SAT Encodings of the At-Most-k Constraint
A Case Study on Configuring University Courses

Paul Maximilian Bittner(✉), Thomas Thüm, and Ina Schaefer

TU Braunschweig, Brunswick, Germany
{p.bittner,t.thuem,i.schaefer}@tu-braunschweig.de

Abstract. At universities, some fields of study offer multiple branches to graduate in. These branches are defined by mandatory and optional courses. Configuring a branch manually can be a difficult task, especially if some courses have already been attended. Hence, a tool providing guidance on choosing courses is desired. Feature models enable modelling such behaviour, as they are designed to define valid configurations from a set of features. Unfortunately, the branches contain constraints instructing to choose at least k out of n courses in essence. Encoding some of these constraints naïvely in propositional calculus is practically infeasible. We develop a new encoding by combining existing approaches. Furthermore, we report on our experience of encoding the constraints of the computer science master at TU Braunschweig and discuss the impact for research on configurability.

1 Introduction

Universities offer various fields of study to graduate in. In our rapidly growing economics and academia, the need for custom variants or even hybrid areas of study arises. Usually, this would lead to the introduction of a new field of study. However, if the changes are only slight or partial, allowing the graduation in different sub-branches within the same field saves bureaucratic effort and thereby time and money. Such branches are usually bound to two constraints in selection of courses. First, some courses are mandatory for graduating in the desired branch. Second, courses from a given list for at least a certain amount of credit points have to be attended. These *compulsory elective* courses and the amount of required credit points often vary for each branch. For example, the TU Braunschweig offers various branches of study in their masters degree program for computer science.[1] That allows not only studying computer science, but also putting emphasis on individual branches like visual computing, networked systems, or robotics.

Usually, informal specifications of the branches tend to be ambiguous and inconsistent, as noticed at TU Braunschweig. Furthermore, students often

[1] https://www.tu-braunschweig.de/informatik-msc/struktur/studienrichtungen.

P. C. Ölveczky and G. Salaün (Eds.): SEFM 2019, LNCS 11724, pp. 127–144, 2019.
https://doi.org/10.1007/978-3-030-30446-1_7

enquire whether they still have the opportunity to graduate in a certain branch after they already completed a number of possibly unrelated courses. A configuration tool for these branches to automate this process and specify the branches precisely is desired.

Feature models are designed to describe relations between individual features (e.g., the courses), such that only specific subsets of features can be chosen [1]. Thereby, they separate configuration logic from the configuration process itself. In contrast to ad-hoc programming and using SMT solvers, we can profit from reusing existing research and tooling on configuration and decision propagation when using feature models [4,14,16–18,20]. For example, guidance for feature selection and explanations for automatic decisions are available [11]. Hence, a configurator comes for free if we can model the branches as a feature model.

As feature models are translated to a boolean formula for analysis, constraints are expressed in propositional calculus [3]. Unfortunately, the *compulsory elective* constraints become a bottleneck. These essentially break down to the atmost$_k$ constraint describing that at most k out of n variables can be set to *true*. This constraint grows exponentially in n when encoded naïvely in propositional calculus. For example, we obtained a formula of about 1 GB text for the branch of Automotive Informatics. Formulas this huge are intractable for common SAT solvers and cannot even be generated in Conjunctive Normal Form (CNF) in a reasonable time even though required by SAT solvers.

Albeit the atmost$_k$ constraint can be encoded to first-order logic naturally, using SMT solvers requires upgrading existing tools and research on boolean feature models. These not only provide configurators with decision propagation already, but are also able to inform the user when and why features are (de-) selected automatically due to model constraints. Thus, course selection would be fully transparent in the resulting tool. Such configurators do not yet exist for first-order logic.

Our research question is: Can we express branches of study as boolean feature models? Therefore we split the problem into multiple steps:

- We developed a new encoding for the atmost$_k$ constraint to minimise formula size by combining existing state-of-the-art encodings [8,12,19]. This is not only useful for feature models but SAT queries in general.
- To describe branches of study we created a Domain Specific Language (DSL) and a compiler, translating the DSL artefacts to a feature model including the constraints.
- We propose a method for generating propositional formulas requiring a sum of weighted variables to be reached. We show its usability for resolving different amounts of credit points.
- We compare performance of different encodings by generating constraints for the branches of study at TU Braunschweig.

In order to test and evaluate our encoding, we implemented each reviewed encoding, our DSL, and our compiler in an open-source project. Our final Branch of study Tool (BroT) and all data are publicly available online at GitHub.[2]

[2] https://github.com/PaulAtTUBS/BroT.

Encoding	$\text{atmost}_1(\{A, B, C\})$
Binomial	$(\neg B \vee \neg C) \wedge (\neg A \vee \neg C) \wedge (\neg A \vee \neg B)$
Binary	$(T_0 \vee \neg A) \wedge (\neg T_0 \vee \neg B_0) \wedge (\neg T_0 \vee \neg B_1)$ $(T_1 \vee \neg B) \wedge (\neg T_1 \vee \quad B_0) \wedge (\neg T_1 \vee \neg B_1)$ $(T_2 \vee \neg C) \wedge (\neg T_2 \vee \neg B_0) \wedge (\neg T_2 \vee \quad B_1)$
Sequential Counter	$(\neg A \vee R_0) \wedge (\neg B \vee R_1)$ $\neg R_0 \vee R_1$ $(\neg B \vee \neg R_0) \wedge (\neg C \vee \neg R_1)$
Commander	$A \vee B \vee \neg c_0$ $(\neg B \vee c_0) \wedge (\neg A \vee c_0) \wedge (\neg A \vee \neg B)$ $C \vee \neg c_1$ $\neg C \vee c_1$ $\neg c_0 \vee \neg c_1$

Fig. 1. With each encoding $\text{atmost}_1(\{A, B, C\})$ is generated. For readability, formulas are split upon multiple rows and are concatenated with $\wedge$. For the same reason, some generated variable indices are shortened.

