Solving Multi-Configuration Problems: A Performance Analysis with Choco Solver

Benjamin Ritz[1,*], Alexander Felfernig[1], Viet-Man Le[1] and Sebastian Lubos[1]

[1]Graz University of Technology, Inffeldgasse 16b, 8010 Graz, Austria

Abstract

In many scenarios, configurators support the configuration of a solution that satisfies the preferences of a single user. The concept of *multi-configuration* is based on the idea of configuring a set of configurations. Such a functionality is relevant in scenarios such as the configuration of personalized exams, the configuration of project teams, and the configuration of different trips for individual members of a tourist group (e.g., when visiting a specific city). In this paper, we exemplify the application of multi-configuration for generating individualized exams. We also provide a constraint solver performance analysis which helps to gain some insights into corresponding performance issues.

Keywords

Knowledge-based Configuration, Multi-Configuration, Performance Analysis

1. Introduction

Configuration is the process of assembling basic components into a complex product while taking into account a set of constraints [1, 2, 3, 4]. Most existing configurators are based on the assumption that a solution (configuration) is developed for a single user. On the contrary, group-based configuration [5, 6] focuses on the configuration of a solution for a group of users. Such a configuration must satisfy the preferences of each individual user as much as possible [7, 8]. Group-based configuration can be further extended to allow the configuration of a set of solutions based on the preferences of one or multiple users. Such a *multi-configuration problem* [9] includes a set of constraints specifying restrictions with regard to (1) the combination of multiple solutions and (2) properties of a specific solution. Scenarios including such configuration sets may benefit from multi-configuration. Related example scenarios are the following.

Multi-exam configuration. individual exams are configured for each student, where related constraints are specified by instructors and possibly also students. A configurator can support instructors during the exam preparation phase and helps in the prevention of cheating through the generation of individualized exams [10].

Project team assignment. persons are assigned to teams such that each team has the expertise to successfully complete the corresponding project (assigned to the team). In this context, fairness aspects can play a role, for example, each team should have at least similar chances to complete a project within pre-defined time limits [9].

Generation of test cases. The automated generation of test cases can be considered as a multi-configuration problem, where input values need to be generated in such a way that given coverage criteria are fulfilled [11].

Holiday planning. Members of tourist groups often do not share the same interests during excursions [12], i.e., which sightseeing destinations to visit. Therefore, a configurator could configure different trips for subgroups of tourists based on their preferences.

Configuration space learning. Many (software) systems (e.g., operating systems) offer a high degree of configurability. In this context, it is difficult to find the optimal configuration settings [13] also due to the fact that it is impossible to test all possible settings to find the optimal one. Multi-configuration can support the identification of test configurations that help to learn dependencies between configuration parameters.

In the context of *multi-exam configuration*, we have built a software library that helps to configure exams. Example inputs are the number of examinees, a pool of questions, and a set of constraints specifying preferences of instructors and examinees. The outcome is a set of questions for each examinee. Constraint solving in our implementation is based on the Choco constraint solver.[1]

In this paper, we show how the problem of multi-exam configuration can be represented as a constraint satisfaction problem (CSP) [14]. We exemplify different types of constraints supported by our configurator and also show

ConfWS'23: 25th International Workshop on Configuration, Sep 6–7, 2023, Málaga, Spain

*Corresponding author.

ritz@student.tugraz.at (B. Ritz); alexander.felfernig@ist.tugraz.at (A. Felfernig); vietman.le@ist.tugraz.at (V. Le); vietman.le@ist.tugraz.at (S. Lubos)

https://www.tugraz.at/ (B. Ritz); https://www.felfernig.eu (A. Felfernig); https://www.tugraz.at/ (V. Le); https://www.tugraz.at/ (S. Lubos)

0009-0000-7774-6693 (B. Ritz); 0000-0003-0108-3146 (A. Felfernig); 0000-0001-5778-975X (V. Le); 0000-0002-5024-3786 (S. Lubos)

[1]https://choco-solver.org/

how the preferences of students can (potentially) be taken into account. In order to analyze constraint solver performance, we evaluate the runtime performance of an open source constraint solver (Choco) on the basis of a typical real-world exam configuration scenario.

