Natural Search Queries a machine learning approach to a search interface for consumer-oriented databases

Marek Lipczak
Evangelos Milios

Technical Report CS-2007-01

March 22, 2007

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

Natural Search Queries – a machine learning approach to a search interface for consumer-oriented databases.

Marek Lipczak[1] Evangelos Milios[2]

[1] Faculty of Electronics and Information Technologies, Warsaw University of Technology, Warsaw, Poland
[2] Faculty of Computer Science, Dalhousie University, Halifax, Canada, B3H 1W5

March 22, 2007

Abstract

Search in consumer-oriented databases is becoming increasingly important, as the computer becomes a commonly used tool. Such databases are at the heart of e-mail managers, flight booking, and other e-commerce systems. Key problems associated with such searches are the structure and interface of the search query. The traditional solution for these problems involves the use of a separate text field for each element of the query structure. However, the requirement to support ever increasing numbers of inexperienced users, who require an efficient and user-friendly interface, is not met by the traditional solution. We present Natural Search Queries (NSQ), a simple and intuitive approach to the search of structured information. Our solution combines the ideas of natural language database interfaces and operator based search; queries, in simplified and intuitive natural language, are entered into a single text field. It is a front-end search interface oriented towards the common user. Our aim is to allow as much freedom in formulating queries as possible, while interpreting such queries as accurately as possible, to automatically extract the elements of the query structure. In our project, we address the problem of e-mail databases, but the results may be applicable to other databases oriented towards consumer users. The report introduces the grammar of Natural Search Queries and probabilistic methods for recognizing the query structure (i.e., parsing, and Hidden Markov Model). In addition, we demonstrate a complete implementation of a system for processing NSQs and presenting retrieved messages. Specific subproblems that were addressed are the deterministic recognition of natural date constraints, and training of the models for query structure recognition. Tests show promising results in processing a broad range of Natural Search Queries.

1 Introduction

Access to structured information is becoming one of the most important and demanded tasks, due to increasing importance of databases in commerce, and new requirements for the World Wide Web. Information contained in the databases has a structure because it models structured world. Free-form data repositories are not able to reflect the world in a way that is easy to understand and process. The trend now is to create structure for such repositories. It can be easily noticed in the Web, where markup languages are used (e.g., XML), and researchers work on automatic or semi-automatic text structuring [5, 13]. This shift in information representation forces us to look for new processing methods. In the report we present a solution for one of the most common tasks – search.

In our project, we address the problem of search in consumer-oriented databases. The proposed ideas can be used in domains like flight-booking, e-library or e-commerce systems. We selected a specific type of database, electronic mail database, to create and evaluate a fully functional system for it. The e-mail database contains few parts, the most important being: subject, sender, recipients, date, attachment and text. Search process in consumer-oriented databases has one standard approach for the interface. Each part of the database has a corresponding control element (i.e., text field, list, radio buttons). This approach, with slight modifications, is known and commonly used for years – we refer to it as the traditional solution. As an example, the traditional interface from Oracle Ultra Search system is presented in Fig. 1. For non-experienced users the traditional solution is:

complicated and hard to use – user needs to use mouse and keyboard at the same time to move from one control element to another.

hard to access - a large number of text fields and other control elements are required, and often have to be hidden, to reduce the complexity of the default interface.

unable to present date constraints in intuitive form – date constraints are an important element of many consumer oriented database interfaces (e.g., e-mails, flight-booking systems). In traditional solutions the user is asked to choose a date from a calendar or specify date borders in two fields **before** and **after**. This date representation is limited and does not take into account date uncertainty. A more natural date description is desired (e.g., "three weeks ago", "between May and July").

too rigid in retrieval and presentation of results – traditional systems are based on Boolean search, only elements that fully match the query are retrieved. Users must create general queries and manually browse through unranked results, and are presented with a "no results found" message every time they misspell a word.

hard to connect with voice recognition – voice is increasingly becoming a standard input device. Traditional solutions are not well suited to voice input.

modified in different systems – search interfaces become more and more popular. A user looking for the cheapest flight visits different travel web sites. Although the query stays the same, there is no copy-paste possibility between sites.

The aim of the project is to create a simple and intuitive search interface that overcomes the limitations presented above. We decided to imitate the natural way search queries are formulated. Humans have already worked out a suitable solution for talking about structured information. Two example sentences presented in Table 1 show the method of formulating search questions. Looking closer into these questions, we discover that they were built from two types of words with specific functions. Words like *"from", "to", "about"* point out the structure element for which we want to specify constraints. We name these words **descriptors**. The constraints are specified by the second type of words – **keywords**: *"Boston", "Miami", "John", "project", "presentation"*. This pattern becomes more visible when we perform repeated searches and the sentences become simplified. Probably the first question we ask would be polite and grammatically correct, but the more questions we formulate the simpler they become, as presented in Table 1. This is a natural way people communicate and we claim that it is the most intuitive solution for formulating search queries in computer system. We named this approach *Natural Search Queries*.

Figure 1: Oracle Ultra Search [4] – an example of traditional search system.

flight booking
What are the available flights from Boston to Miami few days before Christmas?
Flights from New York to Detroit 3rd of July
Miami to Boston 10 days after New Year
e-mail search
Can you find me an e-mail I received from John five months ago, the e-mail is about project presentation?
about project presentation received from Mike last year
group meeting from Betty between May and July

Table 1: Example questions and corresponding Natural Search Queries. NSQ is a simplified and natural way of inter-human communication, as it is used to interact with a computer system.

2 Related work

Similar problems were recognized and solved in two different areas of computer science. Our project tries to combine the strengths of the two solutions.

Intuitive interfaces to databases, and among them, search in structured information have been a popular topic of research since the idea of artificial intelligence was born. A good overview of research area called Natural Language Interfaces to Databases is presented in [1]. NLIDBs are widely used for building interfaces to Question Answering systems. This area of research has been very active recently. Researchers concentrate on processing well-formulated and complicated natural language. Table 2 presents examples from [11, 22, 23]. One group of these interfaces [17, 11, 12, 15, 20, 22] tries to understand complicated questions written in one sentence. These approaches are mostly based on natural language parsers. Their authors focused on understanding complicated questions assuming the sentence was parsed correctly. NLIDB presented in [15] is one of the systems that consider words ambiguity – keywords may have multiple meanings. In this case, a statistical approach, which uses comparison between vectors of n-grams, is used. Another group of QA systems [2, 23] tries to create a human-computer dialog to receive and confirm all information

needed to process the query. The process unfolds step-by-step by small easy to recognize sentences.

NLIDBs are very interesting, but natural language is still too complex to be successfully used in commercial systems. In addition, we must consider if users really want to "talk" with a computer in natural language through dialog and grammatically complete and correct sentences. This is still an open question [21].

The problem of simplified textual access to structured information can be also solved without complicated tools of natural language processing. A widely known commercial project shows the problem from a different perspective. GMail[1] is a commercial system that was created to change the idea of managing e-mails. One of its main new features is "Google[2] based" search. GMail designers tried to overcome some of the limitations of traditional solutions. Users can search the body of e-mail messages through one search text field available on the main webpage. Other parts of the e-mail structure can be searched through a traditional search interface, accessible on a separate page. Date constraints can be formulated more intuitively, but still the number of predefined options is greatly limited. The designers probably considered easy access as the most important element of the search system; the main search field introduces another possibility of formulating structured queries – operators. Operators allow users to formulate textual representations of query structure. Comparing to the idea of Natural Search Queries, the operator is a greatly limited descriptor. Each structure part has one predefined operator. To define appropriate structure part each keyword must be preceded by an operator and a colon to avoid ambiguities. The concept of operators, called labels, is also used in XSEarch [6]. Here labels are not obligatory, however if the label is not specified the system will look for the keyword in all database structure parts.

Operator based interfaces are deterministic and simple to implement. Their drawback is the limitation it places on the user. Queries must match a strict pattern to be processed. The queries are limited (e.g., two operators for date constraints in GMail) and not intuitive.

By Natural Search Queries, we want to maintain or even increase the simplicity and intuitiveness of NLIDBs while preserving the high accuracy of operator-based interfaces. We believe this goal is achievable by addressing a limited and specific area of search in consumer-oriented databases.

Natural Language Interfaces to Databases	
Chat-80	*What is the capital of each country bordering the Baltic?*
Jupiter	*Can you tell me what's the weather like in Boston today?*
NaLIX	*Return every director who has directed as many movies as has Ron Howard.*
Operator Based Search Interfaces	
GMail	*from:John about:project*
XSEarch	*+title: , author:Vianu*
Natural Search Query Interfaces (possible use)	
E-mails	*from John about project presentation five months ago*
Tickets	*from Boston to Miami few days before Christmas*

Table 2: Natural Search Queries connects simplicity and intuitiveness of NLIDBs with high accuracy of operator based interfaces.

[1] http://mail.google.com/mail/help/intl/en/about.html
[2] http://www.google.com/intl/en/about.html

3 Formal definitions

In the introduction, we presented the general idea and examples of Natural Search Queries. Here we state the formal definitions.

Definition 1 (Natural Search Query (NSQ)) *Natural Search Query is a sentence of plain text that describes an entity of a structured model (e.g., message from an e-mail database). NSQ is built from query parts. The number of query parts and their order is chosen by the user. NSQ is natural to construct, and not required to be syntactically correct – queries do not have to follow natural language grammar.*

Definition 2 (Query part) *The expression of some part of the structured model (e.g., sender part in e-mail model) in a search query is called query part. It is made of two types of words: descriptors, and keywords. Query part is built according to a descriptors-keywords pattern. A part can be used in a query only once.*

Definition 3 (Query structure) *A set of query parts forms the query structure. Query structure of traditional search systems is explicit – each structure part is described separately. The Natural Search Query structure is hidden in the plain text of the query.*

Definition 4 (Descriptor) *A word that indicates which part of structured model is referred to by a query part is called descriptor. Query part can contain one descriptor (e.g., "from John"), or a phrase of descriptors (e.g., "send by John"). Descriptors are not mandatory and can be omitted from a query part.*

Definition 5 (Keyword) *A word that must be found in a specified part of an instance of structured model is called keyword. Query part must contain at least one keyword.*

Definition 6 (Descriptors-keywords pattern) *A model by which query parts are created is called descriptors-keywords pattern. A part is created by 0, 1, or more descriptors followed by 1 or more keywords. The date part does not follow the descriptors-keywords pattern strictly. The syntactic and semantic structure of naturally formulated date descriptions is more complicated.*

The definitions describe the general idea of Natural Search Queries, which is independent of the task it works on. To adapt NSQ to a specific database we have to define the query structure, which mirrors the database structure, and the set of descriptors. In our system, we simplify the e-mail structure and leave only four essential parts: "text" (subject and the body), "receivers", "sender", "date", for example Fig. 2. We define a descriptor set for each part. It is possible that a word is present in more than one set (e.g., "from" – can be used to describe sender and date).

part:	text part		sender part		receiver part		date part	
label:	text descriptor	text keyword	from descriptor	from keyword	to descriptor	to keyword	date descriptor operator	text keyword component
example:	about	project	from	Marek	to	John	last	month

Figure 2: An example query. Contains four possible parts of e-mail structure. Each part has two word functions – descriptors and keywords.

4 Problem formulation

The aim of the system is to recognize the hidden structure of Natural Search Queries. After that, we can associate keywords with the corresponding part of the database structure, and interpret date constraints. At this point, the retrieval process becomes as simple as for the traditional queries, where query structure is explicit. The proposed process is shown in Fig. 3. The problem of structure recognition is similar to part-of-speech tagging done for natural language [9]. In both cases, each word in a sentence has a hidden function. In part-of-speech tagging, these functions are described by POS tags (e.g., "noun", "preposition"). In the problem of NSQ recognition, we define labels that describe the role of each word in the hidden structure (Fig. 2). Both POS tagging and NSQ recognition need some additional information – rules by which the sentence is created; the rules together create a grammar. The NSQ recognition grammar was created independently of natural language grammar, based on specific features of search queries presented in section 3.

