Extracting Complex Biological Events with Rich Graph-Based Feature Sets

Jari Björne,[1] **Juho Heimonen**,[1,2] **Filip Ginter**,[1] **Antti Airola**,[1,2]
Tapio Pahikkala[1] and **Tapio Salakoski**[1,2]
[1]Department of Information Technology, University of Turku
[2]Turku Centre for Computer Science (TUCS)
Joukahaisenkatu 3-5, 20520 Turku, Finland
firstname.lastname@utu.fi

Abstract

We describe a system for extracting complex events among genes and proteins from biomedical literature, developed in context of the BioNLP'09 Shared Task on Event Extraction. For each event, its text trigger, class, and arguments are extracted. In contrast to the prevailing approaches in the domain, events can be arguments of other events, resulting in a nested structure that better captures the underlying biological statements. We divide the task into independent steps which we approach as machine learning problems. We define a wide array of features and in particular make extensive use of dependency parse graphs. A rule-based post-processing step is used to refine the output in accordance with the restrictions of the extraction task. In the shared task evaluation, the system achieved an F-score of 51.95% on the primary task, the best performance among the participants.

1 Introduction

In this paper, we present the best-performing system in the primary task of the BioNLP'09 Shared Task on Event Extraction (Kim et al., 2009).[1] The purpose of this shared task was to competitively evaluate information extraction systems targeting complex events in the biomedical domain. Such an evaluation helps to establish the relative merits of competing approaches, allowing direct comparability of results in a controlled setting. The shared task was the first competitive evaluation of its kind in the BioNLP field as the extraction of complex events became possible only recently with the introduction of corpora containing the necessary annotation: the GENIA event corpus (Kim et al., 2008a) and the BioInfer corpus (Pyysalo et al., 2007).

The objective of the primary task (Task 1) was to detect biologically relevant events such as protein binding and phosphorylation, given only annotation of named entities. For each event, its class, trigger expression in the text, and arguments need to be extracted. The task follows the recent movement in BioNLP towards the extraction of semantically typed, complex events the arguments of which can also be other events. This results in a nested structure that captures the underlying biological statements more accurately compared to the prevailing approach of merely detecting binary interactions of pairs of biological entities.

Our system is characterized by heavy reliance on efficient, state-of-the-art machine learning techniques and a wide array of features derived from a full dependency analysis of each sentence. The system is a pipeline of three major processing steps: trigger recognition, argument detection and semantic post-processing. By separating trigger recognition from argument detection, we can use methods familiar from named entity recognition to tag words as event triggers. Event argument detection then becomes the task of predicting for each trigger–trigger or trigger–named entity pair whether it corresponds to an actual instantiation of an event argument. Both steps can thus be approached as classification tasks. In contrast, semantic post-processing

[1] http://www-tsujii.is.s.u-tokyo.ac.jp/GENIA/SharedTask

Proceedings of the Workshop on BioNLP: Shared Task, pages 10–18,
Boulder, Colorado, June 2009.

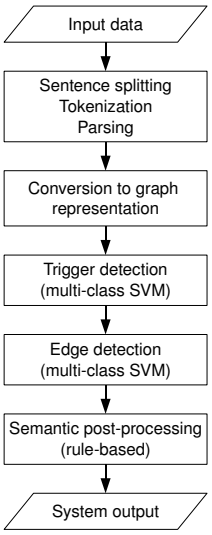

Figure 1: The main components of the system.

is rule-based, directly implementing argument type constraints following from the definition of the task.

In the following sections, we present the implementation of the three stages of our information extraction system in detail, and provide insights into why we chose the approach we did. We also discuss alternate directions we followed but that did not improve performance. Finally, we analyze the overall performance of our system in the shared task as well as evaluate its components individually.

2 The system description

The overall architecture of the system is shown in Figure 1. All steps in the system process one sentence at a time. Since 95% of all annotated events are fully contained within a single sentence, this does not incur a large performance penalty but greatly reduces the size and complexity of the machine learning problems.

