Constraint Detection in Natural Language Problem Descriptions

Zeynep Kiziltan and **Marco Lippi** and **Paolo Torroni**
Department of Computer Science and Engineering – DISI
University of Bologna, Italy
{zeynep.kiziltan, marco.lippi3, p.torroni}@unibo.it

Abstract

Modeling in constraint programming is a hard task that requires considerable expertise. Automated model reformulation aims at assisting a naive user in modeling constraint problems. In this context, formal specification languages have been devised to express constraint problems in a manner similar to natural yet rigorous specifications that use a mixture of natural language and discrete mathematics. Yet, a gap remains between such languages and the natural language in which humans informally describe problems. This work aims to alleviate this issue by proposing a method for detecting constraints in natural language problem descriptions using a structured-output classifier. To evaluate the method, we develop an original annotated corpus which gathers 110 problem descriptions from several resources. Our results show significant accuracy with respect to metrics used in cognate tasks.

1 Introduction

Constraint programming (CP) provides a platform to model and solve complex constraint satisfaction and optimization problems and it is widely used in industry, for instance by Oracle [Bagley, 2015]. However, it has long been recognized that formulating an effective model requires considerable expertise, and efficiently solving it necessitates tailored methods, which constitutes a bottleneck to the broader uptake of CP technology. *Automated modeling and solving* aims at addressing this issue and presents a major challenge for the AI community [O'Sullivan, 2010].

An important aspect of automated modeling and solving is *automated model reformulation*. It refers to generating constraint programs to solve a constraint problem starting from its abstract specification. An *abstract problem specification* is not the model that a constraint solver uses to solve a problem. It contains high-level, mathematical-like notations and describes the problem above the level at which modeling decisions are made. It is *natural*, that is, similar to the style in which humans describe problems informally using natural language (NL). This makes CP technology more accessible to any problem-domain expert with a background in discrete mathematics who is not necessarily an expert in CP technology. Examples of specification languages are ESSENCE [Frisch *et al.*, 2008] and Zinc [Marriott *et al.*, 2008], which are descendants of OPL [Van Hentenryck, 1999].

Yet, a gap remains between the NL informal description of a constraint problem and its formal abstract specification. Indeed, producing the latter from the former is challenging. Different domain experts may produce different specifications. Moreover, a domain expert may not necessarily be a discrete mathematics expert and vice versa. Currently, there is no mechanism that validates that a formal specification complies correctly and completely with its informal description. If we could bridge this gap with a system that transforms the NL description of a constraint problem to its abstract specification in a formal language, we would construct machines that can solve constraint problems described in NL in a fully automated way. Clearly, this is an ambitious vision and certainly not the one we aim to fulfill within this work. We believe, however, that initial, significant steps could be made towards it by developing sub-systems able to isolate significant parts of problem descriptions and map them onto the corresponding entities used in formal specifications.

The focus of this work is thus on the development of a system that automatically detects the parts of text that describe constraints. The system takes as input the NL description of a constraint problem, and outputs a highlighting of portions of such text, identified as belonging to a constraint description. For example, given this text:

> Given a set of items, each with a weight and a value, determine the number of each item to include in a sack so that the total weight is at most a given capacity and the total value is as large as possible.

the system's goal is to automatically detect the underlined problem constraint description. Such a system is the first step towards fully automatizing constraint solving, and it can aid a user in validating his formal constraint specification: by looking at all highlighted text, he can be sure not to miss any constraint description in the text (*completeness*) and that he is looking at constraint descriptions (*correctness*). Such a system also enables a better decoupling between the domain expert (who should focus on providing clear NL problem descriptions) and the discrete mathematics expert (who should focus on producing precise specifications).

The underlying method combines machine learning (ML)

```
given        cst, cap:  int
letting      item be new type of size cst
letting      nat be domain int(1..)
given        wgt, val:  function (total) item → nat
find         x:  set of item
maximising    Σ i :  x .  val(i)
such that    (Σ i :  x .  wgt(i)) ≤ cap
```

Figure 1: The knapsack problem, given in ESSENCE.

and natural language processing (NLP) techniques to classify each word in a sentence as belonging or not-belonging to a constraint. In particular, it uses a structured-output classifier, so as to naturally handle the sequentiality in the data, and thus to exploit collective classification. We show that such a method can detect constraints with significant accuracy, according to the metrics used in cognate ML applications.

