Morphological oriented and Fault tolerant Recognizing of template based natural Language Questions on combinatorial Question Space

Dieter Käppel

Georg-Simon-Ohm Nürnberg University of applied Sciences
Friedrich-Alexander-University Erlangen-Nürnberg [Department of Computer Science 8 (Artificial Intelligence)]
dieter.kaeppel@student.fh-nuernberg.de
drkaeppe@immd8.informatik.uni-erlangen.de
http://bfwspcb9.informatik.uni-erlangen.de/

Abstract. I introduce a new Algorithm mainly based on a modified version of the Smith-Watermann-Algorithm, which is normally used for genetic sequence comparison. I use this modified algorithm for recognizing template based natural language questions. The need for this algorithm is classifying and parameterisation of formal questions to natural language interfaces to databases. Methods will be presented to enable the reader to extract a number of parameters, fixed or varying, from a users question to the system followed by classifying the parameterised question to a given set of questions. The recognition process itself works on morphologic tables to support fault tolerance and permutations of the user input.

Keywords: Parsing of natural Language, Natural Language Interfaces to Databases, Template based Questions, Symbolic Information Processing, Fault tolerant Parsing, Morphological Parsing.

1 Introduction

Since long automated parsing and understanding of natural language is known as a non-trivial process [1] [2]. To break down the complexity of the process I nor try to understand the natural language user input neither to recognize the sense behind it. The natural language interfaces to databases are some approach to exactly define what the interface understands and what not. Considering a database as a closed world, the parameters of the users question can be reduced to the attributes and their values taken from the database. Common knowledge can be considered as far as knowledge is part of the database. The recognition of template based natural language user questions is the outmost part of a natural language interface to a database. The focus of this article is not on the post processing of the resulting formal template representations, which are discussed in [3] and [4].

The advantage of the newly introduced algorithm in contrast to existing algorithms for parsing natural language questions is the fault tolerance and ability of morphological comparison. It is even possible for the algorithm to recognize multi word attributes in the natural language user questions.

2 Recognizing of template based natural language questions

Recognizing of template based natural language questions allow the user to formulate his questions to a given database in his language including aliases for technical and database specific terms. The presented algorithm mainly takes a list of question templates and a list of question attributes as parameters and results in a list of most possibly detected attributes and the most possibly question depending on the natural language user question, which also is a given parameter to the algorithm.

2.1 Algorithm parameters

On the one hand, the algorithm parameters are used to specify the combinatorial question space. On the other hand, the natural language user question the database is given as a parameter to the algorithm.

2.1.1 Question template list

Looking for a most efficient form for the list of the question templates, I suggest to use the following, presented by examples:

What differences are between <arg1> and <arg2>?
What dependencies can be found between <arg1> and <arg2>?
What inductions can be given from <arg1> to <arg2>?
What ordered patterns of <arg1> can be found in <arg2>?
What common occurrences of <arg1> can be found in <arg2>?
What prediction can be given to <arg1>?

Using Extended Backus Naur Form (EBNF), the definitions are:

QuestionTemplate ::= [QuestionPart] { Argument [QuestionPart] } '?'

QuestionPart ::= Word { Word }

Argument ::= '<' ArgumentName '>'

White spaces are not modelled here. *QuestionPart* are defined to be the parts not relevant to the arguments of the questions and are not substituted within the algorithm. *Argument* are defined to be the template arguments of the questions, each with a unique name to distinguish them in further processing steps.

2.1.2 Question attributes list

The attributes for using as question template argument substitutions are given as a simple list. The standard algorithm does not support recursive substitution of the template parameter lists.

2.2 Algorithm result

The algorithms result consists of the most possible question template from one of the given list as defined in 2.1.1 and a list of recognized question attributes as defined in 2.1.2. There exists two modes in recognizing the number of given arguments in the users natural language question, which will further described in 2.3. Independently from the mode the algorithm is running, the resulting list of question arguments is a attribute-value-pair-list, where each attribute is the *ArgumentName* from corresponding argument given by the question template and each value is the extraction from the users given natural language question matched to the most possible attribute from the attribute list.