2 Encoding At-Most-k Constraints

In this section, we elaborate on the atmost_k constraint and how it can be expressed in propositional calculus and introduce our novel *selective* encoding for it. Albeit, the atmost_k constraint is essential for describing *compulsory elective* constraints, choosing at most k out of n elements, where $k, n \in \mathbb{N}, 0 < k < n$, is a common problem. Translated to propositional calculus, the atmost_k constraint requires not more than k variables from a given set V to be true:

$$\bigwedge_{\substack{X \subseteq V, \\ |X|=k+1}} \bigvee_{x \in X} \neg x \tag{1}$$

As this encoding creates $\binom{|V|}{k+1}$ clauses, it is called the *binomial* encoding [8]. Unfortunately, in this representation, formula size grows too fast to be suitable for most use cases. Thus, we further review the encodings *binary* [9,10], *commander* [8,12], and *sequential counter* [19]. Each of them introduces new variables summarizing some information about the original variables' values. Figure 1 exemplifies these encodings for $\text{atmost}_1(\{A, B, C\})$. The *binary* encoding introduces k Bit-Strings of length $\lceil \log_2(n) \rceil$ identifying exactly one variable each. It does not generate a CNF per default as required by SAT solvers. Hence, we use a variation of the *binary* encoding presented by Frisch and Giannaros [8], creating a CNF directly. The *sequential counter* encoding uses n unary registers of size k to count the number of true variables sequentially. An overflow is disallowed because then more than k variables would to be true. The *commander*

encoding recursively groups the variables and assigns k commander variables to each group. These contain information whether no or some of the variables in their corresponding group are true. We refer interested readers for details on those encodings to the paper by Frisch and Giannaros [8].

To minimise the resulting formula size, we develop a meta-encoding, called *selective* encoding, which chooses the most efficient of the reviewed encodings considering formula size:

$$\text{selective}(n,k) = \begin{cases} \text{binomial} & k_{binom}(n) \leq k, \\ \text{binary} & k_{split}(n) < k < k_{binom}(n), \\ \text{seq. counter} & \text{otherwise.} \end{cases} \tag{2}$$

As *selective* encoding is motivated by our evaluation results we present its construction as well as the bounds k_{binom} and k_{split} and the reason for the *commander* encoding not being used in Sect. 4.2.

Our use cases mostly rely on the related atleast_k and exactly_k constraints as illustrated later. We express atleast_k by reducing it to atmost_k. If at least k variables have to be true, not more than the remaining number of variables can be false:

$$\text{atleast}_k(S) = \text{atmost}_{n-k}(\{\neg s \mid s \in S\}) \tag{3}$$

By combining atleast_k and atmost_k, we obtain an expression for choosing exactly k variables:

$$\text{exactly}_k(S) = \text{atleast}_k(S) \wedge \text{atmost}_k(S) \tag{4}$$

We encode exactly_k by using our new encoding for atleast_k and atmost_k respectively.

3 Modelling Configuration of University Courses as Feature Models

In this section, we describe our pipeline for branch configuration. First, we formally define the concept of branches of study to give an unequivocal reference as informal specifications usually tend to be ambiguous. Therefore we refer to the four terms *field*, *branch*, *subject*, and *category*. A whole area of study at a university like physics, biology and computer science is referred to as a *field* of study. *Branches* of study are subtypes of a field of study and are more fine-grained. Students can specialise in a branch fitting their individual interests. Working units granting credit points like lectures, labs and theses are referred to as *subjects*. *Categories* group subjects belonging to a common department. Second, we present our DSL allowing users to create and edit fields of study including branches. Third, as we are interested in the possibility of expressing branches of study as boolean feature models, we present our compiler, translating artefacts of our DSL to a feature model. Thereby, special attention is given to differing amounts of credit points.

***Compulsory* subjects (35 credit points)**
Seminar IT-Security (5 credit points)
Master's Thesis (30 credit points)

***Compulsory elective* subjects (35 credit points)**
Category System Security
Advanced IT-Security (5 credit points)
Machine Learning for IT-Security (5 credit points)
Lab on IT-Security (5 credit points)
Lab on Intelligent System Security (5 credit points)
Project Thesis (15 credit points)
Category Connected and Mobile Systems
Management of Information Security (5 credit points)
Category Distributed Systems
Operating Systems Security (5 credit points)

Fig. 2. Example for specification of *IT-Security* branch at TU Braunschweig (translated from German).

3.1 Formalizing Branches of Study

Branches of study are subtypes of a field of study. They are a concept for dealing with the need for more customised fields of study at universities due to growing complexity of economy and science. Instead of constructing new fields of study, branches can be introduced to existing fields if they are similar enough. For our case study, we look at the branches of study at TU Braunschweig, but nevertheless, the concept of branches is analogous for other universities and institutions. Credit points granted at this university are ECTS-points (European Credit Transfer System points).

Graduating in a branch is optional, but not more than one can be chosen. To complete a branch its constraints for choosing courses have to be fulfilled. These consist of a *compulsory* and one or more *compulsory elective* constraints, containing a set of subjects each:

- *Compulsory* subjects have to be attended.
- *Compulsory elective* subjects have to be attended, such that a certain amount of credit points is reached.