We focus on the detection of constraints because their descriptions provide linguistic cues and features that are more easily identifiable by NLP systems. Identification of other useful elements for formal problem specifications, such as abstract variables, would be possible only after the text has been understood. Indeed, we expect that to be a harder task, as variables are often not explicitly spelled out in the text, but we also believe that such a task can greatly benefit from the insights gained thanks to the work presented here.

Being the first ones to tackle constraint detection, we had to construct a dataset, that is, a corpus of NL problem descriptions where the parts of text containing problem constraints are annotated. Therefore, another significant contribution of this work is the corpus itself, together with the definition of a protocol to produce coherent and sound annotations.

2 Background

Here we first show an example of constraint problem specification language and the corresponding automated model reformulation system, then we provide some background in information extraction from text.

2.1 Constraint Problem Specification and Automated Model Reformulation

Consider again the knapsack problem. It can be specified in the ESSENCE language as shown in Figure 1. ESSENCE provides abstract decision variables with types such as set, multiset, relation and function, as well as nested types, such as set of sets. Constraint solvers, by contrast, typically support variables with atomic types, such as integer or Boolean, and provide limited support for more complex types like sets or multisets. This feature of ESSENCE enables the problem structure to be captured concisely. The users can specify problems in a manner similar to natural yet rigorous specifications that use a mixture of NL and discrete mathematics, without committing to any modeling decisions.

An ESSENCE specification identifies the input parameters of the problem class (`given`) whose values define a problem instance, the abstract variables (`find`), and the constraints to be satisfied (`such that`). In addition, an objective function may be specified (`minimizing/maximizing`), and identifiers declared (`letting`). Starting from an ESSENCE specification, it is possible to automatically obtain several constraint programs, each tailored for the intended constraint solver [Akgun *et al.*, 2011; Nightingale *et al.*, 2014] .

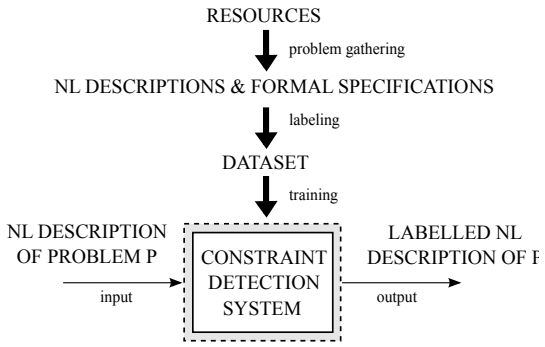

Figure 2: Methodology.

2.2 Information Extraction from Text

Extracting information from textual documents is an appealing AI application [Grishman, 2003]. By employing methods from ML and NLP, a variety of tasks can be addressed, such as text classification, document retrieval, sentiment analysis, document clustering, and lexical and semantic analysis. Based on the application domain and the task to be addressed, a large number of algorithms and techniques have been proposed [Sebastiani, 2002; Miner, 2012]. The task addressed in this work can easily be formalized as a classification task. Classification methods typically rely on an ML system trained on a set of supervised examples (*corpus*, or *dataset*). Examples provide both some representation of the textual information (e.g., *features*) and an associated *class* (or *label*). A predictive model is built, using a training procedure, which can then be used to perform predictions on previously unseen text. Depending on the task, text classification can be performed at different granularity levels, i.e., it is possible to classify the entire document, sentences, or just single words. The design of appropriate features to represent textual information has been the subject of years of ML and NLP research [Sebastiani, 2002; Miner, 2012]. Features capturing the grammatical structure of text (e.g., part-of-speech tags, constituency parse trees), as well as features trying to exploit linguistic and semantic knowledge (e.g., thesauri, ontologies) are frequently used.

3 Constraint Detection System

Here we explain the design of our constraint detection system: we present the methodology we used, the document selection and annotation process that lead to the construction of the dataset, and the ML methods present in the system.

3.1 Methodology

Our methodology is depicted in Figure 2. Thick arrows, top to bottom, indicate the main steps of the process that leads to the *constraint detection system* which is in fact the predictive model, marked with a dotted frame. Thin arrows, left to right, represent the input and output of the system.

