computers

MDPI

Article

Searching Questions and Learning Problems in Large Problem Banks: Constructing Tests and Assignments on the Fly

Oleg Sychev

Software Engineering Department, Volgograd State Technical University, Lenin Ave., 28, 400005 Volgograd, Russia; oasychev@gmail.com

Abstract: Modern advances in creating shared banks of learning problems and automatic question and problem generation have led to the creation of large question banks in which human teachers cannot view every question. These questions are classified according to the knowledge necessary to solve them and the question difficulties. Constructing tests and assignments on the fly at the teacher's request eliminates the possibility of cheating by sharing solutions because each student receives a unique set of questions. However, the random generation of predictable and effective assignments from a set of problems is a non-trivial task. In this article, an algorithm for generating assignments based on teachers' requests for their content is proposed. The algorithm is evaluated on a bank of expression-evaluation questions containing more than 5000 questions. The evaluation shows that the proposed algorithm can guarantee the minimum expected number of target concepts (rules) in an exercise with any settings. The available bank and exercise difficulty chiefly determine the difficulty of the found questions. It almost does not depend on the number of target concepts per item in the exercise: teaching more rules is achieved by rotating them among the exercise items on lower difficulty settings. An ablation study show that all the principal components of the algorithm contribute to its performance. The proposed algorithm can be used to reliably generate individual exercises from large, automatically generated question banks according to teachers' requests, which is important in massive open online courses.

Keywords: exercise generation; question search; knowledge graphs; test-sheet generation; assignment generation; question banks

Citation: Sychev, O. Searching Questions and Learning Problems in Large Problem Banks: Constructing Tests and Assignments on the Fly. *Computers* **2024**, *13*, 144. https://doi.org/10.3390/computers13060144

Academic Editor: Stelios Xinogalos

Received: 11 May 2024
Revised: 31 May 2024
Accepted: 3 June 2024
Published: 5 June 2024

1. Introduction

An increasing number of educational institutions working on different levels of education have adopted e-learning and online learning technologies [1]. That trend was heavily boosted by lockdowns during the COVID-19 pandemic when many institutions were forced to rapidly employ e-learning technologies [2,3]. The transition to online learning is not always smooth: when educational institutions are not ready for it, it often causes problems and disadvantages [4]. One of the problems during the transition to online learning is performing formative and summative assessments online, which is shown to be beneficial for students but puts additional workload on the teaching staff [5,6]. This requires creating and maintaining electronic banks of questions and learning problems. Online learning allows and sometimes requires bigger problem banks than regular classroom learning because of the following reasons:

- When tests and assignments are graded automatically, online learners can do a lot more exercises than they can do in the classroom, especially if the learning system provides feedback;
- Online learning allows greater variety in the exercises performed; they can be chosen either by students or using recommender systems to create personal learning trajectories, but the problem bank should provide enough variety so that all kinds of students have exercises suited for them;

- Performing assessments without supervision from the teaching staff is more vulnerable to cheating, especially with closed-answer or short-answer questions where the number of possible correct answers is limited, so the institutions need more variety in the learning tasks students must perform.

Developing questions and learning problems or gathering them from existing sources takes significant time. Teachers are often overworked and need more time to spare on the routine job of authoring many similar exercises. Developing and adopting Open Educational Resources can help with that, but it has its own problems and concerns [7].

One of the ways to counter this problem is by using artificial intelligence techniques to generate questions and learning problems automatically. Automatic generation saves effort on creating question and problem banks and increases the size and consistency of these banks [8]. Usage of automatically generated problem and question banks provides the following benefits in the learning process [9]:

- Preventing some types of cheating by supplying each learner with a unique set of questions and learning problems;
- Training until mastery because the learner is not limited by the size of the problem bank, which is especially important when these problem banks are used in conjunction with intelligent tutoring systems that can provide automatic feedback on errors and hints;
- Better support adaptive learning in intelligent tutoring systems (ITS) because the bank contains a lot of various learning problems so that the adaptive system can find a learning problem for every educational situation.

However, these benefits come with a cost that impedes the adoption of generated questions in the learning process [10]. While using an existing bank requires far less effort than creating a teacher's own bank, teachers' trust in question and problem banks received from external sources is significantly lower, especially when the bank is so large that the teacher cannot read, solve, and/or evaluate all the learning problems or even a significant portion of them. Teachers are naturally hesitant to use questions and learning problems with which they are not acquainted. Also, large problem banks, often created from different sources, raise the issue of problem homogeneity and consistent evaluation of their difficulties. It is also important to avoid learning problems containing topics the students have not yet learned. Even the concepts (or topics) that do not affect the answer to the question (problem solution) should be taken into account because they can confuse some students. Thus, a good filtering system is a must to deal with any large problem bank, especially when solving the problems is required to complete the course. All these factors restrict the widespread usage of automatically generated banks of learning problems.

Question (problem) banks are used to construct specific exercises (tests, assessments). When the question banks were small, teachers combined questions (or problems) into an exercise that all the students had to pass. When the number of questions increased, the typical organization of a random test involved the teachers grouping similar questions into categories and constructing tests by specifying the number of randomly selected questions from each category. This allowed more diversity in the questions students received, which lessened cheating and made attempting one test several times non-trivial but left the test plan under the teacher's control as categories usually contained similar questions (learning problems). However, that method requires a teacher-defined classification of questions in complex categories according to their purpose, which is not always available for automatically generated questions and learning problems. The same requirements are used in methods of automatic test generation (e.g., [11]).

The problem of automatic generation and the usage of large banks of questions and learning problems is especially important in the field of computer programming education. Learning to program requires learning a significant number of new concepts and mental manipulation of abstract objects. This requires a lot of training by solving well-defined tasks, which creates good conditions for developing and using automatic tutors with explanatory feedback and formal, automatically graded exams [12]. A high failure rate in

introductory programming courses [13] also signals that students need additional support in learning the material.

However, training until mastery by using automatic tutors and the frequent usage of exam tests requires big banks of questions and learning problems so that students do not get the same questions several times, which requires developing techniques of generating questions and generating balanced test sheets from them. This study is aimed at solving the second problem, which has received significantly less attention than the actively developing field of question generation.

Developing automatic question and learning-problem generation methods creates problems for classic methods of building tests and exercises from questions and learning problems. When the question bank becomes too big to be observable by a human teacher, we need to develop new question search and retrieval methods according to the teachers' needs. This is a significant and complex problem because in many subject domains, learning problem features depend on the problem structure, not just the types of its elements. For example, in introductory programming courses, when teaching control-flow statements to assess the problem topic, it is often enough to take into account the statements that are present in the problem formulation (even though they can be in dead code that is never reached during the problem solving). However, the same principles do not work for teaching expressions: rules of operator precedence and associativity influence the answer under certain circumstances. For example, to teach (or verify knowledge of) associativity, you need an expression where two operators with the same precedence stand in a row without parentheses affecting their evaluation. That does not allow constraining learning-problem generation simply by constraining the types of objects (e.g., statements or operators) participating in the formal problem formulation, so we cannot reliably predict classes of generated questions without a complex analysis of the problem formulation.

This study aims at developing an algorithm for constructing exercises from questions and learning problems stored in large problem banks when the problems are labeled according to the knowledge necessary to solve them and difficulty metrics in the absence of significant statistics of the usage of these learning problems because they can be specifically generated to avoid giving the same problem to different students.

The contribution of this study includes:

- A problem formulation for learning problem (question) retrieval from large banks of automatically generated questions, including the required formalization of learning problem metadata, problem request, and retrieval goals that does not require manually assigning slots for questions and statistical data on question usage;
- An algorithm for question search taking into account the subject-domain concepts and rules required to solve them and question difficulties;
- An evaluation of the proposed algorithm on a knowledge bank consisting of thousands of automatically generated questions on the order of expression evaluation.

The rest of the article is organized as follows. Section 2 discusses related work. Section 3 describes the proposed algorithm. Section 4 describes the experimental evaluation of that algorithm on a large problem bank for a particular topic. Section 5 discusses the study's findings. Section 6 contains conclusions and further work. The line between complex questions and easy learning problems are not clearly established, so in the rest of the article, the term "question" is used to describe both questions and learning problems except in places where the distinction between them is important.

2. Related Work

There are two kinds of related work to review. Methods of question and learning problem generation are essential because they determine what kinds of information we can get about the generated questions. Methods of selecting and recommending questions are used to construct exercises from banks.

2.1. Methods of Question and Learning-Problem Generation

Different methods of generating questions and learning problems give different information about their results. It determines the possible usage of the resulting questions and the methods of their retrieval for constructing tests and exercises.

A significant part of the methods of question generation is concentrated on processing natural-language texts and generating questions about the content of that text [8,14–17]. Some researchers add searching texts by keywords to leverage this to generate sets of questions for the given keyword [18]; this approach has obvious drawbacks because the teacher cannot control the content of these texts. Some recent works on Visual Question Generation try to use an image as input for question generation instead of text, using methods of solving the problem of image understanding and summarization [19–21]. These questions are aimed at solving the reading comprehension problem, i.e., verifying that the learner has read and understood the given text or image. Thus, these questions are not linked to studied topics (except the keyword-based generation [18]) but to particular learning materials (textbook fragments, lecture notes, etc.). They can be used to generate quizzes placed right after the learning materials and combined to generate summative quizzes by specifying the number of questions about each material in the topic or course. Chen et al. created a sophisticated dataset of teacher- and learner-created questions, with a significant percentage of questions aimed at high-level thinking skills, and found that modern methods performed poorly compared to humans exercising skills on medium and high levels of Bloom's taxonomy (comprehension, application, analysis) [22]. The chief problem of questions generated by automatic methods, as it was noted by Kurdi et al. [8], is that they have low cognitive levels and simply ask for the facts mentioned in the text (image); they do not verify understanding of these facts, the ability to combine them with other facts (from the same text or other texts) and use them in practical situations.