2.1 Graph representation

We represent the extraction target in terms of semantic networks, graphs where the nodes correspond to named entities and events, and the edges correspond to event arguments. The shared task can then be viewed as the problem of finding the nodes and edges of this graph. For instance, nested events are naturally represented through edges connecting two event nodes. The graph representation of an example sentence is illustrated in Figure 2D.

We have previously used this graph representation for information extraction (Heimonen et al., 2008; Björne et al., 2009) as well as for establishing the connection between events and syntactic dependency parses in the Stanford scheme of de Marneffe and Manning (2008) (Björne et al., 2008).

2.2 Trigger detection

We cast trigger detection as a token labeling problem, that is, each token is assigned to an event class, or a negative class if it does not belong to a trigger. Triggers are then formed based on the predicted classes of the individual tokens. Since 92% of all triggers in the data consist of a single token, adjacent tokens with the same class prediction form a single trigger only in case that the resulting string occurs as a trigger in the training data. An event node is created for each detected trigger (Figure 2B).

In rare cases, the triggers of events of different class share a token, thus the token belongs to several separate classes. To be able to approach trigger detection as a multi-class classification task where each token is given a single prediction, we introduce combined classes as needed. For instance the class *gene expression/positive regulation* denotes tokens that act as a trigger to two events of the two respective classes. Note that this implies that the trigger detection step produces at most one event node per class for any detected trigger. In the shared task, however, multiple events of the same class can share the same trigger. For instance, the trigger *involves* in Figure 2 corresponds to two separate *regulation* events. A separate post-processing step is introduced after event argument detection to duplicate event nodes as necessary (see Section 2.4).

Due to the nature of the GENIA event annotation principles, trigger detection cannot be easily reduced to a simple dictionary lookup of trigger expressions for two main reasons. First, a number of common textual expressions act as event triggers in some cases, but not in other cases. For example, only 28% of the instances of the expression *activates* are triggers for a *positive regulation* event while the remaining 72% are not triggers for any event. Second, a single expression may be associated with various event classes. For example, the instances of the token *overexpression* are evenly distributed among