The remainder of this paper is organized as follows. In Section 2, we introduce a definition of a multi-configuration task and provide an example from the domain of multi-exam configuration. In this context, we also introduce and exemplify different constraint types. Thereafter, in Section 3, we evaluate the performance of Choco when solving multi-exam configuration tasks also including a performance analysis when solving a real-world configuration task. Threats to validity are discussed in Section 4. The paper is concluded with a discussion of open research issues in Section 5.

2. Working Example

We now give a basic definition of a *multi-configuration task* (see Definition 1) (see [9]) and show how multi-exam configuration tasks can be introduced correspondingly.

Definition 1. A multi-configuration task can be defined as a tuple (V, D, C) with $V = \bigcup\{v_{ij}\}$ is a set of finite domain variables (v_{ij} is variable j of configuration instance i), $D = \bigcup\{dom(v_{ij})\}$ a set of corresponding domain definitions, and $C = \{c_1, c_2, ..., c_v\}$ a set of constraints.

In the following, we will use this definition to introduce the task for multi-exam configuration and outline what constraint types are supported by our configurator. In this context, the set of constraints C can be defined by users represented by *instructors* and also *students* (where this is intended).

2.1. Multi-exam configuration

Following Definition 1, a multi-exam configuration task can be defined as follows.

- $V = \{q_{11}..q_{nm}, q_{11}.T_1..q_{11}.T_\pi, .., q_{nm}.T_1..q_{nm}.T_\pi\}$ where q_{ij} is question j of the exam of student i, n is the number of exams (students), m is the number of questions per exam, $q_{ij}.T_k$ denotes the value of the *k-th question property* of question q_{ij}, and π represents the number of question properties per question (in our case, $\pi = 6$). Our configurator supports the following question properties:
  1. *topic* - topic of the question
  2. *level* - difficulty level of the question
  3. *min-duration* - minimum estimated time needed to answer the question
  4. *max-duration* - maximum estimated time needed to answer the question
  5. *type* - type of the question (e.g. single/multiple choice, assignment task, etc.)
  6. *points* - maximum number of points rewarded for correct answers
- $D = \{dom(q_{11})..dom(q_{nm}), dom(q_{11}.T_1)..dom(q_{11}.T_6), .., dom(q_{nm}.T_1)..dom(q_{nm}.T_6)\}$ where $dom(q_{ij}) = \{1..\Omega\}$, with Ω being the total number of questions in the question pool, and $dom(q_{ij}.T_k)$ is a question property domain (one out of the following):
  1. *dom(topic)* = $\{1..\eta\}$ where η is the number of defined question topics
  2. *dom(level)* = $\{1..\mu\}$ where μ is the number of defined question complexity levels
  3. *dom(min-duration)* = $\{1..\tau\}$ with τ indicating the maximum specifiable value
  4. *dom(max-duration)* = $\{1..\kappa\}$ with κ indicating the maximum specifiable value
  5. *dom(type)* = $\{1..\theta\}$ where θ is the number of defined question types
  6. *dom(points)* = $\{1..\phi\}$ where ϕ is the maximum amount of points
- $C = \{c_1..c_v\}$ where c_β is the constraint identifier and v is the number of constraints

Importantly, depending on the question 1..Ω assigned to a question variable q_{ij}, a set of corresponding question properties must hold, for example, if question $q_{11} = 1$, corresponding restrictions such as $q_{11} = 1 \rightarrow q_{11}.topic = A$ indicate the relevant question properties. In Subsection 3.2, we explain in which way we support this aspect in our configuration library. Furthermore, for each student-specific exam i, we need to include an $alldifferent(q_{i1}..q_{im})$ constraint to avoid situations where a questions is assigned to the same exam twice. In our implementation, this aspect is taken into account on the basis of set variables (see also Subsection 3.2).

