LGPSolver - Solving Logic Grid Puzzles Automatically

Elgun Jabrayilzade
Department of Computer Engineering
Izmir Institute of Technology
elgun1999@gmail.com

Selma Tekir
Department of Computer Engineering
Izmir Institute of Technology
selmatekir@iyte.edu.tr

Abstract

Logic grid puzzle (LGP) is a type of word problem where the task is to solve a problem in logic. Constraints for the problem are given in the form of textual clues. Once these clues are transformed into formal logic, a deductive reasoning process provides the solution.

Solving logic grid puzzles in a fully automatic manner has been a challenge since a precise understanding of clues is necessary to develop the corresponding formal logic representation. To meet this challenge, we propose a solution that uses a DistilBERT-based classifier to classify a clue into one of the predefined predicate types for logic grid puzzles. Another novelty of the proposed solution is the recognition of comparison structures in clues. By collecting comparative adjectives from existing dictionaries and utilizing a semantic framework to catch comparative quantifiers, the semantics of clues concerning comparison structures are better understood, ensuring conversion to correct logic representation. Our approach solves logic grid puzzles in a fully automated manner with 100% accuracy on the given puzzle datasets and outperforms state-of-the-art solutions by a large margin.

1 Introduction

Logic grid puzzle (LGP) is a type of word problem where the task is to solve a problem in logic. LGP can be on any domain. Constraints for an LGP are provided as textual clues. The precise understanding of these clues is crucial to correctly solve the puzzle because the representation of clues leads the logical reasoning process.

Automatically solving any word problem paves the way for more equipped digital assistants that take textual commands. Both word algebra problems and logic puzzles appear in admission tests such as the Graduate Record Exam (GRE). Thus, automation improves the understanding of these problems and can be used in the training of students.

In the field of Natural Language Processing (NLP), semantic representations are improved day by day. State-of-the-art BERT (Devlin et al., 2019) representations have boosted performance in a wide variety of NLP tasks. The rising interest is on frameworks that combine neural network-driven representations with logic representations to reason about language and predict correct outputs for tasks such as natural language inference (NLI) (Li et al., 2019). Logic puzzle solving is a task that is considered in this direction, as well.

LGPs are usually defined by a description and clues. The description part introduces categories and instances associated with each of them, and clues provide the definitions of constraints on the relationships between instances. The description can be represented by an $N \times M$ matrix where N is the number of categories, and M is the number of instances of a category. The main rule of logic grid puzzles is that one instance of a category should match only one instance of another category. The solution is provided as a tuple of instances for each category. Here is an example of a simple 3×4 logic grid puzzle taken from puzzlebaron[1]:

- $Students$: {Alex, Emma, Alice, Taylor},
- $Scholarships$:{$25k, $30k, $35k,$40k},
- $Majors$: {Astronomy, English, Philosophy, Physics}

with sample clues such as:

The student who studies Astronomy gets a smaller scholarship than Alice,
Alice is either the one who studies English or the one who studies Philosophy,

[1] https://logic.puzzlebaron.com/

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1118–1123
November 16 - 20, 2020.

The student who studies Physics has a $5000 bigger scholarship than Alex.

There has been some work that can automatically solve an LGP. Puzzler (Milicevic et al., 2012) uses an architecture that is composed of a parser and an inference module. In the parser module, clues are parsed using a Link Grammar parser, and the resultant parse trees are converted into logic representations by a semantic translator. Puzzler runs constraints in logic representation through the Alloy language to provide the solution. The system is designed around the Zebra puzzle and tested on a small dataset. Baral and Dzifcak (2012) use a trained model that consists of clues along with their desired representations in the form of λ-calculus rules to translate clues into Answer Set Programming (ASP). To distinguish between the multiple meanings of words, their system governs word to λ-ASP-Calculus rule association through probabilities. Logicia (Mitra and Baral, 2015) introduces a Maximum-Entropy based model for categorizing clues using features such as dependency trees, POS tags, etc. With the use of Answer Set Programming, it correctly solves 71 puzzles out of 100. LogicSolver (Nordstrom, 2017) generalizes and improves Puzzler's (Milicevic et al., 2012) parser method by adding more regular expression rules. In the solver module, puzzle is solved using custom first-order predicate logic parser on the predicates of type *is*, *not*, *xor*, and comparison predicates. He uses a dataset of 68 puzzles with various difficulties to evaluate LogicSolver.