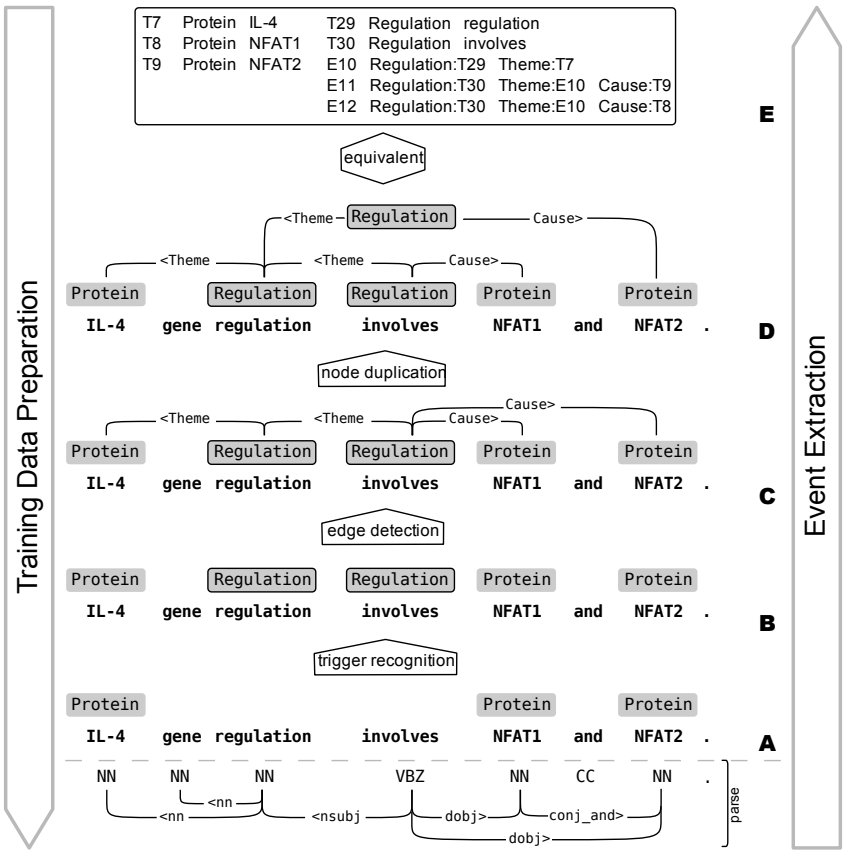

Figure 2: An example sentence from Shared Task document 10069428 (simplified). A) Named entities are given. B) Triggers are detected and corresponding event nodes are created. C) Event argument edges are predicted between nodes. The result is a sentence-level semantic network. D) One node may denote multiple events of the same class, therefore nodes are duplicated in the semantic post-processing step. E) The resulting graph can be losslessly transformed into the Shared Task event annotation. Training data for the trigger recognizer includes named entity annotation (A) and for the edge detector the semantic network with no node duplication (C).

gene expression, *positive regulation*, and the negative class. In light of these properties, we address trigger detection with a multi-class support vector machine (SVM) classifier that assigns event classes to individual tokens, one at a time. This is in contrast to sequence labeling problems such as named entity recognition, where a sequential model is typically employed. The classifier is trained on gold-standard triggers from the training data and incorporates a wide array of features capturing the properties of the token to be classified, both its linear and dependency context, and the named entities within the sentence.

Token features include binary tests for capitalization, presence of punctuation or numeric characters, stem using the Porter stemmer (Porter, 1980), character bigrams and trigrams, and presence of the token in a gazetteer of known trigger expressions and their classes, extracted from the training data. Token features are generated not only for the token to be classified, but also for tokens in the immediate linear context and dependency context (tokens that govern or depend on the token to be classified).

Frequency features include the number of named entities in the sentence and in a linear window around the token in question as well as bag-of-word counts of token texts in the sentence.

Dependency chains up to depth of three are constructed, starting from the token to be classified. At each depth, both token features and dependency type are included, as well as the sequence of dependency types in the chain.

The trigger detector used in the shared task is in fact a weighted combination of two indepen-

dent SVM trigger detectors, both based on the same multi-class classification principle and somewhat different feature sets.[2] The predictions of the two trigger detectors are combined as follows. For each trigger detector and each token, the classifier confidence scores of the top five classes are re-normalized into the $[0, 1]$ interval. The renormalized confidence scores of the two detectors are then linearly interpolated using a parameter $\lambda,\ 0 \leq \lambda \leq 1$, whose value is set experimentally on the development set, as discussed below.

Setting the correct precision–recall trade-off in trigger detection is very important. On one hand, any trigger left undetected directly implies a false negative event. On the other hand, the edge detector is trained on gold standard data where there are no event nodes without arguments, which creates a bias toward predicting edges for any event node the edge detector is presented with. On the development set, essentially all predicted event nodes are given at least one argument edge. We optimize the precision–recall trade-off explicitly by introducing a parameter $\beta,\ 0 \leq \beta$, that multiplies the classifier confidence score given to the negative class, that is, the "no trigger" class. When $\beta < 1$, the confidence of the negative class is decreased, thus increasing the possibility of a given token forming a trigger, and consequently increasing the recall of the trigger detector (naturally, at the expense of its precision).

Both trigger detection parameters, the interpolation weight λ and the precision–recall trade-off parameter β, are set experimentally using a grid search to find the globally optimal performance of the entire system on the development set, using the shared task performance metric. The parameters are thus not set to optimize the performance of trigger detection in isolation; they are rather set to optimize the performance of the whole system.