The first step is to gather problems from a variety of resources. The result is a set of problems each defined in terms of its NL description (*text*) and its formal specification. The second step is to produce a set of annotations that highlight the constraint descriptions in each NL description. This process, called *labeling*, is done by hand on the text and relies on the information contained in the formal specifications. All the NL descriptions and corresponding annotations together form the *supervised dataset*. These are both crucial and nontrivial steps, as they have a direct impact on the performance of the system. The last step is *training*. The result is the system consisting of a *predictive model*, produced by using ML methods trained on the supervised dataset. The specifics of training are given in the Experiments section, since the choices made there are in a sense orthogonal to the methodology explained here. The system takes as input a new NL constraint problem description and produces the corresponding annotations that highlight where the constraint descriptions are.

3.2 Problem Gathering

To gather problems, we explored four different resources: CSPLib[1], the ESSENCE catalog[2], the MiniZinc benchmarks[3] and the OPL examples of IBM® ILOG® CPLEX® Optimization Studio v.12.6.0.[4] In order to avoid repetition, whenever two problems were identical or too similar, we took only one of them. For reasons that will become clear in the next subsection, it was essential to include in the corpus only problems for which a formal specification and a complete NL description were both available. Whenever either of them was missing for a particular problem, we searched for a suitable description or specification in the scientific literature. We did the same for NL descriptions that were either too brief, or citing an external source, or about a particular model rather than about a problem. We dropped the problems for which our literature search did not yield a satisfactory result.

We excluded certain puzzles, games, planning and diagnosis problems whose description did not contain any recognizable constraints. To encompass as many problems as possible, in a few cases (less than 5% of the problems in the dataset) we slightly edited the text so as to obtain a complete description. However, we did that only if the description was almost-complete and the required editing was insubstantial.

Special attention had to be paid to parts of NL descriptions containing supplementary information or elements other than text. Below we list such situations and explain how we systematically dealt with them by hand. We shall remark that this process does not hinder our system's usability. Indeed, our input text was collected from resources prepared independently of our research. In the future we could impose an input NL format and/or automate the text modification process.

- *Tables.* We dropped tables that were unnecessary or long to summarize while keeping references to the table in the text. In all other cases, we converted the table into text.
- *Mathematical formulas.* We translated mathematical formulas into LaTeX, and treated them as regular text.
- *Descriptions of special cases, variants, or generalizations of the problem.* We dropped such descriptions, as they did not change anything from the classifier's point of view.
- *Exemplifications.* We kept only meaningful exemplifications in the text.

In the end, our collection consisted of 110 problems: 46 from CSPLib, 21 from ESSENCE, 8 from MiniZinc and 35 from OPL examples. Each problem has an acceptable NL description and a formal specification, with the descriptions being quite lengthy in many cases. Specifications are formalized in ESSENCE, MiniZinc or OPL, or via a mathematical formula, or via a mathematical program model.

3.3 Labeling Issues

Labeling turned out to be a nontrivial task. Consider for instance the labeled description of the open shop scheduling problem where the labeled constraints are underlined:

> A finite set of operations has to be processed on a given set of machines. Each operation has a specific processing time during which it may not be interrupted. Operations are grouped in jobs, so that each operation belongs to exactly one job. Furthermore, each operation requires exactly one machine for processing. The objective of the problem is to determine the start time of the operations so as to minimize the maximum completion time (makespan) given the additional constraints that: operations which belong to the same job and operations which use the same machine cannot be processed simultaneously.

While the last three constraints are rather easy to catch due to the keywords "so that," "requires" and "additional constraints," the first one might be overlooked.

Another example is in the following fragments of the descriptions of the ACC basketball scheduling and the mystery shopper problems taken from CSPLib: "The schedule is to be played over 18 dates." and "Each sales person will be visited by 4 different shoppers". While both sentences refer to the future, giving the feeling of a constraint to be satisfied, only the second one describes a problem constraint. The first one simply describes an input parameter.

The presence of a formal specification is thus essential to a sound labeling because it permits to double-check that the produced labeling is *correct* (no text wrongly tagged as part of a constraint) and *complete* (no missed constraints).

In some cases, we had to take a decision on how to label. We did so systematically and uniformly throughout the labeling process. Below we list the decisions we took in such cases, which together defined our *labeling protocol*.

- Whenever a verb referred to more than one constraint in the same sentence, we labeled the whole sentence as one constraint.