As an example, the specification of the IT-Security branch at TU Braunschweig is given in Fig. 2. Next to its mandatory seminar and master's thesis, subjects from a given *compulsory elective* pool have to be attended, such that at least 35 credit points are reached. The project thesis is listed optional here, but is actually mandatory because the sum of all other subjects does not reach the required 35 credit points. Our tool BroT will detect this issue and automatically select the project thesis when this branch is picked.

```
Field "Computer Science"
  Category "Master Thesis" [1, 1] {
    "Master Thesis IT-Security" 30 CP
    "Master Thesis Computer Graphics" 30 CP
    ...
  }
  Category "Subjects" {
    Category "IT-Security" {
      "Advanced IT-Security" 5 CP
      "Lab on IT-Security" 5 CP
      ...
    }
    ...
  }
  ...
  Branch "IT-Security"
    Compulsory
      "Master Thesis IT-Security"
      "Seminar IT-Security"
    CompulsoryElective 35 CP
      "Advanced IT-Security"
      "Machine Learning for IT-Security"
      "Lab on IT-Security"
      "Lab on Intelligent System Security"
      "Project Thesis IT-Security"
      "Management of Information Security"
      "Operating Systems Security"
```

Fig. 3. Excerpt of DSL artifact specifying the *IT-Security* branch described in Fig. 2. Subjects referenced by a branch have to be specified with their corresponding credit points.

3.2 A DSL to Describe Fields and Branches of Study

Creating a feature model for a field of study directly is practically infeasible. The branches constraints would have to be written by hand in propositional calculus, but due to their extent and complexity, this task is highly error-prone and time-consuming. Hence, we provide a DSL from which the branches can be translated to a feature model.

Our DSL allows specifying fields, branches and subjects of study (e.g. courses, theses) with their corresponding amount of credit points granted on completion. Branches are described by a set of constraints. A constraint is a subset of subjects and is either *compulsory* or *compulsory elective*. Furthermore, subjects can be grouped. For example, all possible master's theses are grouped in category *Master Thesis*. That allows for specifying cardinalities $[a, b] \subset \mathbb{Z}, 0 \leq a$, describing how many subjects have to be attended at least and at most. For example, the *Master Thesis* in Fig. 3, having cardinality $[1, 1]$, has to be written exactly

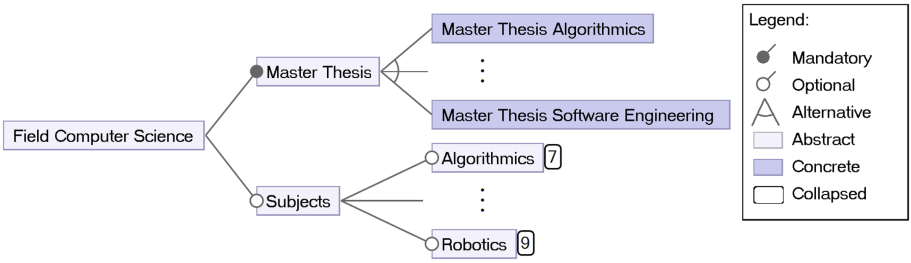

Fig. 4. Example of a feature model describing a simplified version of the field of study computer science at TU Braunschweig. Dots denote that some features are omitted for readability. The modules `Algorithmics` and `Robotics` contain 7 and 9 courses respectively. These are collapsed for readability, too.

once. A value of -1 for b denotes an arbitrary amount, otherwise b has to be greater or equal to a. The default value for cardinalities is $[0, -1]$.

Typically, branches are described by one *compulsory* and one *compulsory elective* constraint, but the official specifications may contain additional side conditions. For example, in *Visual Computing*, auxiliary to the *compulsory* constraint, selecting one of three pre-defined courses is mandatory. The other two courses will then be added to the pool of subjects of the *compulsory elective* constraint. We resolve this issue by introducing a new *compulsory elective* constraint and adjusting the required credit points of the original one.

3.3 Compilation of Our DSL to a Feature Model

As feature models come with dedicated and well investigated analysis tools [4, 11, 16, 18, 20], these can be reused as is for configuring branches of study when encoding the branches as a proper feature model.

Originally introduced to manage configurations of software product lines, feature models are far more general. They define individual features in a tree hierarchy. In Fig. 4, a simplified model describing the field of study *Computer Science* (without the branches) is given. Features can only be selected if their parent feature is selected. Optionally, features can be cumulated in *alternative* or *or* groups. For example, the children of `Master Thesis` in Fig. 4 are in an *alternative* group, as only one master's thesis can be written. Additionally, the `Master Thesis` feature is marked mandatory because it has to be selected. An *or* group requires at least one of its children to be selected. The `Subjects` are grouped by their categories and collapsed in this example. Features can be marked abstract, indicating that they do not have a concrete implementation and are used for modelling purposes only. Auxiliary to parent relationships, each feature can be part of additional arbitrary constraints. These have to be given in propositional calculus, as shown in Fig. 5.

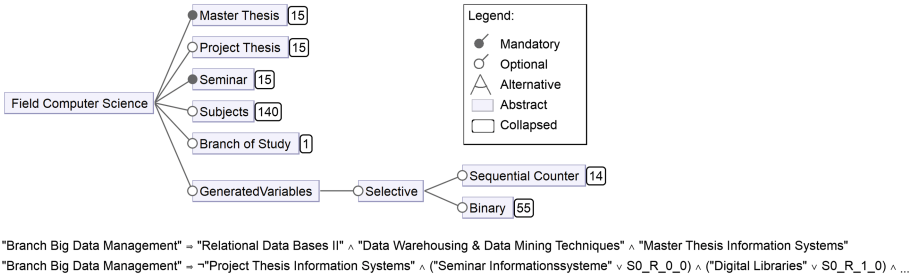

Fig. 5. The feature model generated for branch *Big Data Management*: Numbers behind collapsed features indicate their number of children. The feature `GeneratedVariables` is artificial and groups all variables that were generated for atmost$_k$ constraints by the respective encoding.

