun≫.

2. *Descripteurs individuels et composites.* La base de données PROSITE constitue le premier essai pour proposer une collection complète de motifs individuels qui caractérisent des collections de protéines et de fragments de protéines. Malheureusement, les descripteurs individuels ne conviennent pas à la caractérisation de certaines familles de protéines. Par conséquence, des descripteurs composites ont été conçus. Cette approche repose sur l'hypothèse que plusieurs protéines sont composées de quelques éléments conservés et chacun d'entre eux peut êre représenté par un descripteur.

3. *Séquences d'ADN impliqués dans la régulation et l'expression génique.* Il s'agit ici d'étudier les séquences promotrices et les sites de fixation de facteurs de transcription, et, éventuellement, les sites activateurs/répresseurs. Ces objets biologiques agissent souvent en coopération. En cherchant des motifs séquentiels qui les représentent, il faut prendre en compte cette réalité biologique (e.g., considérer des motifs composites/structurés).

4. *Autres éléments des séquences d'ADN.* On peut aussi rechercher d'autres types de motifs comme les sites d'épissage, les sites activateurs/répresseurs qui contribuent à l'épissage, les sites de restriction (i.e., un site reconnu par l'enzyme de restriction, typiquement un palindrome de longueur de 4-6 nucléotides), les répétitions en tandem (i.e., une collection d'instances multiples d'une combinaison de nucléotides où les instances apparaissent en tandem dans l'ADN; chaque instance étant une forme légèrement modifiée de la même unité de base).

5. *Prédiction de la structure d'ARN.* Ici, la structure est l'arrangement spatial, local ou global, des atomes qui composent la molécule. Les simples éléments structuraux de l'ARN peuvent être identifiés au niveau des séquences nucléiques comme des palindromes et ≪pseudoknots≫.

6. *Prédiction de la structure de protéines.* Ici, l'approche est d'identifier, à partir des séquences seules, les éléments qui sont conservés en termes de structure des protéines, et donc potentiellement impliqués dans la fonction enzymatique ou la topologie de l'ADN.

7. *Analyse de la régulation génique.* Les données provenant de puces à ADN permettent, entre autres, d'étudier la dynamique de régulation génique. L'objectif est d'identifier l'induction ou la répression des gènes en fonction du temps - l'expression différentielle de ces gènes peut être expliquée par un développement naturel de la cellule ou bien comme la réponse à des changements d'environnement ou de signaux.

Dans les cellules des eukaryotes, la capacité d'exprimer les protéines biologiquement actives est contrôlée par la régulation et dépend de plusieurs facteurs: (1) la structure de la chromatine, (2) l'initiation de la transcription qui se fait par l'interaction des éléments promoteurs, activateurs et répresseurs au niveau de la séquence ADN et des multiples protéines activatrices et inhibitrices, (3) le traitement et la modification des ARNm, (4) le transport des ARNm, (5) la régulation de la stabilité des ARNm, (6) l'initiation de traduction, (7) les modifications transactionnelles, (8) le transport des protéines à leurs sites d'activité et (9) la régulation de la stabilité protéique.

A.2 Contributions

A.2.1 Tolérance aux fautes en extraction dans des chaines

Il est courant que les données réelles contiennent des erreurs dues à des problèmes techniques au niveau de la collecte des données, du stockage et de la transmission. Dans certains domaines d'application, elles peuvent aussi être dues aux types de méthodes employées pour les analyser. Par exemple, des données sous la forme de séries numériques temporelles peuvent être analysées au moyen d'algorithmes d'extraction de sous-chaines, à la condition que les données sont préalablement discrétisés et donc encodées comme une séquence d'évènements d'un alphabet "calculé" (ou artificiel). Aussi, les données représentants des phénomènes réels sont souvent intrinsèquement dégénérées [EG01]. Par exemple, de nombreuses variantes de motifs chaine dans les séquences d'ADN peuvent être des sites de fixation du même facteur de transcription ou plusieurs séquences de parcours différents de pages internet peuvent permettent d'accomplir la même tache. Pour réussir à capturer de la connaissance dans de tels jeux de données une tolérance aux fautes est nécessaire.

Le besoin de tolérance aux fautes est présent dans de nombreux domaines. Une application classique est le traitement du signal. Par exemple, en reconnaissance de la parole, certaines parties du signal peuvent être compressées ou non prononcées, l'analyse d'un tel signal sans prise en compte de tolérances aux fautes est en pratique inopérante. Dans le domaine de la correction des erreurs (*error correction*), pour assurer une transmission correcte à travers un canal, il est nécessaire d'être capable de retrouver le signal original après une possible altération due à la transmission. Une autre application est l'extraction de texte ou d'information. L'approximation est un des outils de base pour corriger et reconnaître les mots mal écrits. L'un des plus grand domaine d'application reste la biologie (*computational biology*). Les séquences biologiques peuvent être considérées comme des chaines dans un alphabet spécifique, par exemple les séquences d'ADN peuvent être vues comme des chaines sur un alphabet de quatre lettres et les protéines comme des chaines sur un alphabet de vingt lettres. Un appareillement exact est rarement commode en analyse de séquences biologiques, sachant qu'elles contiennent des erreurs ou fautes dues aux technologies de séquençage et qu'elles sont intrinsèquement dégénérées. Le développement rapide et récent de la bio-informatique a résulté en un nombre intéressant d'approches pour s'attaquer au problème de la tolérance aux erreurs. Le concept de la "recherche en acceptant des erreurs" (search allowing errors) est un opérateur fondamental dans de nombreux problèmes, par exemple en reconstruction de séquences par alignement et reconstruction de sous-séquences, la recherche de parties fonctionnelles dans la séquence d'ADN et la comparaison de séquences pour l'analyse phylogénétique.

Dans ce travail, nous nous concentrons sur les méthodes déterministes par opposition aux méthodes probabilistes. Les motifs déterministes sont des motifs qui correspondent ou pas à un objet donné, c'est oui ou non. Les motifs probabilistes contiennent une probabilité de correspondance à un objet [BJEG98a]. En particulier nous allons nous concentrer sur des stratégies correctes et complètes pour extraire des motifs tolérants aux erreurs dans des données chaine. Dans ce travail, nous ne couvrons pas par exemple les approches non-déterministes comme les modèles de Markov, les réseaux bayésien, les matrices de poids, etc.