Ontologies (and similar representations of structured knowledge) are also a popular approach to generate questions, which has been used to generate questions to teach medicine [23], programming [24], and general knowledge stored in commonsense knowledge graphs [25]. These questions are generated using the graph representation of the required knowledge. Therefore, they can be linked not to a particular textbook fragment but to the subject-domain concepts they are about. This allows the generation of topic-based question banks and requires efficient mechanisms of question retrieval according to the teacher's plans. The difficulty of these questions varies significantly.

More complex tasks are solved in the field of learning problem generation. The chief approaches to generating learning problems are template-based generation [26,27], random generation within subject-domain constraints [28,29], and problem mining [30].

Template-based generation practically means that the learning problems are planned by teachers as templates. A set of problems is created from the same template by changing secondary parameters, which usually affect the correct answer but do not affect the knowledge required to solve problems [31]. Examples of systems using template-based generation are some of Problet tutors [32] (which generate programs and results of their executions from teacher-written templates in pseudo-BNF notation) and QuizJET [33] (a tutoring system aimed at teaching the Java programming language and object-oriented programming which contains about 100 manually created problem templates). This approach can also be used outside of teaching programming, e.g., for generating natural deduction problems [34].

The used templates (and learning problems generated from them can be manually classified; they can use regular methods of exercise generation by specifying the required number of problems of each type. The downside of that method is its low scalability and significant intellectual work from the author of the exercise, who still has to invent every kind of exercise and code it according to the learning problem template. The generation algorithm only multiplies existing learning problems without varying their structure.

The method of random generation within subject-domain constraints has entirely different features. It can generate genuinely new problems (i.e., problems whose structure and

solutions are not planned by a human author), produce problems with different difficulties, and require knowledge within a particular topic while maintaining the correctness of the generated problems using constraints defined by the subject domain. The most popular formalism in this area is Answer Set Programming (ASP), which can be used both to infer solutions to problems and generate problems that can be solved by the same program [35]. It is a widely used approach to procedural generation for designing computer games [36,37]. O'Rourke et al. used ASP to generate algebra problems [29]. However, this approach is limited to learning problems in relatively simple domains where all problem elements can be combined in almost any way (e.g., arithmetic operators in algebraic expressions). When the domain becomes more complex (e.g., algebraic expressions which are homogeneous by data types with expressions in programming languages that are heterogeneous, with different operators applicable to different data types), random generation can only guarantee the basic correctness of the generated problems, but it cannot guarantee their readability and sensibility. For example, Martin and Mitrovich used this approach to generate learning problems for a constraint-based tutor for the SQL language; the generated problems were not instantly applicable and required clarification and validation by a human [28]. Kumar argues that in programming education, randomly generated problems are only helpful for syntax-oriented exercises [38].

Problem mining also relies on subject-domain constraints to solve and classify generated problems [30], but instead of random generation, it uses real data created and verified by humans (e.g., program code, banks of parsed sentences) or digital twins of real-world objects. This allows the large-scale generation of complex, naturally looking learning problems classified by their difficulties and required knowledge in the form of domain concepts and rules required to solve them or possible semantic errors in their solutions [39]. These learning problems often can be used without human verification, which allows scaling generation to problem banks containing many thousands of problems on the topic. It is a promising approach in the fields when the necessary data can be obtained, but it requires developing complex subject-domain models to solve and classify the generated problems [12,34].

Both randomly generated and mined learning problems in complex domains cannot be automatically assigned categories for exercise creation and require special methods of learning problem search according to the particular learning situation. These learning problems can be labeled by the concepts used in the problem definition and domain rules and laws whose knowledge is necessary to solve them. However, the number of combinations of these labels is high, so most of the time, these labels cannot be used directly in exercise construction. For example, the automatically generated problem bank of more than 20,000 learning problems described in [39] contains 613 different kinds of problems (taking into account the concepts and rules whose knowledge is required to solve the problem and the problem difficulty). In some domains, it can be even more difficult: for example, when studying expressions, if we consider knowing the precedence and associativity of each operator a separate kind of knowledge, we obtain more than 840 kinds of learning problems. This makes it impossible for a human teacher to look through all the categories of learning problems in the problem bank and select the categories that should be used in their assignments. Thus, we need to develop algorithms of learning problem search and retrieval from large banks with a reasonable amount of information provided by the teacher.

2.2. Methods of Selecting and Recommending Questions and Learning Problems

One of the most established models of estimating the difficulty of questions and learning problems is Item Response Theory (IRT) [40]. This model is actively used in intelligent tutoring systems for modeling the difficulty of exercise items for students [41]. However, this model relies on gathering statistical data for each item (i.e., a question) in the bank and so cannot be used in large knowledge banks that individualize learning by avoiding using the same question for different learners.

Probabilistic models like Bayesian Knowledge Tracing (BKT) are often used to estimate the students' mastery and choose the next learning problem in exercises. It considers the probabilities of learning (transitioning to the mastery state), guessing (answering correctly without knowledge), and slipping (answering incorrectly while knowing) depending on the skill [42,43]. This theoretically can be used in large generated question banks because students receive different questions but learn the same skills; however, it is impossible to use it in learning-problem banks because one learning problem often requires several skills to solve. Attempts to improve the original BKT model include individualization (i.e., modeling the rate of learning or initial knowledge of each student) [44,45] and item difficulty (i.e., the difficulty of a particular learning problem) [46,47]. While they improve the performance of the original model, they also require significantly more data to train. Pardos et al. found that the model that takes into account item difficulty (KT-IDEM) showed better results over the original BKT model only when the number of solving data per problem exceeded six. This will not work for large, automatically generated problem banks aimed at providing unique learning problems for each student. Deep Knowledge Tracing, a development of Bayesian Knowledge Tracing using Recurrent Neural Networks, also relies on data from previous attempts of existing questions to teach the neural network [48].

Another field of research concerning selecting questions to solve for a student is exercise recommendation systems. That field has been actively researched in the last years [49–51]. Most of these systems use hybrid approaches and rely either on the data concerning item completion by students or student proficiency data. However, their evaluation is often based on prediction accuracy instead of pedagogical effect [52]. This makes them unsuitable for use in the learning process where ideally, each learning item is used only once. For now, the attempts to predict difficulty and necessary knowledge of learning problems and digital course sessions are concentrated on labeling them by difficulty [53] and/or used concepts [54], which does not solve the problem of retrieval of appropriate exercises from a labeled bank.

The problem of generating tests and assignments with necessary characteristics has been studied for many years. Hwang et al. proposed an algorithm of generating test sheets from big question banks [55]. However, their algorithm was aimed at generating a single test sheet with the optimized discrimination coefficient relevant to listed concepts; they did not take into account the problem of generating multiple similar test sheets for different students to avoid cheating. They also relied on question parameters that were measured statistically during their usage and so were poorly suited to large banks of sparsely used questions where the relevant information is not available. In [56], they improved their algorithm to generate a set of test sheets with the requested characteristics, but they used a hard constraint "no question should be identical in two tests", which is impractical when generating a large number of tests (e.g., for each student). They only generated at most four parallel test sheets with the same specification.

Paul proposed generating a set of solution, but the goal of that algorithm was to achieve satisfactory performance regarding several objectives [57], not variable tests for different learners. Sud et al. considered the problem of generating alternative assessments to avoid cheating by sharing information among students [11]. They managed to reduce the standard deviation of questions' difficulty to about half of its regular values. However, their problem formulation required a pre-classification of questions into different slots, which required manual work and could not be done in diverse automatically generated question banks.

Thus, the problem of generating a large number of test sheets from diverse question banks without previous manual question classification and gathering statistics on question usage is not covered in previous works. This study is aimed at solving that problem.

3. Proposed Algorithm

To discuss the algorithm of question retrieval, we must first consider the structure of question metadata available in the question bank and question request to develop a formal problem statement. Then, the algorithm for satisfying that request is proposed.

3.1. Question Metadata and Requests

In order to find questions satisfying tutors' needs (both for human and automatic tutors), we need diverse information about questions and learning problems contained in the bank. Let us consider a question bank Q consisting of questions $q_i \in Q$. We have sets of studied concepts C and rules (domain laws) R for the given subject domain. Rules define the application of concepts in particular situations. In practice, it is often useful to consider possible semantic errors defined as constraints as domain rules because it allows a more precise covering of the relevant knowledge than using rules that govern answering questions and problem solving, i.e., one rule that determines if a particular solution step or answer is possible can be violated in several different ways [12]. We also need to estimate question difficulty, which is based not only on the concepts and rules whose knowledge are necessary to answer the given question but also on the structure of the question definition and the number of objects (concept individuals) in it, i.e., solving an algorithm with more statements, statement nesting, or more trace steps is more difficult than solving a simpler algorithm using the same statements (for example, in [39], it was shown that for code-tracing problems, the optimal difficulty function included the number of used concepts, number of trace steps and cyclomatic complexity of the algorithm.

Thus, the metadata for question q_i can be defined as (1)

$$q_i =< C_i, R_i, d_i >, \tag{1}$$

where $C_i \subset C$ is the set of concepts whose knowledge is necessary to answer the question, $R_i \subset R$ is the set of domain rules that are used to find the correct answer to the question, and d_i is the estimated difficulty of the question.

The non-adaptive exercise consists of groups of questions with defined requests (i.e., one request can specify a series of questions). For each group of questions, the exercise author should divide subject-domain concepts C and rules R into three categories:

- `target` concepts C^t and rules R^t define the content that the teacher intends to teach (or assess learning of) in this exercise (question);
- `denied` concepts C^d and rules R^d must not appear in the exercise (or a particular part of it) because students are not expected to know them;
- `allowed` concepts C^a and rules R^a can appear in the exercise but are unnecessary (e.g., previous knowledge).

Every concept and rule must belong to one of these categories, i.e., for all groups of questions in the exercise, $C^t \cup C^d \cup C^a = C$ and $R^t \cup R^d \cup R^a = C$. This allows flexibility in selecting relevant questions because the concepts and rules that do not belong to the goal of this question (i.e., non-target concepts and rules) can be divided into those that must be excluded and those that can appear in the exercise, thus broadening the range of possible questions that can match the request. They key difference of a question request from a regularly used question category is that the system does not have pre-made categories fitting teachers' requests but chooses the questions from the bank according to their features.