2.3 Edge detection

After trigger detection, edge detection is used to predict the edges of the semantic graph, thus extracting event arguments. Like the trigger detector, the edge detector is based on a multi-class SVM classifier. We generate examples for all potential edges, which are always directed from an event node to another event node (event nesting) or from an event node to a named entity node. Each example is then classified as *theme*, *cause*, or a negative denoting the absence of an edge between the two nodes in the given direction. It should be noted that even though event nodes often require multiple outgoing edges corresponding to multiple event arguments, all edges are predicted independently and are not affected by positive or negative classifications of other edges.

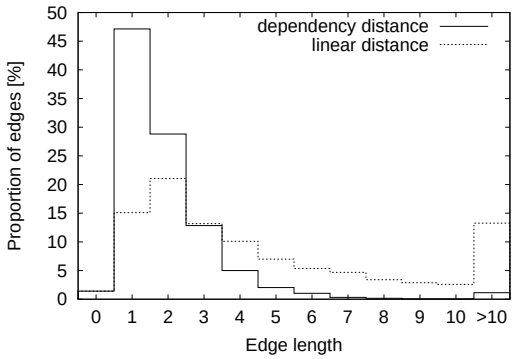

Figure 3: The distribution of event argument edge lengths measured as the number of dependencies on the shortest dependency path between the edge terminal nodes, contrasted with edge lengths measured as the linear token distance.

The feature set makes extensive use of syntactic dependencies, in line with many recent studies in biomedical information extraction (see, e.g. (Kim et al., 2008b; Miwa et al., 2008; Airola et al., 2008; Van Landeghem et al., 2008; Katrenko and Adriaans, 2008)). The central concept in generating features of potential event argument edges is the *shortest undirected path of syntactic dependencies* in the Stanford scheme parse of the sentence which we assume to accurately capture the relationship expressed by the edge. In Figure 3, we show that the distances among event and named entity nodes in terms of shortest dependency path length are considerably shorter than in terms of their linear order in the sentence. The end points of the path are the syntactic head tokens of the two named entities or event triggers. The head tokens are identified using a simple heuristic. Where multiple shortest paths exist, all are considered. Most features are built by combining the attributes of multiple tokens (token text,

[2] This design should be considered an artifact of the time-constrained, experiment-driven development of the system rather than a principled design choice.

POS tag and entity or event class, such as *protein* or *binding*) or dependencies (type such as *subject* and direction relative to surrounding tokens).

***N*-grams** are generated by merging the attributes of 2–4 consecutive tokens. *N*-grams are also built for consecutive dependencies. Additional trigrams are built for each token and its two flanking dependencies, as well as for each dependency and its two flanking tokens. These *N*-grams are defined in the direction of the potential event argument edge. To take into account the varying directions of the dependencies, each pair of consecutive tokens forms an additional bigram defining their governor-dependent relationship.

Individual component features are defined for each token and edge in a path based on their attributes which are also combined with the token/edge position at either the interior or the end of the path. Edge attributes are combined with their direction relative to the path.

Semantic node features are built by directly combining the attributes of the two terminal event/entity nodes of the potential event argument edge. These features concatenate both the specific types of the nodes (e.g. *protein* or *binding*) as well as their categories (event or named entity). Finally, if the events/entities have the same head token, this self-loop is explicitly defined as a feature.

Frequency features include the length of the shortest path as an integer-valued feature as well as an explicit binary feature for each length. The number of named entities and event nodes, per type, in the sentence are defined for each example.

We have used this type of edge detector with a largely similar feature set previously (Björne et al., 2009). Also, many of these features are standard in relation extraction studies (see, e.g., Buyko et al. (2008)).