En fouille de données sous-contraintes, la notion de tolérance aux fautes correspond à deux problèmes disjoints. Le premier est de trouver les motifs qui sont similaires à un motif de référence. Le second problème est de trouver les motifs qui capturent des régularités softs (soft similarities), c'est-à-dire des régularités qui se répètent mais pas de façon complètement exacte. Ce problème peut être formulé comme un problème d'extraction de motifs satisfaisants une contrainte de fréquence avec une fonction de correspondance entre un motif et un objet (des données) qui est vraie si et seulement si l'objet est similaire au motif.

A.2.2 Contrainte de similarité et de soft-fréquence

L'objectif est de formuler des contraintes de similarité et de soft-fréquence de telle sorte qu'elles puissent être résolues efficacement, c'est-à-dire de réussir à les formuler sous la forme de contraintes monotones et anti-monotones. Cela permettrait non seulement une résolution efficace de ces contraintes mais surtout de pouvoir concevoir un algorithme générique `Marguerite-{Sim,SoftFr}` qui emploie les stratégies présentées dans [DJDM02, DD03, DD04] qui permettent de résoudre n'importe quelle combinaison de contrainte monotone et anti-monotone.

La similarité entre deux chaines est le problème algorithmique central dans le domaine de l'approximation de chaines, particulièrement utile en bio-informatique. Plusieurs approches de contraintes de similarité ont été étudiées dans la communauté de fouille de données. Une relation de similarité permet d'identifier les occurrences "soft", éléments indispensables pour définir les motifs tolérants au bruit. Dans la suite, nous garderons le terme anglais "soft" pour ne pas trop détruire son sens. L'extraction de motifs tolérants aux fautes dans des jeux de données de chaines au travers d'une contrainte dite de contrainte de soft-fréquence a beaucoup été étudiée en bio-informatique. En particulier, de nombreux extracteurs ad-hoc ont été développés pour répondre à des contraintes spécifiques.

Une alternative très prometteuse proposée dans [DJDM02, DD03] est de considérer une théorie générale qui est capable de supporter une combinaison arbitraire de contraintes monotones et anti-monotones. Une telle théorie a été implémentée pour des jeux de données type chaine et a conduit aux extracteurs génériques `VST` et `FAVST`. Le problème central pour concevoir un algorithme efficace est d'exploiter l'élagage d'espace de recherche grâce aux propriétés des contraintes. Dans la plupart des domaines d'application, la notion de similarité entre deux entités ε_1 and ε_2 signifie qu'il y a une "petite différence" entre ε_1 and ε_2. Évidemment, la propriété de petite différence ne doit pas être propagée trop loin, c'est-à-dire la propriété de similarité ne doit être transitive. Une contrainte de similarité qui n'est pas transitive est par définition ni monotone ni anti-monotone car une relation de similarité non-transitive ne peut pas être isomorphique à une relation de spécialisation. A cause de cette propriété, les motifs tolérants au bruit ne peuvent pas profiter des avancées récentes réalisées sur les algorithmes génériques.

A.2.2.1 Contrainte de similarité

Les relations de similarité ne peuvent pas être transitives et donc leur contrainte de similarité associée ne peut pas être (anti)-monotone.

Contrainte de similarité LCS

La similarité entre deux chaines peut être évaluée à partir de leur plus longue séquence commune(LCS). Il est intéressant de noter qu'il existe pour deux chaines données σ^1 and σ^2 une relation stable entre la longueur de $LCS(\sigma^1, \sigma^2)$ et les sous-chaines de la chaine σ^1.

Lemma A.1 *Soient σ^1, σ^2 deux chaine de l'alphabet Σ. Soit $\sigma^{1\prime}$ une sous-chaine de σ^1. Alors, nous avons $|LCS(\sigma^1, \sigma^2)| \geq |LCS(\sigma^{1\prime}, \sigma^2)|$.*

Par conséquent, nous proposons de rechercher des propriétés de monotonicité et d'anti-monotonicité dans des contraintes de similarité entre chaines en étudiant leur LCS. D'abord, nous observons que les chaines similaires doivent avoir une LCS suffisamment longue. Nous formulons donc une contrainte de taille minimale de la plus longue séquence commune.

Definition A.17 (Contrainte de LCS minimale) *Soient $\mathcal{L}_\Sigma$ un langage de chaines, ϕ un motif de $\mathcal{L}_\Sigma$, σ une chaine référence et $minLCS \in \mathbb{N}$ un seuil. Nous définissons la contrainte de LCS minimale, notée $\mathsf{MinLCS}_\phi(\sigma, minLCS)$, qui est vraie si et seulement si $|LCS(\phi, \sigma)| \geq minLCS$.*

Theorem A.1 *La contrainte de LCS minimale est monotone.*

Example A.2 *Soit $\sigma = \mathsf{tctggga}$ une chaine référence sur l'alphabet $\Sigma = \{a, c, g, t\}$. Les motifs chaines $\phi_1 = \mathsf{gcggga}$ et $\phi_2 = \mathsf{ctggaga}$ de $\mathcal{L}_\Sigma$ satisfont la contrainte $\mathsf{MinLCS}_\phi(\sigma, 5)$, sachant que $|LCS(\phi_1, \sigma)| = |\mathsf{cggga}| = 5$ et $|LCS(\phi_2, \sigma)| = |\mathsf{ctggga}| =$*

6. *Il faut noter que la* $\phi_3 =$ *attagtgttttgggg* *satisfait aussi la contrainte* $\mathsf{MinLCS}_\phi(\sigma, 5)$ *sachant que* $|LCS(\phi_3, \sigma)| = |\mathit{ttggg}| = 5$.

Cette exemple montre que la contrainte de LCS minimale $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ permet de spécifier un nombre minimal de symboles en correspondance entre deux chaines. Cependant, comme nous avons pu le voir avec la chaine ϕ_3, il n'y a pas de limite aux nombres de symboles qui ne sont pas en correspondance. $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ seule ne peut être utilisée pour spécifier la contrainte de similarité. Nous nous rappelons qu'une manière d'obtenir une sous-chaine d'une chaine est de supprimer certains des symboles de cette chaine. En limitant le nombre de délétions nécessaire à réaliser sur une chaine afin d'obtenir LCS, nous obtenons la contrainte complémentaire souhaitée qui fixe le nombre de fautes.