In real-world tutoring situations, concepts and especially rules that are shown in the tutor's interface and assigned to the target, denied, and allowed categories must not match exactly the concepts and especially the rules that are stored in question metadata. For example, when teaching the order of evaluation of programming-language expressions, the situations of taking into account the precedence of the operator to the left and the precedence of the operator to the right are different rules in the formal model, but nobody teaches them separately—for teachers and students, they are represented as one rule. Similarly, in many expression-related educational situations, it is not necessary to discern

arithmetic operators because they are governed by the same rules and are taught together. Sometimes, human problems of perception and understanding affect the formal model, e.g., while handling two operators with right associativity in a row can be modeled as one rule, for the human perception, it is different for unary (e.g., $!!a$) and binary ($a = b = 0$) operators and should be taught and verified separately because visually, these situations are too different.

Concepts can be connected by links like "requisite" and "parent–child". For example, when teaching control-flow statements, it is impossible to create a question with the "else" concept without the "if" concept, so "if" is considered "requisite" for "else". This allows one to restrict the user interface to avoid requests that are impossible to satisfy (e.g., with target "else" and denied "if"). "Parent–child" relationships allow exercise authors to select several concepts (e.g., all kinds of loop statements) in one click.

Question request can be formalized as (2)

$$qr_j =< Nq_j, C_j^t, C_j^d, R_j^t, R_j^d, d_j >, \quad (2)$$

where Nq is the desired number of questions, C^t, C^d, R^t, R^d are sets of target and allowed concepts and rules, and d is the approximate desired difficulty.

The task of question retrieval can be formulated as follows. Given a question request qr_j, find a set of questions $Q_j \subset Q$ so that:

1. $|Q_j| = Nq_j$ (obtain the required number of questions);
2. $\forall q_{ij} \in Q_j \exists C_{ij}, C_{ij} \in C_j^t \wedge C_{ij} \in C_i$ (each question contains at least one target concept);
3. $\forall q_{ij} \in Q_j \exists R_{ij}, R_{ij} \in R_j^t \wedge R_{ij} \in R_i$ (each question contains at least one target rule);
4. $\forall q_{ij} \in Q_j \nexists C_{ij}, C_{ij} \in C_j^d \wedge C_{ij} \in C_i$ (there are no denied concepts in the found questions);
5. $\forall q_{ij} \in Q_j \nexists R_{ij}, R_{ij} \in R_j^d \wedge R_{ij} \in R_i$ (there are no denied rules in the found questions);
6. The distributions of elements of C^t and R^t across the found questions are as close to a uniform distribution as possible (spend equal effort on all target concepts and rules);
7. The difference $sum(d_i)/Nq - d$ is minimized (the questions are as close to the desired difficulty as possible).

3.2. Algorithm for the Retrieval of Relevant Questions

While filtering out questions with denied concepts is not hard, coming close to an even distribution of target concepts and rules is a challenging goal. First, the question banks that are generated automatically without using human-written templates can have an uneven distribution of different kinds of questions: some concepts and rules can be present several times and more often than others [9]. While this reflects real-world situations where some subject-domain rules are used more often than others, it is bad for learning because learning should prepare students for every situation they can encounter when solving problems in their subject domain. Also, the connection between rules and concepts is not even: one rule can involve many concepts while the other can involve only a few. For example, many operators are left-associative, but in most of the programming languages, there are few right-associative operators, so if we are teaching associativity, we cannot have even coverage of left and right associativity (rules) and different operator groups (concepts).

Using a random selection of appropriate questions is not good for achieving an even distribution of target concepts and rules because of their possible uneven distribution in the question bank. Also, with a small sample size (many learning exercises consist of a small number of questions), a random selection will often produce significant deviations from the uniform distribution. To better achieve an even representation of different learning targets in the exercise, we propose to adjust the selection of each question by rotating targets (i.e., prioritizing the targets that were used less in the previous questions of an exercise attempt and marking the targets that have already been used often as undesirable).

Therefore, for generating each question, we must determine the specific question request (SQR) based on the original questions request. The main goal of the specific

question request is to find the least used target concepts or rules and make them targets for the next question. Algorithm 1 shows how a specific question request sqr is found based on the data from the general question request for this exercise part qr and the data about previous questions in exercise exc.

Algorithm 1 The algorithm for determining specific question request sqr based on the question request qr and exercise data exc.

function SPECQR(qr, exc)
$sqr_{Nq} \leftarrow 1$ ▷ Specific request is for the next question.
$sqr_{C^d} \leftarrow qr_{C^d}$ ▷ Denied concepts and rules remain the same.
$sqr_{R^d} \leftarrow qr_{R^d}$
$sqr_d \leftarrow qr_d \times (1 + \text{RAND}(-0.2, 0.2))$ ▷ Change the question difficulty slightly to broaden the range of available questions.
for all $c_i^t \in qr_{C^t}$ **do**
$cusage[c_i^t] \leftarrow |UC_i|, where\ UC_i = \{exc_q | c_i^t \in exc_{q_C}\}$ ▷ ex_q is a previous question used in the attempt.
end for
$sqr_{C^t} \leftarrow \{C_i^t | C_i^t \in qr_{C_t} \wedge cusage[c_i^t] = \text{MIN}(cusage)\}$ ▷ Make the least-used concepts targets for the next question.
$sqr_{C^a} \leftarrow qr_{C^a} \cup (qr_{C^t} \cap sqr_{C^t})$ ▷ All other target concepts become allowed for this question.
for all $r_i^t \in qr_{R^t}$ **do** ▷ The same applies to building sets of target and allowed rules.
$rusage[r_i^t] \leftarrow |UR_i|, where\ UR_i = \{exc_q | r_i^t \in exc_{q_R}\}$
end for
$sqr_{R^t} \leftarrow \{R_i^t | R_i^t \in qr_{R_t} \wedge rusage[r_i^t] = \text{MIN}(rusage)\}$
$sqr_{R^a} \leftarrow qr_{R^a} \cup (qr_{R^t} \cap sqr_{R^t})$
return sqr
end function

The generated specific question request is then passed to the function that searches relevant questions in the question bank. The algorithm of question search according to a specific question request consists of the following steps:

1. Prepare data for question search.
   (a) Normalize the request's difficulty from the range $[0, 1]$ to the actual range of question difficulties in the question bank.
   (b) Create sets of undesirable concepts and rules: the concepts and rules which were present in the previous N_{prev} questions (with N_{prev} being a search hyper-parameter).
2. Execute a search request in the question bank. The request searches for up to N_{pool} best questions (according to the number of target concepts and rules per question), taking into account the following restrictions:
   (a) Question difficulty must be in range of $\pm Diff_{dev}$% relative to the request difficulty.
   (b) No questions with at least one denied concept or rule are allowed.
   (c) Questions are sorted according to the total number of target concepts and rules in them (descending order), then for the number of their usage in the current exercise (ascending order), so that the most used questions come last.
3. Rank the found questions by three ranking measures:
   (a) The absolute value of the difference between the question difficulty and the requested difficulty (the smaller the better).
   (b) Question "suitability": every target concept and rule adds 3 points of suitability while every undesirable concept and rule subtracts 1 point of suitability (the bigger, the better).

(c) Question diversity: the number of different possible semantic errors while answering the question (the bigger, the better).

4. Calculate the final question rank as a weighted sum of the three ranks discussed above.
5. Return the desired number of best questions according to the final question rank.

That algorithm penalizes repeating the same concepts and rules in subsequent questions to allow the equal representation of all the target concepts and rules in the generated set. It allows a varied range of question difficulties (controlled by hyperparameter $Diff_{dev}$) and more variety in the found questions because of pooling the best-fitting questions and random selection among them.

4. Evaluation

We used a bank of questions generated from program code (see [9] for the description of the generation process) to evaluate the efficiency of the proposed algorithm. The bank used the widely accepted RDF data storage format and had a high variability in the questions because they were generated from randomly selected human-written production program code, not specifically designed for learning purposes. As it can be seen in [9], different concepts and rules (subject-domain laws) were not evenly represented in the bank, so it was well-suited to measure the ability of the proposed algorithm to deal with large, uneven banks of questions.

The bank contained questions about the order of expression evaluation and consisted of 5648 questions with difficulties ranging from 0.54 to 1.0, 11 concept groups, and seven rules.

4.1. Criteria of Evaluation

To evaluate the quality of the generated exercises, first of all, it was necessary to estimate if all the target concepts were present enough times, i.e., if an exercise with N_q questions had N_t target rules (concepts), each target had to appear at least N_q/N_t times. It usually appeared more times because most of the questions contained more than one concept, but N_q/N_t was the required minimum that, ideally, should not be violated in any generated exercise instance. To verify that, we calculated the relative frequency of target concepts (rules) RF_i, which was equal to the number of questions with the i-th target in the attempt divided by the minimum expected number N_q/N_t; see (3).

$$RF_i = |Q_i| * N_t/N_q, \tag{3}$$

where Q_i is the set of the questions with the i-th concept in the generated attempt.

In order to study how well the proposed algorithm adapted to different circumstances, it was decided to study the average number of used rules in questions depending on the number of target rules in the exercise: ideally, both the number of rules per question and question difficulty should remain the same because they should be affected by the exercise difficulty, not by the number of target rules, which should be distributed among different questions in the exercise on lower difficulties. In contrast, average question difficulty, number of rules per question, and number of solution steps per question ideally should monotonically increase with the exercise difficulty, which can be negatively affected if the bank is heavily unbalanced.

4.2. Evaluation of the Algorithm as a Whole

The question bank contained questions asking the student to determine the order of evaluation of the given programming-language expression. The main source of the information in that case was not just concepts but used subject-domain rules, because the rules necessary to answer the questions are determined by the way the operators are placed within the expression, i.e., questions with the same concepts can require knowledge of different rules to solve. Tables 1 and 2 show the distribution of questions with different

concepts and rules in the bank. The bank was diverse, with the number of questions requiring knowing certain rules ranging from 5317 questions for operator precedence to 29 questions for right associativity for binary operators.