Subjects and Categories. Each subject and branch is translated to a feature. For traceability, features are grouped as children of abstract features if their corresponding subjects are grouped. Specific cardinalities of categories can directly be translated to the feature model hierarchy. The feature of a category with cardinality $[c_{min}, c_{max}]$ is mandatory if $c_{min} > 0$, i.e., at least one subject has to be selected. Otherwise, it is optional. Furthermore, the group type can be derived as follows:

- *alternative* if $c_{max} = 1$, i.e., at most one child can be selected,
- *or* if $c_{min} \geq 1$, i.e., at least one child has to be selected.

Moreover, the branches' constraints are translated to propositional calculus, such that they can be added to the feature model. It is important that these constraints are only valid if their corresponding branch is chosen (i.e., feature is selected). We achieve this by means of an implication.

Figure 5 exemplarily shows the feature model of the branch *Big Data Management* compiled from its corresponding DSL artefact. If the feature `Branch Big Data Management` is selected, the two constraints at the bottom will ensure that the right subjects have to be chosen for the configuration to be valid. The first constraint describes the *compulsory* subjects according to Eq. 5. The second constraint describing *compulsory elective* subjects according to Eq. 10 is too long to be shown here entirely. It was computed with our *selective* encoding. Thereby, the *sequential counter* encoding was used to encode atleast$_6(\{1, ..., 8\})$ and produced $14 = (8-1)(8-6) = (n-1)k$ variables. 55 variables were generated by the *binary* encoding for atleast$_3(\{1, ..., 8\})$.

Compulsory and Compulsory Elective Constraints. In the following, we show how our compiler translates constraints consisting of a set of subjects S_B for a given branch B.

In *compulsory* constraints all subjects are mandatory, meaning that they have to be attended if their corresponding branch is chosen:

$$B \implies \bigwedge_{s \in S_B} s \tag{5}$$

Compulsory elective constraints are defined by the amount of credit points p to achieve at least by selecting a subset of its subjects S_B.

$$B \implies \text{atleast}_k(S_B), \tag{6}$$

Assuming that each subject in S_B grants an equal amount c of credit points, we can determine the number k of required subjects via $k = \lceil p \,/\, c \rceil$. For example, if all subjects grant 5 credit points and a *compulsory elective* constraint requires at least 35 credit points (like in Fig. 2), at least $7 = 35$ CP$/5$ CP subjects have to be chosen. Occasionally, some subjects grant a different amount of credit points, wherefore Eq. 6 is insufficient. In these cases a more advanced approach is necessary, which we discuss in the next section.

3.4 Resolving Differing Credit Points

If all subjects grant equally high credit points, all variables have equal weight. Hence, the actual value of credit points must only be considered to obtain the total number of subjects to choose at least. However, many branches' *compulsory elective* subjects differ in their credit points.

We solve this problem by recursively splitting our set of subjects S in two sets H (high) and L (low). All subjects granting the most credit points are put into H, the rest goes to L. Accordingly, all subjects in H grant an equal amount of credit points. By choosing an exact amount of subjects from H, the remaining credit points can be obtained. These remaining credit points can then be chosen from subjects in L. If all subjects in L grant an equal amount of credit points we can use Eq. 6, otherwise we recursively split L again.

Formally, we define $cp(s) \in \mathbb{N}$ as the amount of credit points a subject s grants. If all subjects in a set of subjects S grant the same amount c of credit points, we define $cp(S) = c$. To split S, we introduce the former set H as a function

$$\mathrm{H}(S) = \{x \in S \mid \forall s \in S : \ cp(x) \geq cp(s)\}, \tag{7}$$

which returns all subjects with the highest amount of credit points. For given subjects S and credit points c, our resulting *compulsory elective* constraint CEC is constructed as follows:

$$\mathrm{CEC}(S, c) = \begin{cases} \text{true} & c \leq 0 \\ \text{false} & c > \sum_{s \in S} cp(s) \\ \text{atleast}_{\lceil c / cp(S) \rceil}(S) & \mathrm{H}(S) = S \wedge 0 < c \leq \sum_{s \in S} cp(s) \\ \text{split}(S, c) & \text{otherwise} \end{cases} \tag{8}$$

If the amount of credit points is smaller or equal to zero, enough of them are already reached. If the amount of credit points to achieve c is greater than the credit points all subjects grant together, we never can choose enough subjects. If all subjects grant an equal amount of credit points, the previous solution from Eq. 6 can be used. Finally, we formalise our approach of splitting S into two sets and choosing a definite amount from the higher credit subjects:

$$\text{split}(S, c) = \bigvee_{k=0}^{|\text{H}(S)|} \text{exactly}_k(\text{H}(S)) \wedge \text{CEC}(S \backslash \text{H}(S), c - k * cp(\text{H}(S))) \quad (9)$$

The recursive call to CEC is done on the remaining subjects $S\backslash\text{H}(S)$ and remaining credits $k * cp(\text{H}(S))$. Hence, the problem is simplified to a smaller instance of itself with one amount of credit points removed from the set of subjects. By combining Eqs. 6 and 8, we obtain the final constraint for a given branch B with *compulsory elective* subjects S_B and credit points c_B:

$$B \implies \text{CEC}(S_B, c_B) \quad (10)$$

As a post-processing step, some of the clauses generated by the split function can be omitted in the first place if they are not satisfiable considering the feature model. This commonly occurs if all subjects in $\text{H}(S)$ are in the same category with an upper bound $c_{max} > 0$ where one can never choose more subjects than c_{max}. Thus, $\text{exactly}_k(\text{H}(S))$ can never be true for $k > c_{max}$.

