

Automated Generation of Computationally Hard Feature Models using Evolutionary Algorithms

Sergio Segura^{a,*}, José A. Parejo^{a,**}, Robert M. Hierons^b, David Benavides^a, Antonio Ruiz-Cortés^a

^a*Department of Computer Languages and Systems, University of Seville
Av Reina Mercedes S/N, 41012 Seville, Spain*

^b*School of Information Systems, Computing and Mathematics, Brunel University
Uxbridge, Middlesex, UB7 7NU United Kingdom*

Abstract

A feature model is a compact representation of the products of a software product line. The automated extraction of information from feature models is a thriving topic involving numerous analysis operations, techniques and tools. Performance evaluation in this domain typically relies on the use of randomly generated feature models. However, these only provide a rough idea of the behaviour of the tools with average problems and do not reveal their real strengths and weaknesses. In this article, we propose to model the problem of finding computationally hard feature models as an optimisation problem and we solve it using a novel evolutionary algorithm for optimised feature models (ETHOM). Given a tool and an analysis operation, ETHOM generates input models of a predefined size maximising aspects such as the execution time or the memory consumption of the tool when performing the operation over the model. This allows users and developers to know the performance of tools in pessimistic cases providing a better idea of their real power and revealing performance bugs. Experiments using ETHOM successfully identified models producing much longer executions times and higher memory consumption than those obtained with randomly generated models of identical or even larger size.

Keywords: Search-based testing, software product lines, evolutionary algorithms, feature models, performance testing, automated analysis.

1. Introduction

Software Product Line (SPL) engineering is a systematic reuse strategy for developing families of related software systems [16]. The emphasis is on deriving products from a common set of reusable assets and, in doing so, reducing production costs and time-to-market. The products of an SPL are defined in terms of features where a *feature* is any increment in product functionality [6]. An SPL captures the commonalities (i.e. common features) and variabilities (i.e. variant features) of the systems that belong to the product line. This is commonly done by using a so-called feature model. A *feature model* [32] represents the products of an SPL in terms of features and relationships amongst them (see the example in Fig. 1).

The automated extraction of information from feature models (a.k.a automated analysis of feature models) is a thriving topic that has received much attention in the last two decades [10]. Typical analysis operations allow us to know whether a feature model is consistent (i.e. it represents at least one product), the number of products represented by a feature model, or whether a model contains any errors. Catalogues with up to 30 analysis operations on feature models have been reported [10]. Techniques that perform these operations are typically based on propositional logic [6, 45], constraint programming [9, 76], or description logic [70]. Also, these analysis capabilities can be found in several commercial and open source tools including *AHEAD Tool Suite* [3], *Big Lever Software Gears* [15], *FaMa Framework* [19], *Feature Model Plug-in* [20], *pure::variants* [53] and *SPLIT* [43].

The development of tools and benchmarks to evaluate the performance and scalability of feature model analysis tools has been recognised as a challenge [7,

*Principal corresponding author

**Corresponding author

Email addresses: sergiosegura@us.es (Sergio Segura), japarejo@us.es (José A. Parejo)

10, 51, 62]. Also, recent publications reflect an increasing interest in evaluating and comparing the performance of techniques and tools for the analysis of feature models [4, 25, 26, 31, 45, 39, 50, 51, 52, 55, 64, 71]. One of the main challenges when performing experiments is finding tough problems that show the strengths and weaknesses of the tools under evaluation in extreme situations, e.g. those producing longest execution times. Feature models from real domains are by far the most appealing input problems. Unfortunately, although there are references to real feature models with hundreds or even thousands of features [7, 37, 66], only portions of them are usually available. This lack of hard realistic feature models has led authors to evaluate their tools with large randomly generated feature models of 5,000 [46, 76], 10,000 [23, 45, 67, 74] and up to 20,000 [47] features. In fact, the size of the feature models used in experiments has been increasing, suggesting that authors are looking for complex problems on which to evaluate their tools [10]. More recently, some authors have suggested looking for hard and realistic feature models in the open source community [13, 21, 49, 61, 62]. For instance, She et al. [62] extracted a feature model containing more than 5,000 features from the Linux kernel.

