# Improving Efficiency of Natural-Language Text Generation for Automatic Pedagogical Questions

Yulia Gomazkova, Oleg Sychev(✉), and Marat Gumerov

Volgograd State Technical University, Lenin Avenue, 28, Volgograd 400005, Russia
o_sychev@vstu.ru

**Abstract.** When learning new skills, students may experience problems in understanding the course material and may require a tutor's help in the form of educational scaffolding. That can be organized in the form of learning dialogue consisting of closed-answer questions and explanations for answers to them. However, designing Intelligent Tutoring Systems that support generating pedagogical questions is very labor-intensive. In this paper, we propose a method of improving template-based generation of pedagogical questions by moving the templates from the level of nodes of Thought Process Tree describing the problem-solving reasoning process to the smaller elements like expressions, domain objects, their classes, properties, and relationships. The method allowed to reduce the number of templates used in the intelligent tutoring system for learning programming-language expressions by about half and the number of words used in templates by about 60%.

**Keywords:** Natural language generation · Pedagogical questions · Intelligent Tutoring Systems

## 1 Introduction

It is one of the most important tasks of any tutor to guide students through their learning process, particularly when they encounter obstacles or misconceptions [4], which is why effectiveness of conducted pedagogical interventions is very important [7,23].

The question-based approach to teaching allows teachers to build individualized dialogues with students, assessing the level of their knowledge and adjusting the explanations of the topic. That approach helps to develop the mastery of new educational material more than the usual explanatory monologue [16,22].The role of questions in learning has been repeatedly considered in earlier pedagogical studies, confirming their effectiveness [3,17]. It has been shown that questions asked after studying certain text materials have a significant effect on students' memory and understanding of the source text [3,6].

With rising demand for high-complexity tutoring systems, a growing body of research emerges that focuses on providing increasingly more efficient electronic

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
O. Gervasi et al. (Eds.): ICCSA 2024 Workshops, LNCS 14816, pp. 37–50, 2024.
https://doi.org/10.1007/978-3-031-65223-3_3

tutors capable of performing more tasks that were previously done only by human tutors [8,21]. Today, many such systems aim to provide students with contextualized feedback based on the problem at hand and the individuality of the student. But despite their promising results [9], widespread integration of Intelligent Tutoring Systems (ITS) has been hindered by the substantial effort required to author courses and exercises. Estimates in [2] suggest that the creation of one hour of ITS content requires 200–300 h of expert design input. Reduction of that authoring effort is the focus of a lot of ongoing research in the field.

So, whether delivered by human instructors or sophisticated software systems, the ability to provide clear and comprehensive pedagogical support is paramount in facilitating student comprehension and retention, yet constructing the modern ITS to utilize such methods remains a labor-intensive process. Generating pedagogical dialogue as a set of questions and explanations is particularly challenging in that regard [19]. In this work, we are focusing on improving automatic generation of pedagogical questions with the goal of reducing the effort required to allow the system to engage the student in a natural-language dialogue about the rules of a new subject domain.

## 2 Related Works

The use of natural-language dialogue in tutoring systems goes back at least several decades, with a solid body of research going into it. The approaches to the problem of providing a student with a consistently readable and helpful text are varied across applications, but certain broad groups definitely emerge.

The simplest approach used when creating such systems is relying on the author's expertise by making them provide the questions and/or explanations shown to students explicitly. Such approaches, while straight-forward, can still provide a lot of complexity and adaptability, depending on the method used and the amount of effort spent. A major example of this approach can be seen with applications of the Knowledge Construction Dialogues (KCD) framework in works such as Atlas [5], ProPL [10], and Rimac [1]. With KCD, the author defines both the textual formulations of the questions shown as well as the structure of responses anticipated from the students, with each response either linked to another KCD responsible for providing remedial guidance or categorized as a correct answer. That process results in a hierarchical structure where the main line of dialogue can branch off into sub-dialogues for the purposes of correcting student's mistakes made at the higher level. With that, a number of automatic tutors can be constructed, which can provide a student with educational scaffolding that can help them during misunderstandings, but the authoring effort of such approaches is high.

