Learning Parsers for Technical Drawings

Dries Van Daele[1(✉)], Nicholas Decleyre[2], Herman Dubois[2], and Wannes Meert[1]

[1] Department of CS, KU Leuven, Leuven, Belgium
dries.vandaele@cs.kuleuven.be
[2] Saint-Gobain Mobility | Engineered Components (Seals), Kontich, Belgium

Abstract. From a set of technical drawings, we learn a parser program to interpret the tabular data contained in such a drawing. This enables automatic reasoning and learning on top of a database of technical drawings. For example to help designers find or complete designs more easily.

Keywords: Inductive Logic Programming · Technical drawings

1 Introduction

Technical drawings are the main method in engineering to (visually) communicate how a new machine or component functions or is constructed. They are the result of a design process starting from a set of specifications that the final product needs to comply with. This design process follows a number of strict and soft rules (e.g., material choice as a function of temperature). Figure 1 shows a typical example containing both a 2D and 3D visualisation of the object, and a material list in tabular form specifying its parts and properties. They are carefully crafted documents that act as key deliverables at the end of a design process. As such, they contain a wealth of information. Furthermore, information is laid out according to generally applied conventions.

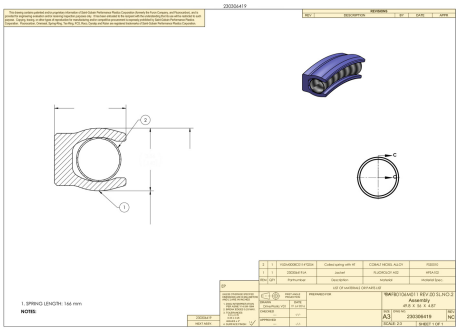


Fig. 1. A technical drawing with highlighted tabular data

Engineering companies have a large database of previous designs, potentially going back decades. They are often underutilized because previous designs can only be search for by title or by a limited set of textual annotations. Ideally, however, this database can also be used to: (1) given a technical drawing, finding other relevant drawings in a large database of previous designs; (2) given a partial description, finding designs that would complete the partial design. In this work we present an approach that can extract the knowledge in a technical

P. Cellier and K. Driessens (Eds.): ECML PKDD 2019 Workshops, CCIS 1167, pp. 51–56, 2020.
https://doi.org/10.1007/978-3-030-43823-4_5

drawing and thus improve the search capabilities significantly to achieve the aforementioned tasks.

To be able to use the data encapsulated in technical drawings, we need to parse the information contained in them, and translate this to a representation that can be handled by automated systems. Furthermore, such a system should be able to deal with both recent digital drawings and historical analog drawings. The latter is important because a great amount of information is captured in legacy drawings. Ideally, extracting the information can be done using a parser, thus a small computer program. The main challenge is that writing and maintaining such a parser is a time-consuming and expensive task. Furthermore, it is error prone since this requires an expert to explain subtle rules to an analyst or a programmer. The approach we present here will learn such parsers directly from expert feedback on the original drawing and allow its output to be used in automated tasks such as searching relevant designs.

We apply Inductive Logic Programming (ILP) to extract structured information from technical drawings, and propose a bootstrapping approach that boosts performance during multitask learning. The feedback used for learning takes the form of annotated technical drawings. Providing such annotations is a trivial task for domain experts. The required number of drawings that need to be annotated is mainly dependent on the number of variations or templates that need to be recognized. Fortunately, since all technical drawings within an organisation are expected to be (loosely) based on a limited set of templates, the number of drawings that need to be annotated is also limited.

In this work we introduce two contributions. First, we introduce the use of ILP to learn parsers from data and expert knowledge to interpret a technical drawing and produce a formal representation. Second, we introduce a novel bootstrapping learning strategy for ILP. The efficacy of this method is demonstrated in experiments on a real-world data set.

2 Identifying Technical Drawing Elements

Archived technical drawings are digitized to varying degrees. Because of this, we consider as a baseline the case where the technical drawing is represented as a bitmap image. A first step involves partitioning the image into its main segments using DBSCAN [2]. The resulting segments are identified using a CNN classifier. Segments identified as tables are further processed using a contour detection algorithm [4]. This enables the extraction of all individual cells. These cells are further processed by a *parser* that is learned from examples (see Sect. 3).

3 Inductive Logic Programs for Parsing

The data contained in a technical drawing is laid out in a manner that facilitates human interpretation. Tabular data in particular tends to have its data organised both spatially and through explicit annotation. Common examples of spatial structuring involve assigning related cells to common rows or columns, while

assigning unrelated cells to different subtables or distant cells. Particularly useful are cells that contain unambiguous keywords such as attribute names. These are helpful to gain insight in the structure of a table. They serve as anchors to cells that are less distinctive on their own but can be described relatively to anchored cells.