[1] http://www.csplib.org

[2] http://conjure.cs.st-andrews.ac.uk/EssenceCatalog

[3] MiniZinc is a simplified version of Zinc, see https://github.com/MiniZinc/minizinc-benchmarks

[4] Documentation available via http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/

- Whenever a given constraint appeared more than once in the description, we labeled each instance separately. We did not label, however, duplicates that were too unclear.
- Whenever a constraint was split into several consecutive sentences, we labeled all the sentences together as one single constraint.
- Sometimes a problem constraint description was spelled out in the NL but it did not appear as such as a problem constraint in the specification. This happened, for instance, when the specification included partitions, permutations, injective functions, or allowed/forbidden assignments on the variables, which were implicitly enforced in the variable and/or domain definitions of the specification. Our decision was to label those as constraints, independently of how they were enforced in the specification. The reason is that different specification languages may have different expressive powers. We plan to use such constraints also for abstract variable and domain detection in our future work.

3.4 Labeling Process

The corpus was labeled by three annotators with expertise in constraint modeling. An initial labeling protocol was given to the annotators based on a first analysis of the dataset. The annotators used the BRAT software[5] for producing and sharing the annotations. They repeatedly discussed their analysis, revised the protocol, and finally agreed on the version reported above. Consensus was mostly easy to reach, thanks to the formal specifications provided together with the NL descriptions. Disagreement situations were rare and insubstantial, mostly due to fine-grained constraint boundaries. The final dataset[6] contains 1,075 sentences, for a total of 25,317 words, among which 6,724 are labeled as belonging to a constraint.

3.5 Constraint Detection

From an ML point of view, the task of detecting constraints within an NL problem description can be formulated as a sequence labeling problem. Given a sentence as a sequence of N words $w_1, w_2, \ldots, w_N$, the goal is to predict a sequence of labels $\ell_1, \ldots, \ell_N$, each associated to the corresponding word. Each label ℓ_i takes its value from a pre-defined set of m possible classes $y = \{y_1, \ldots, y_m\}$ whose semantics depends on the task to be addressed. We use two classes to indicate whether a certain word belongs to a constraint ($y_1 = C$) or it does not ($y_2 = N$). In future work, we plan to introduce further classes to detect other components of the problem specification, such as abstract variables or objective functions. Figure 3 shows an example of sequence labeling applied to the NL description of the knapsack problem.

Sequence labeling problems are widely known in ML. With respect to classical classification problems, where the examples to be classified are assumed to be independent, predictions in sequence labeling tasks have to take into account the order of the elements to be classified, so that relations between consecutive (or close) elements can be exploited. To this end, a *collective classification* is usually performed, by jointly tagging all the elements of a given input sequence, so that the most probable configuration of target elements is returned as a *structured output prediction*.

Among the existing ML methods for sequence labeling, here we consider an approach that currently represents the state-of-the-art for many tasks: Structured Support Vector Machines, and in particular their combination with Hidden Markov Models, named SVM-HMM [Tsochantaridis *et al.*, 2005]. In [Nguyen and Guo, 2007], several sequence labelling algorithms across different domains are compared, showing that SVM-STRUCT (of which SVM-HMM is an instantiation) performs the best. This was confirmed by our initial experiments conducted with Conditional Random Fields (CRFs), which gave poor results. This is explained by the fact that SVM-HMM can be tuned rather easily to deal with imbalanced data, while the same cannot be said for CRFs [De Lannoy *et al.*, 2012].

Given an observed input sequence $x = x_1, \ldots, x_K$, where x_j is the feature vector encoding the information regarding the j-th element in the sequence, SVM-HMM produces a labeling sequence $\hat{y} = \hat{y}_1, \ldots, \hat{y}_K$ as a result of the maximization problem:

$$\hat{y} = \arg\max_{y} \beta^T \Phi(x, y) \quad (1)$$

β being the vector of parameters to be learned, and Φ a joint feature map between input and output spaces. With respect to the traditional SVMs, the structured version considers the inner structure of the examples, as function Φ can encode also the dependencies between output classes. To solve the problem in Eq. 1, a dynamic programming approach can be carried out, by employing a modified instance of the Viterbi algorithm for standard HMMs.

The representation of the input space (the NL description) within a feature vector x associated to each word clearly plays a crucial role for the performance of the algorithm, and different solutions can be designed. In the next section we detail the features we employed.