For example assume the given natural language user question is '*Are there differences between catz and dogz?*' Further, assume the list of possible question templates given in 2.1.1 applies and the question attributes list is *birds*, *cats*, *dogs* and *monkeys*. Then the result will be:

What differences are between <arg1> and <arg2>?

and

arg1=*cats*
arg2=*dogs*

The result can be used to regenerate the question that is most possible to be meant by the user by substituting the argument list into the most possible question template to:

What differences are between cats and dogs?

That not only recognizes the natural language user question to the system, it also corrects the misspelled or malformed question to the well-formed questions given by the question templates.

2.3 Algorithm modes

There are two modes to run the algorithm. The first is, for each question template to find the corresponding number of arguments, substituting them in the template and to compare the template with the natural language user question. The second is first to determine the number of arguments and then only use these question templates for comparing which matches the determined number of arguments. The danger of the second mode is to recognize arguments in the natural language user question, which are part of the template. This will result in the disadvantage of not considering question templates with less arguments and finally not finding the best possible question template.

2.4 Algorithm description

As discussed, the algorithm is mainly divided in two steps. Here, referring to 2.2, the algorithm of mode one is explicitly described. For each question template as described in 2.1.1 we have to search the question attributes in the natural language user question. Hereby the question attributes are given as described in 2.1.2.

2.4.1 Morphological transformation

First, all involved character strings have to be transformed into morphological strings. For an even simpler implementation if the algorithm you can skip this step of the algorithm and directly work on the characters. In this case, whenever talking about morphemes you can read characters. However, beware that not even equivalence between phonemically equivalent characters will be recognized by the following steps and additionally more morphemes must be compared because the reduction effect is not present. This results in a drastically worse performance both in time and in quality.

The transformation process reads the given character string from left to right and outputs for each highest valued morpheme a symbol representing that morpheme. If no entry in the table beginning with the actual character in the input stream is found, the character is directly copied to the output stream and the transformation is continued at the next character.

It is useful to enumerate all known morphemes. The morphemes values should be non-negative numbers, usually the higher the more characters the morpheme counts and the less possible the occurrence of the morpheme is. It is very heuristic to create this table.

2.4.2 Question attribute search

The Smith-Watermann-Algorithm will be used to find the best fitting question attribute. The algorithm results in a numeric value representing the equivalence of two morphological strings. The extension of the original algorithm is the detection of the begin and the end of the search string in the search space. This is necessary to remove the possibly misspelled attribute and allow successive searching next needed attributes. When a found attribute is removed, it is replaced in the natural language user question by a delimiter morpheme to prevent the successive steps finding 'enwrapping' attributes with some morphemes on the left of the removed attribute and some morphemes on the right.

To find the end of the search string, the maximum value of the propagated value of the last search morpheme is searched in the search space. To find the begin of the search string, the algorithm is evaluated a second time on the reverse search string and reverse search space. Equal finding the end of the string, the begin is found at the maximum propagated value in the reverse search space.

2.4.3 Question template search

If the number of needed arguments for one corresponding question template is not found, this question cannot be the users given question. All following templates with equal or more numbers in arguments can be skipped for efficiency reasons. If found, the related template is filled with the original attributes from the question attributes list. This is done by replacing the *<arg1>*, *<arg2>*, ... by the transformed morphemes from the question attributes.

After the preparing steps, the natural language user question and the natural language test question are compared, both in morphological form, by applying the Smith-Watermann-Algorithm a second time. The resulting value from the algorithm is the final equivalence between the tested template and the user question.

When done these steps for all question templates, the one with the highest ranking in comparing with the user question is the question template, which has the highest possibility to match the natural language user question.

2.4.4 Configuring the Smith-Watermann-Algorithm

The Smith-Watermann-Algorithm supports a parameter named gap-penalty, in the following shortly named gap. The gap controls the ability to skip redundant morphemes both in the search string and in the search space. The higher the gap, the less two strings are considered equal, while having more unwanted morphemes.

For practical reasons it is useful to apply a higher gap to the question attribute run while applying a lower gap to the question template run. For the argument gap a value of 0.4 and for the template gap a value of 0.05 has been evaluated to be useful.