There was effort in improving that approach. In the already-mentioned Rimac system [1], for example, an effort was made to increase the system's adaptability to student progress by utilizing a student model based on the Instructional Factor Analysis Model, which would continuously update after each dialogue session.

In more specialized domains, their rules can be utilized to reduce the amount of work necessary to create the teaching dialogue. In Jeliot 3 [13,14], a tutor

aimed at teaching Java programming language via code execution visualization, an approach was proposed for providing students with questions during the code execution. There, a code statement could be assigned a question about the result of its execution, which could then be displayed to the student during the visualization, allowing them to check their understanding. With this approach, the question structure arises naturally from the code execution sequence, and any given question can be reused every time the relevant statement is executed, while its answer options are dynamically adapted to match the context of program execution.

However, manually crafted systems of pedagogical questions remain very labor-intensive to construct. That can be solved by automating the generation of pedagogical questions. There are two broad approaches to the automatic generation of pedagogical questions: the transformational approach and the template approach. Question generation systems using the transformational approach operate on the following premise: a text source (such as a textbook chapter) is used as an input, its sentences are automatically parsed and structurally transformed into questions, keeping the semantics of the source text, in order to assess the student's understanding of it.

An example of such a system can be seen in [18], where textual analysis and transformation are used in order to create "fill in the blank" style questions given a corpus of Wikipedia articles related to the topic. There, the textual transformation is used in its simplest form: replacing a dynamically determined keyword with a blank. However, an interesting approach of analyzing and comparing the context the words appear in is used to determine a set of distractors, which also fit into the blank and can serve as possible answers in multiple-choice questions.

A more complex approach to text transformation is explored in [12]. Mazidi and Tarau propose introducing a Natural Language Understanding aspect into the question generation system. By analyzing the sentences of the input text by the type and arrangement of their syntactic elements, the system classifies sentences into 8 categories.For each of those categories, an internal template can then be defined and used to transform the sentence from a statement to a question while correctly handling its syntactic and semantic contents. The chief problem with that approach is that the questions remain very closely related to single sentences of the input text; they do not use the broad knowledge of the subject domain.

The use of templates defined specifically for a given problem or domain is another approach used in many question-generating systems. The systems in this category generally rely on the human author to provide some natural-language text templates in order to create their questions, rather than gathering this information from an input text. This generally leads to more nuanced questions but requires a lot more effort. Despite that, a study [11] of question generation methods tailored for educational purposes revealed that approaches relying on templates tended to outperform systems that relied on syntactic transformation of the source text.

A promising approach was chosen by O'Rourke et al. [15], who used a formalization of the student's thought process in the form of a custom programming language called Thought Process Language (TPL). The TPL code would describe the actions a student needed to perform to solve a problem, and the structure of the code could be used to generate explanations in a natural language by using predefined textual templates assigned to each type of TPL statement. Those templates could then be populated by the objects from the formal definition of the learning problem that the learner tries to solve in order to create explanations of the actions shown to the learner.

A possible improvement on this approach was proposed in [19], where the similar concept of assigning textual templates to an algorithm-defining structure was used to create sequences of questions, guiding the student through the necessary reasoning steps. This system allowed for a more human-like representation of a Thought Process than TPL, using a specialized tree-like formalization instead of code; it also used a more expressive template system. However, it still required manual creation of a lot of natural-language templates of several kinds.

## 3 Method

### 3.1 Existing Approach and its Flaws

We based this study on the results of automatic generation of pedagogical questions from templates obtained by Sychev and Gumerov [19]. In that approach, single-choice or multiple-choice questions are used, so the natural-language subsystem should generate three kinds of strings (see Fig. 1):

1. question formulations,
2. answer options (both correct answers and distractors),
3. explanations (reply) shown to the student (for wrong answer options).

That method uses Thought Process Trees to represent the reasoning paths a student must follow to correctly solve any problem in the given domain or determine why some solutions are incorrect. That representation is used together with the dictionaries of domain-specific entities (classes, class and object properties and kinds of relationships), and the formal problem formulation containing actual object instances and their relationships is then utilized to construct a sequence of questions which allows to detect where the student strayed from the correct solution path and explain their error.