2.4 Semantic post-processing

The semantic graph produced by the trigger and edge detection steps is not final. In particular, it may contain event nodes with an improper combination of arguments, or no arguments whatsoever. Additionally, as discussed in Section 2.2, if there are events of the same class with the same trigger, they are represented by a single node. Therefore, we introduce a rule-based post-processing step to refine

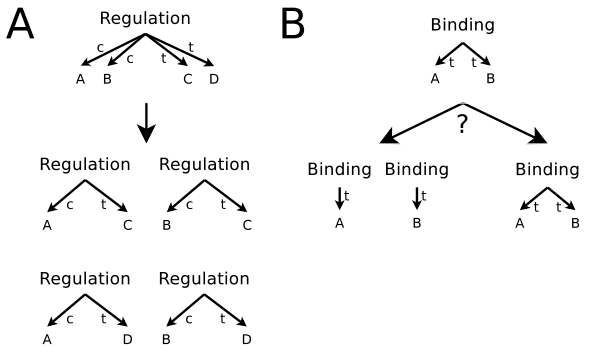

Figure 4: Example of event duplication. A) All *theme–cause* combinations are generated for *regulation* events. B) A heuristic is applied to decide how *theme* arguments of *binding* events should be grouped.

the graph, using the restrictions on event argument types and combinations defined in the shared task.

In Task 1, the allowed argument edges in the graph are 1) *theme* from an event to a named entity, 2) *theme* or *cause* from a *regulation* event (or its subclasses) to an event or a named entity. Edges corresponding to invalid arguments are removed. Also, directed cycles are broken by removing the edge with the weakest classification confidence score.

After pruning invalid edges, event nodes are duplicated so that all events have a valid combination of arguments. For example, the *regulation* event *involves* in Figure 2C has two *cause* arguments and therefore represents two distinct events. We thus duplicate the event node, obtaining one *regulation* event for each of the *cause* arguments (Figure 2D).

Events of type *gene expression*, *transcription*, *translation*, *protein catabolism*, *localization*, and *phosphorylation* must have exactly one *theme* argument, which makes the duplication process trivial: duplicate events are created, one for each of the arguments. *Regulation* events must have one *theme* and can additionally have one *cause* argument. For these classes we use a heuristic, generating a new event for each *theme–cause* combination of outgoing edges (Figure 4A). *Binding* is the only event class that can have multiple *theme* arguments. There is thus no simple way of determining how multiple outgoing *theme* edges should be grouped (Figure 4B). We apply a heuristic that first groups the arguments by their syntactic role, defined here as xthe first dependency in the shortest path from the event

to the argument. It then generates an event for each pair of arguments that are in different groups. In the case of only one group, all single-argument events are generated.

Finally, all events with no arguments as well as *regulation* events without a *theme* argument are iteratively removed until no such event remains. The resulting graph is the output of our event extraction system and can be losslessly converted into the shared task format (Figure 2D&E).

2.5 Alternative directions

We now briefly describe some of the alternative directions explored during the system development, which however did not result in increased performance, and were thus not included in the final system. Whether the reason was due to the considered approaches being inadequate for the extraction task, or simply a result of the tight time constraints enforced by the shared task is a question only further research can shed light on.

For the purpose of dividing the extraction problem into manageable subproblems, we make strong independence assumptions. This is particularly the case in the edge detection phase where each edge is considered in isolation from other edges, some of which may actually be associated with the same event. Similar assumptions are made in the trigger detection phase, where the classifications of individual tokens are independent.

A common way to relax independence assumptions is to use N-best re-ranking where N most-likely candidates are re-ranked using global features that model data dependencies that could not be modelled in the candidate generation step. The best candidate with respect to this re-ranked order is then the final prediction of the system. N-best re-ranking has been successfully applied for example in statistical parsing (Charniak and Johnson, 2005). We generated the ten most likely candidate graphs, as determined by the confidence scores of the individual edges given by the multi-class SVM. A perfect re-ranking of these ten candidates would lead to 11.5 percentage point improvement in the overall system F-score on the development set. While we were unable to produce a re-ranker sufficiently accurate to improve the system performance in the time given, the large potential gain warrants further research.

In trigger word detection, we experimented with a structural SVM incorporating Hidden Markov Model type of sequential dependencies (Altun et al., 2003; Tsochantaridis et al., 2004), which allow conditioning classification decisions on decisions made for previous tokens as well as with a conditional random field (CRF) sequence classifier (Lafferty et al., 2001). Neither of these experiments led to a performance gain over the multiclass SVM classifier.