The problem of generating test data to evaluate the performance of software systems has been largely studied in the field of software testing. In this context, researchers realised long ago that random values are not effective in revealing the vulnerabilities of a system under test. As pointed out by McMinn [42]: “*random methods are unreliable and unlikely to exercise ‘deeper’ features of software that are not exercised by mere chance*”. In this context, metaheuristic search techniques have proved to be a promising solution for the automated generation of test data for both functional [42] and non-functional properties [2]. *Metaheuristic search techniques* are frameworks which use heuristics to find solutions to hard problems at an affordable computational cost. Examples of metaheuristic techniques include evolutionary algorithms, hill climbing, and simulated annealing [69]. For the generation of test data, these strategies translate the test criterion into an objective function (also called a fitness function) that is used to evaluate and compare the candidate solutions with respect to the overall search goal. Using this information, the search is guided toward promising areas of the search space. Wegener et al. [72, 73] were one of the first to propose the use of evolutionary algorithms to verify the time constraints of software back in 1996. In their work, the authors used genetic algorithms to find input combinations that violate the time

constraints of real-time systems, that is, those inputs producing an output too early or too late. Their experimental results showed that evolutionary algorithms are much more effective than random search in finding input combinations maximising or minimising execution times. Since then, a number of authors have followed their steps using metaheuristics and especially evolutionary algorithms for testing non-functional properties such as execution time, quality of service, security, usability or safety [2, 42].

Problem description. Current performance evaluations on the analysis of feature models are mainly carried out using randomly generated feature models. However, these only provide a rough idea of the average performance of tools and do not reveal their specific weak points. Thus, the SPL community lacks mechanisms that take analysis tools to their limits and reveal their real potential in terms of performance. This problem has negative implications for both tool users and developers. On the one hand, tool developers have no means of performing exhaustive evaluations of the strengths and weaknesses of their tools making it hard to find faults affecting their performance. On the other hand, users are not provided with full information about the performance of tools in pessimistic cases and this makes it difficult for them to choose the tool that best meets their needs. Hence, for instance, a user could choose a tool based on its average performance and later realise that it performs very badly in particular cases that appear frequently in their application domain.

In this article, we address the problem of generating computationally hard feature models as a means to reveal the performance strengths and weaknesses of feature model analysis tools. The problem of generating hard feature models has traditionally been addressed by the SPL community by simply randomly generating huge feature models with thousands of features and constraints. That is, it is generally observed and assumed that the larger the model the harder its analysis. However, we remark that these models are still randomly generated and therefore, as warned by software testing experts, they are not sufficient to exercise the specific features of a tool under evaluation. Another negative consequence of using huge feature models to evaluate the performance of tools is that they frequently fall out of the scope of their users. Hence, both developers and users would probably be more interested in knowing whether a tool may crash with a hard model of small or medium size.

Finally, we may mention that using realistic or standard collections of problems (i.e. benchmarks) is equally insufficient for an exhaustive performance eval-

140 uation since they do not consider the specific aspects 191
141 of a tool or technique under test. Thus, feature models 192
142 that one tool finds hard to analyse could be trivially
143 processed by another and vice versa. 193

144 **Solution overview and contributions.** In this article, 194
145 we propose to model the problem of finding computa- 195
146 tionally hard feature models as an optimisation prob- 196
147 lem and we solve it using a novel *Evolutionary algo-* 197
148 *riTHm for Optimised feature Models (ETHOM)*. Given 198
149 a tool and an analysis operation, ETHOM generates in- 199
150 put models of a predefined size maximising aspects such 200
151 as the execution time or the memory consumed by the 201
152 tool when performing the operation over the model. For 202
153 the evaluation of our approach, we performed several 203
154 experiments using different analysis operations, tools 204
155 and optimisation criteria. In particular, we used FaMa 205
156 and SPLOT, two tools for the automated analysis of fea- 206
157 ture models developed and maintained by independent 207
158 laboratories. In total, we performed over 50 million 208
159 executions of analysis operations for the configuration 209
160 and evaluation of our algorithm, during more than six 210
161 months of work. The results showed how ETHOM suc- 211
162 cessfully identified input models causing much longer 212
163 executions times and higher memory consumption than 213
164 randomly generated models of identical or even larger 214
165 size. As an example, we compared the effectiveness 215
166 of random and evolutionary search in generating fea- 216
167 ture models with up to 1,000 features maximising the 217
168 time required by a constraint programming solver (a.k.a. 218
169 CSP solver) to check their consistency. The results re- 219
170 vealed that the hardest randomly generated model found 220
171 required 0.2 seconds to analyse while ETHOM was able 221
172 to find several models taking between 1 and 27.5 min- 222
173 utes to process. Besides this, we found that the hard- 223
174 est feature models generated by ETHOM in the range 224
175 500-1,000 features were remarkably harder to process 225
176 than randomly generated models with 10,000 features. 226
177 More importantly, we found that the hard feature mod- 227
178 els generated by ETHOM had similar properties to re- 228
179 alistic models found in the literature. This suggests that 229
180 the long execution times and high memory consumption 230
181 detected by ETHOM might be reproduced when using 231
182 real models with the consequent negative effect on the 232
183 user. 233