2.2. Instructor constraints

Instructor constraints (defined by instructors) in C restrict the set of questions that may or may not appear in exams. Each exam must fulfil all of these constraints. We distinguish between two types of related constraints: *intra-exam* and *inter-exam* constraints.

2.2.1. Intra-exam constraints

Intra-exam constraints restrict which questions are eligible for being part of an exam. Such constraints refer to each individual exam. A simple form of intra-exam constraints is to directly define a specific question property. For example, let us assume that up to now a course has covered only one (the first) topic (A). As a consequence,

the instructor requires that only questions belonging to topic A are part of the first exam (see Formula 1).

$$\forall q_{ij} \in V \,:\, q_{ij}.topic = A \tag{1}$$

Furthermore, we might want to restrict the complexity level of questions. For example, assuming that four different question complexities exist, for the final exam the instructor would like to increase the overall exam complexity using an intra-exam constraint specifying that all questions of each exam must have a complexity level of at least 2 (see Formula 2).

$$\forall q_{ij} \in V \,:\, q_{ij}.level \geq 2 \tag{2}$$

Intra-exam constraints allow instructors to arbitrarily combine multiple constraints with logical operators. For example, since multiple choice questions can generally be answered rather quickly, an instructor could require that every multiple choice question is of at least difficulty level 3 (see Formula 3 where we assume question type 3 indicates multiple choice questions).

$$\forall q_{ij} \in V \,:\, (q_{ij}.type = 3 \implies q_{ij}.level \geq 3) \tag{3}$$

In many cases, instructors would like to be able to specify intra-exam constraints on a more granular level. It is possible to combine intra-exam constraints with a corresponding scope. Constraint scopes enable instructors to specify how many questions per exam need to satisfy a given constraint. To illustrate this aspect, we will continue our previous example (see Formula 1). By the time the next topic (topic B) is covered in the course, the students will have a follow-up exam consisting of 10 questions. The instructor now wants to focus mainly on the new topic. Therefore, they specify constraints such that for each exam only 2 questions belong to topic A and the remaining 8 to topic B (see Formula 4 and 5).

$$\bigwedge_{i=1}^{n(\#exams)} \left(|\{q_{ij} \in V \,:\, q_{ij}.topic = A\}| = 2\right) \tag{4}$$

$$\bigwedge_{i=1}^{n(\#exams)} \left(|\{q_{ij} \in V \,:\, q_{ij}.topic = B\}| = 8\right) \tag{5}$$

Constraint scopes also support lower and/or upper bounds, for example, the instructor would like to keep the follow-up exam rather simple. For this reason, between 5 and 10 questions of each exam should be easy to solve, which is indicated by complexity level 1 (see Formula 6).

$$\bigwedge_{i=1}^{n(\#exams)} \left(5 \leq |\{q_{ij} \in V \,:\, q_{ij}.level = 1\}| \leq 10\right) \tag{6}$$

Instructors may also specify constraint scopes using percentages in order to describe which amount of questions per exam must satisfy the question property constraint. For example, only between 10 and 20 percent of questions per exam should be solvable in less than five minutes (see Formula 7).

$$\bigwedge_{i=1}^{n(\#exams)} \left(0.10 \leq \frac{|\{q_{ij} \in V \,:\, q_{ij}.min\text{-}duration < 5\}|}{m} \leq 0.20\right) \tag{7}$$

Intra-exam constraints also support aggregations. In the context of our evaluation settings, we support the functions *sum*, *average*, and *distinct count*. For their application, see the examples in Formulas 8–10.

