Counteracting Exam Cheating by Leveraging Configuration and Recommendation Techniques

Viet-Man Le and **Thi Ngoc Trang Tran** and **Alexander Felfernig** and **Müslüm Atas** and **Lisa Weißl** and **Andrei Popescu** and **Martin Stettinger** and **Seda Polat-Erdeniz** [1]

Abstract. Exam cheating indicates behaviors of students to fraudulently achieve their desired grades through various forms, such as item harvesting, item pre-knowledge, item memorizing, collusion and answer copying, and answer checking from available sources. Such dishonesty behaviors become manifest in e-learning scenarios, where exams are often conducted via online assessment platforms without the physical supervision of proctors. In this paper, we propose an approach to counteract exam cheating based on configuration and recommendation techniques. Our approach allows examiners to configure questions and exams using feature models. We support the configuration of parameterized questions, which helps to generate a large number of exam instances. Besides, a content-based recommendation mechanism is integrated into the exam configuration process, which helps examiners to select questions that have not appeared in the latest exams. We also propose mock-ups to show how question and exam generation processes can be proceeded in a real exam generator system.

1 INTRODUCTION

Cheating refers to a tendency of students to fraudulently achieve their desired grades rather than investing a sufficient amount of time and effort in learning and improving their knowledge [42]. In exam scenarios, cheating behaviors can be shown in different forms, such as *item harvesting, item pre-knowledge, item memorizing, collusion and answer copying,* and *answer checking* [11, 13, 34, 41]. *Item harvesting* occurs when a concerted attempt is made to collect exam questions. Students can do this by memorizing exam content, recording it, or transcribing it. *Item pre-knowledge* occurs when students obtain knowledge of the exam questions and/or answers (e.g., through the Internet or other multi-media sources) prior to the exam. *Item memorizing* occurs when a student answers the questions several times to reach an estimated level ability close to his/her true ability. He/she is assumed to use his/her time only to memorize a fixed number of questions. *Collusion* or *answer copying* denotes a scenario where two or more students work together to complete an exam. This type of cheating is triggered when students sit close to each other and try to copy answers from each other during the exam. The final exam cheating type is *answer checking*, in which students try to check the answers to the questions from available resources.

In *e-learning* scenarios where learning and testing activities are done primarily via web-based platforms [5, 10], the mentioned exam cheating behaviors have become even more intensively, which is, therefore, more challenging to be detected and counteracted compared to traditional learning formats [37]. In this context, looking for effective approaches to avoid exam cheating behaviors has become one of the most critical challenges of education institutions. This action is crucial to assure the integrity of student work and to increase trust in online education systems [10].

While extensive research has been conducted to detect cheating reasons as well as factors affecting students' exam cheating behaviors [9, 11, 15, 16, 30], there exist only a few studies that propose solutions for counteracting or avoiding such dishonesty behaviors. Most of these studies target at preventing exam cheating in *online exams* (i.e., *exams* conducted via Internet-based platforms) [37]. Alessio et al. [2] and Dendir and Maxwell [14] proposed approaches to prevent exam cheating using a proctoring software that activates the camera on a computer and then records the exam of students. This software allows examiners to observe the behaviors of students and thereby detect their cheating behaviors. It also helps to prevent students from talking to each other or looking up relevant information in books or other sources. Although this approach helps to mitigate academic dishonesty behaviors in online exams, it could raise privacy issues. Another problem is related to the efficiency of the approach, especially in the context of big courses where exams are conducted with hundreds of students at the same time. Detecting exam cheating of a large number of students by just analyzing students' recorded videos might be a sub-optimal solution since it would consume too much effort of examiners or proctors.

A more efficient approach is to randomize exam questions and answers, which has been widely applied in *Learning Management Systems* such as *WebCT* and *Blackboard*[2]. This approach allows examiners to prepare randomized questions in such a way that no two exams are alike [31]. Besides, in order to increase the probability of generating different exam instances, this approach requires a large question bank that consists of a large number of questions and answers [29]. Additionally, paraphrasing techniques might be needed to reformulate questions that have been selected from the question bank. Golden and Kohlbeck [22] show that paraphrasing questions selected from a question bank is, on the one hand, essential for reducing the benefits of students from cheating in online exams. On the other hand, this helps to increase the performance of students in completing the exam.

Inspired by the ideas discussed by Mccabe [29] and Golden and Kohlbeck [22], we propose in this paper an approach to counteract exam cheating behaviors by generating a large question bank, in which questions and corresponding answers are generated automatically. Our approach supports the generation of different instances

[1] Graz University of Technology, Graz, Austria. Emails: {vietman.le, ttrang, alexander.felfernig, muesluem.atas, andrei.popescu, martin.stettinger, spolater}@ist.tugraz.at, and lisa.weissl@student.tugraz.at

[2] https://www.blackboard.com

of a question topic. For instance, we could create two instances for a question topic regarding *"minimal conflict sets"* using *equivalent terms*. The two instances could be (1) *"What is a minimal conflict set?"* and (2) *"What is a minimal unsatisfiable subset?"*.

In order to support this, our approach enables question configuration mechanisms using *feature models* - one of the core technologies of configuration systems [24]. In the context of exams and questions modeling, where examiners are not always good at technology, feature models might be an appropriate choice. The reason is that the representation of feature models is straightforward and does not require any special expertise of the examiner to create them [6]. Furthermore, a feature model utilizes a *tree-based representation* that provides a good overview of knowledge structure as well as facilitates feature model management [6, 20, 27]. These are the advantages of feature models, which motivate us to leverage them in our approach to exam and question configuration.

Besides, we also encourage the configuration of parameterized questions, in which each question is configured using relationships or constraints defined in the corresponding feature model. With specific question settings, all *instances* are generated. Each instance represents a complete question with a *question statement, correct answers,* and *incorrect answers* (see also *Figure 3*). This way, our approach helps to significantly increase the solution space of questions and, therefore, increase the question bank's size. After the question generation phase, an exam configuration process is activated, which allows an examiner to configure a set of exams by selecting questions that have been generated. The question selection can be made based on constraints specified by an examiner, such as *total number of exam instances, number of questions in each exam instance, duration, the similarity with previous exams*, and *the share of different question types in each exam instance*. Furthermore, question and exam configuration processes are further supported by a *recommendation mechanism* that helps to generate exams that are different from previous exams as much as possible.