Table 1. Number of questions featuring different concepts in the expression bank.

Concept	Number of Questions
Logical operators	4073
Pointer access operators	3749
Object access operators	3682
Comparison operators	3354
Arithmetic operators	1651
Array access operator	1492
Assignment operator	1250
Comma operator	1019
Bitwise operators	987
Increment and decrement operators	79
Assignments with arithmetic operators	67
Total	5648

Table 2. Number of questions requiring knowledge of different rules in the expression bank.

Rule	Number of Questions
Operator precedence	5317
Binary operator with left associativity	3265
Operators with a strict order of evaluating operands	2807
Operators enclosing other operators	1601
Unary operator with right associativity	287
Unary operator with left associativity	30
Binary operator with right associativity	29
Total	5648

Different rule sets were chosen as targets for that knowledge bank, as shown in Table A1. Five rules most frequently learned were chosen, and 45 different kinds of exercises were generated (unary-operator associativity can be understood as "the operator closest to the operand is executed first"; it poses much fewer problems than binary-operator associativity). Each exercise consisted of 10 questions and was generated 20 times. All concepts were allowed to avoid getting into a situation where the system could not find questions for some rules because all the relevant operators belonged to denied concepts.

During this experiment, 9000 questions were retrieved from the bank to create exercises. Among them, there were 3362 unique questions. The mean of the number of times a question was retrieved was 2.68. While the maximum number of retrievals of the same question was 98, 75% of questions were retrieved two or fewer times, and more than 50% of questions were retrieved only once. The most often used question (98 times) had a combination of enclosing operators and binary operator sequences with both left and right associativity affecting the answer, which is a very rare combination.

Figure 1 shows the distribution of relative frequencies of different target rules in the generated exercises. It can be seen that this parameter never fell below the required minimum of one, even for the rules concerning binary operators with the right associativity and strict order of operands that had very few questions. Figure 2 shows the distributions of the number of subject-domain rules in questions retrieved for exercises with the same difficulty but different numbers of target concepts. The average was 2–3 rules per question; uniform exercises (only one target rule) received slightly easier questions because the proposed algorithm did not try to satisfy complex requests at first, while exercises with big numbers of target concepts ($N = 5$) generated slightly more complex questions. Figure 3

shows that the average question difficulty increased slightly when the number of target subject-domain rules increased. The proposed algorithm performed relatively well even with a small question bank, allowing the teacher to set question difficulty separately from the number of target rules.

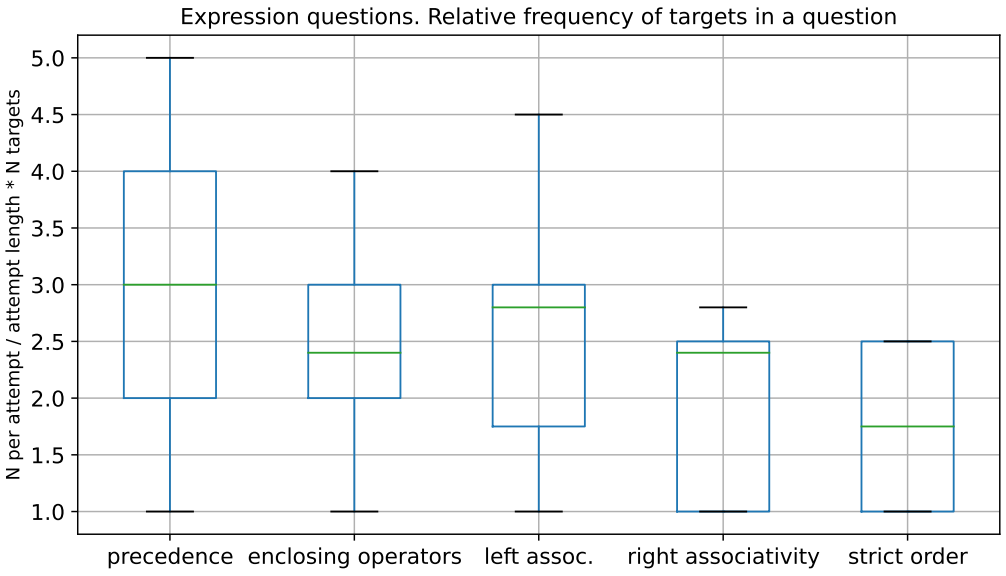

Figure 1. Relative frequencies of different rules in the exercises generated using the expressions bank.

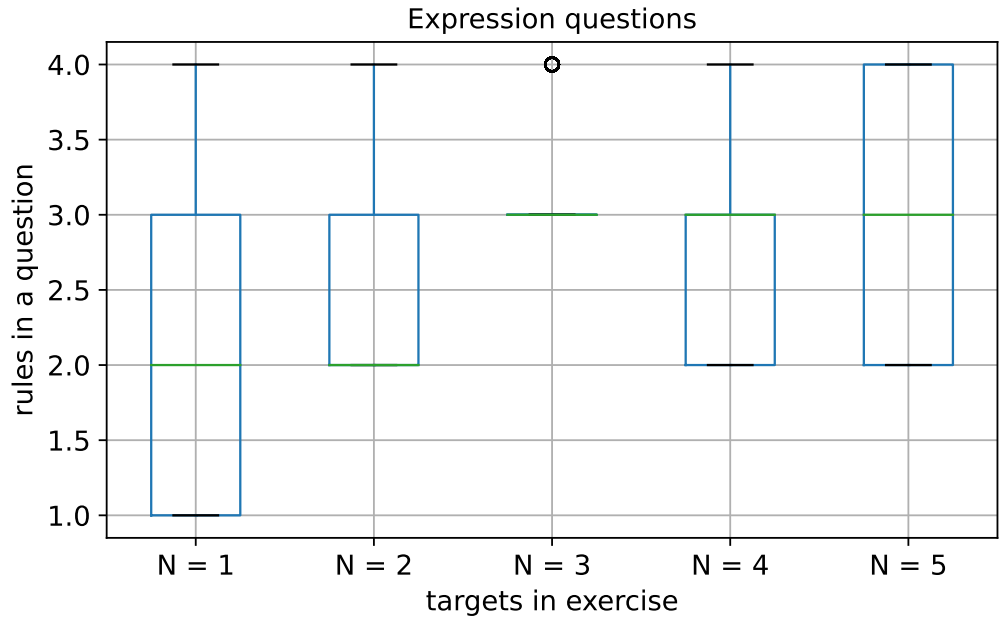

Figure 2. Distribution of the number of subject-domain rules per questions for exercises with different numbers of target rules and the same difficulty. The X axis is the number of target rules in the exercise; the Y axis shows the number of rules in questions.

Figures 4–6 show the average question difficulties, numbers of rules per question, and numbers of solution steps per question depending on the desired difficulty of exercises. Table 3 shows the target rules in different rule groups. The algorithm did not behave ideally for groups #1, #5, and #2, giving a slight decrease in the question difficulty, number of rules per question, and solution steps when the exercise difficulty increased from 0.1 to 0.3. This can be caused by the relatively small size of the question bank and requires further improvement of the algorithm. However, in most situations, the difficulty of included questions increased (or remained the same) when the requested exercise difficulty increased.

The examples of exercises for low, medium, and high difficulty are shown in Appendix B. It can be seen that the low difficulty setting made the algorithm choose the simplest expressions still containing target concepts; medium difficulty settings produced more complex questions and the high-difficulty exercise contained big expressions with a significant number of distracting parentheses and operators.

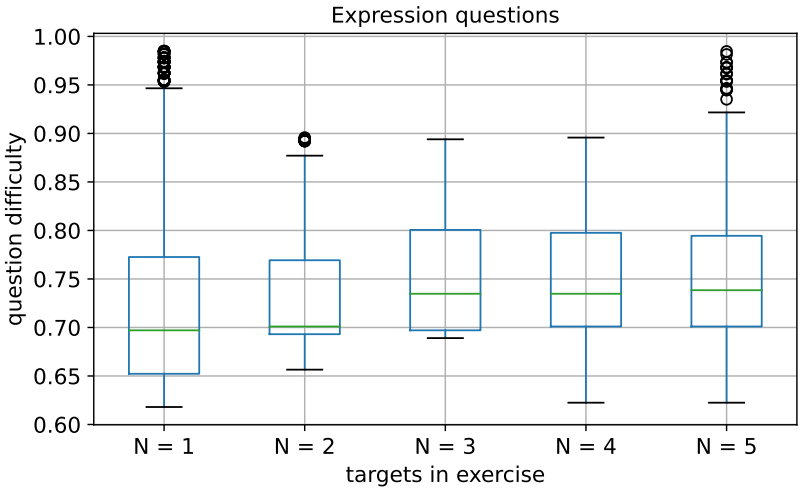

Figure 3. Distribution of the average question difficulty for exercises with different numbers of target rules and the same difficulty. The X axis is the number of target rules in the exercise; the Y axis shows the average difficulties of questions in generated exercises.

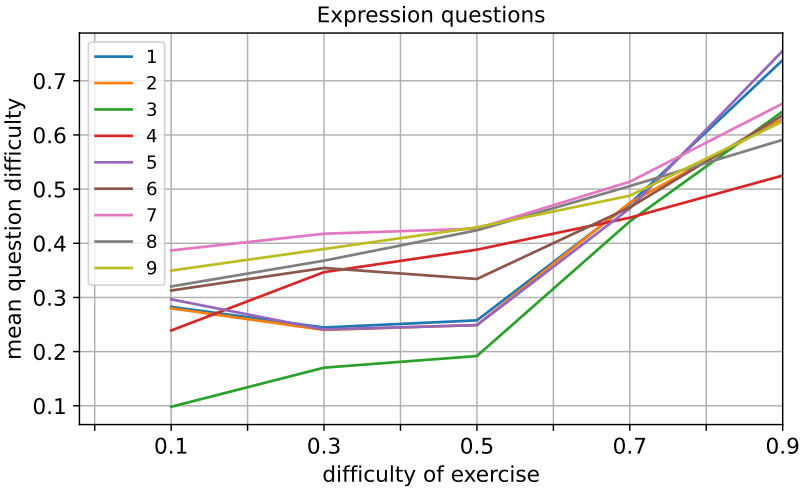

Figure 4. Average question difficulty (Y axis) by exercise difficulty (X axis).

