Current Approaches in Algorithmic Intelligence: Efficient Tolerant Pattern Matching with Constraint Abstractions in Description Logic *

Carsten Elfers, Stefan Edelkamp and Otthein Herzog
Center for Communication and Computing Technologies (TZI)
Bremen, Germany
{celfers;edelkamp;herzog}@tzi.de

Abstract

In this paper we consider efficiently matching logical constraint compositions (called patterns) to noisy observations or to ones which are not well described by existing patterns. The major advantage of our approach to tolerant pattern matching is to exploit existing domain knowledge from an ontological knowledge base represented in description logic in order to handle imprecision in the data and to overcome the problem of an insufficient number of patterns. The matching is defined in a probabilistic framework to ensure that post-processing with probabilistic models is feasible.

Additionally, we propose an efficient complete (and optionally approximate) algorithm for this kind of pattern matching. The presented algorithm reduces the number of inference calls to a description logic reasoner. We analyze its worst-case complexity and compare it to a simple algorithm and to a theoretical optimal algorithm.

Introduction

Conventional pattern matching methods determine if a given pattern is fully satisfied or not. In real-world domains these pattern matching approaches suffer heavily from uncertainty in the environment in form of typical noise or imprecision in the data. This leads to the natural conclusion that matching patterns must become a matter of degree [Dubois and Prade 1993]. Early work on this topic has been done in the context of fuzzy pattern matching in [Cayrol, Farreny, and Prade 1993] and, more recently, in the context of neuro-fuzzy systems, i.e., a combination of fuzzy logic and artificial neural networks [Oentaryo and Pasquier 2009].

Our application area is tolerant pattern matching for improved network security within security information and event management (SIEM) systems. SIEM systems are monitoring systems, which collect events from several sensors distributed in a computer network. These sensors are known as intrusion detection systems, which analyze, e.g., firewall logs, network streams or system health status to detect illegal intruders. The typical huge amount of events collected by a SIEM system is hard to accomplish by users. Therefore, correlation techniques based on pattern matching are typically used to reduce these huge amount of events to the most relevant ones. However, these patterns must be created or adapted regarding the conditions of the network. Due to constantly varying attacks and varying network configurations these patterns must deal with these changing conditions. Nowadays, SIEM systems are essential in most huge business computer networks. Therefore, the following approach can be seen as part of the Algorithmic Intelligence initiative of creating intelligent algorithms for real world problems.

In this paper an alternative to fuzzy pattern matching is proposed. A semantically well-defined and intuitive method is presented to calculate the degree of matching from patterns and data using a probabilistic composition of the patterns’ partially matching constraints. Furthermore, an efficient complete (and optionally approximate) algorithm is presented to calculate the degree of matching. The proposed method differs to fuzzy pattern matching in the way of how the degree of matching is calculated. We use ontological knowledge from description logic (and the inference of subsumption) without the need of specifying a characteristic function as usual in fuzzy pattern matching. This decreases the design efforts of appropriate fuzzy patterns, since the proposed method automatically derives the ”fuzzyness” from an ontology by still supporting most of the expressivity of description logic.

To our knowledge the most similar approach was proposed by [He et al. 2004] which also uses a similarity measurement in an ontology to handle noise in the matching process. However, in contrast, our approach does not need the specification of similarity weights. Additionally, this approach also describes how to handle disjunctions and negations of conditions (or constraints).

Another approach to use logical background-knowledge has been proposed by [Morin et al. 2009]. However, they used first order logic without any abstractions or approximate matching techniques. Instead, this paper is focused on using description logic (and ontologies) to model security background knowledge (e.g. [Undercoffer, Joshi, and Pinkston 2003], [Li and Tian 2010]).

The paper is structured as follows. First, a brief overview of description logic in the context of constraint satisfaction is given and the tolerant matching problem and the abstraction principle is introduced. Then, we turn to the divide-and-conquer algorithm for tolerant matching and analyze its
correctness and complexity. The paper closes with experiments and some concluding remarks.

Tolerant Pattern Matching in Description Logic

The proposed tolerant pattern matching approach is based on constraint satisfaction in description logic. Therefore, a brief introduction of Ontologies in DL is given in the following.