As discussed previously, 4.8% of all annotated events cross sentence boundaries. This problem could be approached using coreference resolution techniques, however, the necessary explicit coreference annotation to train a coreference resolution system is not present in the data. Instead, we attempted to build a machine-learning based system to detect cross-sentence event arguments directly, rather than via their referring expression, but were unable to improve the system performance.

3 Tools and resources

3.1 Multi-class SVM

We use a support vector machine (SVM) multi-class classifier which has been shown to have state-of-the-art classification performance (see e.g. (Crammer and Singer, 2002; Tsochantaridis et al., 2004)). Namely, we use the SVM^{multiclass} implementation[3] which is one of the fastest multi-class SVM implementations currently available. Analogously to the binary SVMs, multi-class SVMs have a regularization parameter that determines the trade-off between the training error and the complexity of the learned concept. We select the value of the parameter on the development set. Multi-class SVMs scale linearly with respect to both the amount of training data and the average number of nonzero features per training example, making them an especially suitable learning method for our purposes. They also provide a real-valued prediction for each example to be classified which is used as a confidence score in trigger detection precision–recall trade-off adjustment and event argument edge cycle breaking in semantic post-processing. We use the linear kernel, the only practical choice to train the classifier with the large training sets available. For example, the

[3] http://svmlight.joachims.org/svm_multiclass.html

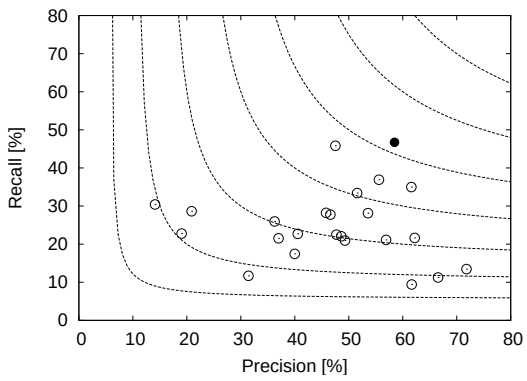

Figure 5: Performance of the 24 systems that participated in Task 1, together with an F-score contour plot for reference. Our system is marked with a full circle.

final training data of the edge detector (8932 sentences) consists of 31792 training examples with 295034 unique features. Training with even this amount of data is computationally feasible, typically taking less than an hour.

All classifiers used in the system are trained as follows. First we optimize the regularization parameter C by training on the shared task training set and testing on the shared task development set. We then re-train the final classifier on the union of the training and development sets, using the best value of C in the previous step. The same protocol is followed for the λ and β parameters in trigger detection.

3.2 Dependency parses

Both trigger detection and edge prediction rely on a wide array of features derived from full dependency parses of the sentence. We use the McClosky-Charniak domain-adapted parser (McClosky and Charniak, 2008) which is among the best performing parsers trained on the GENIA Treebank corpus. The native constituency output of the parser is transformed to the "collapsed" form of the Stanford dependency scheme (de Marneffe and Manning, 2008) using the Stanford parser tools.[4] The parses were provided by the shared task organizers.

4 Results and discussion

The final evaluation of the system was performed by the shared task organizers using a test set whose annotation was at no point available to the task participants. By the main criterion of Task 1, approximate span matching with approximate recursive matching, our system achieved an F-score of 51.95%. Figure 5 shows the performance of all systems participating in Task 1. The per-class results in Table 1 show that *regulation* events (including *positive* and *negative regulation*) as well as *binding* events are the hardest to extract. These classes have F-scores in the 31–44% range, while the other classes fall into the 50–78% range. This is not particularly surprising since *binding* and *regulation* are the only classes in which events can have multiple arguments, which means that for an event to be detected correctly, the edge detector often must make several correct predictions. Additionally, these classes have the lowest trigger recognition performance on the development set. It is interesting to note that the per-class performance in Table 1 shows no clear correlation between the number of events of a class and its F-score.