Comparison of Natural Search Queries and the traditional solution is not possible without creating a complete information retrieval system. Building such a system requires the solution to a number of additional subproblems:

natural date description interpretation – NSQ allows the user to enter date constraints in natural language (e.g., "3 months ago", "between May and July"). Date constraints must be translated to a form understandable by a computer program.

recognition of the correct structure – It is likely for the system to associate more than one possible structure with a query. The system must then decide which of these structures are most likely to be correct.

interface – The system uses simple "Google-like" interface, however some modifications are necessary due to the ambiguity of structure recognition.

training process – The system is based on a number of probabilistic parameters. We used training for setting up the parameters. Training can be also used while the system processes real users' queries. Users tend to enter queries according to individual habits. Training is able to reveal the pattern behind entered queries and tune the system to process these queries with higher accuracy for individual users.

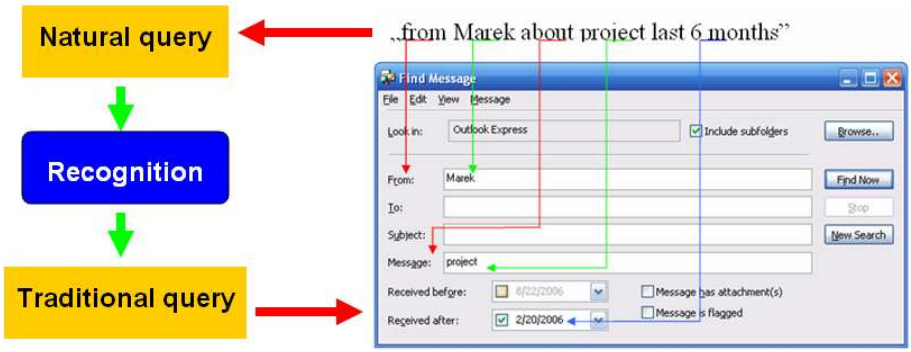

Figure 3: The aim of the system is to recognize the hidden structure of Natural Search Queries. After that, the retrieval process can be as simple as in the traditional solutions.

5 System Structure

System structure contains five steps. In the *Individual Word Labeling* step words are labeled independently of their context, based on knowing that some labels are more likely for particular words and patterns. Labels define the possible functions of the word in the NSQ structure. In the *Probabilistic Structure Recognition* all possible query structures are revealed, based on the context of words in the query. The *Date Recognition* step extracts a time window described by the user in natural language (e.g., "between May and July"). The *Results Ranking* step decides which of the recognized query structures are most likely to be correct, assuming that the answer to the query should be present in the database. Finally, in the *Retrieval* step the recognized NSQ structures and search results are presented to users.

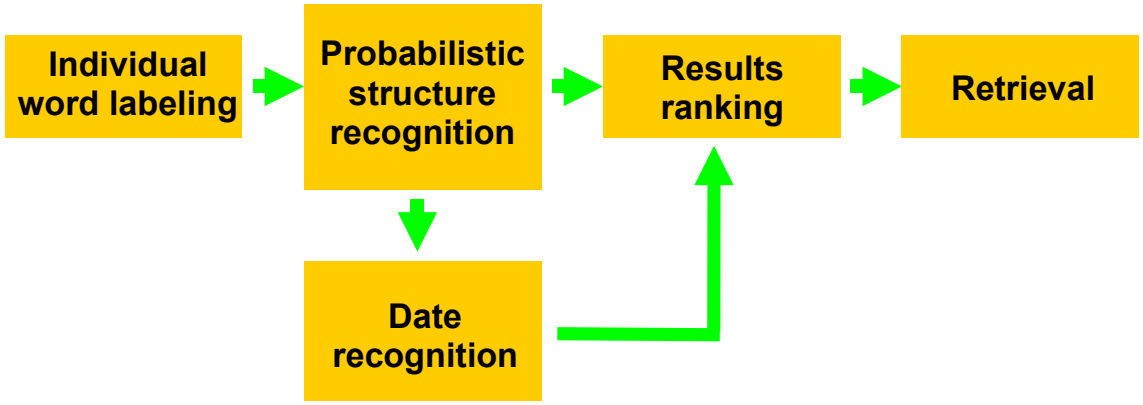

Figure 4: System structure. At each step we use different Natural Search Queries features to process search further. The main task of the system is to recognize the hidden structure of Natural Search Queries.

5.1 Individual Word Labeling (IWL)

Just like in the part-of-speech tagging problem, a given word may have different functions in different sentences. For example, a name can be once used to point out the receiver of a message "to **John** about project" or a sender of another message "from **John** about presentation". In the IWL step, the system finds all possible functions of word in a Natural Search Query; the functions are described by labels (Fig. 2). This step is the equivalent of dictionary lookup step in POS tagging problem [18]. Each word is processed individually and independently of other words in the query. We designed a set of simple methods to check which labels are appropriate for a given word. The methods look for a word in the lists of possible descriptors (see appendix A for the list of descriptors for each part). Some methods look for the word in the database structure to check if it can be a keyword. Finally some methods check if the word matches predefined keywords' patterns to catch dates (e.g., "21/01/2005") or e-mail addresses (see appendix A for the complete list of patterns, and Fig. 5 for the pseudocode of IWL step). Decisions made by different methods are not equally likely (e.g., if a word was found in the senders' group, and is also present in the text of some e-mails, the "from keyword" label is more likely to be correct – the text word is probably just a signature). Each function has a likelihood value, which reflects the probability of giving a correct label. Setting and modifying the likelihood values is described later together with the training process. After the labels are attached to a word the likelihoods are normalized, and the IWL step returns a label probability distribution for each word.

```
Input: Query built from words: word_1, word_2, ..., word_n.
Output: Query tokens sequence.
Definitions:
token – a word with a set of possible labels with their likelihoods.
This character format indicates terms defined in the report's text.

for each word
|   create token from word_k;
|   for each query part
|   |   if word_k exists in predefined set of descriptors
|   |   |   add proper descriptor label to token;
|   |   |   label.likelihood := DescriptorLikelihood;
|   |   if word_k exists in part of the database
|   |   |   add proper keyword label to token;
|   |   |   label.likelihood := KeywordLikelihood;
|   |   for each keywords' pattern
|   |   |   if word_k matches the pattern
|   |   |   |   add proper keyword label to token;
|   |   |   |   label.likelihood := PatternLikelihood;
|   normalize labels likelihoods;
```

Figure 5: Pseudocode of Individual Word Labeling step.

5.2 Probabilistic Structure Recognition (PSR)

This step is the core of the system, as it reveals the hidden structure of Natural Search Queries. The system decides what the functions of words in the query are. The query is treated as a whole sentence, and the system is aware of the NSQ grammar, i.e., the rules that have created the query. The labels given by the IWL step make the possible set of candidate structures smaller – for each word the PSR step considers only functions pointed out by the labels. The number of candidate structures is greatly reduced, although the queries can still be ambiguous (e.g., words from the set of descriptors can be used as text keywords "pictures **from** party"). The ambiguity makes the problem probabilistic in nature. The system does not have to answer the question *"is there a structure that may be represented by a given query?"*; the real question is *"which of many candidate structures is correct?"*. The PSR part does not answer this question definitively. Instead, it points out all sensible candidate structures and their grammatical likelihoods.

To maintain high accuracy and the ability to process a broad range of queries the system uses two complementary approaches. First, it needs to find all possible candidate structures that fully match the grammar using a probabilistic parsing algorithm. The second approach is to relax grammatical constraints, which decrease accuracy for well-defined queries, but allows the processing of extraordinary and misformulated queries. A Hidden Markov Model is used for this approach to the problem. Both subsystems work in parallel, and their results are later combined into one set of candidate structures, which is ranked in the Results Ranking step. The pseudocode of the PSR step is presented in Fig. 6.

Input: Query tokens sequence (tokenSequence). NSQ context free grammar.
Output: All possible query candidate structures (queryStructureSet).
Definitions:
tokenSet – created in IWL step. Set of tokens, each token is a word and its possible labels.
queryStructureSet – set of recognized query structures. Every query structure is a set of tokens with the proposed structural functions of words.
HMMstructure – Query structure recognized using Hidden Markov Model
`This character format indicates terms defined in the report's text.`

queryStructureSet := **do** CYK probabilistic parsing – *tokenSet* as `terminal rules`;
HMMstructure := **do** Viterbi algorithm – *tokenSet* as `evidence values`;
HMMstructure.likelihood := get median likelihood from *queryStructureSet*;
add *HMMstructure* to *queryStructureSet*;

Figure 6: Pseudocode of the Probabilistic Structure Recognition step.

5.2.1 Grammar

Although we do not know the hidden structure of Natural Search Query, we are able to reveal it based on rules by which the query was created. The formal definition of NSQ was presented in section 3. Here we present the general ideas of the grammar. The complete list of grammar rules is presented in appendix B. NSQ grammar rules can be divided into three stages: start, intermediate and terminal.

Start rules construct queries from different parts of the structure. At this stage, users have the most freedom of choice. The number of parts and their order in a query are not predefined. At the same time, each part can be used in the query only once. It makes the number of rules combinatorial in the number of part types. This is nevertheless computationally feasible, because the number of part types is small, and rules can be generated automatically.

"query − > from_part, to_part, text_part, date_part"
"query − > text_part, from_part"

Intermediate rules are based on descriptors-keywords pattern. The date part, as it is more complicated, has to be processed separately at this stage. The first group of the intermediate rules creates structure parts from descriptors and keywords. The second group connects words of the same nature together.

"from_part − > from_descriptors, from_keywords"
"from_part − > from_keywords"
"to_keywords − > to_keywords, to_keyword"
"to_keywords − > to_keyword"

Terminal rules constructed from labels, represent the connection between a single word and its function in the query. These rules are created individually for every query. Each word in a

particular query has only one function, however it may not be easy to choose from the few possible options. This is the reason why even for the particular query a word can be a product of more than one terminal rule. For example the query word "about" in the following two terminal rules:

"text_descriptor − > [about]"
"text_keyword − > [about]"

5.2.2 Parsing – CYK

The Cocke-Younger-Kasami (CYK) algorithm [19] verifies if a sentence can be generated by a given context-free grammar, and returns the grammar rules used in the generation process in the form of a parse tree. An example tree is shown in Fig. 7. The grammar of Natural Search Queries is deterministic. The non-determinism is caused only by multiple word labels. The labels become leaves in the parse tree. The tree chooses one label for each word. These labels are connected according to the NSQ grammar rules – they represent an integral query structure. Because of the labeling ambiguity, it is likely that a query can have many sensible structures. For example the query "Jan about meeting" can represent three different structures: Jan, abbreviation of January, can be an element of the date part, it can also be a person's first name used in sender or receiver part. After simple modifications, CYK is able to create a forest of all possible parse trees. In the NSQ recognition problem, the forest represents possible structures hidden in the query sentence.

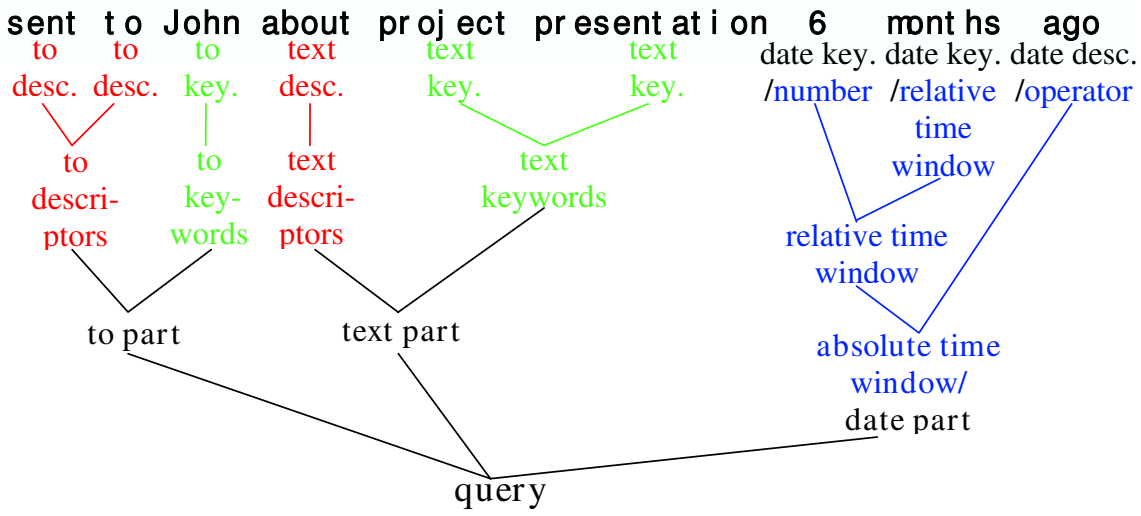

Figure 7: Sample parse tree for e-mail NSQ grammar. Connected leaves represent an integral query structure. At the intermediate stage date part is processed separately.