184 Our work enhances and complements the current 234
185 state of the art on performance evaluation of feature 235
186 model analysis tools as follows: 236

- 187 • To the best of our knowledge, this is the first ap- 237
188 proach that uses a search-based strategy to exploit 238
189 the internal weaknesses of the analysis tools and 239
190 techniques under evaluation rather than trying to 240

191 detect them by chance using randomly generated 192
193 models. 194

- 195 • Our work allows developers to focus on the search 196
197 for computationally hard models of realistic size 198
199 that could reveal performance problems in their 200
201 tools rather than using huge feature models out of 202
203 their scope. If a tool performs poorly with the gen- 204
205 erated models, developers could use the informa- 206
207 tion as input to investigate possible improvements. 208
- 209 • Our approach provides users with helpful infor- 210
211 mation about the behaviour of tools in pessimistic 212
213 cases helping them to choose the tool that best 214
215 meets their needs. 216
- 217 • Our algorithm is highly generic and can be applied 218
219 to any automated operation on feature models in 220
221 which the quality (i.e. fitness) of models with re- 222
223 spect to an optimisation criterion can be quantified. 224
- 225 • Our experimental results show that the hardness of 226
227 feature models depends on different factors in con- 228
229 trast to related work in which the complexity of the 230
231 models is mainly associated with their size. 232
- 233 • Our algorithm is ready-to-use and publicly avail- 234
235 able as a part of the open-source BeTTY Frame- 236
237 work [14, 58]. 238

239 **Scope of the contribution.** The target audience of 240
241 this article is practitioners and researchers wanting to 242
243 evaluate and test the performance of their tools that 244
245 analyse feature models. Several aspects regarding the 246
247 scope of our contribution may be clarified, namely: 248

- 249 • Our work follows a black-box approach. That 250
251 is, our algorithm does not make any assumptions 252
253 about an analysis tool and operation under test. 254
255 ETHOM can therefore be applied to any tool or 256
257 analysis operation regardless of how it is imple- 258
259 mented. 260
- 261 • Our approach focuses on testing, not debugging. 262
263 That is, our work contributes to the detection of 264
265 performance failures (unexpected behaviour in the 266
267 software) but not faults (causes of the unexpected 268
269 behaviour). Once a failure is detected using the 270
271 test data generated by ETHOM, a tool's develop- 272
273 ers and designers should use debugging to identify 274
275 the fault causing it, e.g. bad variable ordering, bad 276
277 problem encoding, parsing problems, etc. 278
- 279 • It is noteworthy that many different factors could 280
281 contribute to a technique finding it hard to analyse 282

a given feature model, some of them not directly related to the analysis algorithm used. Examples including: bad variable ordering, bad problem encoding, parsing problems, bad heuristic selection, etc. However, as previously mentioned, the problem of identifying the factors that make a feature model hard to analyse when using a specific tool is out of the scope of this article.

The rest of the article is structured as follows. Section 2 introduces feature models and evolutionary algorithms. In Section 3, we present ETHOM, an evolutionary algorithm for the generation of optimised feature models. Then, in Section 4, we propose a specific configuration of ETHOM to automate the generation of computationally hard feature models. The empirical evaluation of our approach is presented in Section 5. Section 6 presents the threats to validity of our work. Related work is described in Section 7. Finally, we summarise our conclusions and describe our future work in Section 8.

2. Preliminaries

2.1. Feature models and their analyses

Feature models define the valid combinations of features in a domain and are commonly used as a compact representations of all the products of an SPL. A feature model is visually represented as a tree-like structure in which nodes represent features and connections illustrate the relationships between them. These relationships constrain the way in which features can be combined. Fig. 1 depicts a simplified sample feature model. The model illustrates how features are used to specify and build software for *Global Position System (GPS)* devices. The software loaded in the GPS is determined by the features that it supports. The root feature (i.e. 'GPS') identifies the SPL.