Table 3. Groups of target subject-domain rules for exercise generation with varied difficulty (exercise questions).

Rule Group #	Target Concepts
1	prec
2	encl-ops
3	left-assoc
4	right-assoc
5	strict-order
6	encl-ops prec
7	encl-ops prec left-assoc
8	encl-ops prec left-assoc rightassoc
9	encl-ops prec left-assoc right-assoc strict-order

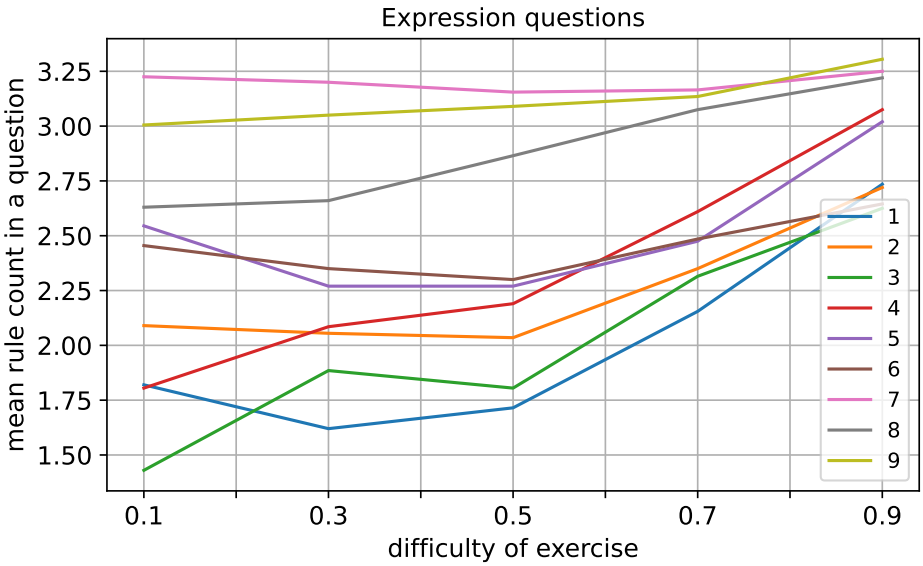

Figure 5. Average number of subject-domain rules per learning question (Y axis) by exercise difficulty (X axis).

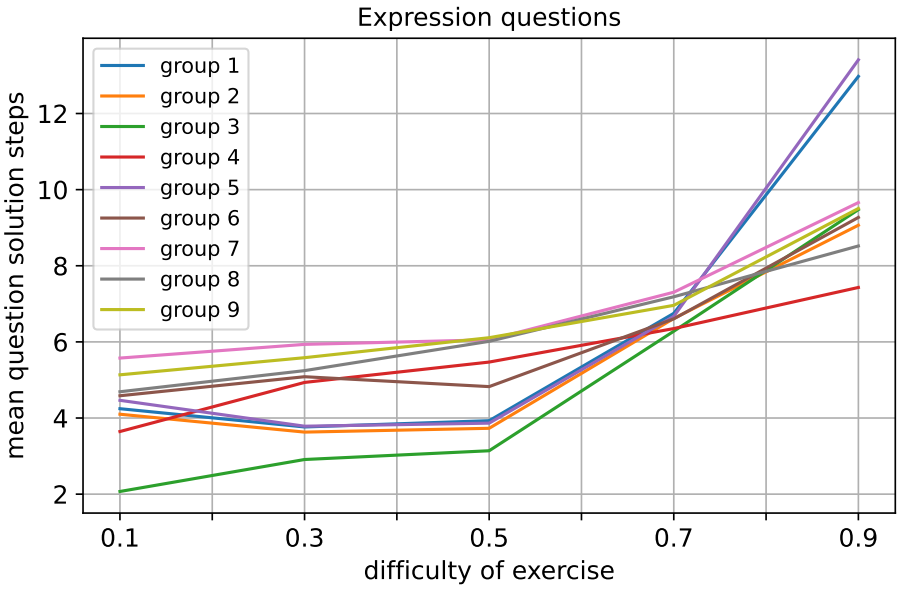

Figure 6. Average number of solution steps per question (Y axis) by exercise difficulty (X axis).

4.3. Ablation Study

In order to verify that all the elements of the presented algorithm were necessary, an ablation study was conducted by turning off essential parts of the algorithm. Three essential components of the algorithm were identified:

1. QR_ADJ—preprocessing of question requests to balance used rules (concepts);
2. SORT_BY_USAGE—taking into account how often a question was used in that exercise before;
3. POST_FILTER—sorting of the found questions by undesirable concepts (rules) and difference to the target question difficulty.

We performed mass exercise generation for the baseline simple random choice among the suitable questions (i.e., the questions of required difficulty without denied concepts and with at least one target concept) which was labeled BASE, the proposed algorithm with all key components turned off (ALL_OFF), the proposed algorithm with one of the components

turned off, and the full version of the proposed algorithm (ALL_ON). Each time, exercises using 9000 questions were generated with different settings. Table 4 shows the number of unique questions in the generated exercises and the statistics on question usage. Figure 7 shows box plots for the relative frequencies of rule usage in different settings.

Table 4. Ablation study of the proposed algorithm: question usage.

Parameter	BASE	ALL_OFF	QR_ADJ Off	SORT_BY_USAGE Off	POST_FILTER Off	ALL_ON
Number of used questions	1530	2964	778	2980	835	3362
Mean number of times a question was used	5.9	3.0	11.6	3.0	10.8	2.7
Maximum number of times a question was used	61	81	142	127	128	98
Maximum number of times a question was used for 50% of questions	4	1	5	1	4	1
Maximum number of times a question was used for 75% of questions	6	3	14	3	11	2

As it can be seen, the naive random question selection (BASE) produced exercises with a relatively low variety of questions (1530 unique questions per 9000 questions in the generated exercises; the median number of times a question was used was 4). Its box plots showed the minimum relative frequency of four rules as zero (which means that there were exercises where these rules were target, but no question on them was included in the attempt, which is unacceptable in practice). For two of them, the lower quartile was zero, and for the "right-associativity" rule even the median was zero (i.e., more than half of exercises targeting the "right-associativity" rule did not have relevant questions).

Introducing the new algorithm significantly increased the number of used questions (lowering question reuse), but the lower quartile for the "right-associativity" rule remained zero. The QR_ADJ component of the algorithm played a chief part in satisfying the condition "every target rule must appear at least once in each exercise", as turning it off brought the "right-associativity" rule to zero. Turning off either QR_ADJ or POST_FILTER resulted in a sharp increase in question re-usage, as the algorithm tried to fulfill the requirements using the same best-fitting questions, which significantly increased the risk of cheating by sharing answers. SORT_BY_USAGE seemed to be the least important component, but turning it on still increased the number of unique questions used from 2980 to 3362 and decreased the maximum usage of one question by almost 25%. Turning SORT_BY_USAGE on also increased the median of occurrence of the less frequent rules, making learners study them more thoroughly.

Thus, the ablation study showed that each component had its role in the performance of the entire algorithm. The proposed algorithm significantly outperformed the baseline random question selection; the baseline algorithm produced practically unacceptable results with unbalanced question banks, totally omitting target rules with low frequency in more than half of the generated exercises.

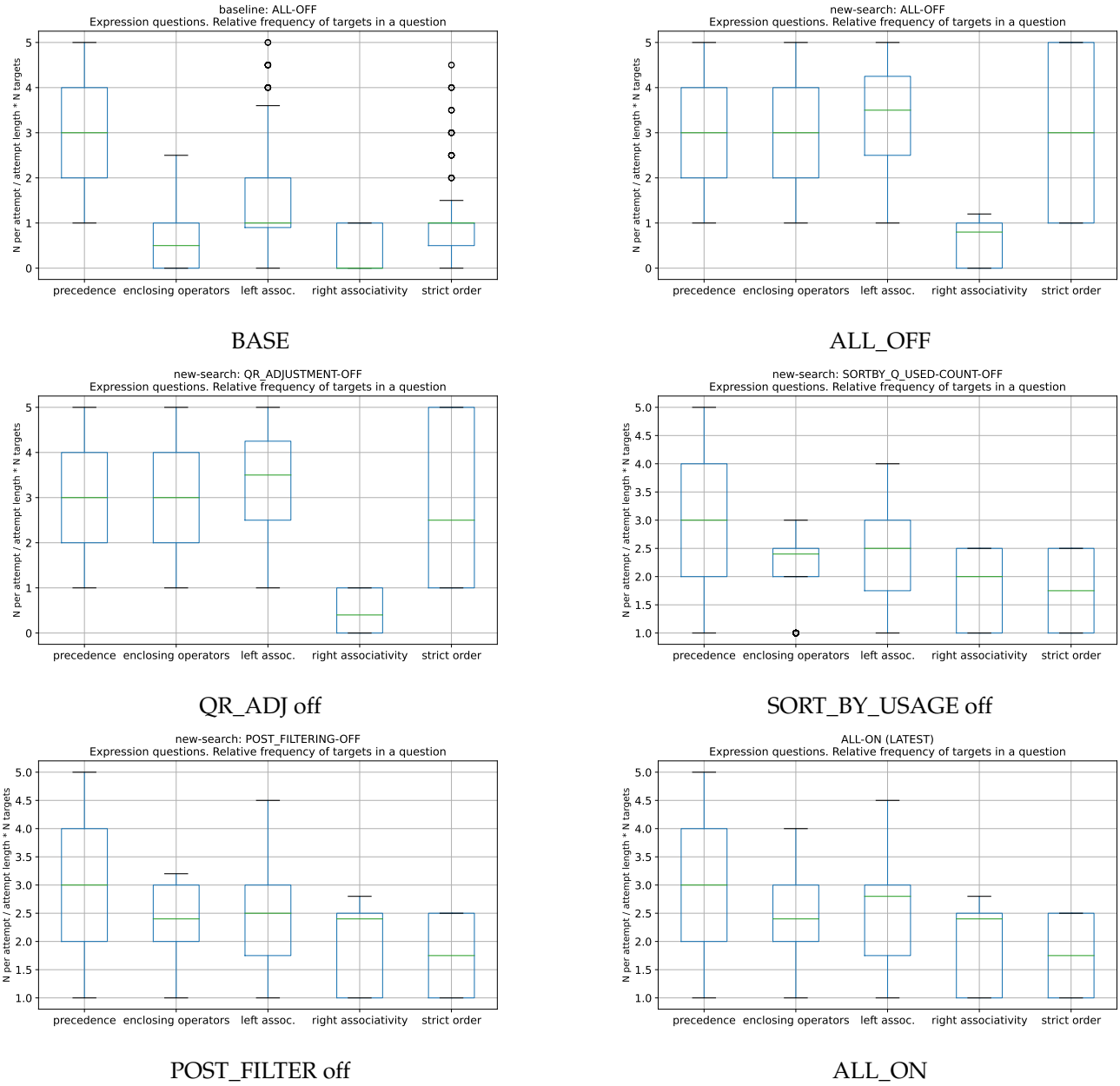

Figure 7. Box plots for relative frequencies of rule usage during the ablation study.

