A Survey of Natural Language Query Builder Interface for Structured Databases using Dependency Parsing

Rohini B. Kokare
Computer Engineering Department,
Vishwakarma Institute of Information Technology,
Savitribai Phule Pune University,

Kirti H. Wanjale
Computer Engineering Department,
Vishwakarma Institute of Information Technology,
Savitribai Phule Pune University,

ABSTRACT
Natural language query builder interface retrieves the required data from database when query is given in natural language. To retrieve the correct data from database, the user should have sufficient technical knowledge of Structured Query Language (SQL) statements. Natural Language Query Builder Interface (NLQBI) will solve this problem. In natural language parsing**,** getting highly accurate syntactic analysis is a crucial step. Parsing of natural languages can be seen as the process of mapping an input string or a sentence to its syntactic representation. One of the parsing technique is dependency parsing. Dependency parsing focuses on relations between words which resolve ambiguity. Most of the recent efficient algorithms for dependency parsing work by factoring the dependency trees. Graph based dependency parsing models are prevalent in dependency parsing because of their state-of-art accuracy and efficiency. This paper covers some recent developments in NLQBI systems and survey on dependency parsing techniques.

Keywords
Natural Language Query Builder Interface(NLQBI), Natural Language Processing(NLP), Dependency parsing, Structured Query Language(SQL), Projective and Non-projective Dependency Parsing, Data-driven Dependency Parsing, Transition-based models, Pseudo-projective parsing, Graph based models, Higher-order factorizations, Span

1. INTRODUCTION
Since the end of 1960's there have been a large number of research works regarding the theories and implementations of NLQBI's. Asking question to databases in natural language is very convenient and easy method of accessing data especially for casually users who do not understand complex database query language. As the usage of databases has spread widely, the concept of user interface presented new challenges to the designers. The main goal of this system is to provide communication between user and computer without recalling any sort of database DDL or DML query syntax.

A Natural language query builder can be developed using dependency parser. Dependency Parsing presents a number of advantages when compared to syntactic parsing. Dependency parsing has three main advantages. The very first, dependency links that are formed between two words are close to semantic relationships needed for the next step of interpretation. Second, the dependency tree contains one node which represents one word, instead of mid-level nodes that represents words as in constituent trees, making the task of parsing more straightforward. Third, dependency parsing lends itself to word-at-a-time operation, i.e., parsing can be done by accepting and attaching words as entered by user. That means it does not wait for complete sentence to be loaded for parsing. The objective of this survey is to study different NLQBI systems, dependency parsing techniques and their limitations. Also study and compare the existing dependency parsers for the future work system.

Rest of the paper is organized as follows: Section II provides a detailed overview of recent developments in NLQBI. In section III different techniques used for developing NLIQBI are described. In section IV provides dependency parsing (DP) overview and different techniques of DP. Finally in section V will conclude the paper.

2. RECENT DEVELOPMENTS IN NLQBI
2.1 NaLIX
NaLIX (Natural Language Interface to XML) is a first generic interactive natural language interface to XML databases. In NaLIX, a complex English language sentence, which includes aggregation, nesting, and value joins, is translated into an XQuery expression that can be evaluated against an XML database. Schema-Free XQuery is used in NaLIX which is a database query language designed mainly for retrieving information in XML.The main advantage of using this Schema-Free Xquery is that mapping a query into exact database schema is not necessarily required. NaLIX uses syntax-based approach for generating, validating and translating the parse tree to an XQuery expression. NaLIX provides an Interactive Query Formulation where user can either write their own natural language queries or they can load query template files, and choose from a variety of preloaded sample natural language queries. NaLIX facilitates the users to save the whole query history or the set of selected queries in it into a new template file so that user can reuse the queries. NaLIX provides three different query result display views to the users: text view, grid view and tree view.

2.2 PRECISE
PRECISE uses the relational database with a SQL as the query language. PRECISE system was evaluated on two databases domains: GEOQUERY domain and ATIS domain. PRECISE uses the Semantic Tractability Model. The parser module of PRECISE is treated as a "plug in" enabling us to force the state of the art in parsing technology. The semantic interpretation in PRECISE can be reduced to a graph matching problem, which can solved by computing maximum flow in the graph. Thus, it facilitates reliable and efficient semantic interpretation. PRECISE adopts a heuristic based approach.