That approach focuses on using the nodes of Thought Process Trees to generate pedagogical questions. That is, the majority of the natural-text templates necessary to generate the pedagogical dialogue are assigned to nodes. That allows for a nuanced control of the formulations presented to students, as the node's human-formulated meaning is often simpler than the literal translation of the expression defining the formal meaning of the node, so it was important to provide separate templates for each node. But assigning templates mostly to nodes did not allow generalizing templates for similar nodes, or indeed for any nodes at all.

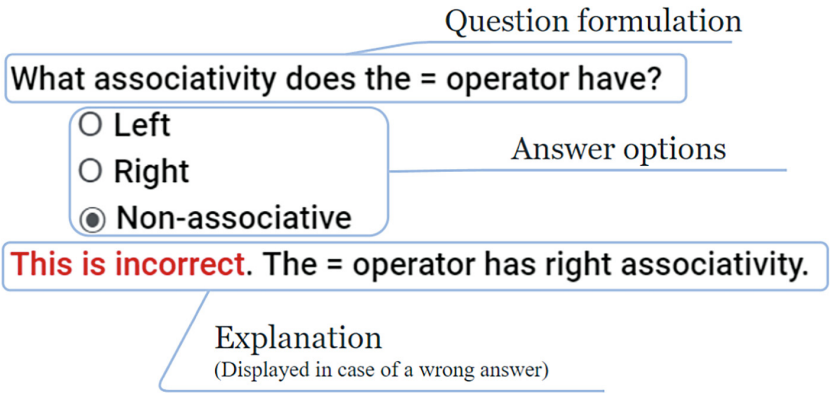


**Fig. 1.** The structure of generated pedagogical questions: the question formulation, answer options and explanation of the chosen answer.

Subject domains often contain similar branches of reasoning. For example, in the task of determining the order of evaluation of a programming-language expression [20], there are two similar branches concerned with the possible competing operators to the left and right of the current operator. The branches of thought were almost identical except that the relations between the operators were changed from "to the left" ("less than") to "to the right" ("more than") and vice versa [19]. But the nodes of the resulting branches were distinct, so there was no way to generalize their descriptions, and so the domain expert (the one responsible for authoring the Thought Process Tree) had to enter almost twice as many templates than was absolutely necessary.

The same problem is even more evident on the level of expressions determining the reasoning and their operands. Many class and object properties and relationships are used in multiple nodes, so their natural-language descriptions should be entered many times.

### 3.2 Improvements

In this study, our main focus was to reduce the amount of work needed to allow the system to generate pedagogical questions and explanations for different answers to them. As it was mentioned before, the previously described methods lacked a way to generalize the text generation for nodes with similar underlying logic, which is what we tried to address.

In the previous work, four kinds of generated questions are defined to be generated by the system:

- Variable Value questions ("What is the value of <variable>?")
- Final Node questions ("Which <reasoning step> determines the outcome in the current situation?")
- Node questions ("What is the outcome of this <reasoning step>?")

– Structural questions("Given the <answer to a node question> what should you do next?")

For the goals of this study, it is important to analyze how those question types relate to nodes and expressions they contain.

Variable Value questions are directly tied to Action Nodes: an action node introduces the relevant variable and contains the expression that determines the value of that variable. Variable Value questions can be generated by either assigning a description to the variable (which would produce more semantically-oriented questions) or deriving the description from the expression used in the node (which would produce a more direct, matter-of-fact question). Generally, the first approach is preferable because students usually refer to temporary values used in reasoning (represented as variables) by their semantic role instead of the exact way needed to determine them.

For example, let us consider the process of determining the order of evaluation of an expression. One of the important variables when considering if the given operator (X) can be evaluated right now is the competing operator to the left (A), which can prevent the given operator's evaluation if the competing operator must be evaluated first. Consider the action node which finds the competing operator—the closest unevaluated operator standing to the left of X—and assigns it to the variable A. The expression for that node is:

```
findExtreme operand A[
    not exists a' in A' {a'->isCloserToThan(X, A)}
]
among operand A' {
    A'->isLeftOf(X)
    AND A'.state == unevaluated
}
```

Our goal is to generate a question verifying that the student determined the value of A correctly in the given circumstances because it is crucial to verifying the further line of reasoning using A.