5. Discussion

Large-scale automatic question and learning-problem generation allows us to overcome traditional limitations of human-authored exercises and assignments: it creates a lot of learning items that can be used for unlimited practice until mastery (many questions for one student), avoiding cheating (different questions for different students), and adapting to the needs of a particular student (many questions for different learning situations). Big banks of questions and learning problems that are generated randomly (e.g., [28]) or mined from real-world data (e.g., [39]) can solve that problem because they provide not only vast numbers of generated questions but also significant variability, unlike template-based generation that increases only the number of questions but not the coverage of different topics (e.g., [32]). However, as these big banks come without any classification of questions according to the learning goals and are too big to be observable by a human teacher, they need special algorithms for constructing exercises depending on the teacher's request.

In this study, the teacher's request was formalized in terms of determining the three sets of concepts and subject-domain rules: target (which should be studied and verified), denied (concepts and rules that students are not expected to know at this point), and allowed (known, but not necessary for this exercise) and setting the desired difficulty. The task of creating a balanced exercise based on the request was formalized, and the algorithm for solving it was proposed.

According to the results of the computational experiment, the proposed algorithm guarantees that in an exercise with N_t target concepts and rules consisting of N_q questions, each target concept (rule) appears in questions at least N_q / N_t times, which guarantees the generation of a balanced exercise each time by spreading the learning targets between different questions in the exercise. In practice, the number of target concepts and rules often exceeds the required minimum because questions often concern more than one concept (rule). The number of used rules in retrieved questions depends on the exercise difficulty instead of the number of target concepts in the exercise. It shows that for easier exercises, when it is useful to have fewer concepts per question, the proposed algorithm can rotate target concepts, diversifying questions without increasing their difficulty if the relevant questions are present in the question bank. This shows that the proposed algorithm discerns between the exercise topic (defined by sets of allowed and denied concepts and rules) and exercise difficulty and can find diverse and relevant questions in the given bank.

The algorithm has a good randomization of retrieved questions: even with many exercises with similar settings and generating each exercise 20 times, the mean number of times the same question was used stood below three, and it was mostly achieved by a high usage of a small number of rare questions: 50% of questions were used only once, and 75% of questions were used no more than 2–3 times. This was achieved by taking into account question usage and active usage of questions that were not retrieved yet.

With smaller, less diverse banks, the algorithm sometimes was forced to choose between requested difficulty and target concepts and rules. In that case, the algorithm chose to keep the requested learning topic by deviating from the necessary difficulty. More research may be necessary to make the results fully stable on smaller sets of questions.

The described features of the proposed algorithm allow us to recommend it for the automatic construction of formative (designed for training) and summative (designed for grading) assignments from big, automatically generated question banks. It can also be used for the cold start in adaptive learning systems until enough data for training adaptive algorithms can be collected and for searching diverse questions using requests generated by adaptive systems.

The results of this study improve the results achieved by previous researchers devoted to generating diverse test sheets. The proposed method allows us to replace the hard constraint "no question should be identical in two tests" used by Hwang et al. [56] by the soft constraint "repeat every question as little as possible", which opens the way to a large-scale sheet generation on diverse question banks that cannot support the hard constraint. The algorithm described in this study generated 20 different sheets by re-using rare questions while using more than half of the questions only once. Compared to the results of Sud et al. [11] and Paul [57], the proposed algorithm does not use manual planning of the test sheet and manual marking of the question in the question bank. The proposed algorithm generates tests with flexible composition, which is aimed at a balanced representation of the target concepts, so that the generated test can automatically adapt to changes in difficulty by changing the number of different target concepts per question according to the desired difficulty level. When using the proposed algorithm, the teacher should only specify the requirements for the entire sheet, not the plan of particular question slots as when using previously published algorithms. By only using an automatic classification of questions by the subject-domain concepts and rules whose knowledge is required to answer the questions, the proposed algorithm can be used on far larger question banks consisting of thousands of automatically generated questions that cannot be marked manually.

The limitations of the proposed algorithm are that it does not use the statistics gathered by question usage (e.g., by using the Item Response Theory) unlike, for example, the algorithms proposed in [55,56]. That comes from the algorithm being aimed at large banks of sparsely used questions in which it is impossible to reliably measure item discrimination (and so on) because each item is used very rarely (ideally, only once) which works best for preventing cheating. When such statistics are available, the algorithm should be improved to use it, which can be a good direction of further research.

6. Conclusions

Currently, the methods of generating questions and learning problems are actively researched. The effective usage of the generated questions has received far less attention. This article was intended to close the gap by considering an algorithm of automatic search and retrieval of questions from a large bank and constructing diverse exercises according to teachers' requests.

The article presented an algorithm that can be used to automatically construct complex exercises with desired topics and difficulty. The required metadata for questions in the bank were their estimated difficulty and concepts and subject-domain rules whose knowledge was necessary to answer the question. A question request allowed the teacher to specify the desired difficulty of the exercise, target concepts and rules (those that should be learned), and denied concepts and rules (those that students are not expected to know when doing the exercise). The presented algorithm relied on the metadata in the question bank generated by the question generation engine and can be used in different areas of programming: expression evaluation (see Appendix B), control-flow statements (see the example in the Supplementary Materials), variable scope and lifetime, accessing objects stored in memory, and other topics. The algorithm did not depend on the programming language in which the questions were formulated; in practice, we used tag questions with the programming language so that the questions with the appropriate language are selected according to the teacher's request. Question generation itself was not covered by this algorithm; we used the automatic banks of programming questions generated by the methods described in [9,39]. Those methods provide a good variability in generated questions and generate questions containing natural-looking expressions and algorithms.

The proposed algorithm was based on generating specific question requests for each question in the exercise, taking into account the previously found question, to solve the problem of automatic exercise generation. The algorithm actively used the principle of parameter rotation (keeping track of parameter usage and searching for the less used values) both to rotate target concepts (rules) between the questions (learning problems) in the exercise and to rotate similar questions in different instances of exercises. Rotation helped to achieve more stable results than random selection.

A computational experiment showed that the algorithm satisfied most requirements, including guaranteeing the minimal number of target concepts (rules) in the exercise (so that each question had at least one target and the targets were distributed evenly among them). It did not increase the question difficulty because of the increase in the number of target concepts, increasing the question variety instead. Question difficulty, number of concepts (rules) per question, and solution steps generally increased with exercise difficulty, though the proposed algorithm was not perfect at that. An ablation study showed that all the components of the algorithm improved performance; the most important component for satisfying the strict conditions was the adjustments of question request each time (specific question request generation algorithm).

The algorithm can be used for question retrieval and exercise construction in online learning systems, replacing the traditional "categorization" approach, which is not viable for large question banks with a large number of possible categories. It can also be used in adaptive testing and intelligent tutoring systems because question requests can be generated automatically based on the learner's progress; in that case, the stage of generating specific question requests can be omitted as the request from the adaptive system can

already be treated as a specific request. That algorithm should increase the adoption of large question and problem banks developed during large-scale question and learning problem generation and fill the gap between theory (generating questions) and practice (their usage in the learning process).

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/computers13060144/s1, Data of the attempts generated during the algorithm's evaluation. Three examples of generated exercises for learning control-flow statements with different difficulties. Source code of the described algorithm.

Funding: The reported study has received no funding.

Data Availability Statement: The data presented in this study are available in the Supplementary Materials.

Acknowledgments: I want to express my thanks to Artem Prokudin and especially Mikhail Denisov for their help in producing the question bank that was used for conducting this study.

Conflicts of Interest: The author declares no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ASP	Answer Set Programming
BKT	Bayesian Knowledge Tracing
IRT	Item Response Theory
ITS	Intelligent tutoring system
LMS	Learning Management System

Appendix A. Features of Exercises Generated during the Experiment

Table A1. Features of exercises that were generated for expression questions. A—allowed rule, D—denied rule, T—target rule.