We stress that the constraint detection task has unique characteristics with respect to classical sequence labeling problems, as the problem descriptions and the phrasing of constraints are very heterogeneous. These characteristics make constraint detection a particularly challenging task.

4 Experiments

We performed experiments on our dataset following the leave-one-problem-out (LOO) procedure. This is a standard ML methodology, where each problem in turn is selected as test set while the remaining ones form the training set. In addition, we conducted a case study so as to detect the constraints of a previously unseen external problem. Our system together with all the reported predictions are available at: `http://nlp4cp.disi.unibo.it`

4.1 Features

A crucial choice is how to represent the input space (the NL description) through a set of features to be used by a statistical classifier. In our sequence labeling setting, the feature

[5] `http://brat.nlplab.org/`

[6] `http://nlp4cp.disi.unibo.it`

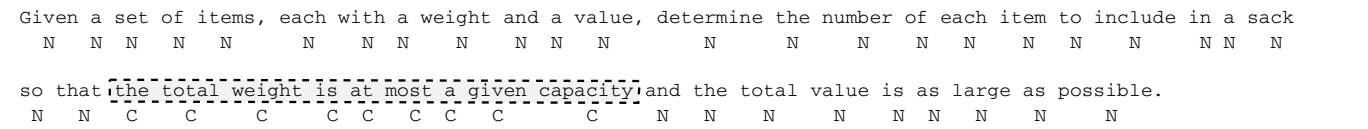

Figure 3: Sequence labeling applied to the NL description of the knapsack problem. Class C stands for constraint, N for none.

vector x_j has to encode the information of the j-th word in the sentence. We rely on a set of features describing the word itself, but also the surrounding words, considering a window of a certain diameter D. More specifically, for each word w_j we keep the original (unchanged) term, and we also extract the part-of-speech and the stemmed word, both obtained with the Stanford CoreNLP library[7]. To improve generalization over mathematical formulas, we transformed each word in the formula into a special token. Then, we build bag-of-words representations [Sebastiani, 2002] for each of these three elements, and we do the same for all the words within a diameter D centered around w_j. Finally, we also add the following bag-of-trigrams both for words and for part-of-speech tags: $[w_{j-2}w_{j-1}w_j]$, $[w_{j-1}w_jw_{j+1}]$, $[w_jw_{j+1}w_{j+2}]$. We thus employ a standard, straightforward feature set for NLP which, as we show later, achieves good performance.

4.2 Performance measurements

The predictions of sequence labeling classifiers can be assessed both by the classical ML performance metrics and by problem-specific evaluation measurements which take into account the sequential nature of the input data. Here, we deal with two classes, but the employed measurements can easily be extended to the multi-class problem with multiple possible tags (e.g., for the detection of abstract variables or input parameters in addition to the constraints).

A typical metric in sequence labeling is the Average Loss per Sequence (ALS), that is the average number of wrongly predicted labels per sequence [Nguyen and Guo, 2007]: $ALS = \sum_{i=1}^{N_{seq}} \sum_{j=1}^{n_i} I(\ell_{ij} \neq \hat{\ell}_{ij})$, where the indicator function $I(x) = 1$ if x holds true, and 0 otherwise. An additional pair of indicators, which better capture the sequential nature of data, are the true-positive hit rate H_T and the false-positive hit rate H_F. These have been used in similar structured output tasks before [Passerini *et al.*, 2012]. H_T is the ratio of constraints (in the sense of consecutive "tag blocks") correctly detected for at least one word: a measure of system *completeness* (the higher, the better). H_F is the ratio of predicted tag blocks that are totally disjoint from the ground truth (no word in common with any labeled constraint): a measure of system *correctness* (the lower, the better). Since we adopt a LOO setting, we report the arithmetic macro-average [Sebastiani, 2002] over all the problems in the dataset for all the measurements.

4.3 Results

Table 1 reports the results obtained on our dataset by different classifiers, as a function of the diameter D used to build contextual features for each word. Overall, the different performance measurements show that the proposed methodology is effective in finding constraints in NL descriptions. Around 80% of the constraints are retrieved for at least one word (H_T), with the hit rate increasing as D grows.