Table 2 shows the performance of the system using various other evaluation criteria defined in the shared task. The most interesting of these is the *strict* matching criterion, which, in order to consider an event correctly extracted, requires exact trigger span as well as all its nested events to be recursively correct. The performance of the system with respect to the strict criterion is 47.41% F-score, only 4.5 percentage points lower than the relaxed primary measure. As seen in Table 2, this difference is almost exclusively due to triggers with incorrect span.

To evaluate the performance impact of each system component individually, we report in Table 3 overall system performance on the development set, obtained by progressively replacing the processing steps with gold-standard data. The results show that the errors of the system are almost evenly distributed between the trigger and edge detectors. For instance, a perfect trigger detector would decrease the overall system error of 46.5% by 18.58 percentage points, a relative decrease of 40%. A perfect edge detector would, in combination with a perfect trigger detector, lead to system performance of 94.69%. The improvement that could be gained by further development of the semantic post-processing step is thus limited, indicating that the strict argument combination restrictions of Task 1 are sufficient to resolve the majority of post-processing cases.

[4] `http://nlp.stanford.edu/software/`

Event Class	#	R	P	F
Protein catabolism	14	42.86	66.67	52.17
Phosphorylation	135	80.74	74.66	77.58
Transcription	137	39.42	69.23	50.23
Localization	174	49.43	81.90	61.65
Regulation	291	25.43	38.14	30.52
Binding	347	40.06	49.82	44.41
Negative regulation	379	35.36	43.46	38.99
Gene expression	722	69.81	78.50	73.90
Positive regulation	983	38.76	48.72	43.17
Total	3182	46.73	58.48	51.95

Table 1: Per-class performance in terms of Recall, Precision, and F-score on the test set (3182 events) using approximate span and recursive matching, the primary evaluation criterion of Task 1.

Matching	R	P	F
Strict	42.65	53.38	47.41
Approx. Span	46.51	58.24	51.72
Approx. Span&Recursive	46.73	58.48	51.95

Table 2: Performance of our system on the test set (3182 events) with respect to other evaluation measures in the shared task.

Trig	Edge	PP	R	P	F	ΔF
pred	pred	pred	51.54	55.62	53.50	
GS	pred	pred	71.66	72.51	72.08	18.58
GS	GS	pred	97.21	92.30	94.69	22.61
GS	GS	GS	100.0	100.0	100.0	5.31

Table 3: Effect of the trigger detector (*Trig*), edge detector (*Edge*), and post-processing (*PP*) on performance on the development set (1789 events). The ΔF column indicates the effect of replacing the predictions (*pred*) of a component with the corresponding gold standard data (*GS*), i.e. the maximal possible performance gain obtainable from further development of that component.

5 Conclusions

We have described a system for extracting complex, typed events from biomedical literature, only assuming named entities as given knowledge. The high rank achieved in the BioNLP'09 Shared Task competitive evaluation validates the approach taken in building the system. While the performance is currently the highest achieved on this data, the F-score of 51.95% indicates that there remains considerable room for further development and improvement.

We use a unified graph representation of the data in which the individual processing steps can be formulated as simple graph transformations: adding or removing nodes and edges. It is our experience that such a representation makes handling the data fast, easy and consistent. The choice of graph representation is further motivated by the close correlation of these graphs with dependency parses. As we are going to explore the interpretation and applications of these graphs in the future, the graph representation will likely provide a flexible base to build on.

Dividing the task of event extraction into multiple subtasks that can be approached by well-studied methods proved to be an effective approach in developing our system. We relied on state-of-the-art machine learning techniques that scale up to the task and allow the use of a considerable number of features. We also carefully optimized the various parameters, a vital step when using machine learning methods, to fine-tune the performance of the system.

In Section 2.5, we discussed alternative directions pursued during the development of the current system, indicating possible future research directions. To support this future work as well as complement the description of the system in this paper we intend to publish our system under an open-source license.