The parsing algorithm is able to recognize the structure only if the query strictly follows the grammar. This contradicts the need of processing a broad range of queries. The solution for processing misformulated or extraordinary queries (e.g., "GMail-like" queries) is to add a large number of additional rules to account for all possible misformulated queries. In this case, the system would finish with a large group of additional senseless structures. We decided to use another approach, a Hidden Markov Model, to process the extraordinary queries.

5.2.3 Hidden Markov Model

Hidden Markov Model [9, 16] has been successfully used in POS tagging [7]. Recognizing the NSQ structure is a similar problem, and we can presume that HMM can be used by analogy in our system.

HMM represents a group of unobservable states. The model switches from one state to another producing a visible outcome – the evidence. In our problem, we have a query sentence, which is a group of words. The task is to recognize the functions of given words. In the model, the state represents a function of a word in a sentence. The states mirror word labels (the labels are enumerated in Fig. 2). The word itself is the evidence. As the system scans two consecutive words in the query, it also switches between two states. Although the states are hidden, we have some probabilistic knowledge about them. We know how likely it is to switch from one state to another (transition model), and how likely it is to get a particular evidence value from a given state (evidence model). Model setting and tuning is described later, as part of the training process.

Creating the evidence model by collecting information about every possible word would not be feasible. We are able to simplify the evidence space thanks to the labels produced in the IWL step. The word is represented by its possible labels, which makes the evidence space equal to the state space. The IWL step returns words with probability distribution of its possible labels. It causes the extraordinary situation where a state can generate evidence with many different values. To handle this situation we had to modify the Hidden Markov Model. In the standard HMM, each evidence value is associated with the probability of its generation according to the evidence model. When we have multiple values, we have to combine their probabilities to come up with the probability of the evidence. The IWL step gives us the probability of label being correct. We use these probabilities as weights while combining the probabilities from the evidence model. The modification allows us to get one evidence probability value that corresponds to the uncertainty of the IWL step.

In the ideal situation, a state produces a word that has the same label. In reality, we have to face word ambiguity, which makes the problem of function recognition more complicated. Based on the sequence of outcomes and the two probabilistic models the system is able to find the most probable sequence of states that produced the outcome using the Viterbi algorithm [8]. This sequence of states is a ready-to-use NSQ structure recognition. The Hidden Markov Model works well for queries with part descriptors (e.g., "**from** John **about** project presentation"). The model is too simple to handle harder queries (e.g., "John project presentation"). It does not consider the query as a whole sentence, because the transition model gives only the information about the most recent state change. Although it may be considered as a disadvantage, this "short memory" allows HMM to handle extraordinary queries. The most important group of such queries are "GMail-like" queries ("from:John **text**:project **text**:presentation"). In this case the "descriptors-keywords" pattern is not strictly followed.

5.3 Training

The presented algorithms are based on a large number of probabilistic parameters. The solution for setting the parameters is to train them using a set of sample Natural Search Queries. Training can be applied to three groups of parameters: the function likelihoods in the Individual Word Labeling step; the rule probabilities for the parsing algorithm; the probabilistic models for the Hidden Markov Model. The training process initializes the parameters. Training can also happen on-line, while the system is processing real user queries. Then it offers the possibility to increase the system accuracy for each individual user. Users tend to enter queries that follow a well-defined pattern (i.e., queries that always contain a text part sometimes followed by a date part). Training

allows the system to adapt to such patterns. To tune the system for an individual user we need to receive feedback information about the recognition correctness. The solution for automatic construction of feedback information is presented later together with the Retrieval step (section 5.6.2). Through feedback, the system is given the correct query structure, namely unambiguous information about each word's function.

Probabilistic parameters reflect the statistical knowledge of all processed NSQ. The knowledge, as well as the parameters, must be updated after each query is processed (online training). The system uses the same update method for all three groups of parameters. Training is based on the intuition that, when an element (i.e., a function in IWL, a rule in parsing, or an element of transition or evidence model in HMM) is successfully used during query processing, it becomes more likely to be used in the future.

During the training process the system has access to the correct query structure. Each word is connected to a label that represents its real function in the query. This information is sufficient to find the labeling functions that have labeled the words correctly, to increase their likelihood. The labels also allow the rebuilding of the parse tree that represents the structure, while probabilities of rules used in the tree are also increased. Updating the parameters of the Hidden Markov Model is a more complicated task. The models are hidden, which means we do not have access to the sequence of states that have generated the outcome, and therefore no clues how to modify the models. In our system the training of HMM is based on the correlation between a visible outcome (represented by the labels) and a state that should have generated it in the ideal case. The correlation is described in section 5.2.3. It allows the system to update the transition model knowing only the evidence values. The training process is described in detail in appendix C.

5.4 Date Recognition

The PSR step finds words that a user might have used to describe date constraints (e.g., "from John **3 months ago**"). Date part contains a natural date description which cannot be processed simply by using "descriptors-keywords" pattern. The description must be interpreted and translated to a form, which is understandable to a computer program. We decided to translate each natural date description to a time window (Fig. 8). The time window is a period presented on the time axis. Finally, time windows are represented as probability distributions for retrieval purposes.

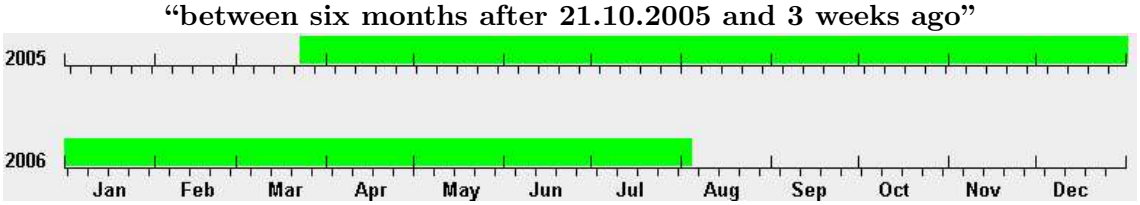

Figure 8: A sample date description in natural language and its representation as a time window. The Date Recognition step interprets natural date descriptions to time windows to make is understandable for a computer system.

To interpret a natural date description we represent it as a group of components and operators.

Definition 1 (Component) *A date description that can be represented on the time axis is called component. Basic component is created from a single word of the natural date description (e.g., "2006-03-08", "January"). During the interpretation process components are connected and modified by operators. To represent all possible date descriptions we defined three types of components: relative time window, relative time distance, and absolute time window.*

Definition 2 (Relative time window (RTW)) *RTW describes the length of the time window, but does not inform where is it placed on the time axis. For example expression "3 weeks" would be represented as a RTW.*

Definition 3 (Relative time distance (RTD)) *RTD describes the distance from a particular point on the time axis. For example expression "3 weeks after" would be represented as a RTD.*

Definition 4 (Absolute time window (ATW)) *AWT describes the interval on the time axis between two points representing start date and end date. The final interpretation of natural date description is presented in the ATW form. For example expressions "3 weeks after June", and "3 weeks ago" would be represented as a ATW.*

Definition 5 (Operator) *A word that indicates a way components should be modified is called operator (e.g., "after", "last"). Each operator has an associated function, which defines how the components should be processed.*

We present the solution for recognizing the natural date descriptions in electronic mail search problem. However, after modification of component and operator sets, the solution can be used generally for other types of consumer-oriented databases.

5.4.1 Date part – processing

The interpretation of the natural date descriptions is done in two steps. First, we connect each word from the natural date description with an appropriate basic component or operator function. The next step is to process iteratively the components using the operator functions. Finally we obtain a component that represents the given natural date description. The two main problems of this process are defining the order of application of the operators, and connecting components with the proper operator. We solved both problems by defining a grammar. Each operator is described by a grammar rule that contains information about the order and type of components that can be processed by operator's function. Additionally, the grammar rule informs what component type will be returned by the operator's function. Sample rules are presented in Table 3. The parse tree, created by the CYK algorithm, indicates the order by which the operators should be used (Fig. 9). The complete list of grammar rules and the example of natural date description interpreting is presented in appendix D.

Sample operator rules
`AFTER_OPR --> [after]`
`AFTER_OPR --> [from]`
`ATW --> AFTER_OPR ATW`
`RTD --> RTW AFTER_OPR`
Sample terminal rules, created independently for every query
`ATW --> [03/21/2002]`
`ATW --> [January]`

OPR stands for operator.

Table 3: Sample date recognition grammar rules.

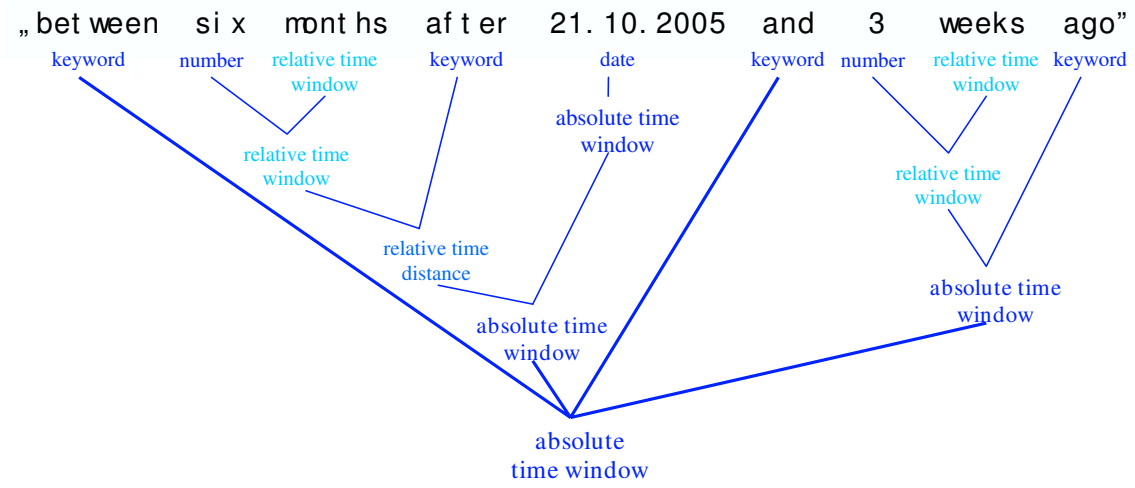

Figure 9: A sample parse tree for natural date description, shows the order by which the components are processed.

5.4.2 Date part – retrieval

If the date description is correct and sensible, the system returns an absolute time window, which represents date boundaries. Similar boundaries are explicitly defined by the user in the traditional solutions (“before:”, “after:”). The date recognition grammar is deterministic, therefore only one answer to given natural date description can be returned. Absolute time window can be represented as a probability distribution where the probability of dates within the borders is uniform and greater than zero, and probability of dates outside the borders is equal to zero. This representation can be useful, however in many cases it is too rigid. The system needs more flexible constraints that take into account user’s uncertainty. Intuitively, when a user wants to retrieve e-mails that were sent for example 6 weeks ago, there is also a chance that the relevant message was sent 5 or 7 weeks ago. The farther off an absolute time window, the smaller is the chance that the relevant e-mail can be found. Gaussian distribution is the intuitive choice to represent user’s uncertainty. The uncertainty grows with time distance – a user is less certain of the date if the message was sent long time ago. This problem is solved by increasing the distribution standard deviation with the time distance growth. Both distributions, uniform and Gaussian, are important and should be combined. More details about calculating the probability distribution are presented in Fig. 10.