Ontologies and Description Logic

An ontological knowledge representation in description logic addresses the investigation “that knowledge should be represented by characterizing classes of objects and the relationships between them” (Baader and Nutt 2007, p. XV). While convenient databases focus on data persistancy, description logic focuses on modeling generic descriptions, building a formal framework for validation and categorization of data. Description logic is, therefore, applied to model knowledge rather than making data persistent. In other words, it is a formal language which is designed to represent knowledge in a more expressive way than propositional logic by still being decidable (in contrast to first order logic).

Ontologies in description logics consist of concepts, roles and individuals. Concepts represent classes of objects. Roles are binary relations between concepts. Individuals represent instances of classes (Gomez-Perez, Fernandez-Lopez, and Corcho 2004, p. 17). The semantic is defined by the interpretation (an interpretation can be regarded as the corresponding domain) of the set of all concept names, the set of all role names and the set of all names of the individuals (cf. Baader, Horrocks, and Sattler 2008, p. 140). A concept C is subsumed by a concept D if for all interpretations I we have C^I ⊆ D^I.

Tolerant pattern matching

In this work tolerant pattern matching is realized by successively generalizing the pattern and determining a remaining degree of satisfaction.

Definition 1 (Entity, Constraint, Satisfaction). An entity E is either (a) an individual, (b) a concept, or (c) a variable. A constraint γ ∈ Γ is defined as γ = e R e’ of a left hand side entity e, a right hand side entity e’ and a relation R between these entities. It is assumed that either e or e’ is an individual (fixed by an observation), or a variable. A constraint γ is satisfied if there exists an interpretation such that relation R holds for e and e’.

Definition 2 (Partially Matching Pattern, Degree of Matching). A pattern p consists of a set of constraints and logical compositions among them. A partially matching pattern p – given the data x – is a real valued function with range [0, 1]. The value of such a function is called degree of matching or matching degree.

\[
p(x) = \begin{cases} 1, & \text{if the pattern fully matches} \\ \alpha \in [0, 1], & \text{if the pattern matches to degree } \alpha \\ 0, & \text{otherwise} \end{cases}
\]

Each constraint in a pattern can be expressed as a query triple in description logic (DL). This allows an easy transformation of patterns into a query language like SPARQL, which can be interpreted by DL reasoners.

Example 1 Let γ_1 be a constraint saying that an observation with the attribute value of product is the same as the individual apple and γ_2 be a constraint saying that the attribute value of product must be the same as the individual pear. Furthermore, the pattern p is specified as γ_1 \lor γ_2. Considering a query “Is pattern p satisfied for product = banana?” the pattern is transformed to SPARQL as

\[
\{\text{ns:banana owl:sameAs ns:apple} \} \cup \{\text{ns:banana owl:sameAs ns:pear} \}
\]

which is obviously not satisfied since a banana is not the same as an apple or a pear.

However, the pattern from the example describes fruits and if no other pattern is satisfied this may give us a good hint of the kind of data presented. This can be easily achieved by using subsumption, i.e. each pear, banana and apple may be subsumed by a concept called “fruit”. E.g. γ_1, γ_2 or both could be abstracted to the condition that the banana must be a fruit instead of being an apple or pear. Each of these abstractions of the pattern will be satisfied. However, the smallest abstraction is desired to maintain as most semantic of the original pattern, i.e., either γ_1 or equivalently γ_2 should be abstracted but not both.

Next, a relation \text{\geq}_g (adapted from Defourneaux and Peltier 1997) describes that a constraint is an abstraction of another constraint. We say γ_1 is more general or equal than γ_2 (notated γ_1 \text{\geq}_g γ_2) if for all interpretations γ_2^I of γ_2 there exists an interpretation γ_1^I of γ_1 such that γ_2^I ⊆ γ_1^I.

In the following we use superscripts to enumerate different levels of specialization. A zero denotes the most abstracted case, while a larger number indicates an increasing specialization, e.g., p^1 is the direct abstraction of p^0. Fig. 1 shows an example of a pattern p^1 with three constraints γ_1^1, γ_2^1, γ_3^1 and their direct abstractions γ_1^0, γ_2^0, γ_3^0.

To abstract a pattern it is first necessary to propagate all negations to the leaves (i.e., the constraint triples) by applying De Morgan’s rules. The construction of this negational normal form can be done before abstracting a pattern or be applied to all patterns in advance (even to the most specific ones). Moreover, each constraint must be abstracted by the following rules:

- A negated constraint is abstracted to a tautology, since the current model of such a constraint includes all individuals except of the negated one, e.g. γ_1^0 in the example has changed to a tautology. In other words, all individuals

\[1\]An overview over several knowledge representations can be found in (Kubat, Bratko, and Michalski 1996).

\[2\]As we will see later, the degree of matching is determined by a fusion function F.
are valid results except of the negated one. Abstracting this (to extend the set of valid results) must include the negated individual and, therefore, is a tautology.