Definition A.18 (Contrainte de délétion maximale) *Soient* $\mathcal{L}_\Sigma$ *un langage de chaines,* ϕ *un motif de* $\mathcal{L}_\Sigma$*,* σ *une chaine référence et* $maxDels \in \mathbb{N}$ *un seuil. On fixe* $LCS(\phi, \sigma)$*, et note par* `dels` *les symboles de* ϕ *qui n'appartiennent pas à* LCS*. Le nombre de* `dels` *noté* $DelsLCS(\phi, \sigma)$*, est égal à* $|\phi| - |LCS(\phi, \sigma)|$*. Nous définissons une contrainte de délétion maximale notée* $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ *qui est vraie si et seulement si* $DelsLCS(\phi, \sigma) \leq maxDels$*.*

Theorem A.2 *La contrainte de délétion maximale* $\mathsf{MaxDels}_\phi(\sigma, maxDels)$ *est anti-monotone.*

Ainsi, deux chaines sont dites dans la relation de similarité de LCS si elles satisfont la contrainte de LCS minimale et de délétion maximale.

Definition A.19 (Relation de similarité de LCS) *Soient* σ^1*,* σ^2 *deux chaines sur l'alphabet* Σ*, et* $minLCS, maxDels \in \mathbb{N}$ *deux seuils. Les chaines* σ^1 *et* σ^2 *sont dans la relation de similarité de LCS, notée* $sim_{LCS}(\sigma^1, \sigma^2, minLCS, maxDels)$*, si et seulement si* $\mathsf{MinLCS}_{\sigma^1}(\sigma^2, minLCS) \wedge \mathsf{MaxDels}_{\sigma^1}(\sigma^2, maxDels) = \mathsf{true}$*.*

A partir de la relation de similarité de LCS, nous définissons la contrainte de similarité de LCS.

Definition A.20 (Contrainte de similarité de LCS) *Soient* $\mathcal{L}_\Sigma$ *un langage de motifs chaine,* ϕ *un motif de* $\mathcal{L}_\Sigma$*,* σ *une chaine référence et* $minLCS, maxDels \in \mathbb{N}$ *deux seuils. Nous définissons une contrainte de similarité de LCS, notée* $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ *qui est vraie si et seulement si le motif* ϕ *et la chaine de référence* σ *sont dans une relation de similarité* $sim_{LCS}(\phi, \sigma, minLCS, maxDels)$*.*

Example A.3 *En reprenant l'exemple A.2, les motifs* ϕ_1 *et* ϕ_2 *satisfont la contrainte* $\mathsf{LCSSim}_\phi(\sigma, 5, 1)$*. Le motif* $\phi_4 = \mathsf{gcgggta}$ *satisfait la contrainte* $\mathsf{LCSSim}_\phi(\sigma, 5, 2)$*, sachant que* $LCS(\phi_4, \sigma) = |\mathsf{cggga}| = 5$*. Le motif* ϕ_3 *ne satisfait ni la contrainte* $\mathsf{LCSSim}_\phi(\sigma, 5, 1)$ *ni la contrainte* $\mathsf{LCSSim}_\phi(\sigma, 5, 2)$*.*

Remark A.1 *La taille d'un motif* ϕ *satisfaisant la contrainte* $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$ *est au moins de* $minLCS$ *et au plus de* $|\sigma| + maxDels$*. Il faut noter que même si la taille maximale de* ϕ *satisfaisant* $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$*, peut être déduite de* $|\sigma|$ *et* $maxDels$*, le fait que* ϕ *satisfait la contrainte* $\mathsf{MinLCS}_\phi(\sigma, minLCS) \wedge \mathsf{MaxLength}_\phi(|\sigma| + maxDels)$ *n'implique pas qu'il satisfait aussi la contrainte* $\mathsf{LCSSim}_\phi(\sigma, minLCS, maxDels)$*.*

Example A.4 *On considère une chaine de référence* $\sigma = \mathsf{agcgac}$ *sur l'alphabet* $\Sigma = \{a, c, g, t\}$*, un motif chaine* $\phi = \mathsf{gagataga}$*, et des seuils* $minLCS = 4$*,* $maxDels = 2$*. Le motif* ϕ *satisfait la contrainte* $\mathsf{MinLCS}_\phi(\sigma, 4) \wedge \mathsf{MaxLength}_\phi(6 + 2)$*, mais il ne satisfait pas la contrainte* $\mathsf{LCSSim}_\phi(\sigma, 4, 2)$*, sachant que même si* $LCS(\phi, \sigma) = |agga| = 4$*, nous avons* $DelsLCS(\phi, \sigma) = 4 > 2$*.*

A.2.2.2 Contrainte de soft-fréquence

Le problème de l'extraction de motifs tolérants aux fautes dans des données chaine peut formalisé comme une tâche d'extraction de motifs ϕ qui satisfont une contrainte de fréquence soft. L'évaluation des contraintes de fréquence soft minimale $\mathsf{MinSoftFr}_\phi(minFr, \mathcal{D})$ et maximale $\mathsf{MaxSoftFr}_\phi(maxFr, \mathcal{D})$ nécessite de calculer la fréquence soft $\mathsf{SoftFr}(\phi, \mathcal{D})$ d'un motif ϕ. Pour cela, nous avons besoin de trouver tous les objets X dans les données $\mathcal{D}$ qui sont similaires au motif, c'est-à-dire d'évaluer la contrainte $\mathsf{Sim}_X(\phi) \wedge \mathsf{MinFr}_X(1, \mathcal{D})$. Comme expliqué précédemment, une contrainte de similarité ne peut pas avoir les propriétés de monotonicité et d'anti-monotonicité. L'étape de calcul de la fréquence soft d'un motif est donc non triviale car les mécanismes d'élagage liés aux propriétés de monotonicité et d'anti-monotonicité ne peuvent pas être employés. En plus, la contrainte de fréquence soft minimale (respectivement maximale) n'est pas forcément anti-monotone car il n'existe pas forcement de relation stable entre les sous chaines et les sur-ensemble d'un motif chaine et la taille de leurs occurrences softs.

La contrainte de similarité LCS introduite précédemment est exprimée comme une conjonction de deux contraintes qui ont les propriétés requises. Ainsi, nous proposons d'utiliser cette contrainte de similarité pour trouver les occurrences softs d'un motif, étape nécessaire pour calculer sa fréquence soft.