5.5 Ranking the results

The Probabilistic Structure Recognition step produces all sensible candidate query structures. The number of these structures is usually large because of the labeling ambiguities, therefore the system must rank the results. The Results Ranking step is introduced to limit the number of candidate structures presented to the user. The step returns the three candidates that are most likely to be correct. In addition, at this step the ranking of e-mail messages, retrieved by candidate query structures, is performed. We present the two score schemes used in the system. The e-mail relevance score describes how well the e-mail matches the Natural Search Query. The structure recognition score shows which NSQ candidate is most likely to be correct.

Input: Set of recognized query structure candidates (queryStructureSet).
Output: Probability distributions corresponding to natural date descriptions made from date tokens.
Definitions:
dateTokenSet – set of tokens that contain date operators or components used in the translation process
TW_B – time window beginning day. (integer number)
TW_E – time window ending day. (integer number)
TODAY – today's day. (integer number)
`This character format indicates terms defined in the report's text.`

for each *queryStructure* that contains `tokens` that were recognized as date words
| **for each** `token` with date word
| | **if** `date part descriptor` token
| | | create token with appropriate `operator`;
| | **else** //`date part keyword` token
| | | create token with appropriate `component`;
| | add the token to *dateTokenSet*;
| `date parse tree` := do CYK parsing – *dateTokenSet* as terminal rules;
| `timeWindow` := recursively solve `date parse tree`;
|
| uniformProbDist := set function $\frac{1}{TW_E - TW_B}$ for days between TW_B and TW_E;
|
| mean := $\frac{TW_E - TW_B}{2}$;
| Std := $0.4 + 0.05 * (\text{TODAY} - mean)$;
| gaussianProbDist := set function Gaussian(mean, Std);
|
| **for each** day in timeWindow
| | day.likelihood := uniformProbDist(day) + gaussianProbDist(day);
| scale day likelihoods so maximal value is 1.0;

Figure 10: Pseudocode of Date Recognition step.

E-mail relevance score There are some factors, which may cause a relevant message not to fully match the query. Misformulated or incorrectly recognized queries can still be useful if the mistakes were minor. The system must also take into account date uncertainty. Boolean search, used in traditional systems, retrieves only the documents that fully match the query. This approach is too rigid. Our system retrieves messages that partially match the query. More flexible constraints force the system to rank the messages before presenting them to the user. E-mail messages that match the query better are more likely to be relevant – this is the basic idea for the ranking scheme. E-mail relevance score is the sum of scores computed for each structure part (text, sender, receiver, and date), where parts are equally weighted. The maximal score for a message that fully matches the query is 100%. For the parts based on keyword search (text, sender, and receiver) the score is simply the number of words from the query part that can be found in the responding part of e-mail, divided by the total number of words in the query part. The date part must be processed individually. The date constraints included in the NSQ are represented as a probability distribution. The distribution is scaled – the maximal value is equal to 1.0 to achieve 100% score for messages that fully match the query. The relevance score for the date part is simply the value of scaled distribution for the given message date.

Structure recognition score The parsing algorithm returns a list of all possible candidate structures with their likelihoods calculated from the probabilities of grammar rules. The likelihoods are the first step of the structure candidates ranking. At this step the HMM result must be added to the list returned by the parsing algorithm. As it was evaluated independently, there is no clue how likely it is to be correct. We decided to add the HMM result to the list with the likelihood of its median element.

The second step is based on the intuition that, the query structure is more likely to be correct if it is able to retrieve valuable information. For example if one of the structures of the query: "Jan about meeting", presented in section 5.2.2, is able to retrieve messages that match it well, then it is more likely to be correct. Retrieved e-mail messages give the system important information about the recognition correctness. The system retrieves messages matching each NSQ candidate structure, and computes a factor that represents the quality of retrieved messages based on the automatically computed e-mail relevance scores as explained earlier. The factor is a combination of the best e-mail relevance score, and the average e-mail relevance score of all retrieved messages. The system modifies the grammatical likelihood produced by the parsing algorithm by a factor computed from the relevance scores of retrieved e-mails.

The nature of the e-mail search makes the system sometimes incapable of choosing a single correct structure. For example, a user exchanged a few e-mail messages about a project presentation with a friend named John. The user tends not to use descriptors in NSQs. The query "John project presentation 3 months ago" can address the messages either sent or received by John. In this case, the system is unable to guess which messages should be retrieved. To deal with this type of problem the system retrieves messages based on the three most probable NSQ structure candidates. The structures as well as the messages are ranked. Combined scores of structure recognition and e-mail relevance create the order of the final message list presented to the user. The next section presents the process of creating the list.

5.6 Retrieval

The physical retrieval of e-mail messages has been performed in the previous step. The main task for the Retrieval step is to present retrieved messages in the form that makes finding the relevant message simple and intuitive. Another task for this step is to automatically prepare feedback

Input: Set of possible candidate query structure(queryStructureSet).
Output: Maximally three query structure candidates that are most likely to be correct.
Definitions:
`This character format indicates terms defined in the report's text.`

for each `candidate structure`
| retrieve matching e-mails;
| **for each** e-mail
| | compute `e-mail relevance score` (ERS);
compute `query structure score` (QSS) $//QSS = 0.8 * Max(ERS) + 0.2 * Mean(ERS)$;
normalize `candidate structures` likelihoods (QSL);
for each `candidate structure`
| update candidate structure likelihood $//QSL = QSL + QSS - QSL * QSS$;
normalize updated `candidate structure` likelihoods;
choose three best solutions, or all `candidate structures` if less than three;

Figure 11: Pseudocode of Results ranking step.

information for system training.

5.6.1 User interface

The system uses a simple "Google-like" interface. However the problem of search results presentation is more complicated here – results of three different top ranking candidate structures are presented together. To make this possible, the e-mail relevance score is modified by the score of the structure that retrieved it. The scores of messages retrieved by the most probable structure stay at the same level, while the scores of other messages are reduced. This makes the results of the most probable structure likely to be placed at the top of the list. It allows browsing the list without prior interaction with the system, with a high probability that the relevant message is close to the top. The user is allowed to turn the incorrect queries off to remove irrelevant messages from the list. If the most probable query structure is turned off, the next active NSQ structure takes its place, and the scores are rescaled.

5.6.2 Automatic feedback information

Feedback information allows the system to tune its parameters. For this to be possible, the system must know which of the presented structures is correct. Asking the user to manually define the correct structure recognition, after entering each query, is not feasible. The feedback must be gathered in a way that is not annoying for the user. This is done by having the system examine user behavior when browsing the retrieved messages list, i.e., tracking chosen NSQ structures and messages. The system collects the user's feedback automatically, in the case that the user did not choose any of the three presented structures to train the system.

When the user does not make the decision manually, the system accepts the most probable NSQ structure as the correct one. When the most probable structure is turned off by the user, using the

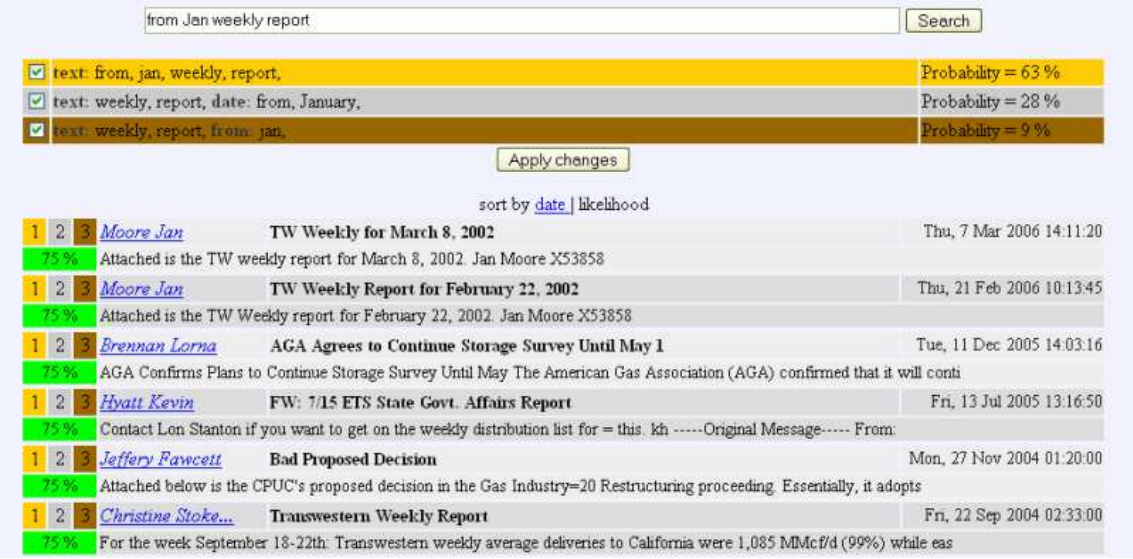

Figure 12: System interface. E-mail relevance score and structure recognition score combined together produce the order of the messages. The messages that match the most probable query structure well are presented close to the top.

button on the left side of the query structure description, the next one on the list takes its place. The automatic feedback information is processed only if the message is found (user clicks on the message link). This method allows reducing the number of incorrect NSQ structures training the system, at the expense of not accepting some correect structures during the training process.

6 System evaluation

The project connects two areas of computer science – Artificial Intelligence and Human Computer Interaction. We should evaluate it from both points of view. The measurement important from AI perspective is system effectiveness, which is determined by the accuracy of recognizing the NSQ structure. From the HCI perspective the aim is simpler to define but harder to measure – we have to answer a general question "do the users feel that the system helps them to search e-mail messages simpler and faster?" While evaluating the system accuracy, we show the system results for the four standard types of query formulations and the hardest realistic task for the system. In addition, we present the design of a formal user study to evaluate the system from the HCI perspective.

6.1 Dataset

For testing purposes we use the Enron dataset [10]. In our tests we work on the dataset preprocessed by Ron Bekkerman[3] [3]. He cleaned the data and left only the seven largest user's directories. We have chosen one directory "lokay-m" to perform tests, since our system is meant to be an individual user search assistant.

The set of queries on the Enron dataset, used to evaluate the system accuracy, is generated automatically. We have designed and implemented the Random Query Generator (RQG), a stand-alone system for the simulation of queries made by real users . The generator is an unlimited source

[3] www.cs.umass.edu/ronb/enron_dataset.html Date of access: 2007-10-01

of query data. Although it is unable to produce the full spectrum of possible queries, its flexibility allows us to generate a broad range of queries.

Random Query Generator is a system that generates Natural Search Queries according to rules presented in section 5.2.1. The rules are well defined, but there is still room for parameters, the values of which must be defined or randomly assigned. The most significant are: the number of parts; parts order; and probability of a descriptor being present. The last one makes the biggest impact on recognition difficulty. An informal user study shows that users tend to forget about descriptors while formulating the queries. These facts make the number of descriptors the key feature of RQG. To generate a query, RQG randomly chooses an e-mail message from the database. It then decides which parts of the message will be used in the query, and randomly chooses the descriptors and keywords. The date part is an exception for this pattern. A natural date description is generated based on predefined formulas, to keep it semantic sensible. The generator's pseudocode is available in appendix E.

Another much smaller set of queries was gathered from real users in an informal user study, and it was used for manually tuning the system parameters, before the evaluation process.

6.2 Accuracy evaluation

System effectiveness is defined by the accuracy of the system's core part, query structure recognition. If words are associated with their functions correctly, the retrieval process becomes as simple as in the traditional solution (Fig. 3).We have formulated four main tasks, corresponding to the standard types of queries, to evaluate the system. In addition, we present the hardest realistic task. Examples presented below were created by the Random Query Generator.

Well described queries – 100% probability that descriptors are present in each part. For example, "before 11 months ago about beverly author kimberly watson". The query structure is well described.

Queries with text keywords only – Queries are created from the text part only. Descriptors are not used. For example, "employee selected plays feedback". Such queries should be especially popular among users who are used to the pattern of keyword-based search engines (i.e. Google).