The contributions of our work are therefore two-fold:

1. We propose an exam creation approach supporting question and answer parameterization, which significantly increases the solution space and automatically generates many exam instances. This way, each student will receive a different exam, which therefore helps effectively counteract cheating behaviors, especially in exams for big courses.
2. We develop mock-ups of a real exam creator system to support the mentioned approach.

The remainder of the paper is organized as follows. In *Section 2*, we provide basic knowledge regarding feature models, feature model configuration, and recommendation techniques. *Section 3* and *Section 4* are the main parts of our work, in which we present how configuration and recommendation techniques are exploited in our approach to generate exams. Finally, we conclude the paper and discuss open issues for future work in *Section 5*.

2 PRELIMINARIES

2.1 Feature Models

Feature models are used to specify the variability and commonality of complex items, such as software artifacts, configurable products, and highly-variant services [3, 6, 26]. Applications based on feature models help users to decide which features should be included in a specific configuration.

A feature model is a hierarchical representation of a set of *features* and their *interrelationships* [6, 26]. In such a representation, features are represented by *nodes*, and relationships between features are represented by *links*. The root of the feature model is a so-called *root feature* (f_r), which is involved in every configuration ($f_r = true$).

A feature model can be exploited in exam scenarios to represent a set of questions for an exam that share common features. For instance, given a set of two multiple-choice questions Q_1 and Q_2 shown in *Table 1*, a corresponding feature model representing these questions is depicted in *Figure 1*.

Table 1: An example of two multiple-choice questions that share common features.

Q_1	**What is a minimal diagnosis?** 1. an arbitrary subset 2. a minimal deletion subset ***(correct answer)*** 3. a minimal unsatisfiable subset 4. a maximal subset
Q_2	**What is the definition of a minimal conflict set?** 1. an arbitrary subset 2. a minimal deletion subset 3. a minimal unsatisfiable subset ***(correct answer)*** 4. a maximal subset

Feature models can be distinguished with regard to the used knowledge representation [6]. In this section, we present three well-known feature models (*basic feature models* [26], *cardinality-based feature models* [12], and *extended feature models* [4]), where their notations are used in our approach.

Basic Feature Models. A *basic feature model* [26, 27] consists of two parts: *structural part* and *constraint part*. The former establishes a hierarchical relationship between features. The latter combines additional constraints that represent so-called *cross-tree constraints*.

Structurally, the relationship between a feature and its sub-features can be typically classified as follows: *mandatory, optional, alternative,* and *or*. A *mandatory relationship* indicates that a child feature will be included in a configuration *if and only if* its parent feature is included in the configuration (e.g., see the relationship between f_1 and f_3). An *optional relationship* denotes the fact that the inclusion of a child feature is optional if the parent feature is included (e.g., see the relationship between f_1 and f_4). An *alternative relationship* describes the fact that exactly one child feature has to be included if the parent feature has been included (e.g., see the relationships between f_8 and its child features f_{10} and f_{11}). Finally, an *or relationship* indicates that at least one of the child features should be included if the parent feature has been included (e.g., see the relationships between f_9 and its child features $f_{12}..f_{15}$).

In the *constraint part*, *cross-tree constraints* are integrated graphically into the model to set cross-hierarchical restrictions for features. There are two constraint types, *requires* and *excludes*, that can be used for the specification of feature models [6]. A *requires* constraint shows that if one feature is included in the configuration, then another feature must be included as well (e.g., f_5 *requires* f_{10}). An *excludes* constraint denotes that two certain features must not be included in the same configuration (e.g., f_6 *excludes* f_{12}).

Cardinality-based Feature Models. *Cardinality-based feature models* [12] extend the *basic* ones to allow cardinalities with an upper bound > 1 of feature relationships. These feature models support two new relationships: *feature cardinality* and *group cardinality*. *Feature cardinality* is a sequence of intervals denoted $[n..m]$ (n - lower bound, m - upper bound), determining the number of instances of the feature that can be part of a product. *Group cardinality* is an

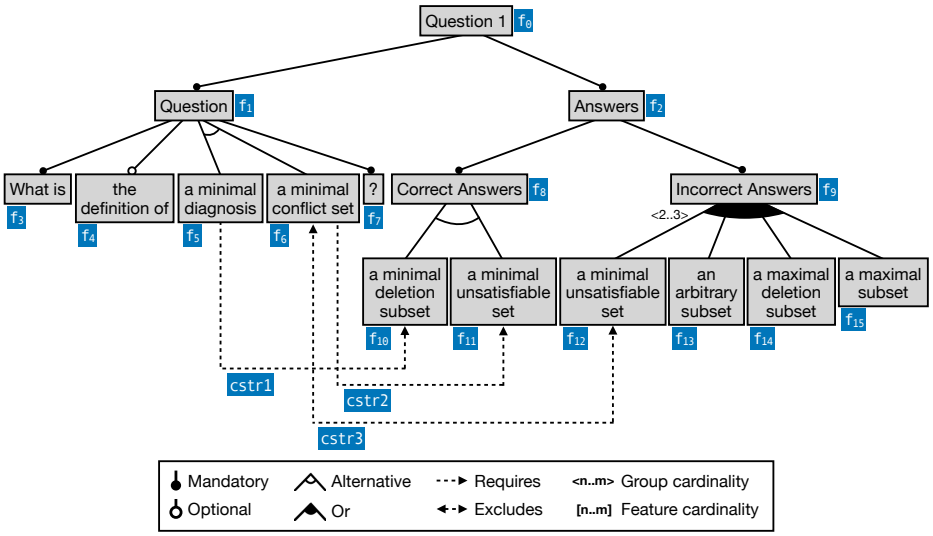


Figure 1: Feature model for a set of multiple-choice questions.

interval denoted $\langle n..m \rangle$ (n - lower bound, m - upper bound), limiting the number of child features that can be part of a product when its parent feature is selected. For instance, the group cardinality $\langle 2..3 \rangle$ between feature f_9 and its child features $f_{12}..f_{15}$ indicates that a configuration for `Question 1` has minimum *two* and maximum *three* incorrect answers.