As mentioned, one approach is to define the textual description for the variable A itself—for example, "the competing left operator of ${X}" (where ${} denotes a sub-template—in this case, the textual description of the object contained in variable X). With this, a question asking to "Find the competing left operator of ${X}" can be generated by utilizing a higher-level template "Find ${variableDescription}" assigned to the variable value question. The answer options for this question are then generated from the textual descriptions of actual operators present in a given problem, including the actual correct value of A.

Another approach is to use the search expression in order to generate the description of A. As all the properties and relationships used in expressions in the Thought Process tree are defined in dictionaries by the problem-solving domain's author, the system can utilize that information to analyze expressions

and generate natural-language descriptions based on their structure and the used concepts. In the example above, the system can recognize the following pattern:

```
findExtreme <class> <extremeVariable>[
    not exists <rangeVariable> in <variableRange> {
        <rangeVariable>-><comparativeRelationship>(
            <otherVariable>,
            <extremeVariable>
        )
    }
]
among <class> <variableRange> {
    ...
}
```

meaning that <extremeVariable> should be described using an extreme formulation assigned to <comparativeRelationship>—(in this case, A) can be described as "${obj} closest to ${X}" based on the template assigned to the `isCloserToThan` relationship. The conditions in the `among` block can similarly be analyzed to get the "$obj to the left of ${X}" and "unevaluated ${obj}" templates, which, along with the variable's class name, could then be combined into the A's final description "unevaluated operator closest to ${X} to the left of it', which can then be used in the question template shown above.

The system we developed allows for both described approaches because they have their advantages and disadvantages. Directly assigning the text to a variable is conceptually simpler; it allows for more nuanced and semantically correct descriptions, which is especially necessary in cases where the meaning (role) of the variable is more important than and different from the procedure of determining its value. It, however, requires more effort from the domain expert and does not allow reusing the textual descriptions. The generation of text based on the structure of expressions has the opposite set of characteristics: while it generally requires less authoring effort, it is strictly tied to the expression's structure, unable to grasp the overall meaning of the variable. The two kinds of questions play different pedagogical roles and can be used in different circumstances. The question using the semantic role of the variables verifies if the student knows how to determine the object for that semantic role (e.g., a competing operator to the left) in the given circumstances. The question generated from the node expression only verifies if the student can perform the instructions for finding an object satisfying certain criteria well without understanding the semantic role of the found object.

Another problem to consider is the ability of the text-generation algorithm to handle arbitrarily complex expressions. Currently, our method can recognize relatively simple patterns in the expressions' structure: a unique object among a set of others, an object related in a certain way to another object, an object with a given property, an object of a given class, and their combinations. If no relevant pattern was detected, the tutoring system must ask for the expert's support

because no reliable way to translate that expression into natural-language text has been found as of yet.

Our work was based on the analysis of seven subject domains that were created for the tutoring system mentioned above: determining the order of a programming-language expression's evaluation; determining the order of adjectives in the English language; determining object visibility in the C++ programming language; constructing access expressions in the C++ programming language; determining the control flow of a given algorithm; determining the lifetime of a variable; determining the data types of expressions in programming languages. Reviewing the Thought Process Trees constructed for those domains allowed us to draw conclusions on the commonalities and patterns occurring within them. They also allowed us to test the method in different situations and make sure it could be generalized to other subject domains.

Generating text from the structure of expression is particularly important for Question Nodes, which represent questions that determine the line of reasoning in the given circumstances. Together, Question Nodes and Action Nodes constitute the majority of nodes in a typical Thought Process Tree (about 92% out of all nodes) so improving text generation for them is more important. Also, question nodes cannot be linked to a variable with certain semantic role so their only source of information is the node expression.