La contrainte de similarité LCS notée $\mathsf{LCSSim}_X(\phi, minLCS, maxDels)$ est paramétrée par $minLCS$ qui correspond à la taille minimale requise pour $LCS(X, \phi)$ et par $maxDels$ qui est le nombre maximal de délétion autorisé.

Relation de similarité *InsDels*

La contrainte de similarité LCS notée $\mathsf{MinLCS}_\phi(\sigma, minLCS)$ en imposant un LCS suffisamment grand entre deux chaines, peut être reformulée de telle sorte qu'elle n'est plus dépendante directement de le taille $|\phi|$. Elle est alors exprimée en terme d'insertions et est appelée contrainte d'insertion maximale.

Definition A.21 (Contrainte d'insertion maximale) *Soient $\mathcal{L}_\Sigma$ un langage de motifs chaine, ϕ un motif de $\mathcal{L}_\Sigma$, σ une chaine référence et $maxIns \in \mathbb{N}$ un seuil. Nous fixons un $LCS(\phi, \sigma)$, notons par **ins** les symboles de σ qui n'appartiennent pas à cette LCS. Le nombre de **ins**, noté $InsLCS(\phi, \sigma)$ est égal à $|\sigma| - |LCS(\phi, \sigma)|$. La contrainte d'insertion maximale notée $\mathsf{MaxIns}_\phi(\sigma, maxIns)$ est vraie si et seulement si $InsLCS(\phi, \sigma) \leq maxIns$.*

Theorem A.3 *La contrainte d'insertion maximale $\mathsf{MaxIns}_\phi(\sigma, maxIns)$ est monotone.*

Example A.5 *Soit une chaine référence $\sigma = \mathsf{tctggga}$ sur l'alphabet $\Sigma = \{a, c, g, t\}$. Les motifs chaine $\phi_1 = \mathsf{gcggga}$ et $\phi_2 = \mathsf{ctggaga}$ satisfont la contrainte $\mathsf{MaxIns}_\phi(\sigma, 2)$ constraint, sachant que $InsLCS(\phi_1, \sigma) = |\sigma| - |\mathsf{cggga}| = 2$ et $InsLCS(\phi_2, \sigma) = |\sigma| - |\mathsf{ctggga}| = 1$. Le motif chaine $\phi_3 = \mathsf{attagtgttttgggg}$ satisfait aussi la contrainte $\mathsf{MaxIns}_\phi(\sigma, 2)$ sachant que $InsLCS(\phi_3, \sigma) = |\sigma| - |\mathsf{ttggg}| = 2$.*

Cette exemple montre que la contrainte d'insertion maximale $\mathsf{MaxIns}_\phi(\sigma, maxIns)$ comme la contrainte $\mathsf{MinLCS}_\phi(\sigma, minLCS)$, permet de spécifier le nombre minimal de symboles qui correspondent, mais comme avec le motif ϕ_3 elle ne permet pas de borner le nombre de symboles qui ne correspondent pas. Pour borner ce nombre "d'erreurs", nous allons utiliser la contrainte de délétion maximale $\mathsf{MaxDels}_\phi(\sigma, maxDels)$.

Definition A.22 (Relation de similarité *InsDels*) *Soient σ^1, σ^2 deux chaines sur un alphabet et $maxIns, maxDels \in \mathbb{N}$ deux symboles. Les chaines σ^1 et σ^2 sont dans une relation de similarité insertion-délétion (InsDels) notée $sim_{InsDels}(\sigma^1, \sigma^2, maxIns, maxDels)$, si et seulement si $\mathsf{MaxIns}_\phi(\sigma, maxIns) \wedge \mathsf{MaxDels}_{\sigma^1}(\sigma^2, maxDels) = \mathsf{true}$.*

A partir de la relation de similarité $InsDels$, nous pouvons définir la contrainte de similarité $InsDels$.

Definition A.23 (Contrainte de similarité *InsDels*) *Soient $\mathcal{L}_\Sigma$ un langage de motifs chaine, ϕ un motif de $\mathcal{L}_\Sigma$, σ une chaine de référence, et $maxIns, maxDels$*

$\in \mathbb{N}$ deux chaines. Nous définissons une contrainte de similarité $InsDels$ notée $\mathsf{InsDelsSim}_{\phi}(\sigma, maxIns, maxDels)$ qui est vraie si et seulement si un motif ϕ et une chaine référence σ sont dans une relation de similarité insertion-délétion $sim_{InsDels}(\phi, \sigma, maxIns, maxDels)$.

Fréquence soft par la relation de similarité $InsDels$

Nous utilisons la relation de similarité $InsDels$ qui est une conjonction de contraintes monotones et anti-monotones et qui peut être évaluée efficacement pour trouver les occurrences softs d'un motif chaine ϕ.

Definition A.24 (Fonction de correspondance $InsDels$**)** *Soient $\mathcal{U}_{\Sigma}$ un univers de chaines (langage des éléments dans les données) sur an alphabet Σ, S un objet de $\mathcal{U}_{\Sigma}$, et $maxIns, maxDels \in \mathbb{N}$ deux chaines. Soit ϕ un motif d'un langage de motifs chaine $\mathcal{L}_{\Sigma}$. Nous définissons une fonction de correspondance $InsDels$ notée $match_{InsDels}(\phi, S, maxIns, maxDels)$ qui est vraie si et seulement si il existe une chaine σ telle que $\sigma \sqsubseteq S$ et $sim_{InsDels}(\phi, \sigma, maxIns, maxDels) = \mathsf{true}$.*

Definition A.25 (Fréquence soft $InsDels$**)** *La fréquence $\mathsf{Fr}(\phi, \mathcal{D})$ qui est évaluée en utilisant la fonction de correspondance $InsDels$ nommée $match_{InsDels}(\phi, S, maxIns, maxDels)$, est appelée la fréquence soft $InsDels$ et est notée $\mathsf{SoftFr}_{InsDels,S}(\phi, \mathcal{D}, maxIns, maxDels)$.*

Definition A.26 (Contrainte de fréquence soft $InsDels$**)** *La contrainte de fréquence soft minimale $\mathsf{MinFr}_{\phi}(minFr, \mathcal{D})$ qui utilise la fonction de correspondance $InsDels$ est appelée une contrainte de fréquence soft $InsDels$ et est notée $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$. La contrainte de fréquence maximale $\mathsf{MaxFr}_{\phi}(maxFr, \mathcal{D})$ qui utilise la fonction de correspondance $InsDels$ est appelée contrainte de fréquence soft maximale $InsDels$ et est notée $\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels)$.*

L'utilisation de la fonction de correspondance $InsDels$ pour trouver les occurrences soft d'un motif permet non seulement de calculer efficacement les motifs soft mais aussi il garantie les propriétés de monotonicité et d'anti-monotonicité de la contrainte de fréquence soft minimale et maximale.