Extended Feature Models. *Extended feature models* [4] support the description of features with *attributes*. For instance, in an exam feature model, each question is described by two attributes: *question complexity* and *question type*. These feature models can also include *complex* constraints among attributes and features. One example constraint can be: "If the *question complexity* of `Question 1` is *'important to know'*, then this question should be included in the exam".

2.2 Feature Model Configuration

For the discussions in the later sections, we introduce the definitions of a *feature model configuration task* and a *feature model configuration* (solution) [17, 24]. A feature model configuration task can be defined as a constraint satisfaction problem (CSP) [40].

Definition 1 (**Feature model configuration task**) A feature configuration task is defined by a triple $(\mathbf{F}, \mathbf{D}, \mathbf{C})$, where $\mathbf{F} = \{f_1, f_2, ..., f_n\}$ is a set of features, $\mathbf{D} = \{dom(f_1), dom(f_2), ..., dom(f_n)\}$ is a set of feature domains, and $\mathbf{C} = CF \cup CR$ is a set of constraints restricting possible configurations, $CF = \{c_1, c_2, ..., c_k\}$ represents a set of feature model constraints, and $CR = \{c_{k+1}, c_{k+2}, ..., c_m\}$ represents a set of user requirements.

Definition 2 (**Feature model configuration**) A feature model configuration $\mathbf{S}$ for a given feature model configuration task (F, D, C) is an *assignment* of the features $f_i \in F, \forall i \in [1..n]$. S is **valid** if it is *complete* (i.e., each feature in F has a value) and *consistent* (i.e., S fulfills the constraints in C).

2.3 Recommendation

Recommendation techniques have been employed in various domains such as *movies, music, books, tourism destinations, financial services*, and *healthcare* to recommend products/services that meet users' needs and preferences [8, 18, 28, 38, 39, 43]. More recently, recommendation techniques have also been applied in the *e-learning domain* to support learners in choosing courses, resources, or learning materials [43]. Besides, these techniques can also be exploited to support teachers/lecturers/instructors for generating exams [23].

There exist three well-known recommendation approaches that have been extensively studied in the recommender systems research: *collaborative filtering, content-based,* and *knowledge-based* [36]. Each of these approaches has its own characteristics and suitable application scenarios. *Content-based recommendation* builds a user's profile based on his/her past preferences and recommends items that are similar to his/her profile. This approach is suitable for recommending items with abundant content information such as documents or webpages [1]. *Collaborative filtering* suggests a specific item to a user based on the preferences of similar users. This approach is widely used and well-known through the Netflix competition [7]. *Knowledge-based* approaches are usually applied to generate recommendations in domains where the quantity of available item ratings is quite limited (such as *cars*, *apartments*, and *financial services*) or when the user wants to explicitly define his/her requirements for items (e.g., *"the apartment should be close to working area"*). These approaches generate recommendations based on *the knowledge about the items*, *explicit user preferences*, and *a set of constraints* describing the dependencies between users' preferences and items' properties [18].

In this study, we select *content-based recommendation* to be integrated into our approach since the items in our recommendation scenario are exams and questions that are mostly represented in text forms. Our recommendation approach helps to filter exams with a low number of questions that have been used in previous exams (see further details in *Section 4*).

3 QUESTION AND EXAM CONFIGURATION

3.1 Configuring Questions using Feature Models

In this section, we present our approach to model a set of questions using feature models. Although our approach is illustrated by *multiple-choice questions*, it is also applicable to other question types, such as *matching*, *drag-drop*, *reordering*, and *freetext* (i.e., questions whose answers can be entered by students using free texts).

Example question configuration scenario. Assume an examiner wants to create a feature model that represents two multiple-choice questions Q_1 and Q_2 as shown in *Table 1*. For the purpose of generating further instances that are different from Q_1 and Q_2, the examiner sets the minimum and the maximum number of correct/incorrect answers. The number of correct answers to each question is exactly 1 (i.e., $min = max = 1$), the number of incorrect answers stays in the range of [2..3]. In the following, we analyze Q_1 and Q_2, which is the basic to construct the feature model of these two questions:

- The phrases *"What is"* and *"?"* are located at the same relative positions and obligatory parts of the questions. Therefore, they are referred to as *mandatory* phrases.
- The phrase *"the definition of"* appears only in question Q_2, and this question will not change its meaning without this phrase. Hence, this phrase can be referred to as an *optional* phrase.
- The phrases *"a minimal diagnosis"* and *"a minimal conflict set"* can be replaced with each other, they are therefore referred to as *alternative* phrases.
- There are the same incorrect answers such as *"an arbitrary subset"* and *"a maximal subset"*, which are referred to as *or* phrases.
- The correct answers (*"a minimal deletion subset"* and *"a minimal unsatisfiable subset"*) are chosen depending on which phrase (*"a minimal diagnosis"* or *"a minimal conflict set"*) has been selected to tailor the question. If *"a minimal diagnosis"* is selected, then the correct answer should be *"a minimal deletion subset"* (Q_1). If *"a minimal conflict set"* is selected, then the correct answer should be *"a minimal unsatisfiable subset"* (Q_2). These show the *requires* relationships between the mentioned phrases.

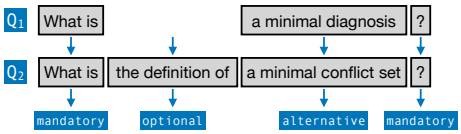


Figure 2: A tokenization for questions Q_1 and Q_2, showing how to identify the relationship of tokens between the questions.