2.3 ENLIGHT[13]
ENLIGHT (IntelligEnt Natural Language Interface) system is a system whose goal is to develop an interface which retrieves correct and exact information in the form of reply to a nonformal query in a natural language English. This interface was expected to be easy to use, quick and most importantly satisfying the actual needs of the user. ENLIGHT uses a shallow parsing approach thus facilitating the interface with quick response time. ENLIGHT handle the linguistic

phenomena like semantic symmetry and ambiguous modification properly.

2.4 C-Phrase[9]
C-Phrase is a natural language interface system which enables users to access the contents of PostgreSQL database. C-Phrase is a web-based natural language front end to relational databases. C-Phrase translates English queries to Codd’s tuple calculus, which is then translated to SQL and sent over the database. C-Phrase also generates natural language descriptions of tuple calculus, so that the queries may be paraphrased back to the user. C-Phrase runs under LINUX environment, connects with PostgreSQL databases via ODBC and supports both select and update queries. C-Phrase can be used in two primary cases: personalized and legacy. In the personalized case, the user is able to build dynamically and query to the personalized database by using administrative tools. In the legacy case, C-Phrase imports the schema of an already existing relational database over which an administrator then authors a natural language interface. This natural language interface is then made available to users through a web-based query interface.

2.5 Intelligent Query Interface(IQI)
Intelligent Query Interface is used for Temporal Database with Natural Language Processing in which Past, Present as well as Future Data is adopted. This system also support for validity time as the Temporal Database that holds the time variant information. IQI for temporal data uses Probabilistic Context Free Grammar by which it can access more than one table as well the temporal data. The main purpose of this system is focused for medical domain, but this is a generalized system.

2.6 GINLINDB[11]
Generic Interactive Natural Language Interface to Databases (GINLIDB) is designed by the use of UML and developed using Visual Basic.NET-2005. The user interacts with GINLIDB system in a user-friendly environment where no knowledge of computers and database terms are required. The interaction with this system is via suitable visual forms, and controls like buttons, and menus. This system is generic in nature given the appropriate database and knowledge base. GINLIDB has additional feature of spell check, by which user can correct the original query. User can extend the knowledge base by adding new vocabulary to the existing knowledge base. This system uses Augmented Transition Network handler procedure.

2.7 HLIDB
Hindi Language Interface to databases (HLIDB) has been developed in which user can input query in Hindi language and can fetch the result in the same language. This system can handle single and multiple column retrieval queries, conditional queries and join queries. In this system, Hindi Shallow Parser (developed by Language Technologies Research Centre (LTRC) at IIIT, Hyderabad) has been used to perform parsing of a sentence given in Hindi language.

2.8 PLIDB[11]
Punjabi Language Interface to databases (PLIDB) system uses the agriculture database, having tables like Farmer, Crop, and Sale. PLIDB accepts query in Punjabi language which is translated into SQL query, by mapping the Punjabi language words, with their corresponding English words with the help of maintained database. Then, the query is executed. For fetching the output in Punjabi language, PLIDB system uses the Unicode driver in MS Access and stored data in Punjabi language itself.

3. TECHNIQUES USED FOR DEVELOPING NLQBI

3.1 Pattern-Matching
Many prototype systems relied on pattern matching where user input is directly matched to the database. If the user input matches one of the patterns then the system is able to build a query for the database. In the pattern matching systems, the database details were inter-mixed within the code which has limitation for specific databases only and to the number and complexity of the patterns. The pattern-matching systems are simple as compared to other approaches. These systems are easy to implement as no elaborate parsing and interpretation modules are needed. Also, pattern-matching systems often come up with some justification, when the input is not in the range of sentences for which the patterns were designed to handle. One of the best NLP system that use pattern-matching approach is ELIZA. ELIZA rephrased the statements of the users as questions and replies the answers of those questions to the patient.

3.2 Syntax-Based Systems
In syntax-based systems, the user query is parsed and the generated parse tree is directly mapped to an expression in database query language. In syntax-based systems the generated grammar defines the possible syntactic structures of the user’s questions. The main advantage of using syntax based approach is that it delivers detailed information about the sentence structure. A parse tree includes a collection of information about the structure of a sentence; starting from a single word and its part of speech extraction, clustering of words to form a phrase structure, how related phrases can be gathered together to form complex phrases. All these are checked till the complete sentence is accessed and built. Having this information, semantic meanings can be mapped to certain production rules. The systems developed till now like LUNAR, LADDER, use this semantic grammar approach.

