Learning Rules to Extract Protein Interactions from Biomedical Text

Tu Minh Phuong, Doheon Lee*, and Kwang Hyung Lee

Department of BioSystems, KAIST
373-1 Guseong-dong Yuseong-gu Daejeon 305-701, KOREA
{phuong, dhlee, khlee}@bioif.kaist.ac.kr

Abstract. We present a method for automatic extraction of protein interactions from scientific abstracts by combing machine learning and knowledge-based strategies. This method uses sample sentences, which are parsed by a link grammar parser, to learn extraction rules automatically. By incorporating heuristic rules based on morphological clues and domain specific knowledge, this method can remove the interactions that are not between proteins and improve the performance of extraction process. We present experimental results for a test set of MEDLINE abstracts. The results are encouraging and demonstrate the feasibility of our method to perform accurate extraction without need of manual rule building.

1 Introduction

The study of protein interactions is one of the most important issues in recent molecular biology research, especially for system biology. Mining biological literature for extracting protein interactions is essential for transforming discoveries reported in the literature into a form useful for further computational analysis [5]. However, most of the interaction data are stored in free text format with irrelevant and confusing text bodies, which makes automatic querying for specific information inefficient. At the same time, manual identification and collection of protein interaction data for storing in databases is time consuming and laborious. This situation has made automatic extraction of protein interaction an attractive application for information extraction researches.

The methods proposed for protein interaction extraction differ from each other by their degrees of using natural language techniques, reliance on pre-specified protein names, statistical or knowledge-based strategies, and ability to identify interaction type.

Marcotte et. al. [5] used Bayesian approach to score the probability that an abstract discusses the topic of interest. The score is computed based on the frequencies of discriminating words found in the abstracts. However, this method allows only ranking and selecting abstracts. Actual extraction must be performed manually.

* Corresponding author

K.-Y. Whang, J. Jeon, K. Shim, J. Srivatava (Eds.): PAKDD 2003, LNAI 2637, pp. 148–158, 2003.

A number of researchers have taken simple pattern-matching approach to extract interactions. Blaschke et. al. [1] used the simple pattern "protein A…interaction verb…protein B" to detect and classify protein interactions in abstracts related to yeast. They simplified the task assuming that protein names are pre-specified by users. Ono and colleagues [7] presented a similar approach. By using part-of-speech tags and simple rules, this approach transforms complex and compound sentences into simple ones before applying pattern matching. However, this approach cannot automatically detect new protein names. Instead, it uses lexical lookup and requires protein name dictionaries.

Several publications proposed using natural language processing techniques for the extraction task. Rindflesch et. al. [10] described EDGAR, a system that used a stochastic part-of-speech tagger, a syntactic parser, and extensive knowledge source (Unified Medical Language System) to identify names and binding relationships. Other linguistic motivated approaches use different grammar formalisms with shallow or full syntactic parsing. Yakushiji et. al. [15] applied a full syntactic parser to produce so called *argument structure* of whole sentences before mapping the argument structure to extraction slots. Park and his colleagues [8] used a syntactic parser with combinatory categorical grammar to analyze sentences. However, without special protein name verification this approach gives recall and precision rates of only 48 and 80 respectively. Thomas et. al. [14] adapted general-purpose information extraction system Highlight to protein interaction extraction by tuning the vocabulary and the part-of-speech tagger. This system groups sequences of words into phrases by cascaded finite state machines. The authors added some kind of protein names verification to the system and reported the interesting fact that more restrict name verification rules lead to higher recall and precision.

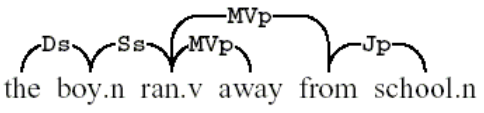
All of the mentioned approaches and systems use hand-crafted extraction rules. Tuning rules manually requires significant time and domain-specific knowledge. To overcome this knowledge-engineering bottleneck, a number of automatic rule learning systems have been made. One of such systems, RAPIER [2], employs inductive logic programming techniques to learn so-called extraction fillers. Another system, WHISK [12], learns extraction rules by gradually adding lexical and/or semantic terms to an empty rule. However, these systems are designed for extracting only independent pieces of data from general text. It is difficult to adapt these systems to extracting relationships between terms – interactions between proteins in our case.