Question feature model. Based on the above analysis, a corresponding feature model that specifies the variability and the commonality of Q_1, Q_2, and all other instances can be generated (see *Figure 1*). The feature model shows two *mandatory* sub-features of the *root feature*, referring to two main parts of a question `Question` - f_1 and `Answers` - f_2. The statement of a question is now modelled based on the sub-features of f_1. The answers of a question are modelled based on the sub-features of f_2.[3]

The feature `Question` has *five* sub-features $f_3..f_7$, where f_3 and f_7 are *mandatory* features, f_4 is an *optional* feature, and f_5 and f_6 are *alternative* features. In the branch of the feature `Answers`, two sub-features f_8 and f_9 have to be added to distinguish between *correct* and *incorrect* answers[4]. The feature `Correct Answers` connects to its sub-features f_{10} and f_{11} using an *alternative* relationship since there is only *one* correct answer to the question. This relationship could be replaced with an *or* relationship with a *group cardinality* if the examiner has specified the maximum number of correct answers is greater than 1. Since the number of incorrect answers stays in the range of [2..3], a group cardinality $\langle 2..3 \rangle$ between the feature `Incorrect Answers` and its sub-features $f_{12}..f_{15}$ is needed. Two cross-tree constraints $\{cstr1$: f_5 *requires* $f_{10}\}$ and $\{cstr2$: f_6 *requires* $f_{11}\}$ should be defined to identify the correct answers. The constraint $\{cstr3$: f_6 *excludes* $f_{12}\}$ indicates that if f_6 is selected then f_{12} cannot be an incorrect answer.

[3] In the context of *free-text questions*, the feature `Answers` does not have sub-features since no answers should be pre-specified (i.e., for a free-text/open question, the answer is entered by students).

[4] Another way is to create direct connections from f_2 to correct and incorrect answers without splitting them into two branches.

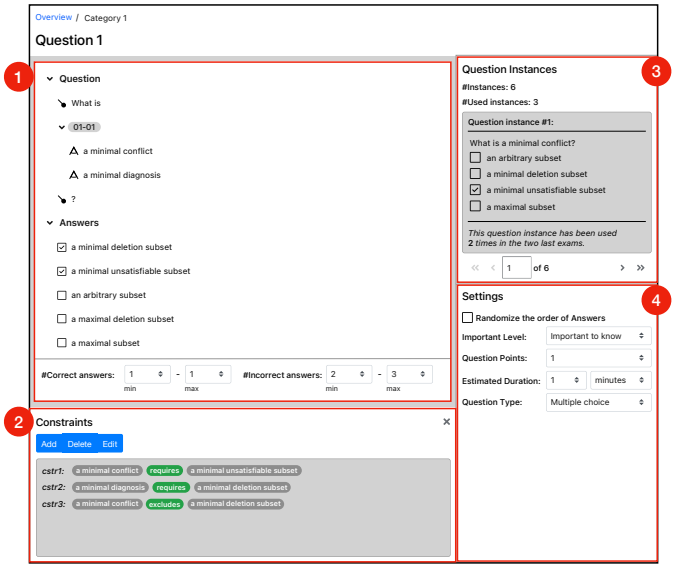


Figure 3: Mock-up for question configuration, consisting of four parts: *Part 1* - Question & Answers Editor, *Part 2* - Constraint Editor, *Part 3* - Question Instances, and *Part 4* - Question-Attribute Settings. The content in *Part 1 & 2* is related to the feature model depicted in *Figure 1*.

Supported tool for question configuration. The question configuration process of an examiner can be supported by an *envisioned exam generator tool*. In the following, we propose a mock-up showing how such a configuration is proceeded.

Figure 3 shows the mock-up for configuring a set of multiple-choice questions, which consists of the following parts:

- *Part 1 - Question & Answers Editor*: The editor represents the structural part of a feature model in a tree view control, where each feature is represented by a node in the tree. The sub-tree `Question` shows phrases used to tailor the question statement. The sub-tree `Answer` shows *correct* answers (with ☑) and *incorrect* answers[5]. At the bottom of this part, the editor asks an examiner to enter the number of correct/incorrect answers to the question.
- *Part 2 - Constraint Editor*: The editor allows an examiner to *add, edit,* or *delete* constraints used in the feature model. When clicking on the *"Add"* or *"Edit"* button, the Constraint Editor dialog is shown to let the examiner create constraints (see *Figure 5*). Besides, when defining a constraint, an *inconsistency detection mechanism* is activated to identify constraints triggering inconsistencies [21, 35]. The identified constraints are highlighted to inform the examiner that these constraints should be adapted for resolving inconsistencies.

[5] In the *Question & Answers Editor*, the correct and incorrect answers are not separated into two branches as shown in the feature model (see *Figure 1*). The reason is to visualize answers in the traditional form of a multiple-choice question.

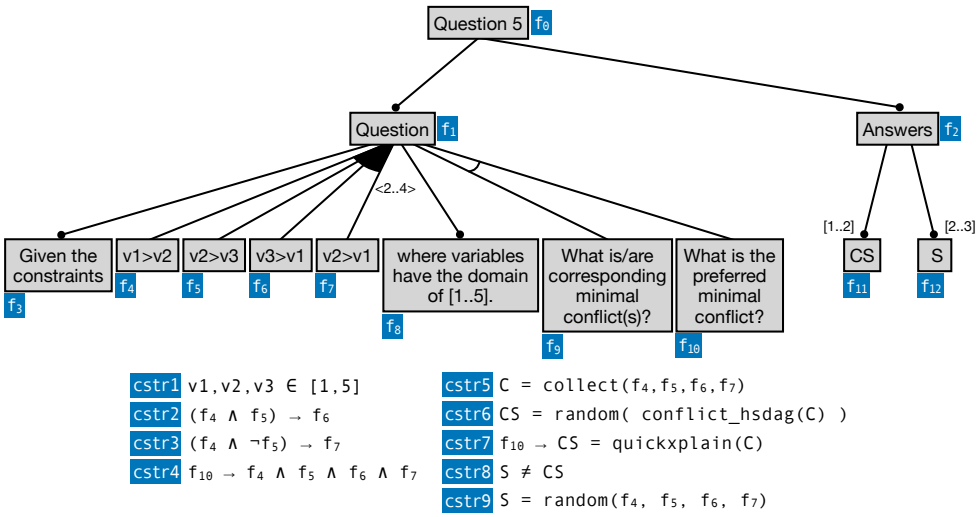


Figure 4: The feature model for a set of parameterized questions related to the identification of minimal conflicts or the preferred minimal conflict from a set of constraints.