- If the entity of the abstracted constraint is a concept or an individual this is replaced by a more general concept due to the definition of $\geq g$.
- The relation of the abstracted constraint might have to be replaced to ensure that the set of interpretations of the constraint increases, e.g., the identity relation must be exchanged to an appropriate (transitive) subclass relation.

**Measuring Abstraction**

In the following, a measure $\theta(\gamma^j, \gamma^k)$ for constraints $\gamma^j$ and $\gamma^k$ is assumed to quantify the similarity of an abstracted constraint $\gamma$ from the original level $j$ to an abstracted level $k$. A simple example of such a measure is $\theta(\gamma^j, \gamma^k) = 1/|j-k| + 1$. We write $\gamma_{\pm}$ for the original constraint on the most specific level $\bot$, and $\theta(\gamma^j, \gamma^k)$ for $\theta(\gamma^j, \gamma_{\pm})$. Independent of a concrete measurement, such a measurement is assumed to be 1, if the constraint is not abstracted, and decreases, if the constraint is getting more abstract; by still being greater than or equal to 0.

This measurement can also be regarded as a similarity function, which says how exactly $\gamma^j$ describes $\gamma^k$, or how similar they are. The properties of a similarity function are assumed to hold for the computation of the degree of matching of a pattern.

**Definition 3** (Similarity, extended from Fanizzi and d’Amato 2006 and Batagelj and Bren 1995). A similarity measure $\theta$ is a real-valued function into $[0, 1]$ defined by the following properties:

- $\forall \gamma^j, \gamma^k: \theta(\gamma^j, \gamma^k) \geq 0$ (positive definiteness)
- $\forall \gamma^j, \gamma^k: \theta(\gamma^j, \gamma^k) = \theta(\gamma^k, \gamma^j)$ (symmetry)
- $\forall \gamma^j, \gamma^k, \gamma^{k+1}: \theta(\gamma^j, \gamma^{k+1}) \leq \theta(\gamma^j, \gamma^k)$ (identity)
- $\forall j < k: \theta(\gamma^j, \gamma^{k+1}) < \theta(\gamma^j, \gamma^k)$ (monotonicity)

Such similarity function values of the constraints are combined to a matching degree of the pattern by applying some fusion operator $F(\theta_1, \ldots, \theta_n)$ similar to fuzzy pattern matching (cf. Cadenas, Garrido, and Herndndez 2005). This is necessary to consider the semantics of the logical operators while abstracting the pattern. Due to the absence of a continuous membership function, a different fusion operator (as the proposed multiplication, minimum, average or fuzzy integral for fuzzy pattern matching) is used. Therefore, a probabilistic fusion approach is suggested by using a Bayesian interpretation of the tree of logical operators in each pattern as follows.

**Definition 4** (Fusion Function). The fusion function $F$ of pattern $p$ is recursively defined with respect to some similarity function $\theta$ of constraints $\gamma$ composed by logical operators.

$$F(\gamma^1 \land \gamma^2) = F(\gamma^1) \cdot F(\gamma^2)$$
$$F(\gamma^1 \lor \gamma^2) = 1 - (1 - F(\gamma^1)) \cdot (1 - F(\gamma^2))$$
$$F(\neg \gamma^i) = \left\{ \begin{array}{ll} 1 - F(\gamma^i), & \text{for } i = 1 \\ \beta \cdot F(\gamma^i), & \text{otherwise} \end{array} \right.$$  

\( \beta \) is penalty factor to additionally penalize the abstraction of negations, since these may have a greater impact on the result.

The factor $\beta$ may depend on the used similarity function and the depth of the ontology.

The fusion function’s conjunction/disjunction can be regarded as deterministic AND/OR node. Therefore, the conditional probability table is fully specified. By the assumption of independent constraints the result can efficiently be calculated.

As an example the pattern $p^4$ from Fig. 1 is represented as $F(p^4) = F(\gamma_1^1) \cdot (1 - (1 - (1 - F(\gamma_2^3))) \cdot (1 - F(\gamma_2^4))))$.