3.3 Semantic Grammar Systems
The basic idea of a semantic grammar system is to make the parse tree as simpler as possible. This is done by simultaneously removing unnecessary nodes or combining some of the nodes. Using this strategy, the semantic grammar system can better reflect the semantic representation without having complex parse tree structures. But semantic grammar approach needs some prior knowledge of the elements in the domain. Semantic grammar approach provides a special way for assigning a name to a tree node thus resulting into less ambiguity as compared to syntax based approach.

3.4 Intermediate Representation Languages
Most current NLQBI systems first transform the natural language question into an intermediate logical query which is expressed in some internal meaning representation language. This is done because of the difficulties of directly translating a sentence into general database query languages using syntax based approach. Here, a sentence is mapped into a logical query language, and translate this logical query language into a general database query language, such as SQL. NLQBI system developed at the University of Essex is a good example which uses a multi-stage transformation process. The first logic query is in the form of λ-calculus, this is converted to first-order predicate logic, which is again transformed to

universal domain relational calculus and domain relational calculus, then tuple relational calculus, and finally SQL. So, in this process there can be more than one intermediate meaning representation languages.

4. DEPENDENCY PARSING

Many existing parsers are used in commercial use but still need some advancement and there was the need of dependency parsers which can improve the overall system. Dependency parsing is widely used in Natural Language Processing. Dependency parsing is a technique where a sentence is given in natural language as an input and produces output in the form of dependency tree. Dependency structures contain much of predicate argument information. Dependency tree consists of lexical items linked by binary asymmetric relations called dependencies which focus on relations between words. The arcs (links) indicate certain grammatical relation between words. Each word of the sentence has dependency on exactly one parent. The tree starts with a root node. Dependency also resolves ambiguity. A dependency analysis of simple sentence is given below.

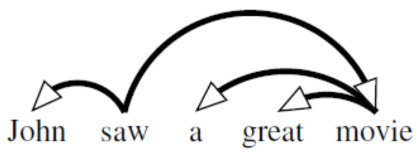

Fig.1. Dependency Parsing

4.1 Projective and Non-projective Dependency Parsing

Projective dependency parsing means dependency tree having no crossing edges. Non-Projective dependency parsing means dependency tree with crossing edges.

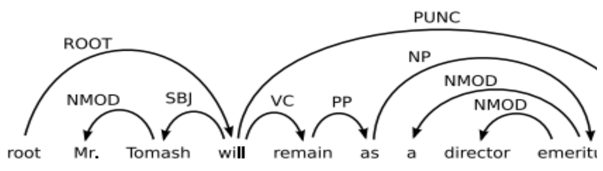

Fig.2. Projective Dependency Graph

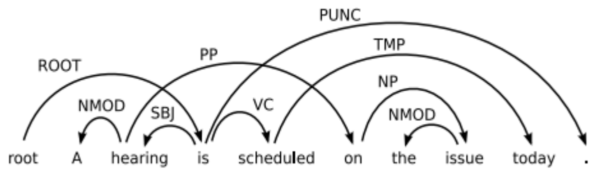

Fig.3. Non-Projective Dependency Graph

4.1.1 Projective Dependency Parsing

Eisner had developed a bottom-up dependency parsing algorithm for Projective Dependency Parsing. Adding one link at a time makes it easy to multiply the model's probability factors. Runtime complexity of this algorithm is O(n^2). Instead of storing subtrees, spans are stored. Span is a substring where no internal word links to any word outside of the span. Non-constituent spans will be concatenated into larger spans. In a span, only the end words are active (meaning they still need a parent).

4.1.2 Non-Projective Dependency Parsing

4.1.2.1 Maximum Spanning tree:

The algorithm finds MST in directed graph which means finding dependency tree with highest score. The scores are independent of other dependencies. Score of dependency is the sum of scores of dependencies in the tree. Runtime complexity of this MST algorithm is O(n^2).

4.1.2.2 Chu-Liu-Edmonds Algorithm

The main goal of Chu-Liu-Edmonds Algorithm [24] is to find the highest scoring tree for the input sentence. For each word in the graph, the incoming edge is found with the highest weight. After this check if the result obtained is a tree. If it is a tree then it's a MST else it is a cycle.