Finally, we can generate feature models representing any fields of study, including branches and subjects with arbitrary credit points. Our upcoming evaluation focuses on their applicability for configuring the modelled branches. Furthermore, we derive our *selective* encoding for the atmost_k constraint by measuring the other encodings performance. We evaluate it by generating the *compulsory elective* constraints for all branches according to Eq. 10.

4 Evaluation

We evaluate the four considered encodings *binomial*, *binary*, *sequential counter*, and *commander*, in terms of produced literals and generated variables. Additionally, we report our experiences when implementing, testing and applying these encodings. We introduce our new encoding, called *selective* encoding, that combines the other encodings to choose the most efficient one considering formula size. By generating the *compulsory elective* constraints for each branch with each encoding, we can evaluate the performance of our new *selective* encoding.

4.1 Tool Support for Implementation

For the opportunity to reuse dedicated libraries and frameworks, we implemented our tool BroT in Java. The FeatureIDE library [13] allows expressing formulas

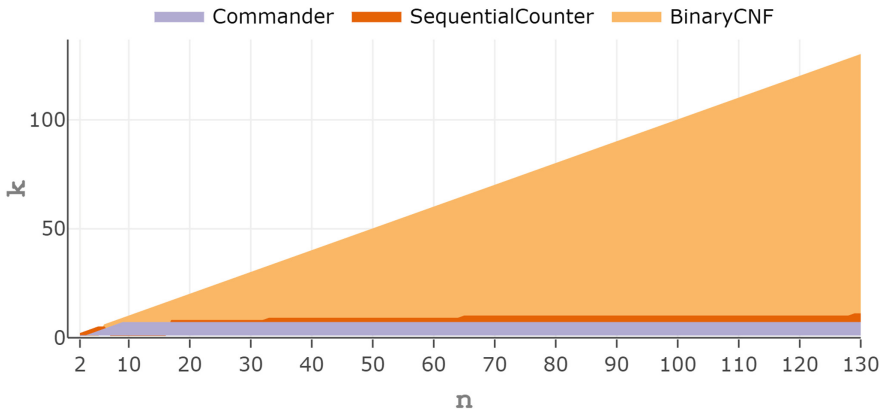

Fig. 6. Encodings producing the lowest number of variables when encoding $\text{atmost}_k(n)$. The *binomial* encoding is not considered because it does not introduce any new variables.

of propositional calculus as well as describing feature models. Additionally, the FeatureIDE plugin [17] for the Eclipse IDE [21] contains graphical editors for feature models and their configurations. We created our DSL with the plugin EMFText[3], which integrates well into the other tools.

4.2 At-Most-k Encoding Performance Comparison

First, we evaluate the atmost_k encodings in terms of generated variables. Here, the *binomial* encoding is not considered because it always produces the lowest amount of variables, namely none. Second, we investigate formula size by counting the number of literals to be independent of clause size. Thereby, we develop our new *selective* encoding, as it is motivated by these results.

To detect the most efficient encoding, we encoded $\text{atmost}_k(n)$ for each $2 \leq n \leq 130, 1 \leq k < n$ with each encoding. Many instances with $n > 26$ became too big for the *binomial* and *commander* encoding, resulting in a memory overflow. In these cases, we assessed them to produce infinitely many variables and literals.

Remarkably, for both criteria, the results split into three connected areas. Hence, stack plots identifying the encoding producing the lowest amount of variables and literals for each k and n are shown in Figs. 6 and 7 respectively. Our plots are available as scatter and stack plots as interactive `HTML` versions bundled with our code to allow investigating the exact values.[4]

[3] https://github.com/DevBoost/EMFText.
[4] https://github.com/PaulAtTUBS/BroT/tree/master/Evaluation/Encodings.

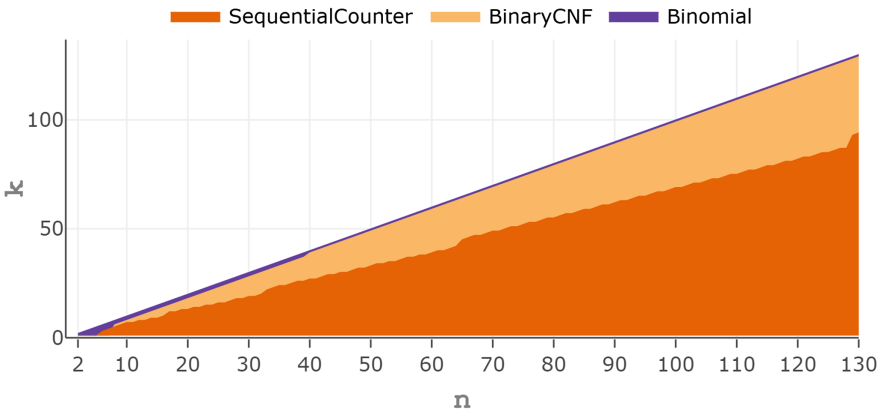

Fig. 7. Encodings producing the lowest number of literals when encoding atmost$_k(n)$. These results serve as the basis for our *selective* encoding.