This shared task represents the first competitive evaluation of complex event extraction in the biomedical domain. The prior research has largely focused on binary interaction extraction, achieving after a substantial research effort F-scores of slightly over 60% (see, e.g., Miwa et al. (2008)) on AIMed, the *de facto* standard corpus for this task. Even if a direct comparison of these results is difficult, they suggest that 52% F-score in complex event extraction is a non-trivial achievement, especially considering the more detailed semantics of the extracted events. Further, complex event extraction is still a new problem — relevant corpora having been available for only a few years.

Acknowledgments

This research was funded by the Academy of Finland. Computational resources were provided by CSC — IT Center for Science Ltd. We thank the shared task organizers for their efforts in data preparation and system evaluation.

References

Antti Airola, Sampo Pyysalo, Jari Björne, Tapio Pahikkala, Filip Ginter, and Tapio Salakoski. 2008. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC Bioinformatics*, 9(Suppl 11):S2.

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. 2003. Hidden Markov support vector machines. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML'03)*, pages 3–10. AAAI Press.

Jari Björne, Sampo Pyysalo, Filip Ginter, and Tapio Salakoski. 2008. How complex are complex protein-protein interactions? In *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM'08)*, pages 125–128. TUCS.

Jari Björne, Filip Ginter, Juho Heimonen, Sampo Pyysalo, and Tapio Salakoski. 2009. Learning to extract biological event and relation graphs. In *Proceedings of the 17th Nordic Conference on Computational Linguistics (NODALIDA'09)*.

Ekaterina Buyko, Elena Beisswanger, and Udo Hahn. 2008. Testing different ACE-style feature sets for the extraction of gene regulation relations from MEDLINE abstracts. In *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM'08)*, pages 21–28. TUCS.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180. ACL.

Koby Crammer and Yoram Singer. 2002. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Marie-Catherine de Marneffe and Christopher Manning. 2008. Stanford typed hierarchies representation. In *Proceedings of the COLING'08 Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8.

Juho Heimonen, Sampo Pyysalo, Filip Ginter, and Tapio Salakoski. 2008. Complex-to-pairwise mapping of biological relationships using a semantic network representation. In *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM'08)*, pages 45–52. TUCS.

Sophia Katrenko and Pieter Adriaans. 2008. A local alignment kernel in the context of NLP. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling'08)*.

Jin-Dong Kim, Tomoko Ohta, and Tsujii Jun'ichi. 2008a. Corpus annotation for mining biomedical events from literature. *BMC Bioinformatics*, 9(1):10.

Seonho Kim, Juntae Yoon, and Jihoon Yang. 2008b. Kernel approaches for genic interaction extraction. *Bioinformatics*, 24(1):118–126.

Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. 2009. Overview of BioNLP'09 shared task on event extraction. In *Proceedings of the NAACL-HLT 2009 Workshop on Natural Language Processing in Biomedicine (BioNLP'09)*. ACL.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*, pages 282–289.

David McClosky and Eugene Charniak. 2008. Self-training for biomedical parsing. In *Proceedings of ACL-08: HLT, Short Papers*, pages 101–104. Association for Computational Linguistics.

Makoto Miwa, Rune Sætre, Yusuke Miyao, Tomoko Ohta, and Jun'ichi Tsujii. 2008. Combining multiple layers of syntactic information for protein-protein interaction extraction. In *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM'08)*, pages 101–108. TUCS.

Martin F. Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

Sampo Pyysalo, Filip Ginter, Juho Heimonen, Jari Björne, Jorma Boberg, Jouni Järvinen, and Tapio Salakoski. 2007. BioInfer: A corpus for information extraction in the biomedical domain. *BMC Bioinformatics*, 8(1):50.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML'04)*, pages 104–111. ACM.

Sofie Van Landeghem, Yvan Saeys, Bernard De Baets, and Yves Van de Peer. 2008. Extracting protein-protein interactions from text using rich feature vectors and feature selection. In *Proceedings of the Third International Symposium on Semantic Mining in Biomedicine (SMBM'08)*, pages 77–84. TUCS.