Queries with operators – "GMail-like" – The query is built from pairs of descriptors and keywords connected by a colon. For example, "text:employee after:2005/3/19 before:2005/7/27 from:announcement". These queries are popular among experienced users working with operator-based database interfaces.

Most common pattern – We observed that users are likely to formulate the queries according to a certain pattern that is individual for each user. The most common pattern is to begin the query with the text part. After the text part users sometimes add date constraints, and/or sender's name, receiver name is added rarely. Users tend not to use descriptors. For example, "resources website michelle 1 year ago".

Hardest task – To find out a lower bound on system accuracy, we created the hardest task, which is still realistic. Queries contain the maximal number of randomly ordered parts. The parts are entered without descriptors. To test how descriptors influence the accuracy we decided to introduce the hardest task gradually. The evaluation procedure consists of four steps, in increasing order of difficulty of the query. We start with a well-described query. At each step, we remove descriptors from a randomly chosen part (not including date). Examples of the query changes while going through the evaluation procedure:

"before 11 months ago author eric.gadd@enron.com about next duke eric sent to james"
"before 11 months ago author eric.gadd@enron.com about next duke eric james"

"before 11 months ago eric.gadd@enron.com next about duke eric james"
"before 11 months ago eric.gadd@enron.com next duke eric james"

6.2.1 Test procedure and results

We investigate how the system is able to handle the five types of queries presented above. The system was manually tuned based on the set of 50 real Natural Search Queries. During the tests the system processed 1000 queries of each type created by the Random Query Generator.

The interface, described in section 5.6, presents to a user the three most probable NSQ structures. We define two accuracy measures based on two different views of correctness. Query recognition is:

strictly correct if the most probable recognition (the first one on the list) is correct,

correct if one of the three presented recognitions is correct.

Evaluation results, i.e., the percentage of correctly recognized queries, are presented in Table 4. We begin the evaluation with the queries created according to the *well-described queries* pattern. The system is designed for this type of query, where high accuracy is a necessity. 100% probability that the descriptor is present in the part makes the parts easily distinguishable. We observe that the number of system mistakes is insignificant. They are mostly caused by including too many words in the text part. Similar results are achieved for the *queries with text keywords only*. These queries are simple and short which makes them easy for the system. The next investigated type is *queries with operators*. This type of query does not strictly follow the grammar. In some cases, such queries can be recognized only by the Hidden Markov Model. In this case, the most probable structure is often not correct. However, the accuracy of the three most probable recognized structures is still very high. CYK, the preferred approach that generally gives better results, makes a lot of mistakes for this type of query. It is more amenable to the Hidden Markov Model, but its result is sometimes not able to reach the top of the list in the Results Ranking step. Last of the four standard types of queries is the *most common pattern*. This task is more difficult for the system. There are no descriptors to help to distinguish the parts, which makes the keyword ambiguities more significant. Although the score of 96.5% (78.7% – strictly correct) is worse than for the first two tasks, the system is still practically usable. Finally we investigate how the system handles the *hardest realistic task*. Results of the four test steps show that accuracy is strongly correlated with the number of descriptors, as shown in Fig. 13. For queries without descriptors the accuracy drops to the level of 37.9% (12.1% – strictly correct), which is definitely inadequate for a practical system. These results show only that there are realistic, but unlikely, queries that the system cannot interpret.

	Strictly correct	Correct
Well described queries	98.7%	99.8%
Queries with text keywords only	99.8%	99.9%
Queries with operators "GMail-like"	56.2%	98.0%
Most common pattern	78.7%	96.5%
Hardest task	12.1%	37.9%

Table 4: Accuracy results for the four standard tasks, and the hardest realistic task. The system is able to process a broad range of queries with high accuracy.

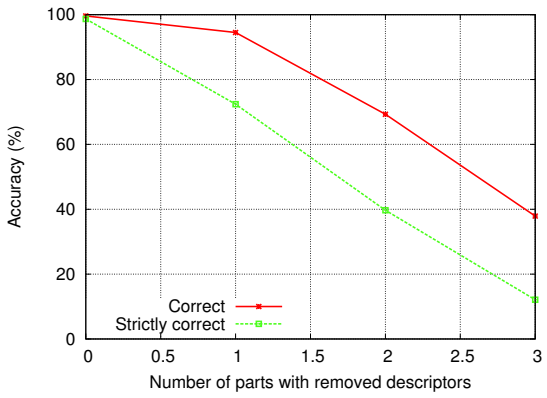

Figure 13: Desriptors significantly determine the query difficulty.

6.3 Training evaluation

The evaluation process becomes more complicated when online training is used. The order in which queries are presented to the system is now important. Each query is recognized by the system that is now already trained by each previously processed query. It is a simulation of a real situation where user enters queries one-by-one. The key element of each test is a pattern by which the queries were created. Training process allows the system to adapt to such patterns. To get unbiased results we decided to start all training tests with plain models (all probabilities uniform). (See appendix F for more information about training test procedure)

6.3.1 Training evaluation – results

By training evaluation, we want to answer two main questions: *"how strong must be the pattern to be recognized?"* and *"how much can the accuracy be increased?"* We specify five tasks, which allow us to describe training results from different perspectives. First, we check tasks for which the training system should get best and worst results to define system's abilities and limitations. Then we process the main tasks presented in section 6.2.1 to check if the accuracy for the most common patterns can be increased. Finally, we consider two problems that are likely to occur while processing real user queries – pattern noise and pattern changes. Detailed training evaluation description is given in appendix F; here we present the overall evaluation conclusions.

Training evaluation shows that the increase of accuracy is mostly dependent on the strength of pattern. The pattern can be considered as strong if there is a large number of factors common in all processed queries (e.g., each query has four, ordered parts). In extreme cases it can improve accuracy from 37.9% (12.1% – strictly correct) to 97% (81.4% – strictly correct), when the pattern is very strong. However, it is unlikely that real users will use this pattern. On the other hand, for queries that simulate user input well the improvement is small. These query patterns are presented and evaluated above, in section 6.2. In this case, the improvement is hard to achieve, because the results for such patterns are high even without training. Two last evaluation tasks show that training has some limitations. In real situations, it is likely that not all entered queries follow one pattern. To simulate it, we noise the main pattern (80% of entered queries) with randomly added queries that do not follow this pattern. The system is able to handle noised queries but the accuracy of trained system is worse, and the training process takes more time. At the same, time

the accuracy for queries that make the noise decrease. This shows the biggest drawback of training – the system becomes more precise for a specific group of queries, when accuracy of other queries is getting worse. The problem becomes clearer in the next task. The system is unable to handle two patterns at the same time. When two patterns are processed simultaneously only the stronger pattern can be learnt, and the quality of learning is low. Training is useable only for individual users.

6.4 User study

The main objective of the formal user study for our project is the comparison of new system, based on the idea of Natural Search Queries, and one of the traditional solutions. We want to test if, thanks to improvements in formulating queries and presenting results, users are able to find relevant messages easier and faster. The user study is in the last stage of preparation, and will be performed soon.

Test procedure tries to simulate common situations when search system is used. A user is asked to find a group of e-mail messages using one of the systems, working on a personal computer. While designing the survey we had to eliminate irrelevant factors that could bias the results. The system interfaces are as similar as possible; they both work in the same environment – an Internet browser. We had to take into account three other elements: Indicators – what measurements should be used to compare the systems; Input – how the messages should be represented to simulate the real situation; Randomization - how to manipulate the variables to minimize the impact of irrelevant factors.

6.4.1 Indicators

Time is the most important indicator when comparing both systems. We have to examine how fast a user is able to find a message. We divide user's interaction with the system into phases to better record detailed time results. First, we must notice that search is likely to be an iterative procedure. Users can modify search constraints and start the search process over many times (i.e. when the constraints were entered incorrectly and no relevant results are found; or when the constraints were too general and huge number of messages is returned). Time measurements for each iteration are stored separately. In each iteration, we distinguish two phases and take independent measurements:

Time user needs to enter the query – measured from the moment user clicks on the "search" link (traditional solution) or starts entering the query (new solution). Finished when the "search" button is pressed.

Time user needs to find a message on the retrieved e-mails list – measured from the moment that the retrieved e-mail messages list appears on the screen. Finished when the user clicks on the relevant message or starts the search process over (relevant message is not found).

Time measurements divided into small units allow us to test the impact of two search phases: query formulation, and retrieved list browsing, on the time user needs to find a message. In addition, we are able to check which system is more likely to give the correct answer in a smaller number of iterations.

The second possibility to compare both systems is to ask about users' opinions directly. After they complete the test, users are asked to fill-in a self-report. It examines users' background – their experience with computers and e-mail search systems. Afterwards we ask questions about specific features of both systems, and the overall impression.

6.4.2 Input

In the real situations, users enter search constraints to find messages they have already seen. While formulating the query they recall only pieces of information from the long-term memory. It is impossible to simulate such situation during the test, and at the same time allow users to formulate freely the queries. We decided to use two different input types:

Hard-copy of a full message – all information about the message is presented explicitly. User is free to choose information that can be useful to find a message. This input type does not simulate the real situation well. However, the freedom of choice is maximal.

Short paragraph describing the message – (i.e. *"You received a message from your brother (John Smith). He sent you his new work email address"*) a user is asked to form the query only from the given information. This input type simulates the real situation, but user is forced to use predefined data.

We decided to run the tests for both input types independently. Each one of them can give interesting information about the way and time in which queries are formulated.

6.4.3 Randomization

Randomization is a solution for removing the impact that irrelevant factors make on the results, by reasonable test procedure preparation [14]. In our system, we have two main factors that can be suppressed: interface learning and message learning. We are not able to remove or neglect these features. By randomization, we can prepare the test in the way that irrelevant factors bias the results equally for both systems.

While working with the system users are getting used to it. They learn the interface and are likely to increase the work effectiveness. We can assume that if a user works on both systems one after another the results for the second system will be better. The solution for this problem in our studies is to randomly choose the first system for every user, and perform the test on the second system after some period (next day).

Not all messages are equally hard to find. To compare both systems they have to work on the same group of messages to ensure equal difficulty. At the same time, users learn about messages and finding them in the second system should be much easier. We decided to create a small set of 20 test messages. Each user randomly draws six messages used in the test; only three of them are used next day for the other system. This solution allows to simultaneously compare the systems while searching for identical messages (bias because of learning) and different messages (bias because of different difficulty). Both measurements should together give the full picture of systems' efficiency.

7 Conclusions and future work

Natural language is the best way for humans to communicate. Many researchers have tried to use it also for human computer interaction. In this technical report, we presented a natural language interface for structured information search. Most of the previous approaches to this problem concentrate on processing complex natural language questions. Their designers try to make a computer system that is able to act like a real interlocutor. It makes the problem too hard to be practically solvable. In our project, we stressed different tasks of natural language interaction: simplicity and intuitiveness. By addressing the well-defined and limited problem of search in consumer-oriented databases, we were able to create a practically usable system for e-mail databases.

The main contribution of our project is the concept of Natural Search Queries. NSQ represents an intuitive way of creating search queries for structured information in simplified natural language. A formal definition of NSQs allowed us to develop the grammar rules that describe the way queries are created. The grammar is the starting point for solving the key problem of NSQ processing – query structure recognition. We used two complementary methods. The CYK parsing algorithm recognizes, with high accuracy, queries that strictly follow the grammar rules. The Hidden Markov Model is able to process misformulated and extraordinary queries, however, its accuracy is lower. Furthermore, we addressed the problem of interpreting date constraints from date descriptions written in natural language.

We presented a complete system that retrieves messages from e-mail databases, using Natural Search Queries. However, the presented ideas can be applied to other consumer-oriented databases. The system successfully overcomes the limitations of the traditional form-filling search interfaces. The intuitive interface of Natural Search Queries makes the system easily accessible for common users without the need of training. The evaluation results demonstrate the potential of the system in processing a broad range of e-mail search queries with high accuracy.