Number of Generated Variables. As shown in Fig. 6, the *binary* encoding performs best in most cases. For each $k > 7$, it introduces the lowest amount of variables in the investigated data range of $1 < n \leq 130$. The *commander* encoding is best for small k. It groups the variables and assigns new commander variables to each group. Thereby, it depends heavily on the size of its groups. We discovered, that an optimal group size can only be chosen for the very rare case of $k(k+2) < n$. We hypothesise this to be the reason for the *commander* encoding producing the lowest amount of variables only for small k. As the *commander* encoding is recursive, it could be further optimised by not using the *binomial* encoding at end of recursion, but a more sophisticated one like *binary*, *sequential counter*, or even our *selective* encoding, introduced in the next section. In some rare cases, the *sequential counter* encoding generates the lowest number of variables, especially for small k.

Formula Size. In this section, we quantitatively assess encoding performance in terms of the number of generated literals. Motivated by these results, we develop our new *selective* encoding by combining the evaluated methods. If two encodings produced the same number of literals, we chose the encoding with fewer total variables.

The *commander* encoding never generated the smallest formula. We hypothesise the usage of the *binomial* encoding at the end of recursion to be the reason. As expected, the *binomial* encoding produces the smallest formula for very small $n < 6$, close to the suggested bound of $n < 7$ by Frisch and Giannaros [8]. Advanced encodings do not decompose to smaller formulas in those cases because the overhead of introducing new variables is too big. Furthermore, *binomial* is the most efficient encoding for $k = n - 1$, where it decomposes to a simple disjunction. Surprisingly, this naïve encoding produces the lowest number of literals for $k = n - 2, n < 40$, too. To describe the cases, where the *binomial* encoding

performs best, we introduce a function giving the lowest k for which it produces the smallest formula:

$$k_{binom}(n) = \begin{cases} 1 & n < 6, \\ n-2 & 6 \leq n < 40, \\ n-1 & \text{otherwise.} \end{cases} \tag{11}$$

The remaining input pairs (k, n) are shared between the *binary* and *sequential counter* encoding. The split between their areas consists of almost linear segments separated by little jumps, which are located at powers of two. When n exceeds a power of two, the binary encoding needs another bit, i.e. another variable. We hypothesise this to be the reason for the *sequential counter* encoding producing less literals than the *binary* encoding at these jumps. To describe the split, we consider the number of literals each encoding produces. For given n, the split is located at k for which both encodings produce the same amount of literals. Thereby, we derive a formula describing exactly the highest k for which the *sequential counter* encoding still produces less literals than the *binary* encoding.

$$\begin{aligned} k_{split}(n) &= \left\lfloor \frac{b + \sqrt{b^2 - 4a}}{2a} \right\rfloor, \text{ with} \\ a &= 1 + 2\lceil log_2(n) \rceil \\ b &= 2(\lceil log_2(n) \rceil (n+1) - 2n + 5) \end{aligned} \tag{12}$$

Finally, we can define our *selective* encoding by choosing the encoding producing the smallest formula in Eq. 2.

4.3 Branches Evaluation

We test our *selective* encoding by comparing its performance when generating feature models for each branch of study at TU Braunschweig with the reviewed encodings. The size of a feature model file in XML turned out to be a good initial indication of a model being usable by our configurator, i.e., can be loaded and handled in feasible time spans. We consider formula size and the total number of variables for all *compulsory elective* constraints at once. The results are shown in Fig. 8. We do not consider the *commander* encoding here because it never produced the lowest amount of literals, as outlined in Sect. 4.2.

Indeed, our *selective* encoding always produces the lowest amount of literals as highlighted in Fig. 9. Thereby, it is able to reduce the amount of literals by up to 20% compared to the respective best of the reviewed encodings. Although we developed it to optimise formula size, it also generates the lowest number of variables in seven out of nine cases as visible in Fig. 10 (without considering the *binomial* encoding). For the branch *Hardware-/Software-System Design* it even nearly halves the amount of variables. In the remaining two branches *Industrial Data Science* and *Networked Systems*, it is also competitive, as it produces only 0.5% and 10% more variables respectively. If we compose all branches

	Binomial			Binary			Seq. Counter			Selective					
Branch of Study	*kB*	*#var.*	*#lit.*	*kB*	*#var.*	*#lit.*	*kB*	*#var.*	*#lit.*	*kB*	*#var.*	*#lit.*	*ΔkB%*	*Δ#var.%*	*Δ#lit.%*
Automotive Informatics	82,682	32	895,347	1,414	1,159	6,541	1,694	2,107	10,135	1,295	1,007	**6,287**	8.42	13.11	3.88
Big Data Management	46	9	338	52	64	256	45	58	237	46	58	**221**	-2.22	0	6.75
H.-/S.-System Design	10,228	36	126,738	728	788	3,834	795	1,135	5,363	538	455	**3,074**	26.10	42.26	19.82
IT-Security	21	7	61	26	23	77	21	17	**50**	21	17	**50**	0	0	0
Industrial Data Science	2,223	22	28,052	170	202	1,018	153	246	1,095	145	203	**930**	5.23	-0.50	8.64
Medical Informatics	3,115	15	31,609	316	313	1,732	254	351	1,630	264	308	**1,513**	-3.94	1.60	7.18
Networked Systems	6,220	27	87,375	201	246	1,241	203	343	1,542	184	270	**1,227**	8.46	-9.76	1.13
Robotics	557	17	6,845	193	212	**977**	239	339	1,570	193	212	**977**	0	0	0
Visual Computing	108	14	1,205	98	124	547	84	131	560	88	124	**524**	-4.76	0	4.20
All Branches (Sum)	105,200	179	1,177,570	3,198	3,131	16,223	3,488	4,727	22,182	2,774	2,654	14,803	13.6	15.2	8.6

Fig. 8. For each branch of study at TU Braunschweig we encoded all its *compulsory elective* constraints with each encoding to compare their performance in terms of model file size, number of total variables, and literals. The minimal number of literals per row is highlighted. The last three columns show the improvement of our encoding in percent compared to the best of the single encodings. The *binomial* encoding is not considered for Δ#var.%.

as necessary for the complete field of study model, *binary* performs best from the reviewed encodings in each category. *Selective* encoding further reduces file size, number of variables, and number of literals by 13.6%, 15.2%, and 8.6%, respectively.