1. Example (*sum*): The total amount of points per exam is 100 (see Formula 8).

$$\bigwedge_{i=1}^{n(\#exams)} \left(\sum_{j=1}^{m} q_{ij}.points = 100\right) \tag{8}$$

2. Example (*average*): The average complexity level of each exam is between 2 and 3. (see Formula 9).

$$\bigwedge_{i=1}^{n(\#exams)} \left(2 \leq \frac{\sum_{j=1}^{m} q_{ij}.level}{m} \leq 3\right) \tag{9}$$

3. Example (*distinct count*): Each exam consists of at least 3 different question topics (Formula 10).

$$\bigwedge_{i=1}^{n(\#exams)} \left(|\{q_{ij}.topic\}| \geq 3\right) \tag{10}$$

2.2.2. Inter-exam constraints

Similar to intra-exam constraints, inter-exam constraints restrict which questions may be part of exams. However, they constrain how often certain question or question properties may or may not appear in the entire exam configuration. Therefore, inter-exam constraints depend on all exams combined, instead of every exam individually. Such constraints count, for example, how many exams have at least one question that fulfills a given constraint. This sum can be lower and/or upper bounded. For example, we assume that a specific question ξ is part of at least 5 exams but at most 10 (see Formula 11).

$$5 \leq |\{q_{ij} \in V \,:\, q_{ij} = \xi\}| \leq 10 \tag{11}$$

As a special case of inter-exam constraints, instructors can restrict the degree of question overlap across exams, i.e., the number of questions that exams have in common. This is especially useful to prevent the generation of identical or very similar exams. The degree of overlap can be lower and upper bounded in order to restrict the

minimum and maximum amount of questions that pairs of exams may share. For example (see Formula 12), the upper bound denotes that no pair of exams exists which shares more than 5 questions, whereas the lower bound states that every pair of exams must share at least 2 questions. This might be useful to create a sense of fairness among students but might lead to cheating. e_λ represents a set of questions comprising all questions of exam λ.

$$\bigwedge_{i=1}^{n(\#exams)} \bigwedge_{j=1}^{n(\#exams)} \big(2 \leq |e_i \cap e_j| \leq 5\big)\,(i \neq j) \tag{12}$$

The amount of exam pairs to be constrained can be further decreased in the context of onsite exams where a pre-defined lecture hall's seating plan (chart) is specified. One goal in such scenarios is to prevent cheating of students positioned in a neighborhood which can be achieved on the basis of constraints avoiding question overlaps in the case of students located next to each other.

Given is the following lecture hall with 5 rows and 8 seats per row (see Figure 1).

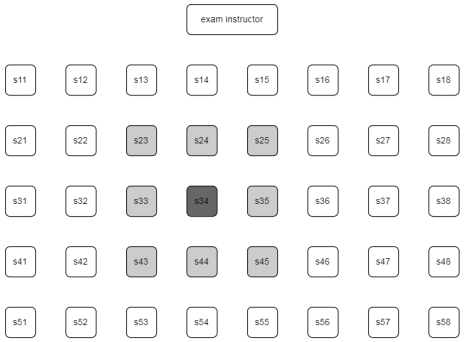

Figure 1: lecture hall seating: seat 34 and its neighbors.

The seats are labeled with the letter s and two digits. The first digit represents its row (top to bottom) and the second digit its position in the row (left to right). We assume no neighboring exams may share even a single question (see Formula 13), where k is the number of neighbors of exam i, e_i is the question set assigned to exam i (student i), and $f(i, j, s)$ describes the set of questions of i's *j-th* neighboring exam according to seating chart s.

$$\bigwedge_{i=1}^{n(\#exams)} \bigwedge_{j=1}^{k} (|e_i \cap f(i, j, s)| = 0) \tag{13}$$

This allows identical exams in the configuration but never for students right next to each other. In order to provide a better understanding of how many constraints approximately need to be added per seat, we have highlighted seat s34 (dark gray) and its neighbors (light gray) as an example (see Figure 1). This particular seat has 8 neighbors requiring 8 constraints to be added.

2.3. Student constraints

Student constraints in C can be specified by each student individually. They constrain only the student's exam, no other exams are affected. Student constraints can only further narrow down instructor constraints.