In this paper, we show how extraction rules can be automatically learned from training samples. We use the link grammar parser to parse sentences. The sentences with links are tagged by the user and serve input for the learning procedure. The learned rules are then used to detect potential protein interactions. We describe the heuristic rules that verify detected nouns and noun phrases if they are protein names or not. The results and evaluation of an experiment on a test set of Medline abstracts related to *Saccharomyces cerevisiae* are presented.

The remainder of the paper is organized as follows. A brief description of a link grammar is given in subsection 1.1. Section 2 describes extraction process and heuristic rules for protein name verification. The rule learning algorithm is described in section 3. Experimental results are presented and discussed in section 4. Section 5 draws our conclusion.

1.1 Link Grammar

Link grammar is a dependency grammatical system introduced by D. Sleator, D. Temperley, and J. Lafferty [11]. Rather than examine the constituents or categories a word belongs to, the link grammar is based on a model that words within a text form links with one another. A sample parse result by a link grammar parser for sentence "The boy ran away from school" is shown in Fig.1.



D: connects determiner to nouns
S: connects subject-nouns to verbs
MV: verbs (adjectives) to modifying phrases
J: connects prepositions to their objects

Fig. 1. A sample parse with links

The arcs between words are called "links" with the labels showing the link types. For example, the verb "ran" is connected to "from school" identifying a prepositional phrase by the link "MV". The part-of-speech tags (nouns, verbs and so on) are also added to some words as suffixes.

The uppercase letters of the link labels indicate the primary types of links (there are 107 primary link types for the link parser version 4.1), and the lowercase letters detail the relationships of words. The meanings of several link labels uppercase letters are given in the right part of Fig.1.

Each word in the link grammar must satisfy the linking requirements specifying which types of links it can attach and how they are attached. These linking requirements are stored in a dictionary. It is easy to express the grammar of new words or words with irregular usage. We can just define the grammar for these words and add them to the dictionary. It is how we deal with domain-specific terms and jargon in our work.

The reason why we have decided to adopt the link grammar in our work is that the grammar provides the simple and effective way to express relationships between terms in a sentence. This feature is very important in detecting protein interactions. We can follow the related links to find participants of an interaction without concerning the rest of the sentence.

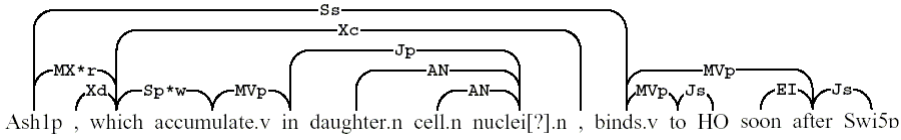
2 Extraction of Protein Interactions

In this section, we describe how to extract protein interactions using extraction rules and heuristic rules for protein name verification. First, sentences are parsed by the link parser; words are stemmed by the Porter stemming algorithm [9] to solve the problem of inflection. Next, our method looks for terms involved in interactions by following the links described in the extraction rules. Finally, the heuristic rules are applied to verify if the terms detected from the previous step are protein names or not. We describe the details for each step below.

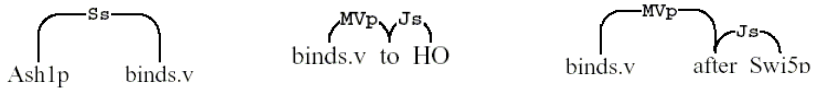
2.1 Rule Representation

In our method, rules are subsets of links that connect terms describing an interaction within a sentence. Our method requires three terms for an interaction: two protein names and a keyword that indicates the type of relationship betweens the proteins. We use the following keywords: "interact", "bind", "associate", and "complex" as well as their inflections.

A rule always begins with a keyword and contains all intermediate links that connect the keyword to the first and the second proteins. Consider a sample sentence, which has the parse result shown in Fig.2a. In this example, the keyword is "binds", the pairs of proteins are "Ash1p" - "HO" and "Ash1p" - "Swi5p". The links that connect "binds" with the pairs of proteins names are shown in Fig. 2b. We can think of the parsed sentence as a graph where words are vertices and links are edges. Then the connection between the keyword and a protein name is the shortest path between these two vertices. This connection is recorded as a rule.