LGPSolver differs from the aforementioned systems in two ways. In LGPs, semantic representation of clues is crucial because it leads the logical reasoning process to solve the puzzle. Considering this, we use a DistilBERT-based (Sanh et al., 2019) classifier, where a transformer model is combined with a Feed-Forward Layer and Softmax to perform clue classification. Language models like DistilBERT take word order and context into account, which are distinctive features of clues. Another notable characteristic of clues is that they consist of comparison structures. In general, a comparison is given among locations, times, or some numbers in the selected domain. We use a collected set of comparative adjectives and a semantic framework to identify comparative quantifiers. By categorizing these comparison quantifiers, LGPSolver can parse comparison clues.

Our experimental results show that our approach outperforms state-of-the-art solutions by a large margin by reaching 100% accuracy on the given puzzle datasets.

2 Methodology

Our approach to solving LGPs consists of four steps: parsing and classifying the given clues, defining the category instances in Prolog, converting the parsed clues into logic representations, and solving puzzles using the reasoning module. The source code with dependencies is provided as a downloadable link [2].

2.1 Parsing and Classifying the Given Clues

First of all, LGPSolver takes the category information and puzzle clues as input. A custom category recognizer is used to extract the category instances that the clue refers to. In the sentence, *"Emma has a $10000 bigger scholarship than Alex"*, the extracted instances are "Emma" and "Alex". The category recognizer returns category instances in clues with the assumption that the given input is in the correct form, meaning that in the description part, each category is given in different lines, and each line contains only those instances that correspond to that category. Thus, LGPSolver does not need to know if Alice is actually a Person or a Subject as long as Alex, Emma, Alice, and Taylor (i.e., people) are given in the same line of input.

The date and time related category instances are tagged using the TimeML (Pustejovsky et al., 2003) annotations provided by HeidelTime (Strötgen and Gertz, 2013) temporal tagger. TimeML provides time expressions as $hh{:}mm$. These expressions are normalized to minutes $(60 * hour + minute)$ to make their Prolog representations invariant.

The next step is the classification of clues. Generally, logic grid puzzles contain only a specific set of clue types. All the clues we have in our dataset can be represented by one of these clue types, as shown in Table 1.

Our observations state that clue types can be classified using some keywords and the order of words in the sentences. For example, the "Pair different" clue type usually starts with the "Of" keyword, whereas the "Comparison" type has a comparative adjective, quantifier, or "than" keyword most of the time. Successful classification of clues requires a model that takes these features into

[2] https://github.com/jelgun/LGPSolver

Clue Type	Example
Is	The Worul is made by Techtrin.
Either	Alice is either the one who studies English or the one who studies Philosophy.
All different	The four pandas were ”A”, ”B”, ”C”, and ”D”.
Pair different	Of Wade’s computer and Jack’s build, one has ”A”, other has ”B”.
Comparison	The student who studies Physics has a $5000 bigger scholarship than Alex.

Table 1: Clue types.

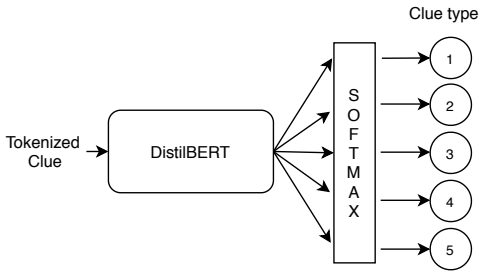

Figure 1: The classification algorithm.

account. Thus, we utilized the fine-tuned version of DistilBERT (Sanh et al., 2019) for the classification method as it can capture the word order and distinctive features of the types of clues. Ktrain (Maiya, 2020) Python package is used for this task as it contains the adjusted algorithm for classification tasks (Feed-forward neural network with Softmax activation on top). The classification workflow is shown in Figure 1.