Example: The instructor specifies a constraint such that between 20% and 50% of the questions of each exam must belong to topic A (see Formula 14).

$$\bigwedge_{i=1}^{n(\#exams)} \left(0.20 \leq \frac{|\{q_{ij} \in V \,:\, q_{ij}.topic = A)\}|}{m} \leq 0.50\right) \tag{14}$$

Student a decides to restrict this constraint even further so that only a maximum of 25% of questions of their exam belong to topic A (see Formula 15).

$$\frac{|\{q_{aj} \in V \,:\, q_{aj}.topic = A)\}|}{m} \leq 0.25 \tag{15}$$

3. Evaluation

We now present a performance analysis of our multi-configuration setting.[2]

3.1. Real world example

We assume that (1) 450 students participate in an exam and (2) a question pool of 45 questions individually associated with one out of four different topic areas (topics) is available. Each exam should consist of $m = 10$ questions. We define the following constraints (C):

1. Each exam should include questions related to at least 2 different topics.

$$\bigwedge_{i=1}^{n(\#exams)} \big(|\{q_{ij}.topic\}| \geq 2\big)$$

2. There is at most one multiple choice question.[3]

$$\bigwedge_{i=1}^{n(\#exams)} (|\{q_{ij} \in V \,:\, q_{ij}.type = 3\}| \leq 1)$$

3. 10% – 20% of the questions are assigned to complexity level 4 being the most complex one.

$$\bigwedge_{i=1}^{n(\#exams)} \left(0.10 \leq \frac{|\{q_{ij} \in V \,:\, q_{ij}.level = 4\}|}{m} \leq 0.20\right)$$

4. Each exam should include questions resulting in 40 points in total.

$$\bigwedge_{i=1}^{n(\#exams)} \left(\sum_{j=1}^{m} q_{ij}.points = 40\right)$$

[2] A link to the source code of our configuration approach will be provided in the final paper version.

[3] Question type 3 was assumed to be multiple choice.

5. Neighboring exams share at most 2 questions (assuming a hall s with 22 rows and 21 seats/row).

$$\bigwedge_{i=1}^{n(\#exams)} \bigwedge_{j=1}^{k} (|e_i \cap f(i,j,s)| \leq 2)$$

Considering these specific requirements, the configurator yielded a solution in about *one second*.

3.2. Dealing with flexible upper bounds

Instead of relying on a constant number of questions per student exam, it is also possible to support flexible lower and upper bounds. We support this aspect by utilizing Choco *set variables*. Every student exam includes a set of questions. The domain of a set variable is (implicitly) defined by lower and upper bound sets. The lower bound is a set of questions that each exam must include, whereas the upper bound defines the maximum possible question set. In our case, the lower bound is empty and the upper bound equals the question pool.

A varying number of questions per exam could trigger the need for further instructor constraints, for example, to restrict the allowed number of questions. Let us assume a defined question pool with $\Omega = 3$ questions ($\{1, 2, 3\}$) and $n = 2$ exams (e_1 and e_2) represented as Choco set variables (*model* is a Choco *model object*).

```
e1 = model.setVar(lb:{}, ub:{1, 2, 3})
e2 = model.setVar(lb:{}, ub:{1, 2, 3})
```

Now, we want to specify that each exam e_i (of student i) needs to include at least two and at most three questions. In Choco, this constraint would be defined as follows.

```
e1.setCard(model.intVar(2, 3))
e1.setCard(model.intVar(2, 3))
```

In this simplified setting, the possible solution sets for both, e_1 and e_2 are: $\{1, 2\}, \{1, 3\}, \{2, 3\}$, and $\{1, 2, 3\}$.