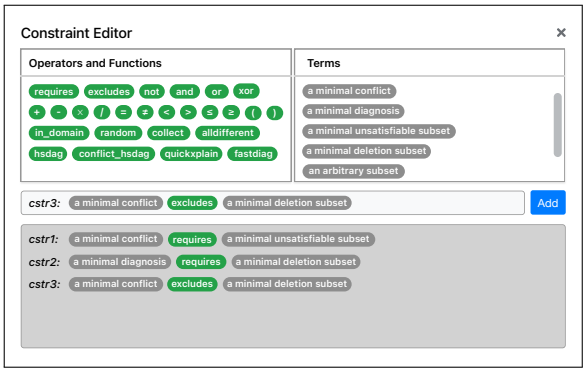


Figure 5: Mock-up for the Constraint Editor dialog.

- *Part 3 - Question Instances*: An examiner is able to see the number of *question instances* generated based on the feature model and constraints defined in *Parts 1 & 2*. In our example, *six* question instances have been generated (*"#Instances: 6"*). The examiner can browse through all instances by using the pagination control. For each instance, a *recommendation mechanism* is activated to specify how often the instance has been used in previous exams (e.g., *"This question instance has been used twice in the last two exams"*). Besides, the system calculates the number of instances used in previous exams. For example, *"#Used instances: 3"* means *three* out of *six* instances have been used in previous exams. For further details of the recommendation mechanism, see *Section 4*.
- *Part 4 - Question-Attribute Settings*: This part allows an examiner to set the attributes of a question, such as *answer randomizing*, *important level*, *question points*, *estimated duration* and *question type*.

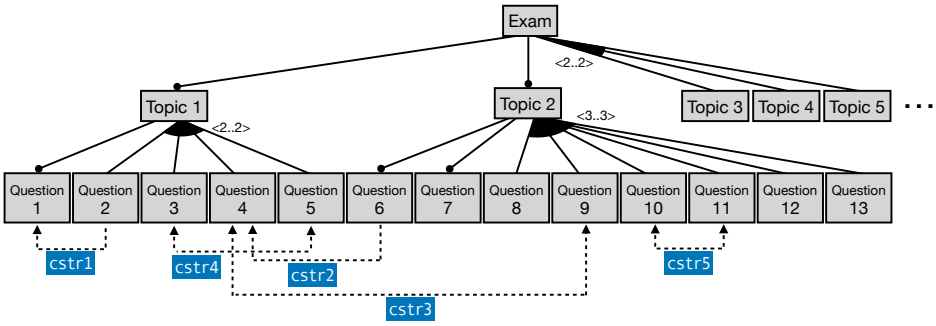
3.2 Configuring Parameterized Questions

A parameterized question is a template with mathematical expressions that are changed based on a specific set of replacement values. A straightforward template for parameterized questions can be: *"What is the result of X + Y?"*, in which X and Y are the parameters whose values are in the range of $[1..5]$. Based on this template, many instances can be generated by randomly selecting different values for X and Y in their domain. This way, we can generate a set of questions related to the sum of two parameters X and Y.

In this work, we support the configuration of parameterized questions for the purpose of increasing the number of exam instances. A set of parameterized questions can be represented using a feature model. The same as discussed in *Section 3.1*, a feature model for a set of parameterized questions represents the relationships between features using basic feature concepts. However, one difference lies in the parameterized features that are often used for a specific calculation (e.g., $X + Y$). Besides, the answers to a parameterized question are not predefined. They are instead automatically calculated depending on the selection of the parameterized features.

In the following, we present an example of parameterized question configuration using the feature model depicted in *Figure 4*:

- Features $f_4..f_7$, f_{11}, f_{12} are parameterized features.
- Features $f_4..f_7$ represent constraints of a CSP problem [40]. Their relationship with feature `Question` - f_1 is represented using a *group cardinality* $\langle 2..4 \rangle$, that specifies the minimum and maximum numbers of constraints to tailor the question statement.
- Features f_{11} and f_{12} represent correct and incorrect answers respectively, which can be automatically calculated depending on which parameterized features have been selected for the question. Assume features $f_4..f_7$ and the statement *"What is/are the corresponding minimal conflict(s)?"* have been selected, *#correct answers = #incorrect answers = 2*. Corresponding correct answers would be $\{c_1, c_2, c_3\}$ and $\{c_1, c_4\}$, and corresponding incorrect answers would be $\{c_2\}$ and $\{c_3\}$.
- Due to the support of parameterized features and an automated answer calculation mechanism, the definition of cross-tree constraints is pretty complex. Instead of using *requires/excludes* constraints, more complex constraints have to be defined. The semantics of the constraints is summarized in the following:
  - $cstr1$ specifies the domain of variables $v1..v3$.
  - $cstr2$ and $cstr3$ assure to trigger at least one inconsistency among the selected features $f_4..f_7$.
  - $cstr4$ ensures the existence of many conflicts.
  - $cstr5$ specifies which of the features (f_4 .. f_7) have been se-



cstr6 `sum(question)=40`
cstr7 `sum(question.estimated_duration)=50`
cstr8 `sum(question.important_level=nice_to_know)/sum(question) = 0.5`
cstr9 `sum(question.important_level=important_to_know)/sum(question) = 0.3`
cstr10 `sum(question.important_level=extremely_important_to_know)/sum(question) = 0.2`
cstr11 `sum(question.type=multiple_choice)/Sum(question) > 0.6`

Figure 6: An example feature model for a set of exams, in which *cstr1..cstr5* represent the relationship between questions, *cstr6* and *cstr7* are resource constraints, *cstr8..cstr10* are question complexity constraints, and *cstr11* denotes a constraint w.r.t. question type.

lected and then adds the selected features to a new variable C.