Future work involves extensive user studies. We want to run the formal user study presented in section 6.4 to check if the system meets user expectations. The system has a modular structure that makes modifications easy. For example, to avoid incorrect recognitions caused by misspelled words we can update the IWL step to perform approximate search.

The general idea of Natural Search Queries can be extended to other consumer-oriented search systems. Each specific implementation can shed new light on the problem. We plan to run the system for flight connections database used in flight booking applications. Only few elements have to be modified in order for the system to work on a different database. New descriptors and grammar rules, representing the database structure, must be defined. The grammar for flight database is more complicated, but less ambiguous, and it is likely to improve system's accuracy.

References

[1] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural Language Interfaces to Databases– an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.

[2] Catherine N. Ball, Deborah Dahl, Lewis M. Norton, Lynette Hirschman, Carl Weir, and Marcia Linebarger. Answers and questions: processing messages and queries. In *HLT '89: Proceedings of the workshop on Speech and Natural Language*, pages 60–66, Morristown, NJ, USA, 1989. Association for Computational Linguistics.

[3] Ron Bekkerman, Andrew McCallum, and Gary Huang. Automatic categorization of email into folders: Benchmark experiments on Enron and SRI corpora. Technical Report IR-418, Center of Intelligent Information Retrieval, UMass Amherst, 2004.

[4] Stefan Buchta. Oracle Ultra Search Architecture 10g. Architecture white paper, Oracle Corporation, 1994.

[5] Michael J. Cafarella, Oren Etzioni, and Dan Suciu. Structured Queries Over Web Text. *IEEE '06: Bulletin of the Technical Committee on Data Engineering*, 29(4):45–51, 2006.

[6] S. Cohen. XSEarch: A semantic search engine for XML. In *VLDB 2003: Proceedings of 29th International Conference on Very Large Data Bases, September 9–12, 2003, Berlin, Germany*, pages 45–56, Los Altos, CA 94022, USA, 2003. Morgan Kaufmann Publishers.

[7] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 133–140, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[8] G.D. Forney. The Viterbi algorithm. In *Proceedings of the IEEE*, pages 268– 278, 1973.

[9] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[10] Bryan Klimt and Yiming Yang. The Enron corpus: A new dataset for email classification research. In *Machine Learning: ECML 2004*, pages 217–226. Springer Berlin / Heidelberg, 2004.

[11] Yunyao Li, Huahai Yang, and H. V. Jagadish. Constructing a generic natural language interface for an XML database. In *In Proceedings of International Conference on Extending Database Technology (EDBT 2006)*, 2005.

[12] Yunyao Li, Huahai Yang, and H. V. Jagadish. NaLIX: an interactive natural language interface for querying XML. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 900–902, New York, NY, USA, 2005. ACM Press.

[13] Jayant Madhavan, Alon Halevy, Shirley Cohen, Xin (Luna) Dong, Shawn R. Jeffery, David Ko, and Cong Yu. Structured Data Meets the Web: A Few Observations. *IEEE '06: Bulletin of the Technical Committee on Data Engineering*, 31(4):19–26, December 2006.

[14] Joseph E. McGrath. *Methodology matters: doing research in the behavioral and social sciences*, pages 152–169. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[15] Wesley W. Chu Frank Meng. Database query formation from natural language using semantic modeling and statistical keyword meaning disambiguation. Technical Report 990003, University of California, Los Angeles, CA 90095, USA, 16, 1999.

[16] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Readings in speech recognition*, pages 267–296, 1990.

[17] Ganesh Ramakrishnan, Soumen Chakrabarti, Deepa Paranjpe, and Pushpak Bhattacharya. Is question answering an acquired skill? In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 111–120, New York, NY, USA, 2004. ACM Press.

[18] Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Comput. Linguist.*, 21(2):227–253, 1995.

[19] Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1,2):3–36, 1995.

[20] Cynthia A. Thompson, Raymond J. Mooney, and Lappoon R. Tang. Learning to parse natural language database queries into logical form. In *Workshop on Automata Induction, Grammatical Inference and Language Acquisition*, Nashville, TN, 1997. Association for Computational Linguistics.

[21] Murray Turoff. Natural language and computer interface design. In *Proceedings of the 18th annual meeting on Association for Computational Linguistics*, pages 143–144, Morristown, NJ, USA, 1980. Association for Computational Linguistics.

[22] David H. D. Warren and Fernando C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *Comput. Linguist.*, 8(3-4):110–122, 1982.

[23] V. Zue, S. Seneff, J.R. Glass, J. Polifroni, C. Pao, T.J. Hazen, and L. Hetherington. JUPITER: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1):5–96, January 2000.

A Individual Word Labeling part

Text part descriptors
about; keyword; keywords; text; with; words; word
From part descriptors
author; by; from; received; send; sender; sent; written
To part descriptors
receiver; send; sent; to
Date part descriptors
-; after; ago; and; before; between; from; in; last; of; on; previous; to

Table 5: Sets of descriptors used in the e-mail search system.

From part/to part keywords' patterns
Name pattern
`[A-Z][a-z]*`
Address pattern
`[\w\'\.\-\&]+@([\w\'\-\&]+\.)+\w+`
Date part keywords' patterns
Date pattern
`\d\d.\d\d.\d\d\d\d`; `\d\d/\d\d/\d\d`... date is later verified.
Year pattern
`200\d`; `199\d`...
Day name
Monday; Mon; Tuesday; Tue ...
Month name
January; Jan; February; Feb ...
Time length
day; days; week; weeks ...
Number
`\d+`; *one; first; two; second ...*

Table 6: Patterns used to recognize keywords. Patterns are presented as regular expressions, or as a list of words.

B NSQ grammar rules

We present a complete list of grammar rules used in the e-mail search system.

B.1 Start rules

Queries produced from one part:
QUERY_1 --> TEXT_PART
QUERY_2 --> FROM_PART
QUERY_3 --> TO_PART
QUERY_4 --> DATE_PART
There are four such rules.
Queries produced from two parts:
QUERY_1_2 --> QUERY_1 QUERY_2
QUERY_1_2 --> QUERY_2 QUERY_1
QUERY_1_3 --> QUERY_1 QUERY_3
QUERY_1_3 --> QUERY_3 QUERY_1
Other rules that produce queries from two parts are created analogically (there are 4*3 = 12 such rules)
Queries produced from three parts:
QUERY_1_2_3 --> QUERY_1 QUERY_2_3
QUERY_1_2_4 --> QUERY_1 QUERY_2_4
QUERY_1_3_4 --> QUERY_1 QUERY_3_4
QUERY_1_2_3 --> QUERY_2 QUERY_1_3
QUERY_1_2_4 --> QUERY_2 QUERY_1_4
QUERY_2_3_4 --> QUERY_2 QUERY_3_4
Other rules that produce queries from three parts are created analogically (there are 4*3 = 12 such rules)
Queries produced from four parts:
QUERY_1_2_3_4 --> QUERY_1_2 QUERY_3_4
QUERY_1_2_3_4 --> QUERY_1_3 QUERY_2_4
QUERY_1_2_3_4 --> QUERY_1_4 QUERY_2_3
QUERY_1_2_3_4 --> QUERY_2_3 QUERY_1_4
QUERY_1_2_3_4 --> QUERY_2_4 QUERY_1_3
QUERY_1_2_3_4 --> QUERY_3_4 QUERY_1_2
There are six such rules.

B.2 Intermediate rules

The rules that collects words of the same nature:
TEXT_DESC_PART --> TEXT_DESC TEXT_DESC_PART
TEXT_DESC_PART --> TEXT_DESC
FROM_DESC_PART --> FROM_DESC FROM_DESC_PART
FROM_DESC_PART --> FROM_DESC
TO_DESC_PART --> TO_DESC TO_DESC_PART
TO_DESC_PART --> TO_DESC
TEXT_KEY_PART --> TEXT_KEY TEXT_KEY_PART
TEXT_KEY_PART --> TEXT_KEY
FROM_KEY_PART --> FROM_KEY FROM_KEY_PART

```
FROM_KEY_PART --> FROM_KEY
TO_KEY_PART --> TO_KEY TO_KEY_PART
TO_KEY_PART --> TO_KEY
DATE_PART --> DATE_WORD DATE_PART
DATE_PART --> DATE_WORD
```

The group of rules based on "descriptors-keywords" pattern:

```
FROM_PART --> FROM_DESC_PART FROM_KEY_PART
FROM_PART --> FROM_KEY_PART
TEXT_PART --> TEXT_DESC_PART TEXT_KEY_PART
TEXT_PART --> TEXT_KEY_PART
TO_PART --> TO_DESC_PART TO_KEY_PART
TO_PART --> TO_KEY_PART
```

Date part is processed individually at this level.

B.3 Terminal rules

Terminal rules are created individually for each query. Rules on this level are created directly from word labels. Here we present two examples of terminal rules:

```
TEXT_DESC --> [about]
TEXT_KEY --> [about]
FROM_DESC --> [from]
DATE_DESC --> [from]
TEXT_KEY --> [from]
```

C Training

Update formula There are three parts of the system, where the probabilistic parameters are used: the Individual Word Labeling step, the parsing algorithm, and the Hidden Markov Model. In all these parts the parameter update is done using one update formula. Intuitively, the more often a particular element (a function in IWL, a rule in parsing, an element of transition or evidence model in HMM) was successfully used, the more probable it should become. To follow this intuition the update formula is based on simple average (Fig. 14), the probabilistic parameters are normalized after update. It is a simplified approach, more sophisticated update methods should be used in the real system, however the basic mechanism of changing the parameters described below will remain.

$$p_{k+1} = \frac{kp_k+\alpha}{k+1}$$
$$k - \textit{number of times the parameter was changed}$$
$$\alpha \in [0,1] \textit{ allows the increase or decrease of the parameter value}$$

Figure 14: Training update formula – the more often an element was successfully used the more probable it should become.

Training in Individual Word Labeling step The labels are triggered by a set of functions, this process is described in section 5.1. Training modifies the function likelihoods. Feedback information (i.e., correct labels) allows the system to find functions that added the correct label to word and increase their likelihoods ($\alpha = 1$ in Fig. 14). At the same time likelihoods of functions that made a mistake are decreased ($\alpha = 0$ in Fig. 14).

Input: Processed query – words and their correct labels. Likelihood values attached to labeling functions.
Output: Updated labeling function likelihood values.
Definitions:
$k_{function}$ – The number of times a certain labeling function was used.
$l_{function}$ – The likelihood of certain labeling function giving a correct label.
`This character format indicates terms defined in the report's text.`

for each word
| get all `labeling functions` that produced a label for the word;
| **for each** `labeling function` that produced the correct label
| | $l_{function} := \frac{k_{function} * l_{function}+1}{k_{function}+1}$;
| **for each** `labeling function` that produced an incorrect correct label
| | $l_{function} := \frac{k_{function} * l_{function}+0}{k_{function}+1}$;
normalize `labeling functions` likelihoods;

Figure 15: Pseudocode of the Individual Word Labeling update function.

Training for parsing algorithm Tuning the rule probabilities used in the parsing algorithm is straightforward. Feedback information contains the correct structure, which defines the correct parse tree and the rules used to build it. Training should increase the probability values of these rules. The changes are applied only to the start and intermediate stages of grammar rules. The probabilities of terminal rules are created according to the label probability distribution returned by the IWL step. These probabilities are trained separately as presented above.

Input: Processed query – words and their correct labels (*labelSet*). Probability values attached to grammar rules.
Output: Updated grammar rule probability values.
Definitions:
k_{rule} – The number of times a certain grammar rule was used.
p_{rule} – The probability of certain grammar rule.
queryStructureRules – The rules that have produced given structure.
`This character format indicates terms defined in the report's text.`

queryStructureRules := **do** CYK probabilistic parsing – *labelSet* as `terminal rules`;
if CYK returned a structure
| **for each** rule from *queryStructureRules*
| | $p_{rule} := \frac{k_{rule} * p_{rule} + 1}{k_{rule} + 1}$;
normalize rule probabilities;

Figure 16: Pseudocode of the grammar rules update function.