If we also want to define restrictions on allowed question properties, the solver needs to know the question properties of each individual question. For scalability reasons, we avoid to define question/property relationships on the basis of constraints. Instead, we support a key-value data structure that allows the identification of question properties on the basis of the corresponding question $ID \in \{1..\Omega\}$. Given such a structure, we are now able to define constraints referring to question properties, for example: in each exam, the number of questions of topic A is exactly 2.

In Choco, no related built-in constraints exist. We have defined a custom constraint by extending the *Propagator* class and implemented the two required methods *propagate* and *isEntailed*.[4] The former is called in each iteration of the solving process. It tries to find solutions by counting the number of questions that belong to the specified topic. If the current branch of the solving process cannot satisfy the constraint, a contradiction is indicated. When taking into account this constraint, the possible solutions for e_1 and e_2 are $\{1, 2, 3\}$ and $\{1, 3\}$.

[4] A *GitHub* source code reference will be included in the final paper.

3.3. Evaluation with synthesized data

We have also evaluated the solver performance with synthesized multi-exam configuration tasks along the dimensions of *number of questions* and *number of exams*. Each task utilizes the same 5 constraints as discussed in Subsection 3.1. We choose the lecture hall size depending on the amount of exams n, using the formula $\lceil\sqrt{n}\rceil$, since this is a fairly simple way to assure that all students will fit in the lecture hall and to keep a good ratio between rows and seats per row. The results of this performance evaluation are summarized in Table 1 showing acceptable runtime performances in the context of typical exam settings as well as extreme cases of around 1000 questions and up to 1000 students inducing solver runtimes up to nearly 3 minutes. Notice that a smaller question pool size does not always result in faster runtime.

4. Threats to Validity

In the context of the reported evaluation, we have applied the standard settings of the used constraint solver. A major topic of further work is to further improve runtime performance on the basis of different approaches supporting the learning of solver search heuristics (see, e.g., [15]). Fairness is a crucial aspect to be taken into account when it comes to the automated generation of exams. In this work, we have taken this aspect into account a.o. on the basis exam-specific criteria regarding the percentage of to-be-included questions that are related to a specific complexity level. For future work, we plan to further refine this aspect, for example, on the basis of optimization functions that help to balance the complexity of individual exams on a more fine-grained level. Finally, in real-world settings, we often have to deal with situations where a given set of constraints is inconsistent, i.e., no solution could be identified. In our future work, we will integrate corresponding repair concepts which will help users to find ways out from the so-called *no solution could be found* dilemma. Such approaches can be based o.a. on model-based diagnosis [16].

5. Conclusions

In this paper, we have introduced multi-configuration as a useful approach in scenarios requiring solution set configuration, for example, exam configuration and project

Table 1
Constraint solver (configurator) performance based on synthesized settings differing in terms of number of questions and number of student-specific exams using the constraints introduced in Subsection 3.1. In this context, *s*=seconds and *m*=minutes. Cells without unit of measurement represent runtimes in *milliseconds*.

		Exams										
		1	**5**	**10**	**25**	**50**	**75**	**100**	**250**	**500**	**750**	**1000**
Questions	**25**	14,33	328,33	397,67	349,00	425,33	475,67	516,00	1183,33	3340,33	6,30s	10,10s
	50	21,67	34,00	80,33	100,67	143,33	153,00	183,33	448,67	1231,33	2572,33	3769,00
	75	26,67	46,33	63,00	88,33	155,33	221,33	234,00	686,67	1689,00	3387,00	5,66s
	100	23,67	55,33	80,33	161,00	172,67	318,00	314,00	849,33	2345,00	4899,00	7,36s
	150	36,00	93,67	102,67	193,67	301,00	462,00	485,00	1163,33	3751,00	7,08s	12,80s
	250	109,00	146,33	185,33	383,33	562,67	725,33	934,33	2540,67	7,21s	14,38s	20,98s
	500	114,00	269,33	407,33	878,00	1638,33	2174,33	3020,33	7,64s	19,27s	35,75s	55,30s
	750	168,33	415,67	719,67	1677,33	3163,33	5,02s	5,09s	15,92s	33,41s	1,14m	1,50m
	1000	250,33	651,67	1052,67	2942,67	5,01s	8,08s	9,04s	27,18s	1,06m	1,81m	2,67m