The application at hand does not only require us to parse a table, but also demands that we learn how to interpret its spatial organisation. A small computer program is required to parse these custom drawings. Programming a parser for each type of drawing is not only an expensive and time consuming task to build and maintain, but also prone to errors. Various errors are potentially introduced while programming a parser. First, the structure of such technical drawings needs to be explained to a non-expert, i.e. a programmer, who interprets the instructions. Second, the tables are typically not simple rectangular tables. They thus require a non-trivial parser that is difficult to understand. Third, a design can change over time requiring periodic maintenance and lead to software erosion. Ideally, these parsers would thus be programmed automatically based on the expert's knowledge. This is possible by means of machine learning techniques that learn programs from examples. The examples in this setting are obtained by annotating technical drawings, a task that is trivial for a domain expert.

The highly relational nature of tabular data and the ease with which tables can sensibly be navigated by visiting adjacent cells suggests the use of Inductive Logic Programming. ILP systems are particularly suitable for learning small programs from complex input data. Two advantages of learning programs using ILP we benefit from in this work are the ability to learn recursive definitions (e.g., row n is defined by row $n+1$) and to reuse learned target labels (e.g., first learning what a header row is helps to define what a content row is).

3.1 Standard ILP

An inductive logic programming system learns from relational data a set of definite clauses. Given background knowledge B, positive examples E^+ and negative examples E^-, it attempts to construct a program H consisting of definite clauses such that $B \wedge H$ entail all, or as many as possible, examples in E^+, and none, or as few as possible, of those in E^-.

We thus need to supply three types of inputs. First, a set of training data, examples E, that contains the properties to describe a cell in a technical drawing. An example can be:

- *Cell text:* The textual contents of each cell. Tesseract 4.0 is used to recognize cell contents [3].
- *Cell location:* The cell's bounding box information (i.e. (x, y) coordinates and cell width and height).

Second, a label for each cell (e.g., author, bill of materials, quantity). A cell can be annotated with multiple labels (e.g., a cell can be a quantity in the bill

of materials). Depending on which target label we want to learn, we split the set of examples E in a tuple (E^+, E^-) where E^+ contains the examples associated with a cell that has the target label and E^- those examples that do not. For standard ILP, the learning task is defined for one target label, so we repeat the standard ILP task for each label in the set of labels.

Third, we can provide background knowledge B that contains generally applicable knowledge for the problem at hand and remains unchanged across examples. In this case we provide:

- *Relative cell positions.* Relations capturing which cells are adjacent to each other, and in which direction (horizontally or vertically) based on their bounding boxes.
- *Numerical order.* The successor relationship. Although not essential, it is useful for learning concise, recursive rules.

The output of ILP, the program H, is a set of definite clauses of the form '`author(A) :- cell_contains(A, drawn)`' which can be read as the rule 'Cell A contains the author if it contains the word 'drawn".

3.2 ILP with Bootstrapping

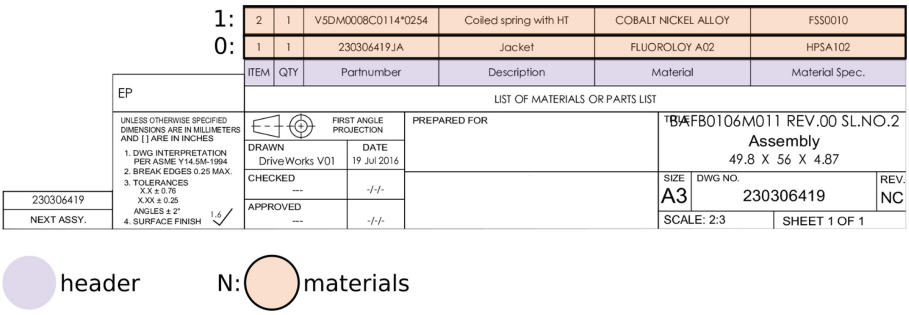
It is expected that learning programs to properly parse the target labels in P will prove simple for some targets and more challenging for others. We propose a bootstrapping extension that supports the construction of sophisticated programs by allowing them to employ the simpler ones in their definition. This is loosely inspired by the ideas raised in [1], but applied to the ILP setting.

This corresponds to a variation of the previously discussed ILP set-up where a dependency graph G is used. The nodes in this directed acyclic graph each represent a possible target label and the edges represent dependencies between those labels. A dependency indicates that one target label might have a natural description in function of another. Although we allow for this dependency graph to be specified manually, our method defaults to a fully automated approach where standard ILP is first applied to learn programs for each target. Then targets are ranked according to the ascending F_1 score of their programs on the training data. Each target in the list then has all subsequent targets as its dependencies. Finally, ILP with bootstrapping learns targets in the order specified by a correct evaluation order of G, and extends the background knowledge B for each target with the programs constructed to parse its dependent target labels. When learning program H using bootstrapping to capture a particular target label l, we define its extended background knowledge $B' = B \wedge (\bigwedge_{i \in descendants(G,l)} H_i)$, where H_i is the program trained for target label i.