(a) Parsed sentence with links



(b) Links from the keyword to protein names

Fig. 2. A sample of connections between a keyword and protein names

In order to store and process rules, we have adopted textual presentation for rules. A rule for the above example is shown in Fig. 3.

```
bind S- @NAME1
bind MV+ to J+ @NAME2
=>bind(@NAME1,@NAME2)
```

Fig. 3. An example of rule representation

Each rule consists of three lines. The first and second lines specify the links to follow for detecting the first and second protein names respectively. The third line is a template for interaction output. For example, the second line of the sample rule in Fig.3 looks for keyword "bind". If the keyword is found, the rule looks further for a word "to" which is on the right of "bind" and which is connected to "bind" by a link "MV". The next step is looking rightward for a word connected to "to" by "J". This word will be extended and considered as a candidate for protein name. The procedure for name extension will be described in the next subsection. Our rules contain only the part of link labels that consists of uppercase letters. The signs "+" and "-" next to

the link labels represent search directions for right and left respectively. We call intermediate words in link paths between keywords and protein names (for example word "to" in the above rule) nodes. A rule line can have any number of nodes.

Given a sentence, a rule is said to be satisfied if we can find all links and words specified in the rule within this sentence. A rule can be applied many times to each sentence to find all interactions satisfying the rule. For instance, applying the above rule to the sentence depicted in Fig.2 will output two candidates of interactions *=>bind(Ash1p,HO)* and *=>bind(Ash1p,Swi5p)*.

Our rules allow a form of disjunction as well as use of wildcard "*". Rule lines can look like the following:

```
bind MV+ to|after J+ @NAME2
```
or
```
bind MV+ * J+ @NAME2
```

the first line looks for either "to" or "after", connected by "MV" to "bind", whereas the second lines allows any word on the right of "bind" which are connected to "bind" with "MV" link.

2.2 Name Extension and Verification

In practice, many protein names are compound words. For example, in the sample sentence shown in Fig.4, protein names are "general transcription factor" and "TATA binding protein" but not "IIA" and "TATA". The rule matching procedure described above can detect only two words "IIA" and "protein". Thus, an additional step is necessary for capturing compound names. The following procedure is designed to solve this problem.

(a) If the leftmost word of a name is connected to the next word on the left by links "G" or "GN" then extend the name to the left one word. From the above example, we have: *IIA=>factor IIA*
(b) If the leftmost word of a name is connected to word or words on the left by links "A" or "AN" then extend the name to the left by adding all the connected words: *factor IIA => general transcription factor IIA*.
(c) If the rightmost word of a name is connected to a word on the right with link "G" or "GN" then extend the name by adding the connected word to the right.

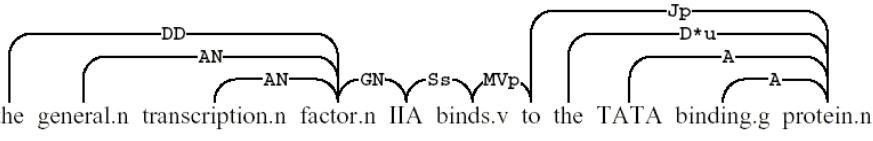


Fig. 4. An example of protein names which are compound words

Having an interaction with names detected as described above, we have to verify that it is a desired interaction between two proteins rather than an interaction between two non-proteins. The most obvious approach is to use a protein name dictionary for

checking whether the detected names are protein names or not. Unfortunately, the number of new proteins is growing rapidly, which challenges maintaining the dictionary up to date. In this work, we verify detected names by applying several heuristic rules (part of rules are adopted from [4], [13] and [14]) that give some score. Examples of rules are:

- If a name contains word or words with uppercase letters, digits, some special symbols, the Greek letters then give the name score 1.0.
- If a name ends with one of the following words (*molecule, gene, bacteria, base*) then reject the name.
- If a name ends with a word that has suffixes "ole, ane, ate, ide, ine, ite, ol, ose" then reject the name.
- If a name is compound words containing functional descriptor (*adhesion, channel, filament, junction*), activity descriptors (*regulated, releasing, promoting, stimulating*), other keywords (*receptor, factor, protein*) then give the name score 1.0.
- If a name is compound words without special words described above then give the name score 0.5.
- If a name is single word with suffix "-in" then give score 0.5.

If a rejecting rule is triggered, the name is removed. If several scoring rules are triggered, the highest score is recorded. Names with the scores higher than a predefined threshold will be accepted as protein names. By adjusting the threshold, we can emphasize the importance of precision (with higher threshold) or the importance of recall (with lower threshold).