team configuration. In the context of multi-exam configuration, we have shown a corresponding configuration task representation as a constraint satisfaction problem. We have evaluated the performance of the proposed approach on the basis of an example real-world configuration task as well as a collection of synthesized configuration tasks (differing in terms of the number of pre-defined questions and the number of "to be generated" exams). Our future work will include the integration of further concepts supporting solver performance optimization. Furthermore, we will include features, for example, in terms of optimization functions, that help to take into account aspects such as fairness in a more explicit fashion. Finally, we plan to include concepts that will allow us to take into account historical data, for example, when generating a set of "new" exams, the frequency of questions already "used" in previous exams should be taken into account in order to avoid situations where specific questions are posed too often.

References

[1] M. Stumptner, An overview of knowledge-based configuration, AICom 10 (1997) 111–125.

[2] U. Junker, Configuration, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006, pp. 837–873.

[3] L. Hvam, N. Mortensen, J. Riis, Product Customization, Springer, 2008.

[4] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based Configuration – From Research to Business Cases, Elsevier, 2014.

[5] A. Felfernig, M. Atas, T. Tran, M. Stettinger, Towards group-based configuration, in: ConfWS'16, Toulouse, France, 2016, pp. 69–72.

[6] V. Le, T. Tran, A. Felfernig, Consistency-based integration of multi-stakeholder recommender systems with feature model configuration, in: 26th ACM Intl. Systems and Software Product Line Conference, ACM, New York, NY, USA, 2022, p. 178–182.

[7] A. Felfernig, M. Stettinger, G. Ninaus, M. Jeran, S. Reiterer, A. Falkner, G. Leitner, J. Tiihonen, Towards open configuration, in: ConfWS'14, Novi Sad, Serbia, 2014, pp. 89–94.

[8] M. Atas, A. Felfernig, S. Polat-Erdeniz, A. Popescu, T. Tran, M. Uta, Towards psychology-aware preference construction in recommender systems: Overview and research issues, J. Intell. Inf. Syst. 57 (2021) 467–489. doi:10.1007/s10844-021-00674-5.

[9] A. Felfernig, A. Popescu, M. Uta, V. Le, S. Erdeniz, M. Stettinger, M. Atas, T. Tran, Configuring multiple instances with multi-configuration, in: ConfWS'21, Vienna, Austria, 2021, pp. 45–47.

[10] V. Le, T. Tran, M. Stettinger, L. Weißl, A. Felfernig, M. Atas, S. Erdeniz, A. Popescu, Counteracting exam cheating by leveraging configuration and recommendation techniques, in: ConfWS'21, Vienna, Austria, 2021, pp. 73–80.

[11] A. Gotlieb, B. Botella, M. Rueher, Automatic test data generation using constraint solving techniques, in: ACM SIGSOFT Intl. Symp. on Software Testing and Analysis, Florida, USA, 1998, pp. 53–62.

[12] A. Jameson, S. Baldes, T. Kleinbauer, Two methods for enhancing mutual awareness in a group recommender system, in: Working Conf. on Advanced Visual Interfaces, Gallipoli, Italy, 2004, pp. 447–449.

[13] J. Pereira, M. Acher, H. Martin, J. Jézéquel, G. Botterweck, A. Ventresque, Learning software configuration spaces: A systematic literature review, Journal of Systems and Software 182 (2021) 111044.

[14] F. Rossi, P. van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier, 2006.

[15] M. Uta, A. Felfernig, D. Helic, V. Le, Accuracy- and Consistency-Aware Recommendation of Configurations, in: SPLC'2022, ACM, 2022, pp. 79–84.

[16] R. Reiter, A theory of diagnosis from first principles, Artificial Intelligence 32 (1987) 57–95.