Training for hidden Markov Model Generally, HMM training is a complicated task. The model is hidden, which means we do not have access to the sequence of states that have generated the outcome, and therefore no clues how to modify the models. In most cases, this problem is solved by variations of the Baum-Welsh algorithm [9, 16]. The algorithm estimates the number of states that were used to generate given sentences. We are able to use some specific features of our problem to simplify the process of HMM training. In section 5.2.3 we show that both state and evidence space mirror the space of possible word labels. We know that the word should have been produced by its corresponding state. Although we do not know which states have actually produced the evidence sequence, we know which ones should have done that in ideal situation, given the correct label from the feedback. It makes updating the transition model straightforward, because we have direct access to the state values. Evidence model is updated by the probabilities of labels produced by IWL step. It is a natural process because IWL and evidence model answer the same question: *"how probable it is to get a particular label (evidence value) from a word (a state hidden behind the word)?"*

Input: Processed query – words and their correct labels (*labelSet*). Transition and evidence model from Hidden Markov Model.
Output: Updated HMM models.
Definitions:
$k_{element}$ – The number of times a certain model element was used.
$p_{element}$ – The probability of model element rule.
`This character format indicates terms defined in the report's text.`

*/*updating the evidence model*/*
for each word
| **for each** label attached to the word
| | choose the element of the `evidence model`
| | that refers to the label;
| | $p_{element} := \frac{k_{element} * p_{element} + l_{label}}{k_{element} + 1}$;
*/*updating the transition model*/*
for each pair of words
| | choose the element of the `transition model` that refers to the switch
| | between the pair of words;
| | $p_{element} := \frac{k_{element} * p_{element} + 1}{k_{element} + 1}$;
normalize models;

Figure 17: Pseudocode of the HMM models update function.

D Date part processing

D.1 Grammar

We present the complete list of rules used in the Time Window Recognition system. To make the list easier to understand we grouped rules and neglect the Chomsky normal form. All rules can be easily transformed to CNF.

The rules are presented in two main groups. The first group shows how date operators, which are represented by words previously recognized as date descriptors, transform components. The second group presents how primal components are created from date keywords.

AFTER_OPR --> [after]
AFTER_OPR --> [from]
ATW --> AFTER_OPR ATW / Function: afterAWT(AWT) /
RTD --> RTW AFTER_OPR / Function: afterRTD(RTW) /
AGO_OPR --> [ago]
AGO_OPR --> [back]
ATW --> RTW AGO_OPR / Function: agoATW(RTW) /
BETWEEN_LEFT_OPR --> [between]
BETWEEN_LEFT_OPR --> [from]
BETWEEN_RIGHT_OPR --> [and]
BETWEEN_RIGHT_OPR --> [to]
ATW --> BETWEEN_LEFT_OPR ATW BETWEEN_RIGHT_OPR ATW /Function: betweenATW(ATW, ATW)/
RTW --> BETWEEN_LEFT_OPR RTW BETWEEN_RIGHT_OPR RTW /Function: betweenATW(RTW, RTW)/
BEFORE_OPR --> [before]
BEFORE_OPR --> [to]
ATW --> BEFORE_OPR ATW / Function: beforeAWT(AWT) /
RTD --> RTW BEFORE_OPR / Function: beforeRTD(RTW) /
LAST_OPR --> [last]
ATW --> LAST_OPR ATW / Function: lastAWT(AWT) /
ATW --> LAST_OPR RTW / Function: lastAWT(RWT) /
CONNECT_OPR --> [and]
CONNECT_OPR --> [-]
ATW --> AWT CONNECT_OPR ATW / Function: connectAWT(AWT, AWT) /
ATW --> AWT ATW / Function: connectAWT(AWT, AWT) /
ATW --> RTD ATW / Function: RTDtoAWT(RTD, AWT) /

OPR stands for operator,
ATW stands for Absolute Time Window,
RTD stands for Relative Time Distance,
RTW stands for Relative Time Window.

Here we present how terminal rules are created from date keywords. Analogically as in Individual Word Labeling step (presented in section 5.1 of the report) these rules are created individually for every query. More information about the character of date keywords can be found in Table 6 placed in appendix A. We distinguish six types of date keywords: date, day, month, year, time length, number. They can create terminal rules:
ATW --> DATE / Function: dateAWT(DATE) /

```
ATW --> DAY / Function: dayAWT(DAY) /
ATW --> MONTH / Function: dateAWT(DATE) /
ATW --> YEAR / Function: dateAWT(DATE) /
RTW --> TIME_LENGTH / Function: createRTW(TIME_LENGTH) /
RTW --> NUMBER TIME_LENGTH / Function: createRTW(NUMBER, TIME_LENGTH) /
```

D.2 Example recognition

In this section, we present how example date description written in natural language is processed by the system. We process the query: "after 2006/03/08 before 2006/05/16". Based on the rules presented above a parse tree is created (Fig. 18).

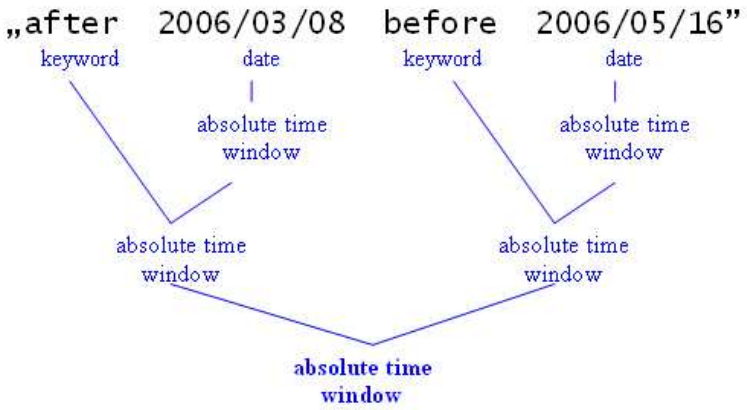

Figure 18: Parse tree of the example query "after 2006/03/08 before 2006/05/16".

Following the parse tree, we can recursively transform components using given functions (Fig. 19).

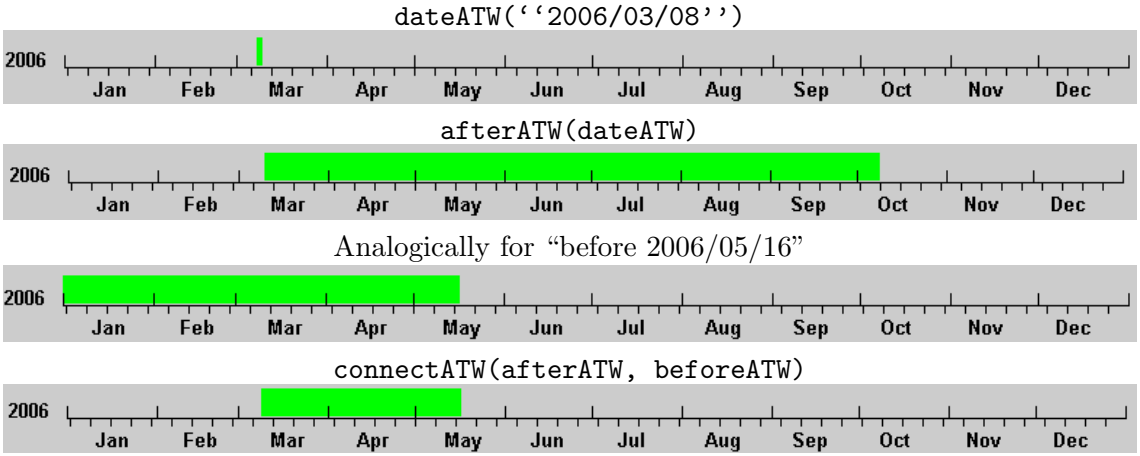

Figure 19: Processing of example query "after 2006/03/08 before 2006/05/16".

The final absolute time window is a natural date description placed on the time axis. In the next step, ATW is translated to two probability distributions: uniform and Gaussian. Both distributions

are combined and then scaled so the maximal date likelihood is 1.0. The algorithm in a form of pseudocode is presented in Fig. 10.

E Random Query Generator

E.1 RQG pseudo-code

Require: User's decisions about all parameters - set to defined value or randomized.
Ensure: A simulation of Natural Search Query.
Definitions:
date pattern - a pattern by which date part is created. Possible patterns
are presented in section E.2.
`This character format` indicates terms defined in section E.2.

choose query parts; */*query parts are chosen by user or randomly*/*
*/*possible parts: text part, from part, to part, date part*/*
set query parts order; */* order is set by user or randomly*/*
randomly choose a message from the e-mail database;

for each query part
| **if** query part describe date
| | randomly choose a pattern;
| | **if** `AGO pattern`
| | | noise message send date;
| | | */*small Gaussian noise to simulate real situation*/*
| | | translate date to `word description`;
| | **else**
| | | randomly choose lower and/or upper limit;
| | | **if** pattern uses `word descriptions` to describe dates
| | | | translate dates to `word descriptions`;
| | fill-in date part according to chosen date pattern;
| **else**
| | decide if descriptors are used; */*randomly or decided by user*/*
| | **if** use descriptors
| | | choose descriptors from part descriptors list;
| | randomly set number of keywords;
| | randomly choose keywords from the database;
| | */*words from defined message and part*/*
| | fill-in the part using chosen descriptors and keywords;

E.2 Patterns used to generate date part

AGO pattern
- with word description *(i.e. "3 weeks ago")* `NUMBER WORD\_DESCRIPTION [ago]`
AFTER pattern
- with DATE *(i.e. "after 03/21/2002")* `[after]DATE`
- with AGO pattern *(i.e. "after 3 weeks ago")* `[after] AGO`
BEFORE pattern
- with DATE *(i.e. "before 03/21/2002")* `[before] DATE`
- with AGO pattern *(i.e. "before 3 weeks ago")* `[before] AGO`
AFTER BEFORE pattern /*with optional `[and]` between `AFTER` and `BEFORE`, the order can be inverted*/
- with DATE *(i.e. "after 02/23/2000 before 03/21/2002")* `[after] DATE [before] DATE`
- with AGO pattern *(i.e. "after 5 months ago before 3 weeks ago")* `[after] AGO [before] AGO`
BETWEEN AND pattern /*with optional `[from]` and `[to]` keywords*/
- with DATE *(i.e. "between 02/23/2000 and 03/21/2002")* `[between] DATE [and] DATE`
- with NUMBER *(i.e. "between 5 and 3 weeks ago")* `[between] NUMBER [and] NUMBER WORD_DESCRIPTION [ago]`
- with NUMBER and word description *(i.e. "between 5 months and 3 weeks ago")* `[between] NUMBER WORD_DESCRIPTION [and] NUMBER WORD_DESCRIPTION [ago]`
- with AGO pattern *(i.e. "between 5 months ago and 3 weeks ago")* `[between] AGO [and] AGO`

F Training evaluation

Training effectiveness is determined by the improvement in the accuracy of processing queries, that can be reached by the system while learning the pattern. This makes the training evaluation doable only by processing a very large number of queries. We use queries generated by the Random Query Generator to evaluate training, analogically as in the system evaluation (section 6.1). We have complete control over patterns processed by the system; it allows us to examine the training system from different perspectives.

Training evaluation needs a complex procedure. Our objective is to simulate the real situation as well as possible. A generated query is used to train the system only if it was previously correctly recognized. This procedure tries to reflect the way that the real feedback is gathered – see section 5.6.2 of the report. In reality, the system should modify its parameters each time the query is entered and correctly recognized. We use the same procedure during the tests. The queries are learnt one-by-one. The objective of training is to tune the system probabilistic parameters. We must define the base parameters, from which the tuning process is started. One solution is to use the manually pretuned parameters, as it was done during the base system evaluation described in section 6. The second possibility is to start the training process from plain models, where all probabilities are equal. We decided to use the second solution to get unbiased results, and observe the real abilities of training.

Training is a complicated process and needs complex evaluation. We decided to group the tests into three stages. First, we check system's abilities and limitations, processing the patterns that are easiest and hardest to train. In the main part of the evaluation procedure, we train the system by patterns that are likely to be used by real users. Finally, we examine how the training process faces difficulties that can happen while processing real queries.