3 Learning Extraction Rules

In this section we describe how to learn extraction rules automatically. Our algorithm requires a set of hand-tagged sentences. During the tagging process the user must explicitly point out the first protein name, the second protein name, and the keyword of each interaction. There may be more than one protein pair linked by one keyword as well as more than one keyword within a sentence. For example, the sentence presented in Fig. 4 would be tagged as

```
The /n1 general transcription factor IIA/n0 /v1
binds/v0 to the /n2 TATA binding protein/n0
```

Our algorithm begins with creating a rule for each interaction being tagged in the training set. To create the first line of a rule, the algorithm looks in the link-parsed sentence for the shortest path from the keyword to any word of the first name. The rule line for the second name is created in the similar way. Applying this processing to all the examples of a training set, we get a set of rules, each per a tagged interaction. Some of rules can have duplicates. These duplicates are removed by a pruning procedure. The rules retained after pruning are referred to in this paper as specific rules.

There are two general design approaches for rule learning systems: compression (or bottom-up) and covering (top-down). Compression-based systems begin with a set

of highly specific rules, typically one for each example, and gradually compress rule sets by constructing more general rules, which subsume more specific ones. This approach has been chosen in designing our algorithm. The reason of our choice is our preference of overly specific rules to overly general ones. In information extraction, there is a trade-off between high precision and high recall. For the potential application of our algorithm – populating protein interactions into databases – precision must be emphasized. The bottom-up approach tends to learn more specific rules, which also are more precise.

The learning algorithm is shown in Fig. 5. Starting with a set of specific rules, the algorithm generalizes rules by repeatedly calling two procedures GENERALIZE_TERM and GENERALIZE_FRAGMENT. Each of the procedures produces a set of more general rules CandidateSet. The rules used to build CandiadetSet are stored in BaseSet. From the CandidateSet returned by GENERALIZE_TERM, the algorithm looks for the best rule r_0. This rule will subsume the rules in BaseSet if its evaluation is not worse than the overall evaluation of BaseSet. In the case of calling GENERALIZE_FRAGMENT, the whole CandidateSet will subsume the whole RuleSet if the former performs not worse than the latter.

To evaluate a rule r we use the Laplace estimation, given by:

$$\text{Score}(r) = \frac{p+1}{n+1}$$

where n is the number of extractions made on the training set by r, and p is the number of correct extractions.

The GENERALIZE_TERM procedure. This procedure performs generalization over rule terms. We use word "term" to refer to any word staying at a node of the link path given by the first line or the second line of a rule. For example, the rule shown in Fig.3 has term "to" in the second line. The keyword and the variables @NAME are special terms and are not considered during term generalization.

The procedure looks for rules that are different only by terms at one node in one of first two lines. Such rules can be found easily by regular expression matching. Consider the following rule lines:

```
interaction M+ between J+ @NAME2
interaction M+ of J+ @NAME2
interaction M+ with J+ @NAME2
```

These rule lines are different only by terms at the second node (denoted by i). If such rules are found, they are added to BaseSet. The procedure then performs generalization by creating two rules, one by replacing term at node i with disjunctions of terms i from rules in BaseSet, and another by replacing term at node i by wildcard '*'. These two rules will form CandidateSet.

From the example above we have the following rules after generalization

```
interaction M+ of|between|with J+ @NAME2
```

and

```
interaction M+ * J+ @NAME2
```

```
RuleSet ← the set of specific rules from examples
loop
    GENERALIZE_TERM (RuleSet, BaseSet, CandidateSet)
    if CandidateSet is not empty
        find rule r0∈ CandidateSet with Score(r0)=max r∈CandidateSet Score(r)
        if Score(r0) >= Σr∈BaseSet Score(r)
            RuleSet = r0∪(RuleSet−BaseSet)
until CandidateSet is empty
loop
    GENERALIZE_FRAGMENT(RuleSet, CandidateSet)
    if CandidateSet is not empty
      if Σr∈CandidateSet Score(r) >= Σr∈RuleSet Score(r)
        RuleSet ← CandidateSet
until CandidateSet is empty

GENERALIZE_TERM (RuleSet, BaseSet, CandidateSet)
    BaseSet ← { }
    CandidateSet ← { }
    for rule r∈RuleSet
      for i=1 to number of nodes of r
        find all rules∈RuleSet that differ from r by only the terms at node i
        if such rules found
            BaseSet ← the found rules
            CandidateSet ← disjuntions of the found rules
            CandidateSet = CandidateSet∪r with term i replaced by '*'
            Return

GENERALIZE_FRAGMENT(RuleSet, CandidateSet)
    CandidateSet ← { }
      For rule r∈RuleSet
        Find a rule r′∈RuleSet that differ from r only by the suffix of one line
        if r′is found
            s ← suffix of r
            s′← suffix of r′
            for each rule p∈RuleSet with suffix sp
                if sp=s OR sp=s′
                    replace sp by s|s′
                    CandidateSet = CandidateSet ∪ p
```