- $cstr6$ identifies all conflicts using the `conflict_hsdag` function [35]. The `random` function is used to randomly select minimal conflicts for the correct answers.
- $cstr7$ indicates that if the question is "*What is the preferred minimal conflict?*", then the correct answer would be the outcome of the `quickxplain` function [25].
- $cstr8$ and $cstr9$ help to generate incorrect answers.

In order to support the configuration of parameterized questions, we propose a mock-up as shown in *Figure 7*, whose design is similar to the mock-up for configuring multiple-choice questions (see *Figure 3*).

3.3 Exam Configuration

A set of exams can be modeled using a feature model, in which each feature represents a topic and/or a question (see *Figure 6*). The relationships between questions, as well as the relationships between exam topics and corresponding questions, can be represented by the constraints described in basic feature models, cardinality-based feature models, and extended feature models. Constraints in *basic feature models* can be used to describe the relationship between topics and questions. For instance, there exists a *mandatory* relationship between a topic and a question, showing that a question should belong to a specific topic. Constraints in *cardinality-based feature models* can be exploited to define the minimum number and the maximum number of questions in a specific topic. For instance, there exists a group cardinality $\langle 2..3 \rangle$ between the feature `Topic 2` and its subfeatures (`Question 8..Question 13`), showing that there are minimum *two* questions and maximum *three* questions to be included in `Topic 2`. Constraints in *extended feature models* can be used to define question complexity constraints. For instance, the distribution of question complexity in the exam should be 50% for *"nice to know"* questions, 30% for *"important to know"* questions, and 20% for *"extremely important to know questions"* (see constraints *cstr8..cstr10*). Further constraints regarding *number of questions*, *duration*, and *question types* could also be defined (see constraints *cstr6, cstr7,* and *cstr11*).

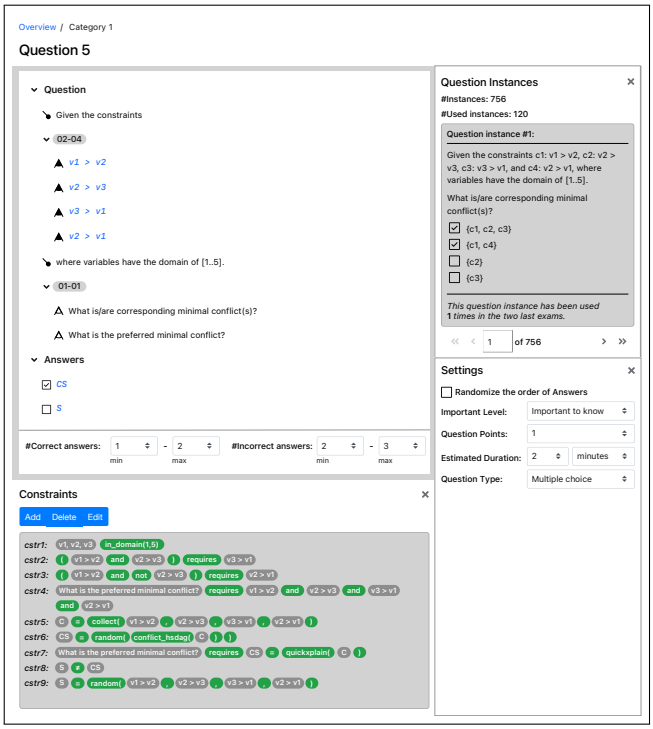


Figure 7: Mock-up for parameterized question configuration representing the feature model depicted in *Figure 4*.

Based on the generated constraints, a set of exam instances can be generated using a *constraint solver*. Before activating the solver, the exam feature model has to be translated into a CSP [40]. On the basis of this representation, solutions (configurations) are directly determined by the solver, such as Excel Solver [19] or Choco Solver [33]. Each configuration indicates an exam instance, which is generated by traversing selected features in the depth-first fashion.

To support the exam configuration process, we propose a mock-up as shown in *Figure 8*. Similar to the mock-up for question configuration, *Exam Editor* (*Part 1*) and *Constraint Editor* (*Part 2*) are shown on the left-hand side, which allow an examiner to describe the exam structure (based on a feature model) as well as corresponding con-

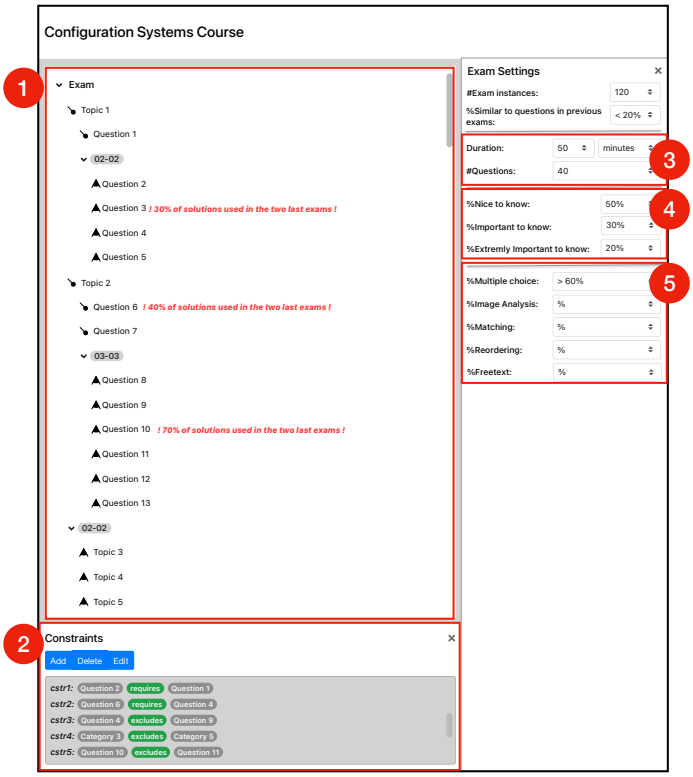


Figure 8: Mock-up for exam configuration, consisting of five parts: *Part 1* - Exam Editor, *Part 2* - Constraint Editor, *Part 3* - Resource Constraints Settings, *Part 4* - Question Complexity Settings, and *Part 5* - Question Type Settings. The content in these *Parts* is related to the feature model depicted in *Figure 6*.

straints between topics and questions. Settings placed in the right-hand slide allow an examiner to specify further constraints regarding resource constraints (*Part 3*), question complexity constraints (*Part 4*), and the distribution of question types in the exam (*Part 5*). Besides these parts, the mockup allows an examiner to specify the number of exam instances (e.g., *#Exam instances = 120*) and how much each exam instance similar to previous exams (e.g., *%Similar to previous exams* $< 20\%$).