One more value can be configured, that indirectly results from the Smith-Watermann-Algorithm when using with natural language comparison: the acceptance. The acceptance is the value that the modified Smith-Watermann-Algorithm at least must result, that a question argument respectively a question template is recognized. Especially when configuring with question attributes, the value is important for not recognizing attributes where no attributes are. Using a too low value can result in finding templates with a high number of arguments in the natural language user question, which are not present. Working values are 0.5 for the argument acceptance and 5 for the template acceptance. Note that two different implementations of the Smith-Watermann-Algorithm to render these two values exist.

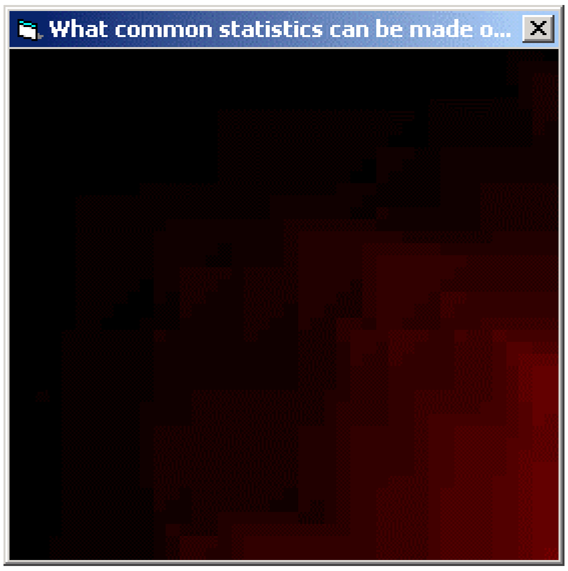
2.5 Algorithm analysis

Given two strings of characters one of the size n and the other of the size m both the memory and the time complexity of the Smith-Watermann-Algorithm are $O(n \cdot m)$. The memory complexity can be reduced to $O(n)$ or $O(m)$ if not the complete result matrix is stored. In every step of the algorithm only the actual and the last row/column is needed to propagate the values through the algorithm. The complexity also is linearly enlarged by the number of question attributes, by the number of question arguments and by the number of question templates. So the total time complexity is approximately $O(w^3)$ where w is the number of words totally used and exactly $O(c^5)$, where c is the number of characters totally used. The complexity related to the number of words cannot exactly be given, because of the variance in words per attribute.

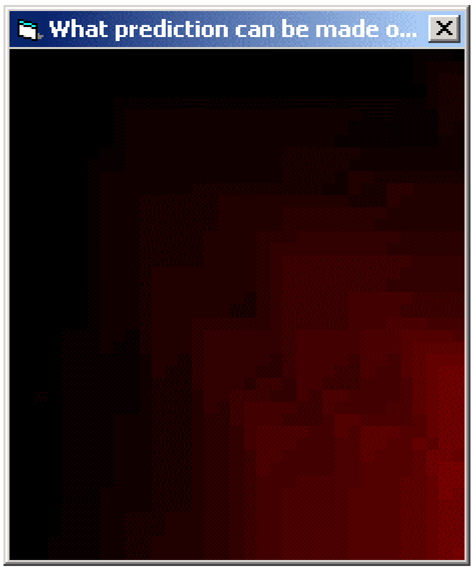
2.6 Algorithm evaluation

Following the example question template list, question attributes and natural language user question from 2.2 visualizations can be made from the template comparison using the Smith-Watermann-Algorithm. The darker the colour, the less the sequences of morphemes are corresponding at a defined position. The two-dimensional integral value of the total algorithm run is a practically good value to determine the equality of two morpheme sequences, as can be seen in the diagrams. All comparisons have been made against '*Are there differences between catz and dogz?*'