The negation is differently interpreted than in a corresponding Bayesian conditional probability table to ensure that an increasing abstraction leads to a decreasing fusion function. This different interpretation results from the negation of an abstraction being a specialization. However, due to our interpretation of abstraction that the set of solutions (models) of a constraint always increases (which is the complement of negation/specialization) we have to use the complement of the Bayesian interpretation for the negation as well. This leads to the monotonicity of $F$ with respect to $\theta$, which is very useful for finding the most specific abstraction as we will see later.

With these properties a partial order of patterns with respect to the generality of their containing constraints is defined. From this basis it is necessary to find the best matching pattern, i.e., the pattern with the biggest $F$. This problem can be postulated for a pattern $p$ containing $d$ constraints $\gamma^{d_1}, \ldots, \gamma^{d_d}$ to find a combination of $x_1, \ldots, x_d$ which satisfies the pattern and maximizes $F$. The solution of interest is in the Pareto front of maximum $x$ due to the monotonicity of $F$ with respect to $\theta$ and the monotonicity of $\theta$ itself (cf. Def. 4). If the level of specialization increases, $F$ increases as well, or – in other words – if any constraint of the pattern is abstracted, $F$ decreases. The next section presents an efficient algorithm to find this Pareto front that includes the most specific solution.

\( ^3 \)These equations naturally result from a Bayesian network (except of the negation) with conditional probability tables equal to the truth table of the corresponding logical operators.
Example 2 For the case of network security, in the example from Fig.\(\text{2}\)\(\gamma_1\) might be a constraint indicating that a port-scan event has been received by the SIEM, \(\gamma_2\) that this event’s source IP is extracted from the internal network and \(\gamma_3\) that the event’s source IP belongs to the administrators PC. Regarding the logical compositions, therefore, this pattern matches on a port-scan from a host of the internal network, which is not an administrator PC. Or in other words this pattern should ensure that no PC of the internal network should perform port-scans except of the administrators PC. This pattern is not intuitive due to the multiple negations, however, it is a good example to show the influence of the fusion function. Now considering a ping event instead of a port-scan event, which relates to a very similar situation, since this is also a reconnaissance event from a possible intruder and, therefore, may be structurally very close in the ontology, e.g., ping and port-scan might be summarized by the concept reconnaissance event. The described pattern is not fulfilled, since this pattern was only designed to match port-scans. However, with tolerant pattern matching the pattern can be abstracted with respect to \(\gamma_1\) in order to include the ping event, e.g., \(\theta(\gamma_1) = 0.5\). Therefore, the pattern is assumed to match by 0.5, since all other constraints are satisfied. Now considering that the pinging PC is the administrator’s PC, this constraint must be abstracted too, e.g., \(\theta(\gamma_3) = 0.1\) (due to \(\beta = 0.2\)). This results in an overall rating of just 0.05, indicating that the pattern is far from being satisfied. If one of these constraints must further be abstracted, the degree of matching decreases.

This measurement is of course highly dependent to the used similarity function and to an appropriate ontology. There are several alternatives of similarity functions in ontologies, but, unfortunately, tackling this issue appropriately is out of scope of this paper. The reader is encouraged to make further experiments with the fusion function to get an impression of the intuitive measurement of logically composed constraints in a Bayesian way.

**Divide-and-Conquer Algorithm**

In this section a divide and conquer algorithm is proposed to efficiently search for most specific satisfied patterns, which correspond to the Pareto front of the constraint abstractions. Each level of abstraction of a constraint is represented as one dimension of the search space. The search space \(X = \{0, \ldots, n-1\}^d\) is divided into satisfied elements (satisfied constraint combinations) \(X^+ \subseteq X\) and unsatisfied elements \(X^- \subseteq X\) with \(X^+ \cap X^- = \emptyset\).

**Example 3** Fig.\(\text{2}\) gives an example how the algorithm works for the two dimensional case (i.e. for \(\gamma_1\) and \(\gamma_2\)). At first the middle of the search space is determined, i.e. point (4, 4). Around this point the search space is divided into (in the two dimensional case) four equal sized areas each including the middle and the corresponding border elements. Two of these areas are marked with a gray background the others as area A and area B. The minus sign at (4, 4) indicates that the pattern with \(\gamma_1^+\) and \(\gamma_2^-\) is unsatisfied, the circle indicates an inference call to test this satisfaction. Therefore all more specific pattern combinations are omitted in the further recursion, i.e. area A. This method is continued for the gray areas but at first for area B. Area B is divided into four equal sized areas around the middle (2, 2). This is a satisfied match therefore we know that each more abstract pattern than \(\gamma_1^2\), \(\gamma_2^3\) is also satisfied, marked as area C which can be omitted in the following. The recursion is continued for the new middle (3, 3). At this point an unsatisfied area can be determined which also affects the currently not investigated gray areas due to we know that from (3, 3) to (8, 8) every solution must be unsatisfied because they are more or equal specific. These temporary results are stored in a list and checked before investigating the gray areas in subsequent recursion steps to omit inference calls for these points.