4 RECOMMENDATION ALGORITHM

As mentioned in *Section 1*, to counteract exam cheating, besides increasing the question bank, the exam generation process should be supported by a recommendation mechanism that helps to select exams that are less similar to previous exams as much as possible. To address this goal, we use a *content-based recommendation* approach that filters exams based on the similarity between the questions of the generated exams and the questions of previous exams.

Given a set of question instances, we need to specify instances that have been used in previous exams as well as their frequency. *Instances that were frequently used in the previous exams should be omitted.* To do this, for a question instance P, we need to calculate the frequency f_P of instance P to be used in a previous exam E_j. We first build the profile for the question using a *vector space model* [32]. The question is represented as a *n-dimensional vector*, in which each dimension corresponds to a term. The value of each term is the frequency of the term appearing in the question. The similarity between P and a question Q_i in exam E_j is calculated using *cosine similarity* [43] (see *Formula 1*).

$$sim(P, Q_i) = \frac{P \cdot Q_i}{||P|| \cdot ||Q_i||} \quad (1)$$

The calculated similarity between two questions P and Q_i is then compared with a *threshold* θ that has been specified by the examiner. In our mock-up shown in *Figure 8*, the examiner can specify the threshold in the item *"%Similar to questions in previous exams"*. If the similarity is greater than θ, we can conclude that P is very similar to Q_i and increases f_P by 1. The same procedure can be done for other previous exams. Finally, the frequency of P to appear in n previous exams can be identified by *Formula 2*. The lower the f_P value, the higher the probability of choosing question instance P for the exam.

$$f_P = |sim(P, Q_i) > \theta : \forall Q_i \in E_j, i \in [1..m], j \in [1..n]| \quad (2)$$

where n is the number of the previous exams and m is the number of questions in E_j.

5 CONCLUSION

The paper has proposed an approach that exploits configuration and recommendation techniques to counteract exam cheating. Thank to question and exam configuration mechanisms, our approach is able to generate a large number of exam instances, which assures the distribution of different exams to students. Supported by a content-based recommendation algorithm, our approach also helps to generate exams that are different from previous exams. This way, it can prevent students from dishonesty behaviors regarding item harvesting, item pre-knowledge, and item memorizing.

Our approach, however, shows some limitations. Automated question and exam generation could trigger issues regarding the preciseness of generated questions and exams, emerging as a gap to be bridged within the scope of future work. Although we have developed mock-ups to support examiners' question and exam generation processes, the implementation of an exam generator prototype is still needed to further analyze user needs, the applicability of the proposed mock-ups, and the effectiveness of our approach.

ACKNOWLEDGMENTS

The presented work has been conducted in the ParXCel project funded by the Austrian Research Promotion Agency (880657).

References

[1] Gediminas Adomavicius and Alexander Tuzhilin, 'Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions', *IEEE Trans. on Knowl. and Data Eng.*, **17**(6), 734–749, (Jun. 2005).

[2] Helaine Alessio, Nancy Malay, Karsten Maurer, A. Bailer, and Beth Rubin, 'Examining the effect of proctoring on online test scores', *Online Learning*, **21**(1), (2017).

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake, *Feature-Oriented Software Product Lines: Concepts and Implementation*, Springer Science & Business Media, 2013.

[4] Don Batory, 'Feature Models, Grammars, and Propositional Formulas', in *International Conference on Software Product Lines*, eds., Henk Obbink and Klaus Pohl, pp. 7–20, Berlin, Heidelberg, (2005). Springer Berlin Heidelberg.

[5] Razan Bawarith, Abdullah Basuhail, Anas Fattouh, and Shehab Gamalel-Din, 'E-exam cheating detection system', *International Journal of Advanced Computer Science and Applications*, **8**(4), 176–181, (2017).

[6] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés, 'Automated Analysis of Feature Models 20 Years Later: A Literature Review', *Information Systems*, **35**(6), 615 – 636, (2010).

[7] James Bennett and Stan Lanning, 'The netflix prize', in *Proceedings of KDD Cup and Workshop*, pp. 3–6, New York, (2007). ACM.

[8] Robin Burke, Alexander Felfernig, and Mehmet H. Göker, 'Recommender systems: An overview', *AI Magazine*, **32**(3), 13–18, (Jun. 2011).

[9] Mason Chen, 'Detect multiple choice exam cheating pattern by applying multivariate statistics', in *Proceedings of the International Conference on Industrial Engineering and Operations Management*, pp. 173–181, Bogota, Colombia, (Oct. 2017).

[10] Chia Yuan Chuang, Scotty D. Craig, and John Femiani, 'Detecting probable cheating during online assessments based on time delay and head pose', *Higher Education Research & Development*, **36**(6), 1123–1137, (2017).

[11] Gregory J. Cizek and James A. Wollack, eds., *Detecting Potential Collusion among Individual Examinees using Similarity Analysis*, chapter 3, 47–69, Routledge, Oct. 2016.

[12] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker, 'Formalizing cardinality-based feature models and their specialization', *Software Process: Improvement and Practice*, **10**(1), 7–29, (2005).

[13] Jennifer P. Davis. Using data forensics to detect cheating: An illustration, 2018.

[14] Seife Dendir and R. Maxwell, 'Cheating in online courses: Evidence from online proctoring', *Computers in Human Behavior Reports*, **2**, 100033, (Aug. 2020).