4 Experiments

4.1 Learning Set-Up

The ILP system Aleph is used to learn possibly recursive programs that parse the chosen targets from the tabular data, ranging from the document's author



(a) A table excerpt from a technical drawing. Its header and materials are highlighted.

```
% Materials hypothesis
materials(A,B) :-
    zero(A),
    above_below(B,C),
    header(C).

materials(A,B) :-
    succ(C,A),
    above_below(B,D),
    materials(C,D).

% Header hypothesis
header(A) :-
    above_below(A,B),
    cell_contains(B, 'LIST').
```

(b) header/1 covers any cell located directly above a cell containing the word 'LIST'. materials/2 parses the indexed parts of the materials table. Its first argument is the index and its second argument represents the cell. materials/2 consists of two clauses. The first clause anchors the table by considering row 0 to consist of the cells above the header. It employs header/1 in its definition. The second, recursive clause indicates that the index is incremented for each row located above another.

Fig. 2. Figure a provides an illustration of the materials table and its header. Listing b shows the associated program learned using bootstrapping.

and its approval date to the attributes covered in the materials table and its indexed components.

Training data consists of a set of fully labeled technical drawings. A custom data labeling tool with a web-based graphical interface was constructed to support domain experts in labeling drawings.

Using this tool, 30 technical drawings with on average 50 cells were labeled with 14 different labels. For each target label, examples that contain that label form its positive example set, while negative examples are automatically derived by taking the complement of all possible examples for that target with its positive example set.

The labeled data is split in a training set consisting of 10 drawings, and a test set containing the remaining 20. Since the choice of training data can heavily affect the capability for finding rules that properly generalize, experiments are repeated 5 times on random samples of the training data. Because the order in which training examples are presented can also affect the rules identified by the coverage-based algorithm employed by Aleph, repeat experiments are performed even when all training data is available for learning, as a sample then corresponds to a different order in which the examples are presented to the learner.

4.2 Results

Figure 3 visualizes the performance with which cell labels and their appropriate index are correctly identified. This shows that only a few annotated designs are required for the bootstrapping method to learn perfect parsers for all labels whereas standard ILP fails to learn a perfect parser. Furthermore, it highlights how ILP with bootstrapping compared to Standard ILP is less sensitive to overfitting when presented with additional training data. This robustness of ILP with bootstrapping lends itself well to incremental learning. Both subtle and drastic variations in template design can be handled by providing the learner with a representative sample as training data. Learning perfect parsers for simple labels such as author or approval date can be achieved by both standard ILP and the bootstrap method with only a few training examples. More interesting is to look at the most complicated label, the indexed components (*materials* in Fig. 2a). The best performing program constructed using standard ILP consists of 14 clauses and yields 17 false negatives. Bootstrap learning, however, succeeds at learning a completely accurate, concise program (see Fig. 2b) whenever more than three technical drawings are provided in the training set. The poor performance when using only a few drawings is due to poor generalization. More specific, in these drawings the materials tables provided for training each consisted only of a single component and there was no pressure on the inductive learner to learn the recursive rule necessary to capture the rows of larger tables.

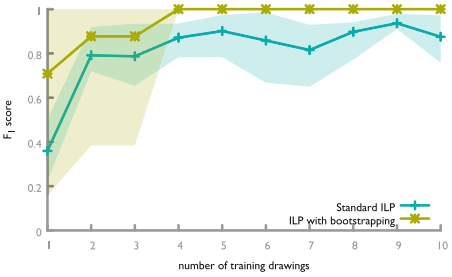


Fig. 3. The performance measured using the F_1 score of programs learning materials/2. Min/max shading is included to indicate the range of performance between the best and worst-performing program over 5 repetitions.

References

1. Dechter, E., Malmaud, J., Adams, R.P., Tenenbaum, J.B.: Bootstrap learning via modular concept discovery. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI 2013, pp. 1302–1309. AAAI Press (2013)
2. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD 1996, pp. 226–231. AAAI Press (1996)
3. Smith, R.: An overview of the tesseract OCR engine. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition, ICDAR 2007, vol. 2, pp. 629–633. IEEE Computer Society, Washington (2007)
4. Suzuki, S., Abe, K.: Topological structural analysis of digitized binary images by border following. Comput. Vis. Graph. Image Process. **30**(1), 32–46 (1985)