4.2 Data Driven Dependency Parsing

In Natural Language Processing a common set of characteristics can be generally assumed about the used grammars: the lexical items that define the nodes are the word forms; the parsing is concerned with only one layer (mono-stratal) of relations and the syntax of dependencies is assumed to be sufficient. Carrol and Charniak [22] propose the first dependency parsers to use data in their models, but their parser is, actually, grammar-driven. Their model is based on a formal dependency grammar and uses the corpus data only to solve disambiguations left by the grammar based model. In other words, this parsing consists in the derivation of all analysis that are permissible according to the grammar and the selection of the most probable analysis according to the generative model. Other parsers improve Carroll and Charniak's results while still being based on a formal dependency grammar in combination with a generative probabilistic model, such as the work of Eisner and Collins[21]. Samuelsson's probabilistic model goes on by allowing non-projective dependency graphs and producing labeled dependencies [19]. However, only recently, models that are not based on a formal grammar and are generated purely based on corpus data have been proposed.

4.3 Transition Based Models

In Transition-based models (or Deterministic Discriminative parsing) a deterministic parser is used to construct dependency structures by having the next action of the parser predicted by a classifier trained in the available data. In this case, no formal grammar is used when inducing the parser model. In Support Vector Machines classifiers predict the next action of a parser in order to build an unlabeled dependency structure [18]. In these systems the parsing is done according to a shift-reduce parsing technique. In this parsing, the parser is considered to be initially located at the beginning of the sentence and during each step, it selects actions from three different actions. Let the target words be w_i –1 the word before the parser – and w_i+1 – the word after the parser, the three different parser actions are mentioned as follows:

1. Shift: The parser simply moves one word along the sentence, adding no dependency relation. The target words change from w_i and w_i+1 to w_i+1 and w_i+2.

2. Right: Builds a dependency relation between words w_i and w_i+1 with the right word w_i+1 as head of the left word w_i; reduces the target words into w_i+1 , making w_i–1 and w_i+1 the new target words.

3. Left: Builds a dependency relation between words w_i and w_i+1 with the left word w_i as head of the right word w_i+1 ;

reduces the target words into w_i, making w_i-1 and w_i the new target words.

The processing of a sentence consists in passing it from left to right until no more dependency relations can be added. Since each passing may use up to n steps and up to n−1 passes may be required, the worst time complexity is $O(n^2)$. Additionally, the framework of inductive parsing proposed by Nivre et al.[19] enhances this approach with three main differences. First, this frame- work builds labeled dependency graphs, i.e., the dependency arcs have types according to what kind of dependency relation they represent. It also constructs the complete dependency graph in only a single pass over the data. Finally, instead of using Support Vector Machines, Nivre et al. [17] uses Memory-Based Learning in its classifiers.

4.4 Pseudo Projective Parsing

One of the major drawbacks in transition-based models is its inability of dealing with non-projective arcs. Only arcs between neighboring words or reduced words can be created when structure of parser is given thus limiting it to arcs under a transitive closure arcs or projective arcs. Nevertheless, a pseudo- projective approach can be applied to overcome this limitation. In pseudo-projective parsing, a preprocess step turns every non-projective arcs into projective ones. Also while doing this, the information regarding the non-projectivity is added in the label of the arc, generating a new label. This new label allows the new projective arc to be turned back to the original non-projective arc. When the parsing classifier learns to correctly label the arcs, it will also learn to predict the new pseudo-projective labels. Therefore, after a pseudo-projective parser is applied, a post processing step that is applied changes the pseudo-projective arcs back into their corresponding non-projective arcs. This pseudo-projective approach significantly improves overall parsing accuracy for non-projective corpus, obtaining the best reported performance for robust non-projective parsing of Czech [25].

4.5 Graph Based Models

Another type of models that do not use a formal grammar as basis in their parsing are graph-based models. While transition-based models try to locally find the best dependency relations, the graph-based models learn a model of the globally best dependency graph given an input sentence. Generally, graph-based models define a scoring or probability function over a set of all possible parsers. During the learning stage, the set of parameters of this function is estimated. Further, during the parsing stage, the graph that maximizes the score given by this function is built, which constructs the dependency graph. Most systems that use graph-based models differ mainly in the type and structure of the scoring function, the method to estimate the function's parameters and the search algorithm that infers the best parse given a score.

Recent advances in dependency parsing have shown that employing higher-order subtree structures in graph-based parsers can substantially improve the parsing accuracy. This work explores a new reranking approach for dependency parsing that can utilize complex subtree representations by applying efficient sub-tree selection methods[1]. The task of reranking is similar to that of parsing, except that the search of the parse tree is performed on a K-best list with selected parse candidates, instead of searching in entire search space and accordingly subtree is extracted.