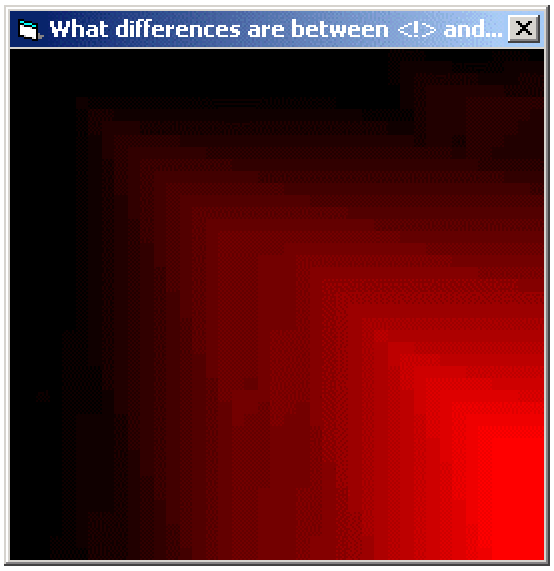
Visualization from comparison to "*What common statistics can be made on cats?*":



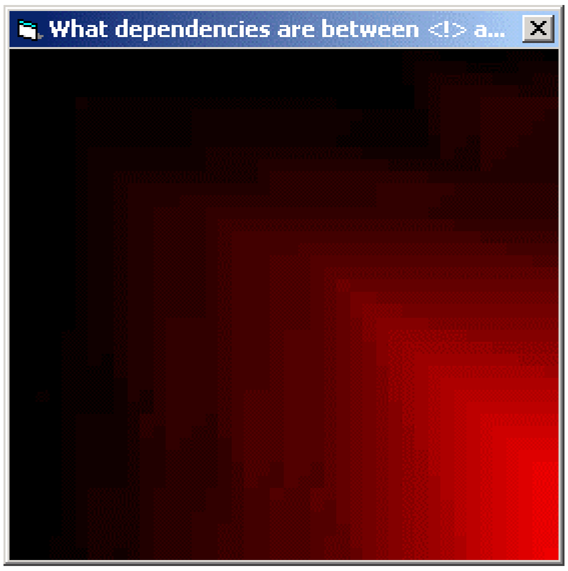
Visualization from comparison to "*What prediction can be made on cats?*":



Visualization from comparison to "*What differences are between cats and dogs?*":



Visualization from comparison to "*What dependencies are between cats and dogs?*":



It is obviously that "*What differences are between cats and dogs?*" is the best fitting question template, because it has the greatest areas of highlighted colouring in the diagram.

3 Conclusions and future work

The problems described in 2.2 lead to the use of the (less time efficient) first mode of the algorithm.

In next of kin to the extension of the algorithm, it seems to be possible to parse more complicated natural language questions through giving up the distinguish between question templates and question attributes by joining the non-terminals (question templates) and the terminals (question attributes) together to one list. Then substitution has to be done until questions only consisting of terminals are produced and can be compared with the given natural language user question. The problem will be to optimise the exponential complexity generated through the recursive substitution of the templates or part of them. A promising method should be the successive comparing and stop when no further substitution seems to be adequate.

Appendix – Code Listings

Public variables, common to all listed functions:

```
Public Head As Integer
Public Tail As Integer

Dim S() As Single, N As Integer, M As Integer
```

The Smith-Watermann-Algorithm searching the templates:

```
Function Distance(Space As String, Search As String, Gap As Single) As Single
    Dim i As Integer, j As Integer, v As Single
    N = Len(Search)
    M = Len(Space)
    Head = 0
    Tail = 0
    ReDim S(N, M)
    For i = 1 To N
        For j = 1 To M
            v = -0.2 - 1.2 * (Mid(Space, j, 1) = Mid(Search, i, 1))
            S(i, j) = max(S(i - 1, j) - Gap, S(i, j - 1) - Gap, S(i - 1, j - 1) + v)
            Distance = Distance + S(i, j)
        Next j
    Next i
    Distance = Distance / (N * M)
End Function
```