Theorem A.4 *La contrainte de fréquence soft minimale $InsDels$ $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$ est anti-monotone si $maxDels \geq maxIns$.*

La contrainte de fréquence soft maximale InsDels $\mathsf{MaxSoftFr}_{\phi}^{InsDels,S}(maxFr, \mathcal{D}, maxIns, maxDels)$ est monotone is monotonic si $maxDels \geq maxIns$.

Example A.6 (Contre-exemple) *Cette exemple montre que la contrainte de fréquence soft minimale InsDels $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(minFr, \mathcal{D}, maxIns, maxDels)$ n'est pas anti-monotone quand $maxDels < maxIns$.*
Soit un alphabet $\{a, c, g, t\}$, $\mathcal{L}_{\Sigma}$ un langage de motifs chaine de Σ et $\mathcal{D} = \{\mathsf{aagc}, \mathsf{gc}, \mathsf{gc}\}$ un jeu de données chaine. La contrainte $\mathsf{MinSoftFr}_{\phi}^{InsDels,S}(2, \mathcal{D}, 2, 1)$ est satisfaite pour le motif $\phi = \mathsf{aagc}$, mais n'est pas satisfaite pour une de ces sous-chaines $\phi' = \mathsf{aa}$, sachant que $\mathsf{SoftFr}_{InsDels,S}(\mathsf{aagc}, \mathcal{D}, 2, 1) = 3$ et $\mathsf{SoftFr}_{InsDels,S}(\mathsf{aa}, \mathcal{D}, 2, 1) = 1$.

A.2.3 Twilight zone

A.2.3.1 Motivation

De nombreux types de motifs locaux comme par exemple les règles d'association, les sou-chaines, les règles d'épisode, les sous-arbres et les sous-graphes, ont été étudiés ces dernières années. De nouveaux types de motifs qui supportent des tâches d'extraction spécifique à certains domaines d'application rendant les motifs plus utilisables sont cruciaux. Évidemment, cela ne résout pas tous les problèmes. En effet, dans des contextes d'application réels, les motifs intéressants sont souvent cachés au milieu de très nombreux autres [CZ06]. Le problème majeur de la plupart des méthodes d'extraction de motifs locaux est le fardeau de ces motifs a priori inintéressants. Le terme inintéressant doit être compris à la fois au niveau subjectif et objectif. Par exemple, la plus part du temps, éviter de fournir ou même de calculer les motifs déjà connus est un facteur clé pour faciliter l'interprétation futur des motifs. Le problème central est de réussir à prendre le meilleur de la connaissance du domaine d'application et de l'utiliser au mieux à chaque étape du cycle de vie de l'ECD. La qualité de la sélection des données et des étapes de pré-traitement résulte principalement de la connaissance du domaine. L'étape de fouille de données doit incorporer la connaissance du domaine si l'on ne veut pas devoir utiliser toutes nos ressources de calcul pour calculer un ensemble gigantesque de motifs inintéressants tout en ratant ceux recherchés. Pour finir, l'étape cruciale de post-traitement des motifs extraits nécessite aussi une prise en compte de la connaissance du domaine par l'intermédiaire des experts des données. Nous sommes convaincu que de nombreux de ces problèmes, en particulier pour améliorer la phase d'extraction et de post-traitement des motifs, peuvent être résolus au moyen du cadre d'extraction de motifs sous-contraintes [BDM05].

Finalement, pour réussir à extraire ces précieuses aiguilles (motifs intéressants) dans la botte de foin (la collection des motifs possibles), il y a un besoin très important de nouvelles méthodes qui permettraient d'extraire ces motifs potentiellement intéressants. Si l'extraction de motifs sous-contraintes est notre approche, nous avons besoin non seulement de spécifier des contraintes subtiles et ingénieuses pour décrire la tache d'extraction dans notre domaine d'application. Nous avons aussi besoin de s'intéresser de près aux autres manières d'exprimer la pertinence des motifs. En particulier, celles qui ne peuvent pas être complètement exprimées sous la forme de contraintes monotones et anti-monotones sur des motifs locaux. C'est le cas de la plupart des mesures statistiques. Même si elles n'ont pas ces jolies propriétés, il serait néanmoins très intéressant de pouvoir les exploiter en recherchant par exemple les motifs qui sont statistiquement inattendus dans un jeu de données particulier connaissant les valeurs des paramétrés de l'extraction. Pour cela, nous pouvons nous inspiré de [KP02b, GMMT07] qui propose d'évaluer la pertinence des motifs locaux (respectivement les chaines et les itemsets) en comparant le nombre de motifs extraits dans le jeu de données et le nombre de motif extraits dans des jeux de données aléatoires exhibant les même caractéristiques (propriétés structurelles du jeu de données comme la taille, nombre d'items et de séquences, longueur des itemsets ou des séquences) que le jeu de données original. Le problème ici est d'être en mesure d'estimer le nombre de motifs locaux qui satisfont une contrainte donnée dans un jeu de données aléatoire ayant certaines caractéristiques.

Estimer le nombre attendus de motifs qui peuvent satisfaire une contrainte est un général bien plus compliqué que d'estimer la probabilité qu'à un motif de satisfaire une contrainte donnée. Le deuxième problème a reçu beaucoup plus d'attention conduisant à de nombreuses mesures statistiques pour évaluer la pertinence de motifs. Concernant le premier problème, seulement quelques solutions ont été proposées. [RMZ03] et [LRS05] analysent les distributions possible des itemsets fréquents, des motifs fermés [LRS05] et des itemsets maximaux [RMZ03].