The DistilBERT-based classifier is a fundamental component of our workflow because it can capture the context of clues. Traditional machine learning classifiers are behind DistilBERT in performance, and generally, they misclassify some Comparison type clues as Is type. For example, those classifiers encounter difficulty in distinguishing the following clues of Is and Comparison type respectively:

Jed Jarvis is the teacher.
Ed Ewing finished before the teacher.

DistilBERT is still a better choice than a rule-based classifier since every little variation in clues introduces an update to the rule base. Instead, a pretrained language model is quite robust and able to classify a clue even there are small variations in the sentence structure. Moreover, DistilBERT easily adapts to a new clue type by increasing the number of clue types in the output layer, while in the case of a rule-based classifier, an explicit regex rule must be written.

2.2 Representation of Category Instances

The designed architecture is implemented so that the instance of one category is defined as the pair of matched instances of other categories. This representation simplifies the solver part described in Section 2.4. For our example scenario, the Prolog statements that represent the instances are:

$Alex = [Alex\_scholarship, Alex\_major]$,
$Emma = [Emma\_scholarship, Emma\_major]$,
$Alice = [Alice\_scholarship, Alice\_major]$,
$Taylor = [Taylor\_scholarship, Taylor\_major]$

Here, $Alex\_scholarship$ and $Alex\_major$ define the scholarship and the major associated with Alex. Additionally, we have an $all\_members$ rule that ensures each student has a different scholarship and major. The predicates for our example are shown below:

$all\_members$([25000, 30000, 35000, 40000], [Alex_scholarship, Emma_scholarship, Alice_scholarship, Taylor_scholarship])

$all\_members$([astronomy, english, philosophy, physics], [Alex_major, Emma_major, Alice_major, Taylor_major])

These Prolog predicates are automatically generated by LGPSolver using the information given in input files.

2.3 Logic Representation of Clues

After defining the instances, clues need to be translated into Prolog statements. The translation method is shown in Table 2. I_k represents the k^{th} referenced instance in the clue. Is relationship matches the given two instances. Other clue types are represented using the combination of Is relationship with and, or, not logical operators.

Comparison clues need additional consideration. They usually contain two instances, a quantifier,

Clue	Prolog description
Is(I_1, I_2)	$I_1 =:= I_2$
Either (I_1, I_2, I_3)	$A = \textbf{Is}(I_1, I_2), B = \textbf{Is}(I_1, I_3)$ $and(or(A, B), not(and(A, B)))$
All-diff($I_1, I_2, I_3, ...$)	$A = \textbf{Is}(I_1, I_2), B = \textbf{Is}(I_1, I_3), C = \textbf{Is}(I_2, I_3), ...$ $not(A), not(B), not(C), ...$
Pair-diff(I_1, I_2, I_3, I_4)	$A = \textbf{Is}(I_1, I_3), B = \textbf{Is}(I_2, I_4), C = \textbf{Is}(I_1, I_4), D = \textbf{Is}(I_2, I_3)$ $or(and(A, B), and(C, D))$
Comparison($I_1, I_2, >, quant$)	$I_1\_val–I_2\_val =:= quant$

Table 2: Clue to Prolog translation table.

and a comparative adjective. These clues are divided into two types: $less$ and $more$ (e.g. "lower than" is a $less$ type while "bigger than" is of $more$ type). These types represent whether the first referenced instance's corresponding value is less or more than the value of the second instance. Types are defined as '$<$' (smaller sign) and '$>$' (greater sign) in Prolog. We have gathered a list of comparative adjectives in English from curso-ingles[3] and with the help of Semantic Framework for comparison structures (Bakhshandeh and Allen, 2015), they are categorized as $less$ type, $more$ type, or none of them. In total, we acquired 41 comparative adjectives that are commonly used in LGPs.