Fig. 5. The learning algorithm

The GENERALIZE_FRAGMENT procedure. We call *rule fragment* (or just *fragment*) any part of a rule line (the first or the second line) that begins with a term, which is not a keyword, and ends with a term (another or the same). In the example above a fragment can be "of J+ @NAME2" or "@NAME2". We call *suffix* any fragment that contains the rightmost term. The procedure looks for a pair of rules that differ from each other only by the suffixes of one line. If such pair is found, the suffixes of the rules are recorded. There may be more than one pair of suffixes for

each pair of rules. Then the procedure looks for all rules in RuleSet with suffixes identical to one of the recorded suffixes. Is such rules are found the procedure builds more general rules from them by replacing their suffixes with the disjunction of the recorded suffixes. The new rules are then added to CandidateSet. In the example below, the first two rule lines have two pairs of suffixes (`to J+ @NAME2; to J+ domain M+ of J+ @NAME2`) and (`@NAME2; domain M+ of J+ @NAME2`)

```
bind MV+ to J+ @NAME2
bind MV+ to J+ domain M+ of J+ @NAME2
bind O+  to J+ @NAME2
```

Generalization using these suffixes produces the following rules

```
bind MV+ (to J+ @NAME2)|(to J+ domain M+ of J+ @NAME2)
bind O+  (to J+ @NAME2)|(to J+ domain M+ of J+ @NAME2)

bind MV+ to J+ (@NAME2)|(domain M+ of J+ @NAME2)
bind O+  to J+ (@NAME2)|(domain M+ of J+ @NAME2)
```

The underlying assumption of fragment generalization is that if two different suffixes are found in the same position of two similar rules, the suffixes probably can appear in similar contexts and therefore can replace each other in other rules.

Example. As an example of the learning process, consider generalizing the rules based on the following three sentences (only fragments are shown)

> "While *Scd2* interacted with the R1 N-terminal domain of *Shk1*…"
> "The interaction between *Sec1p* and *Ssop* is…"
> "… we observed an interaction of *Sp1* and *ZBP-89*…"

The following specific rules are created after parsing and processing links for these sentences (for the purpose of this example, we consider only the second lines of the rules produced)

```
interact M+ with J+ domain M+ of J+ @NAME2
interact M+ between J+ @NAME2
interact M+ of J+ @NAME2
```

During term generalization, the second and third lines are found to have only different terms "`between`" and "`of`". Thus, the lines are generalized to:

```
interact M+ with J+ domain M+ of J+ @NAME2
interact M+ * J+ @NAME2
```

During fragment generalization, these lines give two suffixes "`@NAME2″` and "`domain M+ of J+ @NAME2`", which are used to build the following general rule

```
interact M+ * J+ (@NAME2)|(domain M+ of J+ @NAME2)
```

This rule can be used, for example, to find the interaction between *Spc72p* and *Kar1p* from the sentence:

> "Here we show that the interaction between yeast protein *Spc72p* and the N-terminal domain of *Kar1p*…."

which cannot be detected by the initial specific rules.

4 Experimental Results

We present here experimental results on a set of abstracts from Medline, a literature database available through PubMed [6]. The abstracts were obtained by querying Medline with the following keywords: "Saccharomyces cerevisiae" and "protein" and "interaction". We filtered the returned 3343 abstracts and retained 550 sentences containing at least one of four keywords "interact", "bind", "associate", "complex" or one of their inflections.

We adopt the standard *cross validation* methodology to test our algorithm. The data collection is partitioned several times into a training set and a testing set, rules are learned using the training set and then are evaluated using the test set. In our experiment, ten-fold cross validation was done on the set of sentences. An extracted interaction is considered correct if both extracted protein names are identical to those tagged by the user. The order of protein names (which is the first, which is the second) is not taken into consideration although experimental results show that the order is retained.

We evaluate performance in terms of *precision*, the number of correct extracted interactions divided by the total number of extracted interactions, and *recall*, the number of correct extracted interactions divided by the total number of interactions actually mentioned in the sentences.

In order to analyze the effect of term generalization and fragment generalization on the results, four versions of the learning algorithm were tested. The first version uses only specific rules without any generalization. The other two versions use either term or fragment generalization. The full version uses both type of generalization as shown in Fig. 5. Recalls and precisions of the four versions are given in Table. 1.