[RMZ03] analyse le type de distributions que l'on peut attendre pour différents types de jeux de données. Ils répondent à la question "Existe-t-il une collection contenant un certain nombre d'itemsets ou d'itemsets maximaux de taille donnée ?". [LRS05] calcule le nombre moyen d'itemsets (fermés) fréquents en utilisant des techniques statistiques. Les auteurs se concentrent plus particulièrement sur la contrainte de fréquence minimale en considérant des seuils proportionnels ou absolus. Une autre approche a été proposée dans [GGdB05] qui fournit une borne fine du nombre de motifs candidats qui peuvent être rencontré lors d'une extraction de motifs par niveau. Connaissant le niveau et le nombre courant de motifs fréquents, ils proposent une borne fine du nombre maximal de motifs candidat qui peuvent être générés au niveau suivant. En extraction de chaines, [KP02b] propose une estimation du nombre de motifs issus du bruit de fond et qui probablement seront extraits pour une fréquence minimale donnée sachant la structure du jeu de données. La dénommée Twilight Zone (TZ) est définie comme l'ensemble des valeurs de la fonction de score pour lequel l'on peut s'attendre à avoir des motifs aléatoires ayant de tels scores.

Considérons maintenant ce problème à un niveau plus large en incluant les propriétés structurelles des jeux de données (par exemple le nombre de séquences et la taille des séquences) et les paramètres d'extraction (par exemple la sélection des seuils de mesures d'intérêt et taille des motifs à extraire). Ainsi, la TZ peut être vue comme une région (ou ensemble de régions) dans l'espace des paramètres où il y a très probablement des motifs aléatoires parmi les motifs extraits, ces motifs aléatoires ayant des scores aussi bons (éventuellement meilleurs) que les motifs qui nous intéressent.

Ces propositions sont toutes basées sur un modèle analytique global, c'est-à-dire ce sont des approches intéressantes mais qui néanmoins nécessitent des développements complexes et spécifiques. Finalement, ces méthodes ne peuvent pas être facilement étendues pour supporter des conjonctions complexes de contraintes qui incorporent différentes distributions de symboles ou des sémantiques différentes au niveau des occurrences des motifs.

Cette estimation peut devenir difficile à calculer dès lors que de nombreuses contraintes primitives peuvent être combinées pour former une requête d'extraction et tel que de nombreuses primitives utilisent au moins un paramètre. Nous avons souvent qu'une vision très réduite du comportement des contraintes à l'intérieur de l'espace des paramètres. Une pratique habituelle dans un tel cas est de compter le nombre de motifs obtenus pour différentes configurations de paramètres pour essayer de deviner ce que pourrait être les "bons" paramètres. Dans les contextes simples quand on considère une simple fréquence minimale, un nombre assez réduit d'essais est nécessaire. Ce n'est évidemment pas le cas dans la majorité des cas où la contrainte employée est une conjonction de contraintes primitives, conduisant à un espace de paramètres multidimensionnel.

Trois alternatives principales peuvent être envisagées pour estimer le nombre de motifs dans des séquences aléatoires, qui satisfont une contrainte. D'abord, des jeux de données aléatoires avec les caractéristiques désirées peuvent être générés aléatoirement dans lesquels les motifs sont extraits et comptés. La deuxième solution est de calculer une estimation analytique du nombre de motifs extraits à partir des caractéristiques du jeu de données. La troisième alternative consiste à calculer cette estimation analytique mais sur un échantillon de l'espace de recherche. Il y a compromis important à trouver entre la précision de l'estimation et le temps de calcul. La première méthode permet de fournir une très bonne estimation (si l'on parvient à générer les jeux de données aléatoires avec les bonnes caractéristiques) mais est bien trop gourmande en temps pour être utilisable en pratique sur de vraies jeux de données. Nous avons donc décidé de nous concentrer sur les deux autres alternatives et avons proposé pour chacune d'elle une méthode d'estimation du nombre de motifs.

A.2.3.2 Estimation analytique d'une mesure d'intérêt des motifs

Nous considérons des jeux de données composées de motifs chaine de l symboles et ayant la même longueur, noté G. Comme dans [KP02b], nous supposons que les séquences du jeu de données sont composées de symboles indépendants et uniformément distribués ayant la même probabilité d'occurrence, et tel que le chevauchement des motifs a un impact négligeable sur le nombre de motifs extraits (sachant que $L \ll G$). Nous souhaitons estimer le nombre de motifs exactes et tolérants aux fautes de taille L qui seraient extraits avec les seuils α_{min}, α_{max} et α_{dist} dans deux jeux de données. Nous supposons que les deux jeux de données sont indépendants.

Sans rentrer dans le détail du calcul, nous fournissons ici l'estimation analytique obtenue :

$P(M \ sat. \ min. \ and \ max. \ supp.) \times l^L =$
$P(M \ sat. \ min. \ supp.) \times P(M \ sat. \ max. \ supp.) \times l^L = \times$
$\sum_{z=0}^{\alpha_{max}} \binom{N^-}{z} \times P(exists \ soft \ occ. \ of \ M \ in \ a \ seq.)^z$
$(1 - P(exists \ soft \ occ. \ of \ M \ in \ a \ seq.))^{N^- - z} \times$
$\sum_{i=1}^{N^+} (P(X = i) \times P(M \ sat. \ min. \ supp.|X = i)) \times l^L$

où

si $i \geq \alpha_{min}$ alors $P(M \ sat. \ min. \ supp.|X = i)) = 1$

si $i < \alpha_{min}$ alors $P(M \ sat. \ min. \ supp.|X = i) = P(M \ sat. \ min. \ supp.|X = i)) = \sum_{z=\alpha_{min}-i}^{N^+-i} (\binom{N^+-i}{z} \times P(exists \ strict \ soft \ occ. \ of \ M \ in \ a \ seq.)^z \times (1 - P(exists \ strict \ soft \ occ. \ of \ M \ in \ a \ seq.))^{N^+-i-z})$

A.3 Estimation par échantillonnage

Soit $S_{\mathcal{C}}$ l'ensemble de motifs de $\mathcal{L}$ qui satisfont la contrainte $\mathcal{C} \equiv \mathsf{MinFr}^{\phi}_{(}(,)\mathcal{D}_1, f_1)$ $\wedge \mathsf{MaxFr}_{\phi}(\phi, \mathcal{D}_2, f_2) \wedge \mathcal{C}_{synt}(\phi)$. Ici, nous traitons à la fois les motifs chaine exacts et tolérants aux fautes. Nous présentons dans cette partie une proposition pour estimer $|S_{\mathcal{C}}|$ par échantillonnage de l'espace des motifs en utilisant une fonction qui donne $P(\phi \ sat. \ \mathcal{C})$ pour tout motif ϕ, c'est-à-dire la probabilité que le motif satisfasse la contrainte $\mathcal{C}$.