An effective POS (Part-Of-Speech) tag pruning strategy which can greatly improve the decoding efficiency is a related work with graph-based DP for Chinese language. Several joint models and their corresponding decoding algorithms which can incorporate different feature sets are proposed in this paper. The experimental results show that joint models can significantly improve the state-of-the-art POS tagging parsing accuracies [3].

4.5.1 *Graph Based Algorithms*

4.5.1.1 *First Order Factored Parsing Algorithm[8]*

A "first-order" factorization is a technique, which decomposes a dependency tree into its individual dependencies. Eisner [20] introduced a widely-used dynamic programming algorithm for first-order parsing which laid the foundation for higher order factorization in the field of graph-based dependency parsing. The algorithm has two main components the complete span and the incomplete span. An incomplete span is constructed from a pair of complete spans, and a complete span is created by aggregating the incomplete span with the other half of the constituent. Hence it is a recursive process. A complete span is a 'half-constituent' of a dependency tree part. This half-constituent is headed by a head h and is modified by a modifier m. Similarly, an incomplete span can be thought of as a 'partial half-constituent', because, modifiers are added so that it can be extended to m. This can be given in figure 4

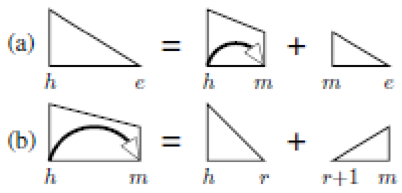

Fig. 4. First Order Parsing

4.5.1.2 *Second Order Factored Parsing Algorithm[8]*

In a second order factored algorithm a part contains two dependencies. This can be done in the following ways:

A. Sibling Factorization: This was introduced by McDonald et al. [14] where the dynamic programming structures were modified to explore the possibility of extending the parts to include the sibling information i.e. it decomposes the dependency tree into sibling parts. That is two words with a shared head word. Here, a sibling information is a triplet < h,m,s >. Extending from the previous algorithm, (h,m) and (h,s) are dependencies and s and m are successive modifiers to the same side of h. In this case, the dynamic programming structure has been augmented to include an extra structure called sibling spans. The region between successive modifiers and of a head is represented by the sibling spans. The parser combines incomplete span that represents the innermost dependency with a sibling span. Even in this case, each derivation is still defined by a span and split point only. This is given in figure 5.

B. Grandchild Factorization: In this algorithm, the information of a grandchild is a triplet < h,m,c >. Extending from the same first order factorization, (h,m) and (m,c) are now the dependencies here. Again, for this case, the dynamic programming structure is modified to include the identity of the outermost modifier of the head of the complete span. The grandchild relation changes the parsing algorithm in terms of the computational complexity. That is the complexity increases from O(n^3) to O(n^4).

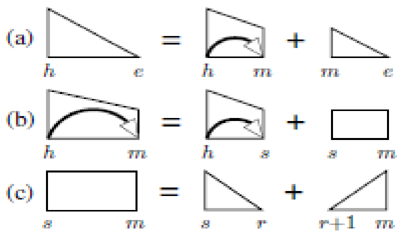

Fig 5. Second Order Algorithm

4.5.1.3 Third Order Factored Parsing Algorithm[2][8]

Third order parsing algorithms was introduced by Koo & Collins [8] which basically extends the above approaches. This is mostly by augmenting the grand-parent index. The efficiency of the third order algorithms is due to a fundamental asymmetry in the structure of a directed tree. The parsing algorithm is divided into three different models.

Model 0: All grandchildren: Here, a grandchild is a part that contains the information of the triplet < g,h,m >, where (g,h) and (h,m) are dependencies. This is showm in figure 6. For this both complete and incomplete spans are augmented with g-spans. Hence, in other words, it basically represents the same first-order algorithm, but now it includes the indices of the grandparent. Here, each derivation copies the grandparent index g into smaller g-spans. This actually causes each g-span to have non-contiguous structure. This is basically an extension of the second order grand-child factorization.