In order to generate Node Questions for Question Nodes, we had to classify the kinds of questions used in the nodes in Thought Process Trees. Based on the trees from the domains that we used, we identified the following kinds of questions:

- Checking a class's or object's property: further reasoning depends on the value of an object's property.
- Comparing a class's or object's property to a constant: further reasoning depends on how the value of an object's property relates to some explicit value.
- Comparing values of the same property of different objects: further reasoning depends on how the properties of the objects relate to each other.
- Checking for the existence of a relationship between two objects.
- Checking for the existence of an object defined by certain criteria: further reasoning depends on if such an object exists in the given learning situation or not.
- Checking for an object's class; the further reasoning depends on the class.

This classification allows to choose the appropriate meta-patterns for dynamically constructing question formulation for question nodes. This is done as follows.

Question nodes that check an object's property are defined by the expression of the following structure: <variable>.<property> (For example, 'X.associativity'). Those questions have a simple structure, but there are different kinds of properties and in natural languages, different syntactic templates are used for them. So, in this case, we propose to assign a question template to the property. For example, the template "What associativity does ${class}

have?" fits the `associativity` property of operators. Other examples are "How many tokens does $obj} have?" for integer property `countOfTokens` and "Does ${class} need a left operand?" for boolean property `needsLeftOperand`. If no template is defined for the class or object property, the system can use the default template "What is the ${property name} of ${class/object}?", which uses the textual description of the property. Properties can belong either to individual objects or their classes (static properties). So the templates use either the class description ("operator +") or the object description ("operator + at position 5") in the question.

The answer options for this type of questions can be created from all the possible values of the property depending on its type—or, when the value range of the property is large, some subset of the values close to the actual value of the property for the corresponding object and, possibly, some nearby objects in the same learning problem. The explanations, then, are constructed like the question formulation: when the student chooses an incorrect value for the checked value, the system can create a reply like "That is incorrect, because ${statement}.", where the `statement` placeholder is either constructed from the statement template assigned to the property itself (e.g., "${obj} has ${value} associativity"), or from the default "${propertyName} of ${class/object} is ${value}" template.

When comparing a class's or object's property to a constant, the expressions look like <object/variable>.<property> <operator> <constant> (for example, 'X.associativity == left'). For boolean and enumerated properties, which support only equality and inequality comparisons, the statement template assigned to the property can be reused. The questions then take the form "Is it true that ${statement}?", with the `statement` placeholder being filled in as mentioned above, just inserting the description of the constant in question in place of the `value` placeholder. With numeric (or other linear-scale) properties, whose values can be less or greater than others, the questions are constructed differently. In that case, the question takes the form "Is ${propertyName} of ${object} less than/greater than/equal to ${value}", with the comparison formulation chosen based on the used comparison operator.

The answer options for this type of question are "Yes" or "No"; explanations are generated similarly to the previous type of questions: it is enough to show the student the correct value of the property in question in order to show why their answer was incorrect. It should be noted that inequality operator (!=) as well as non-strict comparisons (>= and <=) do not produce their own formulations in the question (in order to reduce the question's complexity), but are rather treated as the inversion of another operator. So, rather than asking, "Is it true that the associativity of ${X} isn't ${value}?", the system asks, "Is it true that the associativity of ${X} is ${value}?", and then inverts the student's answer.

Another type of comparison recognized by the system is the comparison between the values of the same property of different objects, which have the form '<object1>.<property> <operator> <object2>.<property>' for the regular (boolean-typed) comparisons and 'compare( <object1>.<property>, <object2>. <property>)' for three-way (greater, equal, or less) comparisons

(for example, 'X.precedence == Y.precedence' and 'compare(X.precedence, Y.precedence)' respectively). Depending on the property's type, the answer options are constructed differently. If the property allows for less/greater comparisons, the system constructs the question of the form of "Compare ${propertyName} of ${object1} and ${object2}", with placeholder values being filled as it was described before, and answer options are "${propertyName} is greater for ${object1}", "${propertyName} is greater for ${object2}", and "They are equal". If the property type only allows for equality comparisons, the question generated is "Is ${propertyName} of ${object1} equal to that of ${object2}?" with answer options "Yes." or "No.", with explanations being generated similarly to the previous two types of questions.