These are good figures sharing similar accuracy levels with most existing information extraction applications [Nguyen and Guo, 2007]. With larger contexts for features, though, the system tends to overpredict constraints, as the false-positive hit rate (H_F) also grows. Notice that the true-positive hit rate H_T remains high even if one counts as correct only those constraints detected for at least half of their words. For example, with $D = 5$, we would correctly predict 74% of the constraints (such results are not shown in Table 1). The last row shows the performance of a Random Baseline (RB) classifier, for which a higher H_T is to be expected, since it is easy to make a hit by selecting random strings of text. The tricky part is to exclude irrelevant text, thus the poor RB figures in correctness (H_F) and overall performance (ALS). Moreover, if we count as correct only the strings that include at least half of the relevant words, the RB H_T goes down to a mere 19%.

A qualitative analysis of the results also enabled us to identify classes of errors which might be reduced by a more comprehensive classification task. For instance, one of the classifier's frequent mistakes is to consider a problem's input parameter as a problem constraint, as illustrated by the dotted-underlined text in the following sentence:

> We assume that all vehicles are identical and have the same capacity D.

Our guess is that by the time input parameters are also labeled in the dataset, it will become easier for our system to distinguish them from constraints, yielding an even lower H_F.

To illustrate the quality of the output produced by our system, we report here below the description of the wood cutting problem. Solid-underlined sentences are correctly detected constraints, while dashed-underlined sentences are false negatives (constraints that go undetected) and dotted-underlined sentences are false positives.

> A wood factory machine cuts stands (processed portions of log) into chips. Each stand has a certain length, diameter, and specie. The machine can cut a limited number of stands at a time with some restriction on the sum of the diameters that it can accept. Only one species can be processed simultaneously. Finally, each stand has fixed delivery dates and a processing status being either standard or rush. Any delay on rush stand will cost a penalty. The objective is to minimize the total cost of operating and delay while meeting the system constraints: 1) the truck fleet that carry stands to machines for

[7] http://nlp.stanford.edu/software/corenlp.shtml

Table 1: Results on our dataset, as a function of diameter D for feature context. Performance measurements are macro-averaged on an LOO evaluation. For each column, bold results indicate the best system.

	H_T	H_F	ALS
$D = 1$	62.2	34.8	21.5
$D = 3$	72.4	33.4	20.2
$D = 5$	77.2	32.1	**19.5**
$D = 7$	**80.1**	**31.4**	19.9
$D = 9$	79.2	32.2	20.1
$D = 11$	79.9	32.0	20.6
RB	90.3	68.1	36.8

> processing is limited; 2) the machine is a discrete resource with capacity specified in terms of the sum of stands capacity that can be cut at the same time; 3) beside the diameter constraint, only limited number of stands can be loaded at the same time; 4) to express that only one type of species can be cut at the same time a state resource is used. At any given time, this resource indicates the state in terms of available to cut.

All the constraints except one are detected. The only false-positive sentence is not a misleading one. For some well-known problems like (temporal) knapsack, round robin tournaments, traveling tournament, job-shop scheduling, flow-shop scheduling, multi-machine assignment scheduling, assembly scheduling, social golfers, balanced incomplete block design, steel mill slab design, warehouse location, traveling salesman, Golomb rulers, and covering arrays, all the constraints are detected with zero or one false positive.

As a case study, we considered the torpedo scheduling problem. This is an industrial problem proposed for the 2016 ACP challenge `http://cp2016.a4cp.org/program/acp-challenge/`. Considering that the data source is completely different from our dataset, the results are encouraging, with only 7 false negatives out of the 22 labeled constraints and no false positives (although in this case we do not yet have a formal specification/model to be used as ground truth, because the challenge is still open).

5 Related Work

The bottleneck caused by the expertise required in modeling led to *constraint acquisition* [Bessiere *et al.*, 2015], the task of acquiring constraints from a set of examples. It relies on the expertise of a user who already solved several instances of a given problem without the help of the solver and knows how to classify an example as a (non-)solution. Similarly, techniques were devised for discovering constraints for inductive process modeling [Todorovski *et al.*, 2012] and to aid the search for appropriate constraints in the global constraints catalog [Beldiceanu and Simonis, 2011; 2012]. None of these works aims to identify constraints in NL. To the best of our knowledge, ours is the first attempt.