Comparison quantifiers (e.g. $5000) in clues are recognized using the regex patterns and expressions provided by HeidelTime (Strötgen and Gertz, 2013). Furthermore, due to the limitation of the HeidelTime tagger in capturing fractional time units (e.g., half an hour), we have extended the tagger's ruleset to include them.

The comparison clue's Prolog description includes I_k_val keyword to represent the matched instance of the compared category with the I_k. In our case, the compared category is the $scholarship$. Generally, the compared category in LGPs is the one that has numerical instances. In the case of multiple numerical categories, the instances of comparison clues are analyzed. LGPSolver chooses the unmentioned category as the compared one (the subject of comparison). For example, in clue *"The student who studies Astronomy gets a smaller scholarship than Alice"*, the categories of mentioned instances are $students$ and $majors$. In contrast, no instance of the $scholarships$ category is mentioned.

[3]https://www.curso-ingles.com/en/resources/cheat-sheets/adjectives/list-of-comparatives-and-superlatives

2.4 Solving the Puzzle

To get the puzzle's solution, the instances of one category should be given as a query to the Prolog. For example, the query of $(Alex, Emma, Alice, Taylor)$ will return the matched scholarships and majors for each of these students. For a given query, Prolog recursively binds the query parameters to their possible values and returns the matched values if all the predicates are true. This is accomplished by the Prolog's built-in backtracking search algorithm. Pyswip (Tekol, 2020) package is used to execute the generated Prolog scripts inside a Python code.

3 Results

For the work's evaluation, we have used the datasets provided by Logicia (Mitra and Baral, 2015) and LogicSolver (Nordstrom, 2017). Logicia dataset has 150 LGPs, whereas LogicSolver has 68 LGPs. We have used the 50 LGPs (the training set in Logicia) from the Logicia dataset for DistilBERT training, and the remaining 100 LGPs are used for the testing purposes. The 68 LGPs in the LogicSolver dataset are also used as a test set without additional training. In brief, there are 50 training and 168 test samples. The details of these datasets are given in Table 3 and Table 4.

The evaluation is based on two factors: parser and solver accuracies. Parser accuracy is defined as the percentage of the correctly parsed clues (including classification), while solver accuracy is the percentage of correctly solved puzzles. The performance of LGPSolver was compared to LogicSolver and Logicia. The experimental results are shown in Table 5.

The DistilBERT-based classifier successfully classified all the clues in the test puzzle sets, and LGPSolver has correctly solved all the LGPs in the given datasets. It should also be noted that to solve

# of puzzles	50
# of clues	245
# of category instances in clues	584
Avg. # of clues / category instance	0.42
Avg. # of clues / puzzle	4.9
# of Is clue type	57
# of Either clue type	36
# of Comparison clue type	120
# of All-diff clue type	16
# of Pair-diff clue type	16

Table 3: Details of the training set.

	LogicSolver	**Logicia**
# of puzzles	68	100
# of clues	297	457
# of category instances in clues	756	1112
Avg. # of clues / category instance	0.39	0.41
Avg. # of clues / puzzle	4.36	4.57
# of Is clue type	65	114
# of Either clue type	53	74
# of Comparison clue type	125	203
# of All-diff clue type	15	22
# of Pair-diff clue type	39	40

Table 4: Details of the test set.

LGPs successfully with the reasoning module of Prolog; the puzzle description should be parsed without any errors.

4 Conclusion and Future Work

This paper presents a system that automatically solves logic grid puzzles. Better identification of comparison structures in clues and using a DistilBERT-based clue classification solution are the two highlights of the system. LGPSolver achieves full accuracy in a fully automated manner.