Nous avons choisis trois distributions différentes pour les symboles dans les séquences d'entrées pour montrer que la méthode peut être utilisée avec différents modèles. Malgré tout, ce choix n'est pas central dans ce travail et suivant le domaine d'application d'autres modèles peuvent être choisis.

Les trois modèles retenus sont:

- μ_E: indépendance de tous les symboles avec fréquence égale pour chaque symbole;
- μ_D: indépendance de tous les symboles avec fréquence d'apparition différente pour chaque symbole;
- μ_M: chaine de Markov d'ordre un.

Pour chacun des trois modèles mentionnés, il est facile de calculer la probabilité qu'un motif donné apparaisse dans une chaine, d'obtenir la probabilité que le motif soit fréquent avec une loi binomiale et finalement de calculer $P(\phi\ sat.\ \mathcal{C})$.

A chaque motif ϕ est associé une variable aléatoire X_ϕ de telle sorte que $X_\phi = 1$ quand ϕ satisfait $\mathcal{C}$ et $X_\phi = 0$ sinon. Alors, nous avons $|S_\mathcal{C}| = \sum_{\phi\in\mathcal{L}} X_\phi$. En calculant la valeur de $|S_\mathcal{C}|$ grâce à la linéarité de l'opérateur d'espérance, nous avons $E(|S_\mathcal{C}|) = \sum_{\phi\in\mathcal{L}} E(X_\phi)$. Sachant que $E(X_\phi) = 1 \times P(X_\phi = 1) + 0 \times P(X_\phi = 0)$ alors $E(|S_\mathcal{C}|) = \sum_{\phi\in\mathcal{L}} P(\phi\ sat.\ \mathcal{C})$. Soit $S_{\mathcal{C}_{synt}}$ un ensemble de motifs de $\mathcal{L}$ qui satisfont $\mathcal{C}_{synt}$. Comme $P(\phi\ sat.\ \mathcal{C}) = 0$ pour tous les motifs qui ne satisfont pas $\mathcal{C}_{synt}$ alors nous avons $E(|S_\mathcal{C}|) = \sum_{\phi\in S_{\mathcal{C}_{synt}}} P(\phi\ sat.\ \mathcal{C})$. Il n'est pas envisageable de calculer cette somme sur $S_{\mathcal{C}_{synt}}$ sachant que l'on souhaite calculer $E(|S_\mathcal{C}|)$ pour un grand nombre de points dans l'espace des paramètres. Ainsi, nous estimons $E(|S_\mathcal{C}|)$ en utilisant un échantillonnage des motifs de $S_{\mathcal{C}_{synt}}$. Soit S_{samp} un tel échantillonnage, alors nous utilisons la valeur suivante comme estimation de $E(|S_\mathcal{C}|)$:

$$\frac{|S_{\mathcal{C}_{synt}}|}{|S_{samp}|} \times \sum_{\phi\in S_{samp}} P(\phi\ sat.\ \mathcal{C})$$

En pratique, de nombreuses techniques peuvent être utilisées pour calculer l'échantillonnage. Dans nos expériences, nous utilisons le processus suivant :

- Etape 1: Construction d'un échantillon initial S_{samp} de $\mathcal{C}_{synt}$ (avec replacement) correspondant à 5% de $|\mathcal{C}_{synt}|$ et ensuite calcul de $E(|S_\mathcal{C}|)$.
- Etape 2: Ajout de 1000 nouveaux échantillons à S_{samp}. Calcul de l'estimation et si la valeur absolue de la différence entre la nouvelle estimation et l'ancienne estimation est supérieure à 5% de l'estimation précédente alors l'étape 2 est répétée.

A.4 Application

A.4.1 Background

Nous avons appliqué

- notre implémentation de notre solveur `FAVST` [DD04] pour réaliser des extractions différentielles correctes et complètes avec des contraintes de fréquence exacte,
- notre solveur générique `Marguerite-H` pour réaliser des extractions correctes et complètes avec des contraintes de fréquence soft avec distance de Hamming,
- la mesure d'intérêt de type Twilight Zone que nous avons développé permettant d'évaluer l'intérêt des motifs obtenus,

pour trouver des motifs signature dans l'ensemble des promoteurs de gènes différentiellement exprimés. Ce travail a été accompli en collaboration avec Dr. Olivier Gandrillon et son équipe de recherche " Bases Moléculaires de l'Autorenouvellement et de ses Altérations" du "centre de génétique moléculaire et cellulaire" (CNRS UMR 5534). L'auto-renouvellement, qui est une propriété caractéristique des cellules souches et qui est une notion encore mal comprise, est le sujet de recherche de l'équipe BM2A. Une dérégulation de ce processus se produit fréquemment dans le cancer. L'équipe BM2A étudie ce processus au travers de la découverte de gènes différentiellement exprimés par la technique SAGE [VEVK95] sur le modèle T2EC (normal chicken erythroid progenitors [GSBS99]). Ces cellules peuvent s'auto-renouvelé ou alors se différentier suivant certaines conditions contrôlables. L'expression de l'oncogène v-erbA provoque une transformation en bloquant le processus de différentiation [GJP+89]. L'équipe BM2A a décidé d'identifier les gènes cibles de v-erbA responsables du processus de transformation provoqué par v-erbA. Pour cela, ils comparent le transcriptome des T2ECs en exprimant une forme oncogénique de v-erbA avec le transcriptome des des T2ECs en exprimant le mutant S61G de v-erbA. Ce mutant est défectueux dans sa capacité à inhiber la différentiation et à provoquer une transformation erythroid [SP91]. Ainsi, la comparaison entre le transcriptome des cellules exprimant soit une forme transformée de v-erbA ou le mutant S61G de v-erbA, a permis de générer une liste de 110 gènes différentiellement exprimés entre ces deux condition [BKF+07]. Nous avons utilisé ce jeu de données pour extraire des motifs exactes et les motifs tolèrants aux fautes avec une extraction différentielle. Nous avons sélectionné les motifs les plus inattendus grâce à la mesure d'intérêt TZI que nous avons développé. L'évaluation biologique de ces motifs confirment leur rôle fonctionnel potentiel et ainsi illustre le potentiel de notre methode de découverte de motifs. Afin d'évaluer la généralité de notre approche, nous avons aussi appliqué notre méthode sur un second jeu de données obtenu à partir de promoteurs de gènes ayant une expression différentielle entre deux situations [DKGG+04].