[15] Martin Dick, Judy Sheard, Cathy Bareiss, Janet Carter, Donald Joyce, Trevor Harding, and Cary Laxer, 'Addressing student cheating: Definitions and solutions', *SIGCSE Bull.*, **35**(2), 172–184, (Jun. 2002).

[16] George M. Diekhoff, Emily E. LaBeff, Robert E. Clark, Larry E. Williams, Billy Francis, and Valerie J. Haines, 'College cheating: Ten years later', *Research in Higher Education*, **37**, 487–502, (1996).

[17] Alexander Felfernig, David Benavides, José Galindo, and Florian Reinfrank, 'Towards Anomaly Explanation in Feature Models', in *ConfWS-2013: 15th International Configuration Workshop (2013)*, volume 1128, pp. 117–124, (Aug. 2013).

[18] Alexander Felfernig and Robin Burke, 'Constraint-based recommender systems: Technologies and research issues', in *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC'08, pp. 1–10, New York, NY, USA, (2008). ACM.

[19] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Christian Russ, and Markus Zanker, 'Developing Constraint-Based Applications with Spreadsheets', in *Developments in Applied Artificial Intelligence*, eds., Paul W. H. Chung, Chris Hinde, and Moonis Ali, volume 2718 of *IEA/AIE 2003*, pp. 197–207, Berlin, Heidelberg, (2003). Springer.

[20] Alexander Felfernig, Viet Man Le, and Trang Tran, 'Supporting feature model-based configuration in microsoft excel', in *22nd International Configuration Workshop*, (2020).

[21] Alexander Felfernig, Monika Schubert, and Christoph Zehentner, 'An efficient diagnosis algorithm for inconsistent constraint sets', *Artif. Intell. Eng. Des. Anal. Manuf.*, **26**(1), (Feb. 2012).

[22] Joanna Golden and Mark Kohlbeck, 'Addressing cheating when using test bank questions in online Classes', *Journal of Accounting Education*, **52**(C), (2020).

[23] Hicham Hage and Esma Aïmeur, 'Exam question recommender system', in *Proceedings of the 2005 Conference on Artificial Intelligence in Education: Supporting Learning through Intelligent and Socially Informed Technology*, p. 249–257, NLD, (2005). IOS Press.

[24] Lothar Hotz, Alexander Felfernig, Markus Stumptner, Anna Ryabokon, Claire Bagley, and Katharina Wolter, 'Chapter 6 - Configuration Knowledge Representation and Reasoning', in *Knowledge-Based Configuration*, eds., Alexander Felfernig, Lothar Hotz, Claire Bagley, and Juha Tiihonen, 41 – 72, Morgan Kaufmann, Boston, (2014).

[25] Ulrich Junker, 'QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems', in *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, p. 167–172. AAAI Press, (2004).

[26] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson, 'Feature-Oriented Domain Analysis (FODA) Feasibility Study', Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, (1990).

[27] Viet-Man Le, Thi Ngoc Trang Tran, and Alexander Felfernig, 'A conversion of feature models into an executable representation in microsoft excel', in *Intelligent Systems in Industrial Applications*, eds., Martin Stettinger, Gerhard Leitner, Alexander Felfernig, and Zbigniew W. Ras, pp. 153–168, Cham, (2021). Springer International Publishing.

[28] Greg Linden, Brent Smith, and Jeremy York, 'Amazon.com recommendations: Item-to-item collaborative filtering', *IEEE Internet Computing*, **7**(1), 76–80, (Jan. 2003).

[29] Donald McCabe, 'Cheating on tests: How to do it, detect it, and prevent it (review)', *The Journal of Higher Education*, **73**, 297–298, (Jan. 2002).

[30] Donald L. McCabe, Kenneth D. Butterfield, and Linda Klebe Treviño, 'Academic dishonesty in graduate business programs: Prevalence, causes, and proposed action', *Academy of Management Learning and Education*, **5**(3), 294–305, (Sep. 2006).

[31] James Moten Jr, Alex Fitterer, Elise Brazier, Jonathan Leonard, and Avis Brown, 'Examining online college cyber cheating methods and prevention measures', *Electronic Journal of e-Learning*, **11**, 139–146, (2013).

[32] Michael J. Pazzani and Daniel Billsus, *Content-Based Recommendation Systems*, 325–341, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[33] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca, *Choco Solver Documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016.

[34] Hong Qian, Dorota Staniewska, Mark Reckase, and Ada Woo, 'Using response time to detect item preknowledge in computer-based licensure examinations', *Educational Measurement: Issues and Practice*, **35**, n/a–n/a, (Feb. 2016).

[35] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).

[36] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, *Recommender Systems Handbook*, Springer-Verlag, Berlin, Heidelberg, 1st edn., 2011.

[37] Neil C. Rowe, 'Cheating in online student assessment: Beyond plagiarism', *Online Journal of Distance Learning Administration*, **7**(2), (2004).

[38] Thi Ngoc Trang Tran, Müslüm Atas, Alexander Felfernig, and Martin Stettinger, 'An overview of recommender systems in the healthy food domain', *Journal of Intelligent Information Systems*, **50**(3), 501–526, (Jun. 2018).

[39] Thi Ngoc Trang Tran, Alexander Felfernig, Christoph Trattner, and Andreas Holzinger, 'Recommender systems in the healthcare domain: state-of-the-art and research issues', *Journal of Intelligent Information Systems*, 1–31, (Dec. 2020).

[40] Edward Tsang, *Foundations of Constraint Satisfaction*, Academic Press, London, 1993.

[41] W. J. van der Linden and Guo Fanmin, 'Bayesian procedures for identifying aberrant response-time patterns in adaptive testing', *Psychometrika*, **73**, 365–384, (2008).

[42] George Watson and James Sottile, 'Cheating in the digital age: Do students cheat more in online courses?', *Online Journal of Distance Learning Administration*, (Jan. 2010).

[43] Qian Zhang, Jie Lu, and Zhang Guangquan, 'Recommender systems in e-learning', *J Smart Environ Green Comput*, **1**, 76–89, (Jun. 2020).