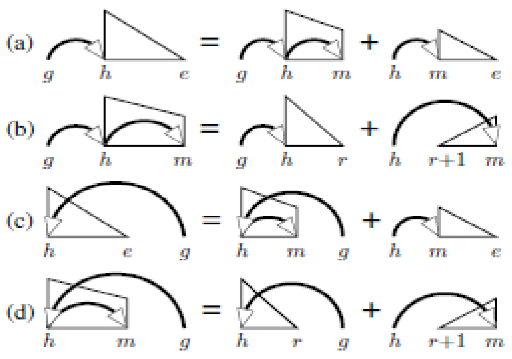

Fig. 6. Third Order Model 0

Model 1: All grand-siblings: In this case, decomposition of each tree into a set of grand-sibling parts is done which consist of the sibling parts and the grandchild parts. i.e., a grand-sibling is a quadruple < g,h,m,s > where (h,m,s) is basically the sibling part from above and (g,h,m) is the grandchild parts. It's almost like a hybrid of the aforementioned approaches. This is explained in figure 7.

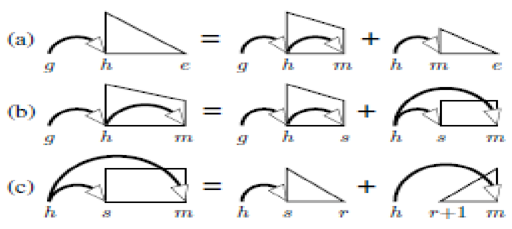

Fig. 7. Third Order Model 1

Model 2: Grand-siblings and tri-siblings: In this model, the g-span-based techniques is combined with a third-order sibling parser, resulting in a parser that captures both grand-sibling parts and tri-sibling parts—4-tuples of indices (h,m,s,t) such that both (h,m,s) and (h,s,t) are sibling parts. This is explained in figure 8. To parse this factorization, a new type of dynamic-programming structure is introduced: sibling-augmented spans, or s-spans. Here an incomplete s-span is denoted as $I_{h,m,s}$ where $I_{h,m}$ is a normal incomplete span and s is an index lying in the strict interior of the range [h,m], such that (h,m,s) forms a valid sibling part. Unlike Model 1, Model 2 produces grand-sibling parts only for the outermost pair of grandchildren.

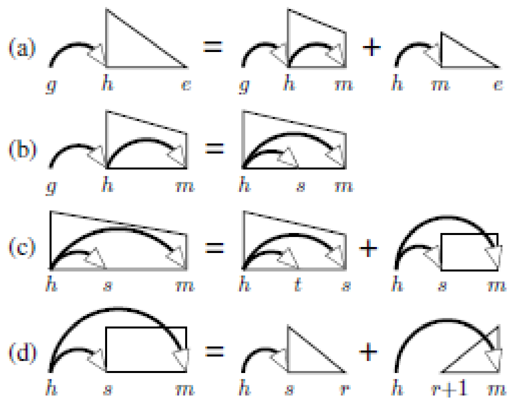

Fig. 8. Third Order Model 2

5. EXISTING PARSERS

There are some existing parsers available which are developed by using different dependency parsing techniques. These are mentioned in table 1 with their techniques used and their limitations.

Table 1 Comparison of existing dependency parsers

Parser	Methodology	Limitations
MaltParser	Data-driven transition based deterministic parser, grade of incrementality	No automatic feature engineering so big impact on quality of results
MSTParser	Graph-based parser, it is non-deterministic and non-incremental by nature	Very complex machine learning approach, lot of time and resources to train the model required
Standford Parser	Phrase structure grammar parser, works with plain text	Compact due to two stages so long parsing time
Minipar	Rule-based dependency parser for English, neither deterministic nor incremental	Performs worse as far as quality of results are concerned
MDParser	Transition-based system, which has all the application-oriented properties	Relies on machine learning, projectivity is not considered

6. CONCLUSION

Recently there are many developments in the field of NLQBI's in the last few decades. Some NLQBI systems which are discussed in this survey have been developed for the commercial use but still need some advancement. Hence there was need of different parsing techniques like dependency parsing which improves the overall system. NLQBI systems are more efficient, simple, precise and user friendly. Different dependency parsing algorithms are discussed with limitations of one overcome in the next algorithm. One of the advantage of dependency parsing is that it resolves ambiguity. There are many natural language query builder interfaces that are developed but there are no query builders that are developed using dependency parsing approach. There are many parsers that are developed using dependency parsing techniques but have some performance issues which will be solved in future research. At the last, the purpose of this research is to study the techniques and limitations of existing systems and overcome the problems in the future work.