Table 1. Results of different versions of the learning algorithm

	Recall (%)	Precision (%)
Without generalization	41	93
With term generalization only	48	89
With fragment generalization only	49	91
Full algorithm	60	87

These results show that whereas the generalization slightly decreases precision, it leads to valuable improvement on recall. The results also show a little advantage of fragment generalization over term generalization.

There are a number of publications addressing the similar task of extracting protein interaction. Unfortunately, it is not simple to quantitatively compare our method with these alternatives because they use different text corpora, different assumptions about protein names, and different treatment of errors. For instance, Ono et.al. [7] describe an extraction method with high recall and precision of 86% and 94% respectively. However, they required the presence of protein name dictionaries, which are not always available. It is more reasonable to compare our approach with those that do not require pre-specified protein names or dictionaries. Thomas and his colleague [14] present one of such systems with 58% recall and 77% precision. Park et. al. [8] describe an extraction method using a combinatory categorical grammar for parsing and detecting interactions. They report recall and precision rates of 48 and 80 respectively. Both of the systems require hand-crafted patterns or rules for detecting protein interactions.

5 Conclusion

We have described an algorithm that automatically learns information extraction rules from training sentences. The rules learned by our algorithm can be used in combination with heuristic rules, which verify whether a noun phrase is protein name or not, to extract protein interactions from scientific abstracts. The learning and extraction algorithms exploit the link grammar parser to parse input sentences. The grammar has been shown appropriated for expressing relationships between words. This makes it possible to design a relative simple learning algorithm that can achieve accurate extraction performance without the need of manual rule building.

Acknowledgement. We thank Dokyun Na and Hyejin Kam for tagging training examples. The first author was supported by The Korea Foundation for Anvanced Studies and The Natural Science Council of Vietnam.

References

1. Blaschke, A., Andrade, M.A., Ouzounis, C., Valencia, A.: Automatic extraction of biological information from scientific text: protein-protein interactions. In: Proceedings of the 5th Int. Conference on Intelligent Systems for Molecular Biology. AAAI Press (1999).
2. Califf, M.E.: Relational learning techniques for natural language information extraction. PhD thesis. University of Texas, Austin (1998).
3. Freitag, D.:Machine learning for information extraction in informal domains. In: Machine learning, 39. Kluwer Academic Publishers.(2000).
4. Fukuda, K., Tamura, A., Tsunoda, T., Takagi, T.: Toward information extraction: identifying protein names from biological papers. In: Proceedings of the Pacific Symposium on Biocomputing. (1998).
5. Marcotte, E.M., Xenarios, I., Eisenberg, D.: Mining literature for protein-protein interactions. Bioinformatics. 17(4). Oxford University Press (2001)
6. Medline Pubmed: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed
7. Ono, T., Hishigaki, H., Tanigami, A., Takagi, T.: Automated extraction of information on protein-protein interactions from the biological literature. In: Bioinformatics. 17(2) (2001).
8. Park, J.C., Kim, H.S., Kim, J.J.: Bidirectional Incremental Parsing for Automatic Pathway Identification with Combinatory Categorial Grammar. In: Proceedings of the Pacific Symposium on Biocomputing (2001).
9. Porter, M.F.: An algorithm for suffix stripping. In: Program 14 (1980).
10. Rindflesch, T.C.,Tanabe, L.,Weinstein, J.,Hunter, L.: EDGAR: extraction of drugs, genes and relations from the biomedical literature. In: Proceedings of the Pacific Symposium on Biocomputing (2000).
11. Sleator, D., Temperley, D.: Parsing English with a Link Grammar. In: Proceedings of 3d International Workshop on Parsing Technologies (1993).
12. Soderland, S.: Learning information extraction rules for semi-structured and free text. In: Machine learning, 34. Kluwer Academic Publishers.(1999).
13. Tanabe, L., Wilbur, W.: Tagging gene and protein names in biomedical text. In: Bioinformatics. 18(8). Oxford University Press (2002).
14. Thomas, J., Milward, D., Ouzounis, C., Pulman, S., Carroll, M.: Automatic extraction of protein interactions from scientific abstracts. In: Proceedings of the Pacific Symposium on Biocomputing (2000).
15. Yakushiji, A., Tateisi, Y., Miyao, Y., Tsujii, J.: Event Extraction from Biomedical Papers Using a Full Parser. In: Proceedings of the Pacific Symposium on Biocomputing (2001).