We developed our *selective* encoding in favour of formula size. Nevertheless, the solving time is an important metric for efficiency of formula generation [15, p. 413]. We found branch models loaded to our configuration tool BroT to be configurable without any lags. However, the loading times for the models exceed several minutes for most of the branches. This time span is mainly caused by the configuration initialisation. As it is branch specific and only necessary a single time, it could be pre-computed and stored on disk. Thus, BroT could enable instantaneous branch loading and configuration. Unfortunately, we were not able to load the branches *Automotive Informatics*, *Hardware-/Software-System Design*, and *Medical Informatics* yet as the configuration generation took too much time. We suspect our resolving of different credit points (Eq. 8) to impair performance immensely because the expression it generates is not in CNF.

4.4 Threats to Validity

In this section we reflect on our experiment design for evaluation of *selective* encoding in Sect. 4.3. We compare its performance with the reviewed encodings by generating the constraints for each branch of study at TU Braunschweig. Although, this is a very special use case, it emerges from a real-world problem for which a solution was even enquired at TU Braunschweig. At other universities or institutions this problem may arise analogously. Furthermore, each branch demands 5 to 45 atmost$_k$ formulas with $k \in [0, 19] \subset \mathbb{N}$ and $n \in [1, 20] \subset \mathbb{N}$ as specified at the universities website and in our DSL files delivered with our tool.

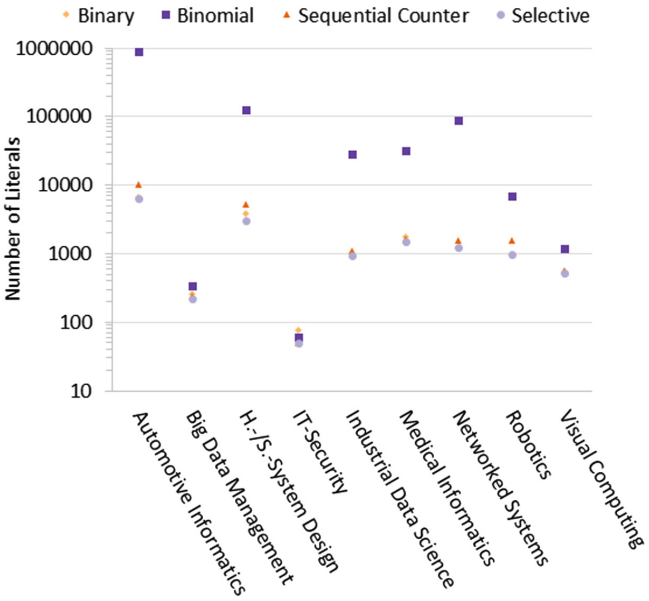

Fig. 9. Number of literals in *compulsory elective* constraints generated by each encoding per branch.

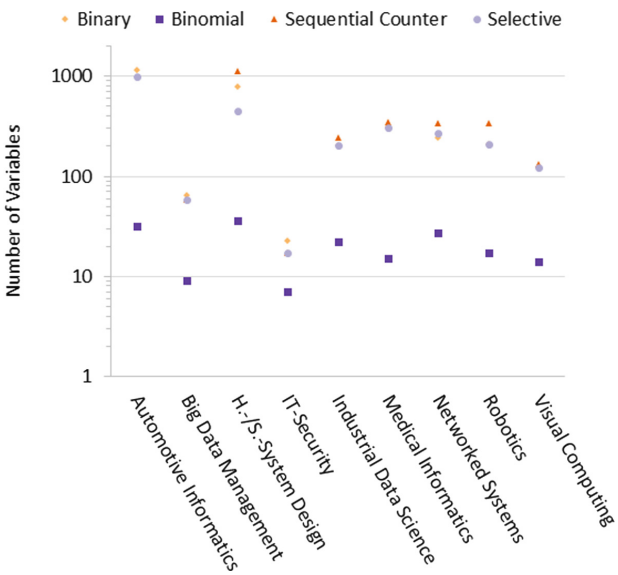

Fig. 10. Number of total variables in *compulsory elective* constraints generated by each encoding per branch.

The resulting formulas for each individual branch on which we count number of literals and variables include various atmost$_k$ constraints. Nevertheless, our usability results are unimpaired, as these branch specific descriptions are constant for all encodings. Therefore, our *compulsory elective* formulas (see Eq. 8) are fixed per branch, too. Particularly, the base feature model describing the field of study computer science is equal for the whole evaluation. Thus, for each branch the atmost$_k$ queries are the same for each encoding.

5 Related Work

This work is primarily based on two fields, namely feature-oriented software development and SAT encodings of the atmost$_k$ constraint.

First, we describe the branches of study with feature models [1]. A feature model can be converted directly to a propositional formula [3]. This enables analysis based on satisfiability queries [4,16,18,20]. We create these models with the FeatureIDE Framework [17] and use its analysis and configuration tools for testing and evaluating our results. We use the FeatureIDE library [13] to implement the encodings independently from the main FeatureIDE plugin. We detected a new application for feature models, as we use them for configuration of courses. Cardinality-based feature models assign cardinalities to features, allowing them to occur multiple times [6,7]. Our new *selective* encoding can be used to express the bounds of group cardinalities and, thus, cardinality-based feature models could profit from our encoding. Formulating alternative groups is the special case of choosing atmost$_1$ and is a common task for feature models. Hence, our *selective* encoding could improve the generation of these constraints. Here, our results exhibit the *sequential counter* encoding as a reasonable choice for $n > 5$.