Please note that the algorithm may be limited in the search space to give approximate results. By increasing the search depth the solution is more and more appropriately approximated.

Algorithm \(\text{1}\) shows the full implementation of the approach. This algorithm is initialized with an empty set of solutions (representing the most specific satisfied patterns) \(S^+\) and \(S^-\) (representing the most abstract unsatisfied patterns). The individual search spaces are specified by a most specific bound \(\text{msb}\) and a most abstract bound \(\text{mab}\), where \(\text{msb}\) and \(\text{mab}\) are coordinates of the search space. Initially, for all \(i\) we have \(\text{msb}_i = 0\) and \(\text{mab}_i = n\) (to ensure completeness \(\text{mab}\) is located outside of the actual search space). For reasons of simplicity, each constraint is assumed to have an equal amount of specializations/abstractions, however the algorithm is also capable of differing amounts.

In Algorithm \(\text{1}\) we find \(\text{Eval}\), the call to the reasoner. The other method that enumerates the sublattice structure is called \(\text{Hypercubenodes}(\text{msb}, \text{mab})\) (without \(\text{msb}, \text{mab}\).
themselves), formally defined as
\[ \bigcup_{i=1}^{2^d-2} \text{msb} \otimes \text{bin}(i) + \text{mab} \otimes \overline{\text{bin}(i)}, \]
where \( \text{bin}(i) \) denotes the binary representation of a number \( i \), \( \overline{\text{bin}(i)} \) denotes its (first) complement, and \( \otimes \) the component-wise multiplication of two vectors.

Algorithm 1 Pareto

\begin{itemize}
  \item \textbf{Input:} Most specific bound \( \text{msb} \in X = \{x_1, \ldots, x_d\}^d \)
  
  \item Most abstract bound \( \text{mab} \in X = \{x_1, \ldots, x_d\}^d \)
  
  \item 1: \( m \leftarrow \lfloor (\text{mab} + \text{msb})/2 \rfloor \)
  
  \item 2: \text{CHECK IF RESULT IS ALREADY KNOWN}
  
  \item 3: if \( \exists x \in S^+ \) with \( (m \geq_g x) \) then
  
  \item 4: \( s = 1 \)
  
  \item 5: else
  
  \item 6: if \( \exists x \in S^- \) with \( (m \leq_g x) \) then
  
  \item 7: \( s = 0 \)
  
  \item 8: else
  
  \item 9: \( s = \text{Eval}(m) \)
  
  \item 10: if \( s = 1 \) then
  
  \item 11: \( S^+ = \{ x \in S^+ \cup \{m\} \mid \forall x' \in S^+ \cup \{m\} : x' \geq_g x \} \)
  
  \item 12: else
  
  \item 13: \( S^- = \{ x \in S^- \cup \{m\} \mid \forall x' \in S^- \cup \{m\} : x' \leq_g x \} \)
  
  \item 14: end if
  
  \item 15: end if
  
  \item 16: end if
  
  \item 17: // TERMINATION
  
  \item 18: if \( \text{mab} = m \) then
  
  \item 19: return
  
  \item 20: end if
  
  \item 21: if \( s = 1 \) then
  
  \item 22: // EXPLORATION IN MORE SPECIFIC DIRECTION
  
  \item 23: Pareto(msb, m)
  
  \item 24: end if
  
  \item 25: // EXPLORATION IN MORE ABSTRACT DIRECTION
  
  \item 26: Pareto(m, mab)
  
  \item 27: end if
  
  \item 28: // EXPLORATION IN REMAINING DIRECTIONS
  
  \item 29: for each \( h \in \text{Hypercubenodes}(\text{mab}, \text{mab}) \) do
  
  \item 30: for \( i = 1 \) to \( d \) do
  
  \item 31: \text{msb}' = \max\{h_i, m_i\}
  
  \item 32: \text{mab}' = \min\{h_i, m_i\}
  
  \item 33: end for
  
  \item 34: Pareto(msb', mab')
  
  \item 35: end for
\end{itemize}

The following definitions express the previous considerations transferred to the \( d \) dimensional search space \( X \) which can be interpreted as a coordinate system:

**Definition 5 (Domination).** Let
\[ \geq_g = \{(x, x') \in X^2 \mid \forall i (x_i \leq x'_i)\}. \]
We say that \( x \in X^- \) dominates \( x' \in X \) if \( x' \leq_g x \) and \( x \in X^+ \) dominates \( x' \in X \) if \( x' \geq_g x \).