Expressions that require assessing the object's class ('<object>.class' and '<object> is <class>') are technically separate, but operate very similarly to the property-related expressions described above (the object's class may be thought of as a property of it), so the questions and explanations are generated by a similar method. For them, the question template "What is ${obj}?" and the statement template "${obj} is ${class}" are used. The answer options are constructed from subclasses of the class that was used to define the variable holding the object.

Another kind of questions the system recognizes is the question about the existence of a certain relationship between objects. The expressions in this case have the form '<subject>->relationship>(<objects>)' (For example, 'Y->isLeftOf(X)'). This type of questions has similar text to the comparison of a property to a constant: the statement template is defined for the relationship checked (for example, "${subj} is to the left of ${obj}") and then the question is generated using the same "Is it true that ${statement}?" template, with answers and explanations being constructed in the same manner.

The last question type we identified is a check if an object matching a certain set of criteria exists. The expression structure in this case is '[not] exists <Var> {<boolean criteria>}'. The Question Nodes containing such expressions are similar to the Action Nodes for finding the object that matches those criteria, except that for Question Nodes, the object itself does not matter in the long term and so should not be remembered. This means that we can reuse the method of generating descriptions for Action Nodes to create an object description and use the general template "Is there any ${objectDescription}?". The answers here are once again "Yes" and "No"; the explanation are generated like "That's incorrect, because the ${object}, matches all the criteria".

Introducing those distinct question types allowed us to specialize Node Questions generated by the system, which led to using low-level templates which can be reused in several questions each. While we discussed this method with respect to Question Nodes only, it should be noted that this approach can be utilized in other nodes that contain expressions—for example, Loop Nodes that iterate over all the objects matching certain criteria can use the same method of generating descriptions as the Action Nodes and existence-checking Question Nodes.

## 4 Results

The proposed approach allows to replace a lot of repeated text in the descriptions of the nodes of Thought Process Trees with a smaller set of shorter templates assigned to the domain entities. That can considerably reduce the effort required to enable the system to generate all the necessary questions and explanations for a subject domain. We used the Thought Process Tree for the order of expression evaluation to evaluate the gains of the proposed method.

Out of 237 textual templates that are required for the question generation of that tree, 172 correspond to the nodes that contain expressions and can benefit from our approach. Those templates were replaced by 54 much smaller ones, assigned to individual domain entities like properties and relationships, which reduced the number of required templates by more than three times. The templates themselves also became smaller (see Table 1).

The remaining 65 templates were not reduced by this approach because they correspond to the nodes related to the control structures of Thought Process Tree—specifically Branching Nodes and their branches. There was no Loop nodes in the that tree so they are not represented in the table; those nodes also are not covered by our method. Optimizing natural-language message generation for Branching and Loop nodes requires further work.

By assigning templates to domain entities like object properties and relationships, we also reduced the conceptual load on the domain expert because it is easier to create lists of domain entities and assign them all of the necessary information (including templates, now) than it is to go through the entire Thought Process Tree to create the templates for each node.

**Table 1.** Improvements made to the amount and size of the templates.

| | before | | | after | | |
|---|---|---|---|---|---|---|
| | total templates | avg. words per template | total words | total templates | avg. words per template | total words |
| action nodes | 38 | 13.7 | 521 | 54 | 5.5 | 297 |
| question nodes | 134 | 12.6 | 1681 | | | |
| branching nodes | 65 | 15.2 | 985 | 65 | 15.2 | 985 |
| total | 237 | 13.4 | 3187 | 119 | 10.8 | 1282 |

## 5 Discussion

One of the important questions to discuss is the generalizability of the proposed method. It was initially created to lower the workload of generating pedagogical questions and relevant explanations for subject domains described by Thought Process Trees. However, the basic patterns of natural languages, utilized in the proposed method, can be used in other contexts.

For example, the Thought Process Language model (see [15]) lends itself well to such an integration. TPL already employs templates connected to formal language constructs for the generation of natural-language textual explanations;

however, the approach is not described in detail, and it seems to have similar limitations to the first version of Thought Process Tree natural-language generation, that is, the rigidity of templates and their inability to substitute names of domain-specific entities. As such, TPL-based systems can also benefit from adapting the proposed approach to lower the workload of creating templates and allow for question-based dialogue (currently, it only supports explanatory monologue).