#	Difficulty [0...1]	Precedence	Enclosing	Binary Left Assoc.	Binary Right Assoc.	Strict Order
1	0.1	T	A	A	A	A
2	0.3	T	A	A	A	A
3	0.5	T	A	A	A	A
4	0.7	T	A	A	A	A
5	0.9	T	A	A	A	A
6	0.1	A	T	A	A	A
7	0.3	A	T	A	A	A
8	0.5	A	T	A	A	A
9	0.7	A	T	A	A	A
10	0.9	A	T	A	A	A
11	0.1	A	A	T	A	A
12	0.3	A	A	T	A	A
13	0.5	A	A	T	A	A
14	0.7	A	A	T	A	A
15	0.9	A	A	T	A	A
16	0.1	A	A	A	T	A
17	0.3	A	A	A	T	A
18	0.5	A	A	A	T	A
19	0.7	A	A	A	T	A
20	0.9	A	A	A	T	A
21	0.1	A	A	A	A	T
22	0.3	A	A	A	A	T
23	0.5	A	A	A	A	T
24	0.7	A	A	A	A	T
25	0.9	A	A	A	A	T
26	0.1	T	T	A	A	A
27	0.3	T	T	A	A	A
28	0.5	T	T	A	A	A
29	0.7	T	T	A	A	A
30	0.9	T	T	A	A	A
31	0.1	T	T	T	A	A
32	0.3	T	T	T	A	A

Table A1. *Cont.*

#	Difficulty [0…1]	Precedence	Enclosing	Binary Left Assoc.	Binary Right Assoc.	Strict Order
33	0.5	T	T	T	A	A
34	0.7	T	T	T	A	A
35	0.9	T	T	T	A	A
36	0.1	T	T	T	T	A
37	0.3	T	T	T	T	A
38	0.5	T	T	T	T	A
39	0.7	T	T	T	T	A
40	0.9	T	T	T	T	A
41	0.1	T	T	T	T	T
42	0.3	T	T	T	T	T
43	0.5	T	T	T	T	T
44	0.7	T	T	T	T	T
45	0.9	T	T	T	T	T

Appendix B. Examples of Exercises Generated during the Experiment Using Different Difficulty Settings

The following exercises were generated to learn solving the question evaluation order problem with the following target rules: operator precedence, binary left-associative operators, binary right-associative operators, and enclosing operators. The question bank contained questions for those rules with normalized difficulties from 0.53 to 1.0.

Appendix B.1. Low Difficulty

Determine the order of evaluation of the following expressions:

1. sqlite3Strlen30 (pStr2 -> u . zToken) − 1
2. i = j = 0
3. cachedRegex = rkl_lastCachedRegex = (0)
4. nMinKey = nMaxKey = info . nKey
5. pIdx = pProbe = pProbe -> pNext
6. len = i = sqlite3VdbeSerialTypeLen (serial_type)
7. flag_alternateform = flag_altform2 = flag_zeropad = 0
8. rc = db -> errCode = p -> rc
9. * prNotFound = rMayHaveNull = ++ pParse -> nMem
10. pMem -> zMalloc = pMem -> z = z

Appendix B.2. Medium Difficulty

Determine the order of evaluation of the following expressions:

1. (memcpy (& uiwork , (acc + 6) + 34 − 4 , 4) , uiwork) == 0 && (memcpy (& uiwork , (acc + 6) + 34 − 8 , 4) , uiwork) == 0
2. pTabList -> a [0] . iCursor = iCur = pParse -> nTab ++
3. pPager -> sjfd = (pPtr += (((pVfs -> szOsFile) + 7) & 7))
4. flag_leftjustify = flag_plussign = flag_blanksign = flag_alternateform = flag_altform2 = flag_zeropad = 0
5. pMem -> zMalloc = pMem -> z = z
6. u . ad . pMem = p -> pResultSet = & aMem [pOp -> p1]
7. addrBrk = pLevel -> addrBrk = pLevel -> addrNxt = sqlite3VdbeMakeLabel (v)
8. pPage -> cellOffset = cellOffset = hdr + 12 − 4 * pPage -> leaf
9. mxReadMark = pInfo -> aReadMark [i] = pWal -> hdr . mxFrame
10. iCur = pTabList -> a [0] . iCursor = pParse -> nTab ++

Appendix B.3. High Difficulty

Determine the order of evaluation of the following expressions:

1. (as -> src . alpnid == srcalpnid) && Curl_strcasecompare (as -> src . host , srchost) && (as -> src . port == srcport) && (versions & as -> dst . alpnid)

2. lo == 147483648 && hi == 2 && ((((& result) -> words [(16/4) − 1 − (0)]) & 2147483648) != 0)
3. p -> aOp = aOp = u . by . pProgram -> aOp
4. g_use_ansi | | ! isatty (fileno (fp)) | | (− 1) == (stdo = _get_osfhandle (fileno (fp)))
5. p -> iLimit = iLimit = ++ pParse -> nMem
6. * prNotFound = rMayHaveNull = ++ pParse -> nMem
7. u . cb . ctx . pMem = u . cb . pMem = & aMem [pOp -> p3]
8. ((h -> flags [i » 4] » ((i & 15) « 1)) & 1) | | ! ((h -> keys [i]) == (key))
9. g_use_ansi | | ! isatty (fileno (fp)) | | (− 1) == (stdo = _get_osfhandle (fileno (fp))) | | ! GetConsoleScreenBufferInfo (stdo , & csbi)
10. ! ((((df) -> words [(8/4) − 1 − (0)]) & 2080374784) == 2080374784) && DECCOMBMSD [sourhi » 26]

References

1. Baig, M.I.; Shuib, L.; Yadegaridehkordi, E. E-learning adoption in higher education: A review. *Inf. Dev.* **2022**, *38*, 570–588. [CrossRef]
2. Qiao, P.; Zhu, X.; Guo, Y.; Sun, Y.; Qin, C. The Development and Adoption of Online Learning in Pre- and Post-COVID-19: Combination of Technological System Evolution Theory and Unified Theory of Acceptance and Use of Technology. *J. Risk Financ. Manag.* **2021**, *14*, 162. [CrossRef]
3. Mensah, I.K.; Zeng, G.; Luo, C.; Lu, M.; Xiao, Z.W. Exploring the E-Learning Adoption Intentions of College Students Amidst the COVID-19 Epidemic Outbreak in China. *SAGE Open* **2022**, *12*, 21582440221086629. [CrossRef]
4. Coman, C.; Țîru, L.G.; Meseșan-Schmitz, L.; Stanciu, C.; Bularca, M.C. Online Teaching and Learning in Higher Education during the Coronavirus Pandemic: Students' Perspective. *Sustainability* **2020**, *12*, 10367. [CrossRef]
5. Ali, L.; Dmour, N.A.H.H.A. The Shift to Online Assessment Due to COVID-19: An Empirical Study of University Students, Behaviour and Performance, in the Region of UAE. *Int. J. Inf. Educ. Technol.* **2021**, *11*, 220–228. [CrossRef]
6. Mate, K.; Weidenhofer, J. Considerations and strategies for effective online assessment with a focus on the biomedical sciences. *FASEB BioAdvances* **2021**, *4*, 9–21. [CrossRef] [PubMed]
7. Otto, D. Adoption and Diffusion of Open Educational Resources (OER) in Education: A Meta-Analysis of 25 OER-Projects. *Int. Rev. Res. Open Distrib. Learn.* **2019**, *20*, 122–140. [CrossRef]
8. Kurdi, G.; Leo, J.; Parsia, B.; Sattler, U.; Al-Emari, S. A systematic review of automatic question generation for educational purposes. *Int. J. Artif. Intell. Educ.* **2020**, *30*, 121–204. [CrossRef]
9. Sychev, O.; Penskoy, N.; Prokudin, A. Generating Expression Evaluation Learning Problems from Existing Program Code. In Proceedings of the International Conference on Advanced Learning Technologies, ICALT 2022, Bucharest, Romania, 1–4 July 2022; pp. 183–187. [CrossRef]
10. Wang, X.; Fan, S.; Houghton, J.; Wang, L. Towards Process-Oriented, Modular, and Versatile Question Generation that Meets Educational Needs. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Seattle, WA, USA, 10–15 July 2022; pp. 291–302. [CrossRef]
11. Sud, P.; West, M.; Zilles, C. Reducing Difficulty Variance in Randomized Assessments. In Proceedings of the 2019 ASEE Annual Conference & Exposition, Tampa, FL, USA, 16–19 June 2019. [CrossRef]
12. Sychev, O.; Anikin, A.; Penskoy, N.; Denisov, M.; Prokudin, A. CompPrehension-Model-Based Intelligent Tutoring System on Comprehension Level. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; 12677 LNCS; Springer: Berlin/Heidelberg, Germany, 2021; pp. 52–59. [CrossRef]
13. Luxton-Reilly, A.; Simon; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.N.; Ott, L.; Paterson, J.; Scott, M.J.; Sheard, J.; et al. Introductory Programming: A Systematic Literature Review. In Proceedings of the Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018 Companion, New York, NY, USA, 2–4 July 2018; pp. 55–106. [CrossRef]
14. Du, X.; Shao, J.; Cardie, C. Learning to Ask: Neural Question Generation for Reading Comprehension. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, BC, Canada, 30 July–4 August 2017; pp. 1342–1352. [CrossRef]
15. Subramanian, S.; Wang, T.; Yuan, X.; Zhang, S.; Trischler, A.; Bengio, Y. Neural Models for Key Phrase Extraction and Question Generation. In Proceedings of the Workshop on Machine Reading for Question Answering, Melbourne, Australia, 19 July 2018; pp. 78–88. [CrossRef]
16. Qiu, J.; Xiong, D. Generating Highly Relevant Questions. 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), Hong Kong, China, 3–7 November 2019; pp. 5983–5987. [CrossRef]
17. Kim, Y.; Lee, H.; Shin, J.; Jung, K. Improving Neural Question Generation Using Answer Separation. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 6602–6609. [CrossRef]