The objective of the first test stage is to determine training abilities and limitations. To achieve it we process queries for which training should get best and worst results. The factor that indicates training effectiveness is the increase in the accuracy of processing queries. Training is able to tune the system to a given pattern better if the pattern is strong, which means that a large number of factors (i.e., parts number, parts order, etc.) is common in all processed queries. We process two groups of queries with a strong pattern, and one group that does not follow any pattern:

The hardest task for system without training - Each query has four, randomly ordered parts. As presented in section 6.2.1 the accuracy for this type of queries is quite low (37.9% – correct; 12.1% – strictly correct). Such queries have a strong pattern. Training allows the system to expect keywords from every possible query part in each query.

The strongest pattern – In this test we remove the only weakness of the pattern presented above – randomly ordered parts. We fix the order to see how it changes the effectiveness and efficiency of training. This change is unobservable for the not trained system, but it should be a significant facility for the trained system.

The weakest pattern – Number of parts and their order is randomized. Parts have no descriptors to keep the queries hard to recognize. This test shows how training works when there is no pattern to follow.

In the main part of the training evaluation, we test patterns that are most likely to be used in real queries: **"well described queries", "text keywords only", "most common pattern"** (for a detailed description of these patterns please refer to section 6.2.1). The results for this type of queries indicates the real usability of training, because such queries are most likely to be entered by real users.

The last tested group of patterns represents the problems that are likely to occur while processing real queries. We cannot expect that users will enter queries that always follow a certain pattern.

We simulate two situations:

Noised pattern. – We noise the strongest pattern (four ordered parts with no descriptors), 20% of processed queries is fully randomized. It is unlikely that a user will strictly follow one pattern; we have to expect some off-pattern queries. Queries that do not match the main pattern create noise for the training system. We test how the noise queries change system's ability to learn.

Mixed patterns – When the number of noise queries is comparable with the number of pattern queries we can consider input queries as a mixture of two patterns. We have mixed two query types: a strong pattern presented above and "well described queries". This test shows if a system can be used by two users simultaneously. A modification of this test, where queries are entered to the system in groups of 50, shows how already trained system handles the change of pattern.

F.1 Training evaluation – results

The first stage of training evaluation states the borders of the training system. Results for strong patterns suppose to present the potential of the system. The hardest pattern for system without training (four randomly ordered parts, without descriptors) shows that training can significantly increase the accuracy. The accuracy is increased from 37.9% – correct (12.1% – strictly correct) to about 64.7% (46.1% – strictly correct) (presented numbers are averaged accuracy for the last 100 queries in the test). Fig. 20 shows that training process is slow; the system needs over 100 queries to start increasing the accuracy. The system is trained only by the correctly recognized queries. Their percentage at the beginning of training process is low, which is the reason for a slow start. Accuracy and time system needs to learn the pattern are improved significantly when processing the strongest pattern, where the parts order is fixed (Fig. 21). A stable score of 97% (81.4% – strictly correct) allows the system to successfully process queries. On the other side, the system parameters after training are very sharp, which makes the results high only for a specialized group of queries. As we assumed, training is unable to capture weak patterns, i.e., queries where all parameters are randomized. For this type of queries, system parameters are changed slightly and no increase in accuracy is observed (Fig. 22).

The first group of tests proved that for some specialized patterns the accuracy of the system can be increased by training. The next group of tests is supposed to present training abilities for the most common patterns. The task is hard because all these patterns have very good results in untrained system. Surprisingly training, in its early stage, can slightly decrease system accuracy – it can be noticed in results for "well described queries" (Fig. 23). In this case, the queries pattern is weak, so the system has problems with catching it, and the learning process is long. After processing big number of queries in all cases the results are slightly improved, or, as for the "text keywords only" queries – where all processed queries were strictly correct (Fig. 24), the accuracy is not reduced. In the ideal cases, when all entered queries follow one pattern the training process does not decrease system's accuracy, however we should not expect significant improvement.

The third group of evaluated patterns simulates the situations that are likely to happen it the real world. First, we process the noised queries. The system is still able to tune to the main pattern however tuning is not as strict as for queries without noise. Training process takes more time and gives worse results. After processing 400 queries, noised results 86.7% (64.9% – strictly correct) are lower that accuracy for not noised pattern 97% (81.4% – strictly correct). The test shows that training from noised queries is still feasible, although the accuracy is reduced for both pattern and noise queries. In the next test, we mix a strong pattern that gives poor results, but can be easily learnt, and a weak pattern with good results and no ability for training. The results (Fig. 27) show the biggest drawback of training – it gets specialized for the strongest pattern, accuracy for other queries is reduced. Weak patterns are also able to abuse training process, which can be

noticed when mixed patterns are entered to the system in groups of 50 queries. The graph (Fig. 28) shows that the system is unable to memorize parameters setting. Training process is started over each time the pattern changes. The system is not able to handle two patterns simultaneously, and should be used only for individual users.

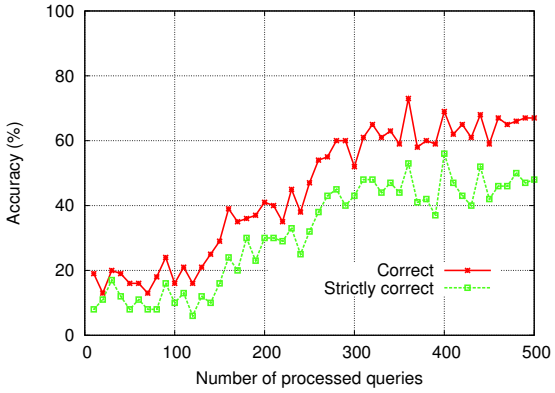

Figure 20: Hardest task for system without training. System's accuracy can be significantly improved

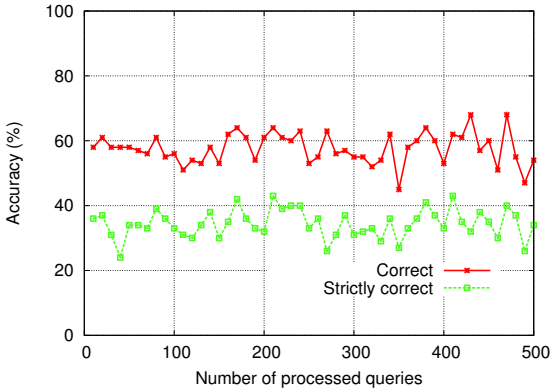

Figure 22: The hardest task for training. Randomized number and order of parts make the pattern weak.

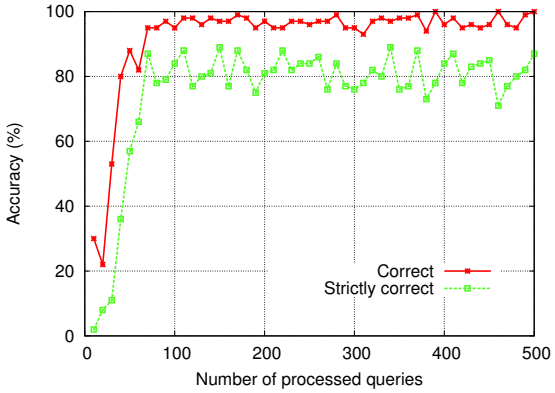

Figure 21: The strongest possible pattern - four, ordered parts, and no descriptors. Stronger pattern is easier to learn.

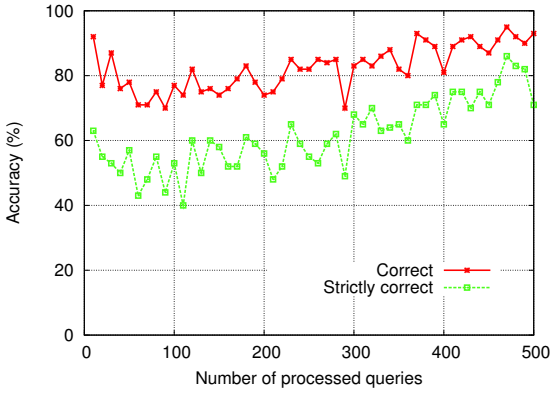

Figure 23: Well described queries

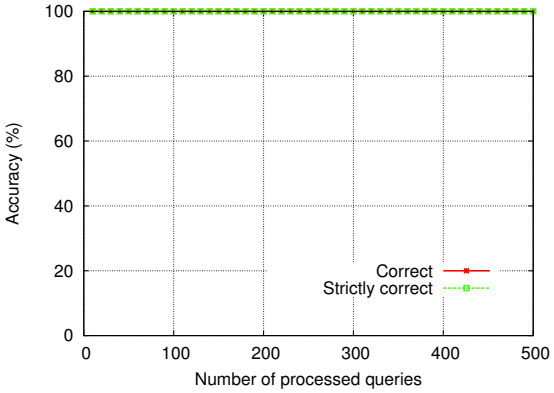

Figure 24: Text keywords only

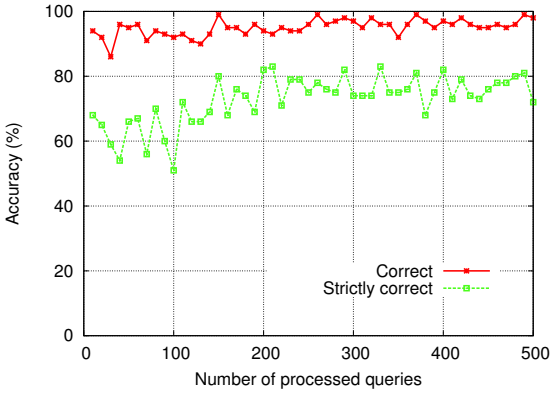

Figure 25: Most common pattern

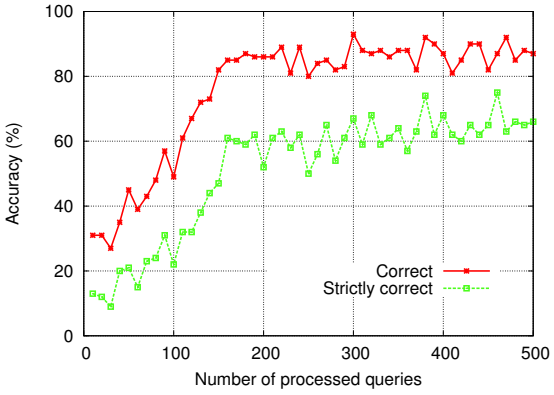

Figure 26: Strong pattern, noised.

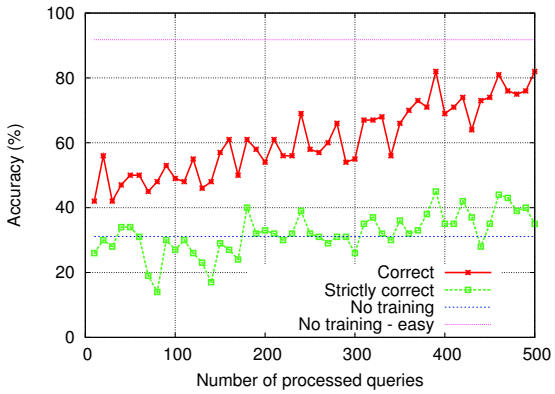

Figure 27: Two patterns, randomly mixed.

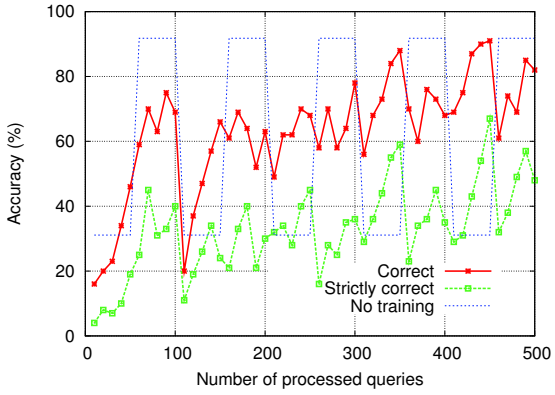

Figure 28: Two patterns. Entered in groups of 50 queries.