All more specific patterns than an unsatisfied one are still unsatisfied and all more general patterns than a satisfied one are still satisfied. In other words, we have
\[ \forall x \in X^-, x' \in X. (x' \leq_g x) \Rightarrow x' \in X^- \]
and
\[ \forall x \in X^+, x' \in X. (x' \geq_g x) \Rightarrow x' \in X^+. \]

The algorithm computes the Pareto frontier:

**Definition 6 (Pareto Frontier).** The Pareto frontier is the set of extreme points \( E = E^+ \cup E^- \) with \( E^+ \cap E^- = \emptyset \) containing each element of \( X^+ \) with no element in \( X^- \) being more general
\[ E^+ = \{ x \in X^+ \mid \forall x' \in X^+ (x' \geq_g x) \} \]
and each element of \( X^- \) with no element in \( X^- \) being more specific
\[ E^- = \{ x \in X^- \mid \exists x' \in X^- (x' \leq_g x) \}. \]

No element in \( E \) is dominated by another element in this set, i.e., the compactest representation of the set of satisfied / unsatisfied solutions.

Next, we show that Algorithm 1 computes \( E^+ \).

**Theorem 1 (Correctness and Completeness of Algorithm 1).** The algorithm determines the whole set of satisfied constraints, i.e., \( E^+ = S^+ \).

**Proof. (Correctness)** To show the correctness of the algorithm we ensure that each element of the expected result set \( E^+ \) is in the solution set \( S^+ \) of the algorithm and, vice versa, i.e., \( e^+ \in E^+ \Rightarrow e^+ \in S^+ \) and \( s^+ \in S^+ \Rightarrow s^+ \in E^+ \).

**Lemma 1.** \( s^+ \in S^+ \Rightarrow s^+ \in E^+ \)

If the search is exhaustive (this is shown later) Line 11 implies that \( s^+ \in S^+ \Rightarrow s^+ \in E^+ \), since it computes \( S^+ \) as \( \{ x \in (S^+ \cup \{m\}) \mid \forall x' \in (S^+ \cup \{m\}) : x' \geq_g x \} \), which is the same as the expected result \( E^+ \) with \( S^+ \cup \{m\} \subseteq X^+ \).

**Lemma 2.** \( e^+ \in E^+ \Rightarrow e^+ \in S^+ \)

We investigate four conditions under which an element is inserted into (and kept in) the solution set of the algorithm \( S^+ \) in Line 11. These conditions, directly derived from the algorithm, are as follows

1. Line 10 implies that each element from \( S^+ \) must be contained in \( X^+ \) which is exactly the same condition as in definition of \( E^+ \).

2. The following assumption, derived from Lines 3 and 5, must hold for \( e^+ \) to be inserted into \( S^+ \)
\[ \neg \exists x' \in S^+. (e^+ \geq_g x'). \]

This condition is not fulfilled if an equivalent solution \( e^+ \) is already in the set \( S^+ \) or if \( e^+ \) dominates another element from \( S^+ \). In both cases \( e^+ \) is not inserted into the result set \( S^+ \).

3. The next statement, derived from Line 6 and 8, is
\[ \neg \exists x' \in S^-.(e^+ \leq_g x'). \]

This condition is always fulfilled, since we consider the case that \( e^+ \in X^+ \) and from eqn. 1 we know that this implies that \( x' \in X^+ \), which cannot be the case since \( x' \in S^- \subseteq X^- \).
4. Line 11 is does not drop solutions because for all \( m \in \mathbb{X}^+ \) we have eqn. \[1\]

Analogically, the proof can be made for \( E^- \).

Proof. (Completeness)
From Line 22 to 39 we obtain that the recursion is omitted for

- \( \{ x \in \mathbb{X} \mid \text{msb} \leq g \ x \leq g \ m \} \) if \( m \in \mathbb{X}^- \) and for
- \( \{ x \in \mathbb{X} \mid \ m \leq g \ x \leq g \ \text{mab} \} \) if \( m \in \mathbb{X}^+ \).