Other systems can also benefit from that approach. For example, the question system in Jeliot 3 [14], in which the questions provided to the students are manually entered by teachers, can be improved using that method. Jeliot uses Java code as both the model and the subject of study, so the automatic question generation can be implemented there by assigning templates to Java expressions and classes (matching node expressions and domain-specific entities in our work, respectively).

Systems using the Knowledge Construction Dialogues framework [1,5,10] can also benefit from the reduced workload that our method provides because they rely heavily on the domain expert to provide both formulations and structure of the dialogue. However, those systems do not use any kind of formal model to describe the process by which the questions are asked; the templates must be not only created but also used by the question authors manually. The question of applying the proposed method to other systems requires further research.

## 6 Conclusion

In this study, we proposed a method to reduce the workload of creating natural-language templates for generating pedagogical questions, answer options for them and explanations of wrong answers required to construct an ITS capable of having a question-based dialogue with the student. The previous approach required specifying natural language templates for each node of a Thought Process Tree describing the problem-solving reasoning separately, which resulted in the domain expert needing to write many similar texts.

We proposed a method that allows attributing the text templates to domain elements like object classes, properties and their relationships, and generating descriptions based on the expressions contained in the nodes. By recognizing the patterns within these expressions, we were able to classify nodes based on the natural-language patterns required to describe them and use that classification for question generation. The placeholders inside the templates are substituted with the descriptions of the domain elements used in the expression.

That allowed us to significantly reduce both the number of templates and the average template length, making the system of templates more modular to allow for template reuse, while still maintaining the possibility for the expert to create more nuanced, higher-level templates if they so choose.

There are several ways this approach can be further improved. The directions of further research include identifying more expression patterns to describe; choosing the appropriate language pattern for question formulation based on the

question type and property (relationship) description (e.g., if the main word in the description is a noun or a verb) and so on. It is also important to extend this approach to the structural nodes of Thought Process Trees: Branching Nodes and Loop Nodes.

## References


1. Albacete, P., Jordan, P., Katz, S.: Is a dialogue-based tutoring system that emulates helpful co-constructed relations during human tutoring effective? In: Conati, C., Heffernan, N., Mitrovic, A., Verdejo, M.F. (eds.) AIED 2015. LNCS (LNAI), vol. 9112, pp. 3–12. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19773-9_1
2. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R.: The cognitive tutor authoring tools (CTAT): preliminary evaluation of efficiency gains. In: Ikeda, M., Ashley, K.D., Chan, T.-W. (eds.) ITS 2006. LNCS, vol. 4053, pp. 61–70. Springer, Heidelberg (2006). https://doi.org/10.1007/11774303_7
3. Anderson, R.C., Biddle, W.B.: On asking people questions about what they are reading. In: Bower, G.H. (ed.) Psychology of Learning and Motivation, vol. 9, pp. 89–132. Academic Press (1975). https://doi.org/10.1016/S0079-7421(08)60269-8
4. Belland, B.R.: Scaffolding: definition, current debates, and future directions. In: Spector, J.M., Merrill, M.D., Elen, J., Bishop, M.J. (eds.) Handbook of Research on Educational Communications and Technology, pp. 505–518. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-3185-5_39
5. Freedman, R., Rosé, C.P., Ringenberg, M.A., VanLehn, K.: ITS tools for natural language dialogue: a domain-independent parser and planner. In: Gauthier, G., Frasson, C., VanLehn, K. (eds.) ITS 2000. LNCS, vol. 1839, pp. 433–442. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45108-0_47
6. Ge, X., Land, S.: Scaffolding students' problem-solving processes in an ill-structured task using question prompts and peer interactions. Educ. Technol. Res. Dev. **51**, 21–38 (2003). https://doi.org/10.1007/BF02504515
7. Gruppen, L., Irby, D.M., Durning, S.J., Maggio, L.A.: Interventions designed to improve the learning environment in the health professions: a scoping review. MedEdPublish **7**, 211 (2018). https://doi.org/10.15694/mep.2018.0000211.1
8. Hughes, J.: The deskilling of teaching and the case for intelligent tutoring systems. J. Ethics Emerg. Technol. **31**(2), 1–16 (2021). https://doi.org/10.55613/jeet.v31i2.90
9. Koedinger, K., Anderson, J., Hadley, W., Mark, M.: Intelligent tutoring goes to school in the big city. Int. J. Artif. Intell. Educ. **8**, 30–43 (1997)
10. Lane, H.C., VanLehn, K.: Teaching the tacit knowledge of programming to novices with natural language tutoring. Comput. Sci. Educ. **15**(3), 183–201 (2005). https://doi.org/10.1080/08993400500224286
11. Le, N.-T., Kojiri, T., Pinkwart, N.: Automatic question generation for educational applications – the state of art. In: van Do, T., Thi, H.A.L., Nguyen, N.T. (eds.) Advanced Computational Methods for Knowledge Engineering. AISC, vol. 282, pp. 325–338. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06569-4_24
12. Mazidi, K., Tarau, P.: Automatic question generation: from NLU to NLG. In: Micarelli, A., Stamper, J., Panourgia, K. (eds.) ITS 2016. LNCS, vol. 9684, pp. 23–33. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39583-8_3