18. Foucher., S.; Pascual., D.; Richter., O.; Wattenhofer, R. Word2Course: Creating Interactive Courses from as Little as a Keyword. In Proceedings of the 14th International Conference on Computer Supported Education CSEDU, Online, 22–24 April 2022; INSTICC, SciTePress: Setúbal, Portugal, 2022; Volume 1, pp. 105–115. [CrossRef]
19. Chen, F.; Xie, J.; Cai, Y.; Wang, T.; Li, Q. Difficulty-Controllable Visual Question Generation. In *Proceedings of the Web and Big Data: 5th International Joint Conference, APWeb-WAIM 2021, Guangzhou, China, 23–25 August 2021, Proceedings, Part I*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 332–347. [CrossRef]
20. Bi, C.; Wang, S.; Xue, Z.; Chen, S.; Huang, Q. Inferential Visual Question Generation. In Proceedings of the 30th ACM International Conference on Multimedia, MM '22, New York, NY, USA, 10–14 October 2022 ; pp. 4164–4174. [CrossRef]
21. Patil, C.; Patwardhan, M. Visual Question Generation: The State of the Art. *ACM Comput. Surv.* **2020**, *53*, 1–22. [CrossRef]
22. Chen, G.; Yang, J.; Hauff, C.; Houben, G.J. LearningQ: A Large-Scale Dataset for Educational Question Generation. In Proceedings of the International AAAI Conference on Web and Social Media, Washington, DC, USA, 25–28 June 2018; Volume 12. [CrossRef]
23. Leo, J.; Kurdi, G.; Matentzoglu, N.; Parsia, B.; Sattler, U.; Forge, S.; Donato, G.; Dowling, W. Ontology-Based Generation of Medical, Multi-term MCQs. *Int. J. Artif. Intell. Educ.* **2019**, *29*, 145–188. [CrossRef]
24. Kumar, A.N. Solving Code-Tracing Problems and Its Effect on Code-Writing Skills Pertaining to Program Semantics. In Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '15, Vilnius, Lithuania, 4–8 July 2015 ; pp. 314–319. [CrossRef]
25. Rodríguez Rocha, O.; Faron Zucker, C. Automatic Generation of Quizzes from DBpedia According to Educational Standards. In Proceedings of the Companion The Web Conference 2018, Republic and Canton of Geneva, CHE, WWW '18, Lyon, France, 23–27 April 2018; pp. 1035–1041. [CrossRef]
26. Thomas, A.; Stopera, T.; Frank-Bolton, P.; Simha, R. Stochastic Tree-Based Generation of Program-Tracing Practice Questions. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, SIGCSE '19, Minneapolis, MN, USA, 27 February–2 March 2019; pp. 91–97. [CrossRef]
27. Russell, S. Automatically Generated and Graded Program Tracing Quizzes with Feedback. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 2, ITiCSE '21, Virtual, 26 June–1 July 2021; p. 652. [CrossRef]
28. Martin, B.; Mitrovic, A. Automatic Problem Generation in Constraint-Based Tutors. In Proceedings of the 6th International Conference on Intelligent Tutoring Systems, ITS '02, Biarritz, France and San Sebastian, Spain, 2–7 June 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 388–398.
29. O'Rourke, E.; Butler, E.; Tolentino, A.D.; Popović, Z. Automatic Generation of Problems and Explanations for an Intelligent Algebra Tutor. In Proceedings of the 20th International Conference on Artificial Intelligence in Education, Chicago, IL, USA, 25–29 June 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 383–395. [CrossRef]
30. Sychev, O. From Question Generation to Problem Mining and Classification. In Proceedings of the 2022 International Conference on Advanced Learning Technologies (ICALT), Bucharest, Romania, 1–4 July 2022; pp. 304–305. [CrossRef]
31. Sadigh, D.; Seshia, S.A.; Gupta, M. Automating Exercise Generation: A Step towards Meeting the MOOC Challenge for Embedded Systems. In Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education, WESE '12, Tampere, Finland, 12 October 2012. [CrossRef]
32. Shah, H.; Kumar, A.N. A Tutoring System for Parameter Passing in Programming Languages. *SIGCSE Bull.* **2002**, *34*, 170–174. [CrossRef]
33. Hsiao, I.H.; Brusilovsky, P.; Sosnovsky, S. Web-based Parameterized Questions for Object-Oriented Programming. In Proceedings of the E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2008, Las Vegas, NV, USA, 17 November 2008; Bonk, C.J., Lee, M.M., Reynolds, T., Eds.; Association for the Advancement of Computing in Education (AACE): San Diego, CA, USA, 2008; pp. 3728–3735.
34. Ahmed, U.Z.; Gulwani, S.; Karkare, A. Automatically Generating Problems and Solutions for Natural Deduction. In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13, Beijing, China, 3–9 August 2013; AAAI Press: Washington, DC, USA, 2013; pp. 1968–1975.
35. Polozov, O.; O'Rourke, E.; Smith, A.M.; Zettlemoyer, L.; Gulwani, S.; Popovic, Z. Personalized Mathematical Word Problem Generation. In Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, Virtual, 13–15 April 2021; AAAI Press: Washington, DC, USA, 2015; pp. 381–388.
36. Whitehead, J. Spatial Layout of Procedural Dungeons Using Linear Constraints and SMT Solvers. In Proceedings of the International Conference on the Foundations of Digital Games, FDG '20, Bugibba, Malta, 15–18 September 2020. [CrossRef]
37. Smith, A.M.; Andersen, E.; Mateas, M.; Popović, Z. A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game. In Proceedings of the International Conference on the Foundations of Digital Games, FDG '12, Raleigh, NC, USA, 29 May–1 June 2012; pp. 156–163. [CrossRef]
38. Kumar, A.N. Generation of Problems, Answers, Grade, and Feedback—Case Study of a Fully Automated Tutor. *J. Educ. Resour. Comput.* **2005**, *5*, 3–es. [CrossRef]
39. Sychev, O.; Prokudin, A.; Denisov, M. Generation of Code Tracing Problems from Open-Source Code. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Toronto, ON, Canada, 15–18 March 2023; pp. 875–881. [CrossRef]
40. Baker, F.B.; Kim, S.H. *Item Response Theory: Parameter Estimation Techniques*; CRC Press: Boca Raton, FL, USA, 2004.

41. Johns, J.; Mahadevan, S.; Woolf, B. Estimating Student Proficiency Using an Item Response Theory Model. In *Proceedings of the Intelligent Tutoring Systems*; Ikeda, M., Ashley, K.D., Chan, T.W., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 473–480.
42. Corbett, A.T.; Anderson, J.R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Model. User-Adapt. Interact.* **1995**, *4*, 253–278. [CrossRef]
43. Martori, F.; Cuadros, J.; González-Sabaté, L. Studying the Relationship between BKT Fitting Error and the Skill Difficulty Index. In Proceedings of the Sixth International Conference on Learning Analytics & Knowledge, LAK '16, Edinburgh, UK, 25–29 April 2016; pp. 364–368. [CrossRef]
44. Pardos, Z.A.; Heffernan, N.T. Modeling Individualization in a Bayesian Networks Implementation of Knowledge Tracing. In *Proceedings of the User Modeling, Adaptation, and Personalization*; De Bra, P., Kobsa, A., Chin, D., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 255–266.
45. Yudelson, M.V.; Koedinger, K.R.; Gordon, G.J. Individualized Bayesian Knowledge Tracing Models. In *Proceedings of the Artificial Intelligence in Education*; Lane, H.C., Yacef, K., Mostow, J., Pavlik, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; pp. 171–180.
46. Pardos, Z.A.; Heffernan, N.T. KT-IDEM: Introducing Item Difficulty to the Knowledge Tracing Model. In *Proceedings of the User Modeling, Adaption and Personalization*; Konstan, J.A., Conejo, R., Marzo, J.L., Oliver, N., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 243–254.
47. Meng, L.; Zhang, M.; Zhang, W.; Chu, Y. CS-BKT: Introducing item relationship to the Bayesian knowledge tracing model. *Interact. Learn. Environ.* **2021**, *29*, 1393–1403. [CrossRef]
48. Piech, C.; Bassen, J.; Huang, J.; Ganguli, S.; Sahami, M.; Guibas, L.J.; Sohl-Dickstein, J. Deep Knowledge Tracing. In *Proceedings of the Advances in Neural Information Processing Systems*; Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2015; Volume 28.
49. Chavarriaga, O.; Florian-Gaviria, B.; Solarte, O. A Recommender System for Students Based on Social Knowledge and Assessment Data of Competences. In *Proceedings of the Open Learning and Teaching in Educational Communities*; Rensing, C., de Freitas, S., Ley, T.; Muñoz-Merino, P.J., Eds.; Springer: Cham, Switzerland, 2014; pp. 56–69.
50. Wang, S.; Ma, Z.; Ji, H.; Liu, T.; Chen, A.; Zhao, D. Personalized exercise recommendation method based on causal deep learning: Experiments and implications. *STEM Educ.* **2022**, *2*, 157–172. [CrossRef]
51. Troussas, C.; Krouska, A. Path-Based Recommender System for Learning Activities Using Knowledge Graphs. *Information* **2023**, *14*, *9*. [CrossRef]
52. Da Silva, F.L.; Slodkowski, B.K.; da Silva, K.K.A.; Cazella, S.C. A systematic literature review on educational recommender systems for teaching and learning: Research trends, limitations and opportunities. *Educ. Inf. Technol.* **2022**, *28*, 3289–3328. [CrossRef] [PubMed]
53. Hussain, M.; Zhu, W.; Zhang, W.; Abidi, S.M.R.; Ali, S. Using machine learning to predict student difficulties from learning session data. *Artif. Intell. Rev.* **2018**, *52*, 381–407. [CrossRef]
54. Pereira, F.D.; Fonseca, S.C.; Wiktor, S.; Oliveira, D.B.F.; Cristea, A.I.; Benedict, A.; Fallahian, M.; Dorodchi, M.; Carvalho, L.S.G.; Mello, R.F.; et al. Toward Supporting CS1 Instructors and Learners With Fine-Grained Topic Detection in Online Judges. *IEEE Access* **2023**, *11*, 22513–22525. [CrossRef]
55. Hwang, G.J.; Lin, B.M.; Lin, T.L. An effective approach for test-sheet composition with large-scale item banks. *Comput. Educ.* **2006**, *46*, 122–139. [CrossRef]
56. Hwang, G.J.; Chu, H.C.; Yin, P.Y.; Lin, J.Y. An innovative parallel test sheet composition approach to meet multiple assessment criteria for national tests. *Comput. Educ.* **2008**, *51*, 1058–1072. [CrossRef]
57. Paul, D.V. Metaheuristic Algorithms for Designing Optimal Test Blueprint. *Comput. Sist.* **2020**, *24*, 1627–1642. [CrossRef]