This does not affect the set of solutions due to the definition of domination and the definition of \( E \) that there should not be any value in the result set that is dominated by another element. Note that \( m \) has already been checked by the algorithm at this point.

The remaining space under investigation is getting smaller in each recursion path until \( m \) is getting equal to \( \text{mab} \) (the termination criteria from Line 18). This is only the case if each edge of the space under investigation is smaller or equal one. This can be derived from Line 1 of the algorithm. At some time in the recursion the space of possible solutions is divided into a set of spaces with edges of the length one or less by still covering the whole space of possible solutions as previously shown. Further if any point of such a smallest area is a possible solution (these are the corners), this point is under investigation in another space due to the recursive call with overlapping borders (cf. Lines 24, 27 and 35) except of the borders of the whole search space at the specific borders due to there is no \( \text{mab} \) of any area including these specific border elements, e.g., there is no \( \text{mab} \) for the one element area \( (8, 8) \) in the example from Fig. \[2\]. For this border case the algorithm is called with a lifted \( \text{msb} \) to ensure that the unlifted specific bound is included in some smallest (one element) area as \( \text{mab} \) visualized as a light gray border in Fig. \[2\]. Therefore, each element of the search space which is a possible solution is investigated as \( \text{mab} \) in some recursive path.

After computing the pareto front, the fusion function \( F \) is used on the remaining set of candidates to identify the most specific matching pattern abstraction.

**Complexity Considerations**

The worst-case running time is dominated by the number of calls to the reasoner. So we distinguish between the number of recursive calls \( T(n) \) and the number of inference calls \( C(n) \) (for the sake of simplicity, we assume \( n_1 = \ldots = n_d \) and \( n = 2^{k} \)). Of course, a trivial algorithm testing all possible elements in \( S \) induces \( C(n) = T(n) = O(n^d) \). We will see that the algorithm \text{Pareto} is considerably faster.

With \( \lg n \) we refer to the dual logarithm \( \log_2 n \), while \( \ln n \) refers to the natural logarithm \( \log_e n \).

**Recursive Calls**

Let us first consider the 2D case. The number of calls of the divide-and-conquer algorithm in a 0/1 \((n \times n)\) matrix is bounded by

\[
T(k) = \sum_{i=0}^{k} 3^i = \frac{3^{k+1} - 1}{2}
\]

Assuming \( n = 2^k \) we have

\[
T(n) = \left(3^{\lg n} - 1\right)/2 = \left(n^{\lg 3} - 1\right)/2 = O(n^{1.5849})
\]

Let us now consider the 3D case. The number of calls of the divide-and-conquer algorithm in a 0/1 \((n \times n \times n)\) (hyper-)cube is bounded by

\[
T(n) = \left(7^{\lg n} - 1\right)/2 = \left(n^{\lg 7} - 1\right)/2 = O(n^{2.8073})
\]

We also see that for larger dimensions \( d \) the complexities \( O(n^{\lg (2^d-1)}) \) rise. In the limit for large \( d \) the exponent to \( n \) converges to \( d \).

**Inference Calls**

Again, we first consider the 2D case. We observe that the structure of the recursion corresponds to find a binary search to the SAT/UNSAT boundary. The recursion depth is bounded by \( \lg n \). Therefore, the worst-case number of calls to the reasoner of the divide-and-conquer algorithm in a 0/1 \((n \times n)\) matrix is defined by

\[
C(n) = 2C(n/2) + O(\lg n)
\]

The \( O(\lg n) \) term is due to the binary search. In the worst case the boundary between SAT and UNSAT cells is in the middle, where one quarter of SAT and one quarter of UNSAT elements are omitted.

Using the Akra-Bazzi theorem (Akra and Bazzi 1998) (a generalization to the well-known master theorem), the above recursion can be shown to reduce to \( C(n) = O(n) \) as follows.