Future work is to develop a natural language query builder interface for structured databases using dependency parsing approach. To build an algorithm which will improve the overall system in terms of feature models(like word forms, POS i.e part-of-speech, dependency type)

7. REFERENCES

[1] Mo Shen, Daisuke Kawahara, and Sadao Kurohashi, "Dependency Parse Reranking with Rich Subtree Features" IEEE transactions on audio, speech, and language processing, vol.22, no.7, July 2014

[2] Emily Pitler, "A Crossing-Sensitive Third-Order Factorization for Dependency Parsing", Transactions of the Association of Computational Linguistics -- Volume 2, Issue 1, 2014

[3] Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, and Wenliang Chen, "Joint Optimization for Chinese POS Tagging and Dependency Parsing" IEEE transactions on audio, speech, and language processing, vol.22, no.1, Jan 2014

[4] Preeti Verma, Kulwant Kaur, "Recent Developments in Natural Language Interface to Database Systems", International Journal of Innovation and Research in Computer Science, 2014

[5] Martins, M. Almeida, and N. A. Smith, "Turning on the turbo: Fast third-order non-projective turbo parsers" In Proceedings of ACL (Short Papers), pages 617–622, 2013

[6] Bohnet and J. Kuhn, "The best of both worlds – a graph-based completion model for transition-based parsers." In Proceedings of EACL, pages 77–87. 2012

[7] Himani Jain, Parteek Bhatia "Hindi Punjabi Language Interface to databases",Journal of Global Research in Computer Science, Volume 2, No. 4, Part 1 (1995), 29–81, April 2011

[8] T. Koo and M. Collins, "Efficient third-order dependency parsers", in Proc. ACL '10, pp. 1–11, 2010

[9] C-Phrase System Guide (version 1.0 beta)Michael Minock Copyright 2010 http://www.cs.umu.se/~mjm/guide.pdf

[10] Amandeep kaur "Punjabi Language Interface to databases", ME Thesis,Thapar University, june 2010

[11] Faraj A. El- Mouadib,Zakaria Suliman Zubi,Ahmed A. Almagrous, I. El- Feghi, "Interactive Natural Language Interface (GINLIDB)", ISSN: 1109-2750 664 Issue 4, Volume 8, April 2009

[12] Yunyao Li, Huahai Yang, and H. V. Jagadish, "NALIX:an Interactive Natural Language Interface for Querying XML", 2006

[13] Manish R. Joshi, "The ENLIGHT SystemIntelligEnt Natural Language Interface", Department of Computer Science, North Maharashtra University, Jalgaon 2006

[14] Ryan McDonald, "Discriminative Training and Spanning Tree Algorithms for Dependency Parsing", Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA, July 2006

[15] Ryan McDonald, Fernando Pereira ,"Non-projective Dependency Parsing using Spanning Tree Algorithms" 2005

[16] J. Nivre and J. Nilsson, "Pseudo-projective dependency parsing", In Proc. ACL, 2005

[17] Nivre, J., Hall, J. and Nilsson, J, "Memory-Based Dependency Parsing", In Ng, H. T. and Riloff, E. (eds.) Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL), pp. 49-56, 2004

[18] J. Nivre, "An efficient algorithm for projective dependency parsing", In Proc. of IWPT-2003, pages 149–160, 2003

[19] Christer Samuelsson, "A theory of stochastic grammars", In Proceedings of NLP-2000, pages 92{ 105. Springer Verlag, 2000

[20] Eisner J, "Bilexical grammars and their cubic-time parsing algorithms", In Bunt, H. C. and A. Nijholt, editors, New Developments in Natural Language Parsing, pages 29–62. Kluwer Academic Publishers, 2000

[21] Jason M. Eisner,"Three New Probabilistic Models for Dependency Parsing: An Exploration", CIS Department, University of Pennsylvania 200 S. 33rd St., Philadelphia, PA 19104-6389, USA, 1996

[22] Glenn Carroll and Eugene Charniak, "Two Experiments on learning probabilistic dependency grammers from corpora", Technical Report, TR-92, Department of Computer Science, Brown University, 1992

[23] J. Edmonds, "Optimum branchings", Journal of research of National Bureau of standards, 71B:233-240, 1967

[24] Y. J. Chu and T. J. Liu, "On the shortest arborescence of a directed graph Science" Sinica, 14:1396–1400, 1965

[25] Kubon, V, "A Robust Parser for Czech", Dissertation at MFF UK, Praha, manuscript.

[26]