La compréhension de la régulation de l'expression des gènes reste un des challenges les plus importants en biologie moléculaire. Un des points clés est l'initialisation de la transcription par l'interaction entre les éléments promoteurs d'un gène au niveau de la séquence d'ADN et les protéines activatrices et inhibitrices appelées facteurs de transcription (TF). Ces intéractions se produisent quand un TF se fixe sur un site de fixation de TF sur un site promoteur. De nombreuses méthodes pour découvrir de nouveaux sites de fixation ont été développées. Parmi elles, deux familles peuvent être distinguées: les méthodes statistiques ou stochastiques et les approches combinatoires [VMS99a].

Concernant la famille des métodes statistiques et stochastiques, une récente étude des algorithmes les plus utilisées montre que les résultats obtenus avec ces méthodes sont encore limités [TLB$^+$05], et conclu sur la nécessité d'explorer d'autres alternatives. Il y a de nombreuses raisons pour leurs succès limités mais il semble que la difficulté de séparer les motifs du bruit de fond est une des principales. Les méthodes statistiques font des hypothèses sur les modèles de distributions pour des raisons d'efficacité, mais personne ne connait le processus stochastique réel employé par la nature et ce qu'est l'aléatoire biologique. En plus, les processus stochastiques semblent différents entre espèces: de nombreux outils fonctionnent bien mieux sur la levure que sur d'autres espèces [TLB$^+$05, DD07]. En plus de cela, avec les mesures d'intérêt employées, la pertinence statistique est très dépendante de la taille des séquences promotrices : en considérant des sequences plus longues permet d'identifier des éléments régulateurs plus loin du gène mais malheureusement des séquences aléatoires deviennent aussi statistiquement significative que les éléments régulateurs [KP02b].

Nous nous sommes concentrés sur les approches combinatoires qui ont comme objectif l'extraction exhaustive de motifs sans hypothèses a priori sur les processus stochastiques. Selon [KP02a], probablement les meilleur outils de recherche de motifs consensus dans les séquences ADN sont ceux qui testent tous les 4^l motifs de longueur $l = 4$, score chaque motif avec le nombre d'occurrences approchées et cherche les motifs ayant les meilleurs scores. La recherche exhaustive de tous les motifs de taille l devient impossible lorsque l est grand. Or la taille des sites de fixation de FT est habituellement entre 5 et 15 paires de bases [Bul03]. Il est assez raisonnable de penser qu'avec de telles tailles les extractions exhaustives sont possibles en pratique. Malgré tout, les méthodes exhaustives ne sont souvent pas assez sélectives pour discriminer les vrais sites des faux sites, à cause du grand nombre de motifs obtenus. Le développement des méthodes exhaustives et optimales (tous les motifs ayant les plus hauts scores) pour la découverte de motifs dans les séquences biologiques a conduit à de nombreux algorithmes de recherche de sites de fixation de FT comme [QWK82, WAG84, Sta89, SEVS95b, SV96b, BJVU98b, RF98a] . En pratique, ils ont tous besoin d'une mesure de fitness utilisée pour classer ou sélectionner les motifs.

En ayant à l'esprit les difficultés de modéliser statistiquement l'aléatoire biologique, nous proposons de repousser la phase de la sélection des motifs pertinents et d'utiliser les information biologiques dont nous disposons pour contraindre la recherche et ainsi réduire le nombre de motifs extraits. Ces informations supplémentaires proviennent d'un second jeu de données représentant des situations biologiques opposées. Pour utiliser ces informations, la méthode commence avec une opération standard en biologie moléculaire : la recherche de gènes différentiellement exprimés. Cela permet d'obtenir deux groupes de gènes dont on peut dériver deux ensembles de sites promoteurs. Pour rechercher des sites de fixation putatifs de FT qui régulent les gènes sur-exprimés, nous choisissons le premier ensemble de gènes comme le jeu positif et le second comme jeu négatif. Ensuite, nous recherchons les motifs qui sont présents sur au moins $minFr$ promoteurs du jeu de données positif et sur au plus $maxFr$ promoteurs du jeu de données négatif. L'originalité de la méthode proposée par rapport aux autres algorithmes combinatoires est qu'elle permet d'extraire des motifs de plusieurs jeux de données (e.g., SPEXS [BJVU98b] or DRIM [ELYY07]) en fixant explicitement le seuil de fréquence maximal. Deux types de motifs peuvent être extraits par notre méthode : des motifs exacts et des motifs tolérants aux fautes (ayant une fonction de correspondance approchée). L'utilisation du jeu de données négatif permet de réduire considérablement (plusieurs ordres de magnitude) le nombre de motifs extraits. Malgré cela, le nombre de motifs extraits reste important, ce qui nous a poussé à développer d'autres solutions pour aider à ajuster les paramètres d'entrée afin de n'avoir à analyser qu'un ensemble réduit et potentiellement intéressant de motifs. L'étape suivante, nous sélectionnons les motifs inattendus c'est-à-dire non "subtile" : un motif ϕ est considéré comme subtil si l'on peut s'attendre à ce que des motifs aléatoires soient au moins aussi fréquents que ϕ dans le jeu de données positif et pas plus fréquents que ϕ dans le jeu de données négatif. Finalement, nous vérifions quels motifs sont des facteurs de transcription connus.

A.4.2 Les solveurs utilisés

Nous avons utilisé utilisé notre implémentation de `FAVST` [DD04] et pour extraire les motifs tolérants aux fautes (SMP) nous avons utilisé notre solveur générique `Marguerite-H` . Ces solveurs permettent d'évaluer n'importe quelle combinaison de contraintes monotones et anti-monotones en particulier les contraintes de fréquence exacte et de fréquence soft avec Hamming. Avec une conjonction de fréquence minimale et de fréquence maximale, nous pouvons alors réaliser des extractions différentielles.

A.4.3 Résultats

Parmi les motifs extraits sur nos données réelles avec nos solveurs, certains ont été validés comme étant des sites de fixation de facteurs de transcription impliqués dans le mécanisme d'auto-renouvellement [MRS+08].