The Akra-Bazzi theorem (for \( k = 0 \)) states that for recurrence equation \( T(n) = g(n) + aT(n/b) \) with \( a = b^p \) we have the following closed form

\[
T(n) = O \left(n^p \cdot \left(1 + \int_1^n g(u)/u^{p+1} \, du\right)\right).
\]

Here, \( g(n) = \lg n = \ln n/\ln 2 \) and \( a = b = 2 \) so that \( p = 1 \) and

\[
T(n) = O \left(n + n \cdot \int_1^n \ln(u)/u^2 \, du\right) = O \left(n + n \cdot \left[-\ln u/u^{3/2}\right]\right) = O(n).
\]

Let us now consider the 3D case. The number of calls to the reasoner in the divide-and-conquer algorithm in a 0/1 \((n \times n \times n)\) (hyper-)cube is bounded by

\[
C(n) = 6C(n/2) + O(\lg n).
\]

Again the \( O(\lg n) \) term is due to the binary search. In the worst case the boundary between SAT and UNSAT cells is in
the middle, where one eighth of SAT and one eighth of UNSAT elements are neglected.

The recursion yield non-linear time complexity. Here, \( g(n) = \lg n \) and \( a = 6, b = 2 \) so that \( p = 2.5849 \) and

\[
T(n) = O\left(n^{2.5849} \cdot \left(1 + \int_1^n \frac{\ln(u)}{u^{0.5849}} \, du\right)\right)
\]

\[
= O\left(n^{2.5849} \cdot (1 + 0.3868 \ln 1 + 0.1496 - (0.3868 \ln n + 0.1496/n^{2.5849}))\right)
\]

\[
= O(n^{2.5849}).
\]

In the general case for \( d \) dimensions we have \( g(n) = \lg n \) and \( a = 2^d - 2, b = 2 \) so that \( p = \lg(2^d - 2) \) and

\[
T(n) = O\left(n^p \cdot \left(1 + \int_1^n \frac{\ln(u)}{u^{p+1}} \, du\right)\right)
\]

\[
= O\left(n^p \cdot (1 + p \ln 1 + 1 - (p \ln n + 1/(p^2 n^p)))\right)
\]

\[
= O(n^p) = n^{\lg(2^d - 2)}.
\]

We see that for larger dimensions \( d \) the complexities \( O(n^{\lg(2^d - 2)}) \) rise. In the limit for large \( d \) the exponent to \( n \) converges to \( d \).

**Evaluation**

We have evaluated the efficiency of the algorithm with respect to the number of inference calls.

In Fig. 3 two tolerant matching algorithms and the result of a perfect guessing algorithm (a lower bound) are visualized. It is assumed that the lower bound algorithm checks exactly the Pareto border of satisfied and unsatisfied elements. Therefore, the best possible algorithm needs at least \( |S^+| + |S^-| \) inference calls.

The proposed divide-and-conquer algorithm **Pareto** with pruning the recursive calls as in Algorithm 1 – but without using the lists \( S^+ \) and \( S^- \) – is called **Pareto-0**. This is the first efficient algorithm one might think of. The proposed algorithm is visualized as **Pareto** and the lower bound as **LOWERBOUND** for the two dimensional case in Fig. 3 (no log-scale) and for the four dimensional case in Fig. 4 (log-scale).

**Conclusion and Outlook**

In this paper we have shown how to use ontological background knowledge in the form of description logic to overcome the problem of noisy and imprecise data in the method of pattern matching (of logical terms). The proposed tolerant pattern matching can even address erroneous or missing patterns by successively abstracting them.

This kind of pattern matching suffers from the necessity to call a description logic reasoner several times, which might be - depending on the size and structure of the knowledge base - very time consuming. Therefore, this paper considered an efficient algorithm to reduce the amount of these inference calls. It has been shown that the presented algorithm is correct and complete. Moreover, it can be parameterized to have the ability to infer approximate results, yielding a number of inference calls near to an optimal lower bound.

By maintaining temporary result lists in form of a dictionary to prune the search, the algorithm has been shown to outperform a divide-and-conquer algorithm.

Both figures show that the inference calls of **Pareto** is near to the optimal lower bound **LOWERBOUND** and considerably better than the typical divide-and-conquer algorithm **Pareto-0** in both 2D and 4D. These results are reasonable for the proposed pattern matching algorithm due to the depth of an ontology being typically smaller than 30 and the patterns having typically a small amount of constraints. Furthermore, not all constraints share variables, so that some of the constraints can be checked independently to the full pattern by using a simple binary search.

It can be seen that the amount of pareto results, which is around the half of the **LOWERBOUND** value, is very small. For this set the degree of matching must be computed with respect to the fusion function to find the most optimal solution out of the set of Pareto-optimal solutions. This search can be done without to call the DL reasoner, since we already know that these solutions are satisfied.
References