13. Moreno, A., Myller, N., Sutinen, E., Ben-Ari, M.: Visualizing programs with jeliot 3. In: Proceedings of the Working Conference on Advanced Visual Interfaces, AVI 2004, pp. 373–376. Association for Computing Machinery, New York (2004). https://doi.org/10.1145/989863.989928
14. Myller, N.: Automatic generation of prediction questions during program visualization. Electron. Notes Theor. Comput. Sci. **178**, 43–49 (2007). https://doi.org/10.1016/j.entcs.2007.01.034
15. O'Rourke, E., Andersen, E., Gulwani, S., Popović, Z.: A framework for automatically generating interactive instructional scaffolding. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, pp. 1545–1554. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2702123.2702580
16. Peverly, S.T., Wood, R.: The effects of adjunct questions and feedback on improving the reading comprehension skills of learning-disabled adolescents. Contemp. Educ. Psychol. **26**(1), 25–43 (2001). https://doi.org/10.1006/ceps.1999.1025
17. Rothkopf, E.Z.: Learning from written instructive materials: an exploration of the control of inspection behavior by test-like events. Am. Educ. Res. J. **3**(4), 241–249 (1966). https://doi.org/10.3102/00028312003004241
18. Shah, R., Shah, D., Kurup, L.: Automatic question generation for intelligent tutoring systems. In: 2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA), pp. 127–132 (2017). https://doi.org/10.1109/CSCITA.2017.8066538
19. Sychev, O., Gumerov, M.: Generating pedagogical questions to help students learn. In: Frasson, C., Mylonas, P., Troussas, C. (eds.) Augmented Intelligence and Intelligent Tutoring Systems, pp. 195–208. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-32883-1_17
20. Sychev., O., Penskoy., N., Terekhov., G.: Using software reasoning to determine domain-law violations and provide explanatory feedback: expressions tutor example. In: Proceedings of the 14th International Conference on Computer Supported Education, vol. 1: CSEDU, pp. 116–123. INSTICC, SciTePress (2022). https://doi.org/10.5220/0011070100003182
21. Uglev, V., Sychev, O.: Synthesizing didactic explanatory texts in intelligent tutoring systems based on the information in cognitive maps. In: Augmented Intelligence and Intelligent Tutoring Systems, pp. 233–246. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-32883-1_20
22. VanLehn, K., Graesser, A.C., Jackson, G.T., Jordan, P., Olney, A., Rosé, C.P.: When are tutorial dialogues more effective than reading? Cogn. Sci. **31**(1), 3–62 (2007). https://doi.org/10.1080/03640210709336984
23. Wise, A.F.: Designing pedagogical interventions to support student use of learning analytics. In: Proceedings of the Fourth International Conference on Learning Analytics and Knowledge, LAK 2014, pp. 203–211. Association for Computing Machinery, New York (2014). https://doi.org/10.1145/2567574.2567588