The DistilBERT-based classifier is a fundamental component of our workflow because it can capture the context of clues. The traditional classifiers (e.g., Naive Bayes, SVM, Logistic Regression) have a lower accuracy, which can be attributed mostly to the misclassification of Comparison type clues as Is type clues. Furthermore, rule-based classifiers (e.g., regex patterns) were not preferred due to their generalizability issues as every little variation in clues introduces an update to the rule base. On the other hand, a pretrained language model is quite robust and able to classify a clue even there are small variations in the sentence structure.

The parser module requires to recognize and normalize time and date related information in clues

	LGPSolver	**LogicSolver**
Parser	100%(297/297)	74.4%(221/297)
Solver	100%(68/68)	83%($\approx$ 56/68)

(a) 297 clues and 68 puzzles.

	LGPSolver	**Logicia**
Parser	100%(450/450)	90.9%(410/450)
Solver	100%(100/100)	71%(71/100)

(b) 450 clues and 100 puzzles.

Table 5: Accuracy of parser and solver modules.

to process comparisons in the correct way. Temporal taggers can be used for this purpose. However, temporal taggers' numeric normalizers have limitations in capturing fractional time units (Chang and Manning, 2012)(Angeli and Uszkoreit, 2013). Thus, we have extended the rule set of HeidelTime (Strötgen and Gertz, 2013) to resolve the issue.

As LGPs contain only a specific set of clue types, the problem of clue classification is formulated on a fixed number of clue types. A more sophisticated system would be able to learn the number of clue types automatically by the processing of clues. Thus, to further reason about language, seeking an automatic mapping between an NLP-based semantic representation and a logic representation is an important future direction.

Acknowledgments

We thank Tuğkan Tuğlular for his helpful suggestions on an earlier version of this paper.

We also thank anonymous reviewers for their valuable comments.

References

Gabor Angeli and Jakob Uszkoreit. 2013. Language-independent discriminative parsing of temporal expressions. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 83–92. The Association for Computer Linguistics.

Omid Bakhshandeh and James Allen. 2015. Semantic framework for comparison structures in natural language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 993–1002, Lisbon, Portugal. Association for Computational Linguistics.

Chitta Baral and Juraj Dzifcak. 2012. Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 573–577. AAAI Press.

Angel X. Chang and Christopher D. Manning. 2012. Sutime: A library for recognizing and normalizing time expressions. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pages 3735–3740. European Language Resources Association (ELRA).

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. 2019. A logic-driven framework for consistency of neural models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3922–3933. Association for Computational Linguistics.

Arun S. Maiya. 2020. ktrain: A low-code library for augmented machine learning. *CoRR*, abs/2004.10703.

Aleksandar Milicevic, Joseph P Near, and Rishabh Singh. 2012. Puzzler: An automated logic puzzle solver. Technical report, Massachusetts Institute of Technology (MIT).

Arindam Mitra and Chitta Baral. 2015. Learning to automatically solve logic grid puzzles. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1023–1033. The Association for Computational Linguistics.

Ross Nordstrom. 2017. Logicsolver - solving logic grid puzzles with part-of-speech tagging and first-order logic. Technical report, University of Colorado, Colorado Springs.

James Pustejovsky, José M. Castaño, Robert Ingria, Roser Saurí, Robert J. Gaizauskas, Andrea Setzer, Graham Katz, and Dragomir R. Radev. 2003. Timeml: Robust specification of event and temporal expressions in text. In *New Directions in Question Answering, Papers from 2003 AAAI Spring Symposium, Stanford University, Stanford, CA, USA*, pages 28–34. AAAI Press.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Jannik Strötgen and Michael Gertz. 2013. Multilingual and cross-domain temporal tagging. *Lang. Resour. Evaluation*, 47(2):269–298.

Yüce Tekol. 2020. Pyswip v0.2.9. `https://github.com/yuce/pyswip`.