Machine understanding of user's needs by way of NLP has been a long standing vision and challenge in software engineering, especially in requirements engineering (RE) [Ryan, 1993]. The challenge is still open [Mich *et al.*, 2004]. Work in this domain has enabled (limited) automated support in some specific tasks such as ontology extraction [Kof, 2005; Sadoun *et al.*, 2013] using a combination of lexical, syntactical and semantic methods, and semantic annotation [Körner and Landhäußer, 2010] using part-of-speech tagging, statistical parsing and named entity recognition. None of these works uses SVM-HMM. Moreover, the tasks are different from ours. Apart from RE, work has been done also in the business process modeling (BPM) domain to support the development of executable models from NL descriptions. In particular, [Bajwa *et al.*, 2010; 2011; Njonko and Abed, 2012] addressed automatic translation of business rules written in NL into UML/OCL constraints. Although the concept of constraint in BPM is related to that in CP, we remark that these business constraints do not need to be identified in the text. They instead have to be translated and formalized to produce executable models. This task is complementary to the one we tackle, as in the mapping process from NL descriptions to formal specifications, translation is possible only after constraint detection.

Finally, if we widen our scope to include information extraction from non-technical documents, a task related to constraint detection is claim detection [Levy *et al.*, 2014; Lippi and Torroni, 2015] whose purpose is to identify portions of text containing claims. However, the datasets, genres and techniques used there are quite different from ours.

6 Conclusions and Future Work

We focused on the gap between NL descriptions and formal specifications of constraint problems, and proposed a method for constraint detection: a task that had never been attempted before. The task is nontrivial even for human experts. We constructed a system using a sophisticated sequence labeling technique and trained it with an original, carefully built corpus, achieving over 80% hit rate while keeping false-positive hit rate below 35%. Our work is intended to alleviate the well-known constraint modeling bottleneck, by enabling a better decoupling between experts, and by helping producing formal specifications that comply correctly and completely with their informal descriptions. It is also a step towards the grand vision of machines that can solve problems described in NL.

Work done on the construction of the corpus, including the definition of the labeling protocol, paves the way to the detection of other elements useful for formal problem specifications, such as abstract variables, input parameters, objective functions, for some of which the same sequence labeling techniques could deliver comparably accurate results. We expect that a combined classification task where different types of information are considered together will improve the performance of each individual task.

We are now in the process of analysing and classifying the constraints detected by our system, with the objective of developing a method for mapping the information obtained from the text onto the corresponding entities in the formal specifications.

References

[Akgun *et al.*, 2011] O. Akgun, I. Miguel, C. Jefferson, A. M. Frisch, and B. Hnich. Extensible automated constraint modelling. In *Proc. AAAI Conf. Artif. Intell.*, 4–11. AAAI Press, 2011.

[Bagley, 2015] C. Bagley. Constraint-based problems and solutions in the global enterprise, 2015. Invited talk, *Int. Conf. Principles Practice of Constraint Programming*.

[Bajwa *et al.*, 2010] I. Sarwar Bajwa, B. Bordbar, and Mark G. Lee. OCL constraints generation from natural language specification. In *Proc. Int. Enterprise Distrib. Object Comput. Conf.*, 204–213. IEEE Comp. Soc., 2010.

[Bajwa *et al.*, 2011] I. Sarwar Bajwa, M. G. Lee, and B. Bordbar. SBVR business rules generation from natural language specification. In *Proc. AAAI Spring Symp. on AI for Business Agility, Tech. Rep. SS-11-03*. AAAI, 2011.

[Beldiceanu and Simonis, 2011] N. Beldiceanu and H. Simonis. A constraint seeker: Finding and ranking global constraints from examples. In *Proc. Int. Conf. Principles and Practice of Constraint Programming, LNCS 6876*, 12–26. Springer, 2011.

[Beldiceanu and Simonis, 2012] N. Beldiceanu and H. Simonis. A model seeker: Extracting global constraint models from positive examples. In *Proc. Int. Conf. Principles and Practice of Constraint Programming, LNCS 7514*, 141–157. Springer, 2012.

[Bessiere *et al.*, 2015] C. Bessiere, F. Koriche, N. Lazaar, and B. O'Sullivan. Constraint acquisition. *Artif. Intell.*, 2015. Online preprint.