Second, we are interested in encodings of the atmost$_k$ constraint. Frisch and Giannaros present a convenient summary of the state-of-the-art encodings [8]. Additionally, they lift some of the encodings from their atmost$_1$ form to atmost$_k$. We use their work compared with some of the original introductions [12,19] for further detail as a reference for implementing and using encodings correctly. The lifted version by Frisch and Giannaros of the *product* encoding by Chen [5] requires a dedicated number sequence. Because the generation of such a sequence is described only vaguely and informally, we have not considered this encoding. Our new *selective* encoding of the atmost$_k$ constraint is useful in any application the other encodings are used in, as it can replace them without any adaptions.

6 Conclusion and Future Work

We presented a new hybrid encoding, called *selective* encoding, for the atmost$_k$ constraint by combining existing techniques. By construction, our encoding produces the lowest amount of literals and, nevertheless, introduces a comparatively low amount of new variables.

We used *selective* encoding successfully for generating feature models that can be used for configuring branches of study. We showed that choosing a suitable

encoding for the atmost$_k$ constraint makes a difference of up to 20% in terms of literals for our branch study. Furthermore, our tool BroT using our DSL as input can be useful for universities and institutions facing similar problems. Our approach for resolving different amounts of credit points can be generalised to domains, where each element is weighted. Hence, it can be useful in any task where a sum of weights has to be reached by choosing arbitrary elements.

To improve our *selective* encoding and results on *compulsory elective* constraint generation, further encodings like *parallel sequential counter* by Sinz [19] or *totalizer* by Bailleux and Boufkhad [2] could be investigated, too. Especially the second one is of interest, as it can handle atmost and atleast constraints simultaneously, which could optimise our frequent exactly constraints in Eq. 9 when dealing with different amounts of credit points. Additionally, our *selective* encoding could be tested on handling alternative groups in feature models as these require the special case of atmost$_1$. Furthermore, the question why the encodings count for generated literals and variables split into distinct connected areas that allowed deriving our encoding, is still open.

Acknowledgements. We thank Moritz Kappel, Chico Sundermann, Timo Günther, Marc Kassubeck, Jan-Philipp Tauscher, and Moritz Mühlhausen for reviewing our paper in the earlier stages. Additional thanks go to the SEFM reviewers for giving very detailed and constructive remarks.

References

1. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines (2013)
2. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45193-8_8
3. Batory, D.: Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer, Heidelberg (2005). https://doi.org/10.1007/11554844_3
4. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–708 (2010)
5. Chen, J.: A new SAT encoding of the at-most-one constraint. In: Proceedings of the Constraint Modelling and Reformulation (2010)
6. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Softw. Process: Improv. Pract. **10**, 7–29 (2005)
7. Czarnecki, K., Kim, C.H.P.: Cardinality-based feature modeling and constraints: a progress report, pp. 16–20 (2005)
8. Frisch, A.M., Giannaros, P.A.: SAT encodings of the at-most-k constraint. some old, some new, some fast, some slow. In: Proceedings of the Ninth International Workshop of Constraint Modelling and Reformulation (2010)
9. Frisch, A.M., Peugniez, T.J.: Solving non-boolean satisfiability problems with stochastic local search. In: IJCAI, vol. 2001, pp. 282–290 (2001)
10. Frisch, A.M., Peugniez, T.J., Doggett, A.J., Nightingale, P.W.: Solving non-boolean satisfiability problems with stochastic local search: a comparison of encodings. J. Autom. Reason. **35**(1–3), 143–179 (2005). https://doi.org/10.1007/s10817-005-9011-0

11. Günther, T.: Explaining satisfiability queries for software product lines. Master's thesis, Braunschweig (2017). https://doi.org/10.24355/dbbs.084-201711171100. https://publikationsserver.tu-braunschweig.de/receive/dbbs_mods_00065308
12. Klieber, W., Kwon, G.: Efficient CNF encoding for selecting 1 from n objects. In: Proceedings of the International Workshop on Constraints in Formal Verification (2007)
13. Krieter, S., et al.: FeatureIDE: empowering third-party developers, pp. 42–45 (2017). https://doi.org/10.1145/3109729.3109751
14. Krieter, S., Thüm, T., Schulze, S., Schröter, R., Saake, G.: Propagating configuration decisions with modal implication graphs, pp. 898–909, May 2018. https://doi.org/10.1145/3180155.3180159
15. Kučera, P., Savický, P., Vorel, V.: A lower bound on CNF encodings of the at-most-one constraint. In: Gaspers, S., Walsh, T. (eds.) SAT 2017. LNCS, vol. 10491, pp. 412–428. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66263-3_26
16. Mannion, M.: Using first-order logic for product line model validation, pp. 176–187 (2002)
17. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G.: Mastering Software Variability with FeatureIDE. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61443-4
18. Mendonça, M.: Efficient reasoning techniques for large scale feature models. Ph.D. thesis, University of Waterloo, Canada (2009)
19. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005). https://doi.org/10.1007/11564751_73
20. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: Analysis strategies for software product lines: a classification and survey, pp. 57–58, Gesellschaft für Informatik (GI), Bonn, Germany, March 2015
21. Wiegand, J., et al.: Eclipse: a platform for integrating development tools. IBM Syst. J. **43**(2), 371–383 (2004)