The Smith-Watermann-Algorithm searching the attributes:

```
Function DistanceExt(Space As String, Search As String, Gap As Single) As Single
    Dim i As Integer, j As Integer, v As Single
    N = Len(Search)
    M = Len(Space)
    Head = 0
    Tail = 0
    ReDim S(N, M)
    For i = 1 To N
        For j = 1 To M
            v = -0.2 - 1.2 * (Mid(Space, j, 1) = Mid(Search, i, 1))
            S(i, j) = max(S(i - 1, j) - Gap, S(i, j - 1) - Gap, S(i - 1, j - 1) + v)
        Next j
    Next i
    For i = 1 To M
        If S(N, i) > S(N, Tail) Then Tail = i
    Next i
    DistanceExt = S(N, Tail)
    For i = 1 To N
        For j = 1 To M
            v = -0.2 - 1.2 * (Mid(Space, M + 1 - j, 1) = Mid(Search, N + 1 - i, 1))
            S(i, j) = max(S(i - 1, j) - Gap, S(i, j - 1) - Gap, S(i - 1, j - 1) + v)
        Next j
    Next i
    For i = 1 To M
        If S(N, i) > S(N, Head) Then Head = i
    Next i
    Head = M + 1 - Head
    If Tail + 1 <= Head Then
        DistanceExt = -10000
        Exit Function
    End If
    DistanceExt = DistanceExt / max2(N, Tail - Head + 1)
End Function
```

The Function to replace the patterns:

```
Function ReplacePattern(ByRef Space As String, Search As Collection, _
                        Gap As Single, Accept As Single) As Pattern
    Dim act As Single, max As Single, Pattern As Variant, Tmp As String
    Dim User As String, Repl As String
    Dim Pos As Integer
    For Each Pattern In Search
        Tmp = Pattern
        act = DistanceExt(Space, Tmp, Gap) ' Gap = 0.4
        If act > max Then
            max = act
            User = Mid(Space, Head, Tail - Head + 1)
            Repl = Tmp
            Pos = Head
        End If
    Next Pattern
    If max < Accept Then Exit Function ' Accept = 0.3
    Set ReplacePattern = New Pattern
    ReplacePattern.User = User
    ReplacePattern.Pattern = Repl
    ReplacePattern.Pos = Pos
End Function
```

The functions to extract the patterns. Two functions are used, because the patterns are found in descending order of equivalence but they are needed in order of the question template arguments:

```
Private Function PatternsT(ByVal Space As String, Search As Collection, Gap As Single,
Accept As Single, MaxCount As Integer) As Collection
    Dim Str As String, i As Integer, Pattern As Pattern
    Set PatternsT = New Collection
    Do
        i = i + 1
        If MaxCount > 0 And i > MaxCount Then Exit Do
        Set Pattern = ReplacePattern(Space, Search, Gap, Accept)
        If Pattern Is Nothing Then Exit Do
        Space = Replace(Space, Pattern.User, "<!>")
        PatternsT.Add Pattern
    Loop
End Function

Public Function Patterns(Space As String, Search As Collection, Gap As Single, Accept
As Single, Optional MaxCount As Integer = -1) As Collection
    Dim List As Collection, max As Integer, i As Integer
    Set List = PatternsT(Space, Search, Gap, Accept, MaxCount)
    Set Patterns = New Collection
    While List.Count > 0
        max = 1
        For i = 2 To List.Count
            If List(i).Pos < List(max).Pos Then max = i
        Next i
        Patterns.Add List(max)
        List.Remove max
    Wend
End Function
```

References

[1] I. Androutsopoulos et al.: *Natural Language Interfaces to Databases – An Introduction* ; Journal of Natural Language Engineering, Cambridge University Press; Research Paper no. 709, Department of Artificial Intelligence, University of Edinburgh, 1994.

[2] W. A. Woods et al. The Lunar Sciences Natural Language Information System : Final Report. BBN Report 2378, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1972.

[3] O. Hogl: Konzeption und Realisierung eines Data-Mining-Front-Ends zur Konkretisierung von Benutzerinteressen und eines Data-Mining-Back-Ends zur Abstraktion von Data-Mining-Ergebnissen; Diplomarbeit Friedrich-Alexander-Universität Erlangen-Nürnberg, 1998.

[4] Hogl, O. et al.: On Supporting Medical Quality with Intelligent Data Mining, in: Sprague, R. (Hrsg.): Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Sciences (HICSS-01), Maui, Hawaii, IEEE Press, January 2001.