[De Lannoy *et al.*, 2012] G. De Lannoy, D. François, J. Delbeke, and M. Verleysen. Weighted conditional random fields for supervised interpatient heartbeat classification. *IEEE Trans. Biomed. Eng.*, 59(1):241–247, 2012.

[Frisch *et al.*, 2008] A. M. Frisch, W. Harvey, C. Jefferson, B. Martínez Hernández, and I. Miguel. Essence : A constraint language for specifying combinatorial problems. *Constraints*, 13(3):268–306, 2008.

[Grishman, 2003] R. Grishman. Information extraction. In *The Handbook of Computational Linguistics and Natural Language Processing*, 515–530. Wiley-Blackwell, 2003.

[Kof, 2005] L. Kof. Natural language processing: Mature enough for requirements documents analysis? In *Proc. Int. Conf. Applic. of Nat. Lang. to Inf. Sys., LNCS 3513*, 91–102. Springer, 2005.

[Körner and Landhäußer, 2010] S. J. Körner and M. Landhäußer. Semantic enriching of natural language texts with automatic thematic role annotation. In *Proc. Int. Conf. Applic. of Nat. Lang. to Inf. Sys., LNCS 6177*, 92–99. Springer, 2010.

[Levy *et al.*, 2014] R. Levy, Y. Bilu, D. Hershcovich, E. Aharoni, and N. Slonim. Context dependent claim detection. In *Proc. Int. Conf. Comput. Ling.*, 1489–1500. ACL, 2014.

[Lippi and Torroni, 2015] M. Lippi and P. Torroni. Context-independent claim detection for argument mining. In *Proc. Int. Joint Conf. Artif. Intell.*, 185–191. AAAI Press, 2015.

[Marriott *et al.*, 2008] K. Marriott, N. Nethercote, R. Rafeh, P. J. Stuckey, M. G. de la Banda, and M. Wallace. The design of the zinc modelling language. *Constraints*, 13(3):229–267, 2008.

[Mich *et al.*, 2004] L. Mich, M. Franch, and P. Novi Inverardi. Erratum: Market research for requirements analysis using linguistic tools. *Requir. Eng.*, 9(2):151, 2004.

[Miner, 2012] G. Miner. *Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications*. Academic Press, 2012.

[Nguyen and Guo, 2007] N. Nguyen and Y. Guo. Comparisons of sequence labeling algorithms and extensions. In *Proc. Int. Conf. Machine Learning, ICML 2007*, 681–688. ACM, 2007.

[Nightingale *et al.*, 2014] P. Nightingale, O. Akgun, I. P. Gent, C. Jefferson, and I. Miguel. Automatically improving constraint models in savile row through associative-commutative common subexpression elimination. In *Proc. Int. Conf. Principles and Practice of Constraint Programming, LNCS 8656*, 590–605. Springer, 2014.

[Njonko and Abed, 2012] P. B. Feuto Njonko and W. El Abed. From natural language business requirements to executable models via SBVR. In *Proc. Int. Conf. Systems and Informatics*, 2453–2457. IEEE, 2012.

[O'Sullivan, 2010] B. O'Sullivan. Automated modelling and solving in constraint programming. In *Proc. AAAI Conf. Artif. Intell.*, 1493–1497. AAAI Press, 2010.

[Passerini *et al.*, 2012] A. Passerini, M. Lippi, and P. Frasconi. Predicting metal-binding sites from protein sequence. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 9(1):203–213, 2012.

[Ryan, 1993] K. Ryan. The role of natural language in requirements engineering. In *Proc. Int. Symp. Requir. Eng.*, 240–242. IEEE, 1993.

[Sadoun *et al.*, 2013] D. Sadoun, C. Dubois, Y. Ghamri-Doudane, and B. Grau. From natural language requirements to formal specification using an ontology. In *Proc. Int. Conf. Tools with Artif. Intell.*, 755–760. IEEE Computer Society, 2013.

[Sebastiani, 2002] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[Todorovski *et al.*, 2012] L. Todorovski, W. Bridewell, and P. Langley. Discovering constraints for inductive process modeling. In *Proc. AAAI Conf. Artif. Intell.*, 256–262. AAAI Press, 2012.

[Tsochantaridis *et al.*, 2005] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

[Van Hentenryck, 1999] P. Van Hentenryck. *The OPL Optimization Programming Language.*. The MIT Press, 1999.