Feature models were first introduced in 1990 as a part of the FODA (Feature-Oriented Domain Analysis) method [32]. Since then, feature modelling has been widely adopted by the software product line community and a number of extensions have been proposed in attempts to improve properties such as succinctness and naturalness [56]. Nevertheless, there seems to be a consensus that at a minimum feature models should be able to represent the following relationships among features:

- **Mandatory.** If a child feature is mandatory, it is included in all products in which its parent feature appears. In Fig. 1, all GPS devices must provide support for *Routing*.

- **Optional.** If a child feature is defined as optional, it can be optionally included in products in which its parent feature appears. For instance, the sample model defines *Multimedia* to be an optional feature.

- **Alternative.** Child features are defined as alternative if only one feature can be selected when the parent feature is part of the product. In our SPL, software for GPS devices must provide support for either an *LCD* or *Touch* screen but only one of them.

- **Or-Relation.** Child features are said to have an or-relation with their parent when one or more of them can be included in the products in which the parent feature appears. In our example, GPS devices can provide support for an *MP3 player*, a *Photo viewer* or both of them.

Notice that a child feature can only appear in a product if its parent feature does. The root feature is a part of all the products within the SPL. In addition to the parental relationships between features, a feature model can also contain *cross-tree constraints* between features. These are typically of the form:

- **Requires.** If a feature A requires a feature B, the inclusion of A in a product implies the inclusion of B in the product. GPS devices with *Traffic avoiding* require *Auto-rerouting*.
- **Excludes.** If a feature A excludes a feature B, both features cannot be part of the same product. In our sample SPL, a GPS with *Touch* screen cannot include a *Keyboard* and vice-versa.

The automated analysis of feature models deals with the computer-aided extraction of information from feature models. It has been noted that in the order of 30 different analysis operations on feature models have been reported during the last two decades [10]. The analysis of feature models is usually performed in two steps. First, the analysis problem is translated into an intermediate problem such as a boolean satisfiability problem (SAT) or a Constraint Satisfaction Problem (CSP). SAT problems are often modelled using Binary Decision Diagrams (BDD). Then, an off-the-shelf solver is used to analyse the problem. Most analysis problems related to feature models are NP-hard [7, 51]. However, solvers provide heuristics that work well in practice. Experiments have shown that each technique has its strengths and weaknesses. For instance, SAT solvers are efficient when checking the consistency of a feature model but

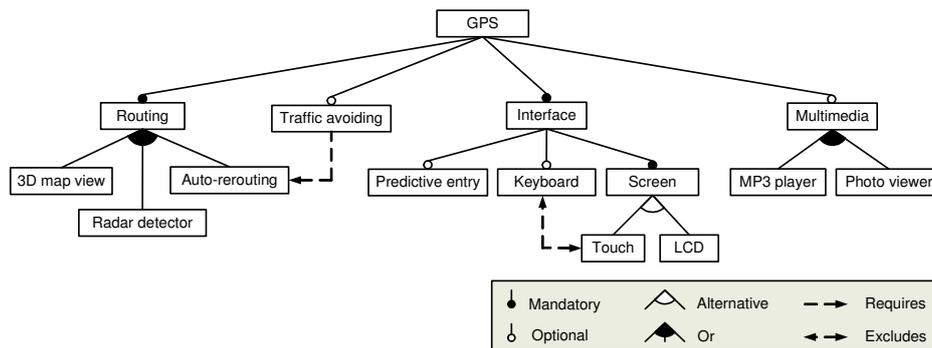


Figure 1: A sample feature model

333 incapable of calculating the number of products in a
 334 reasonable amount of time [11, 45, 51]. BDD solvers
 335 are the most efficient solution known for calculating the
 336 number of products but at the price of high memory con-
 337 sumption [11, 46, 51]. Finally, CSP solvers are espe-
 338 cially suitable for dealing with numeric constraints as-
 339 sociated with feature models with attributes (so-called
 340 extended feature models) [9].

341 2.2. Evolutionary algorithms

342 The principles of biological evolution have inspired
 343 the development of a whole branch of optimisation tech-
 344 niques called *Evolutionary Algorithms (EAs)*. These al-
 345 gorithms manage a set of candidate solutions to an opti-
 346 misation problem that are combined and modified iterat-
 347 ively to obtain better solutions. Each candidate solution
 348 is referred to as an *individual* or *chromosome* in analogy
 349 to the evolution of species in biological genetics where
 350 the DNA of individuals is combined and modified along
 351 generations enhancing the species through natural se-
 352 lection. Two of the main properties of EAs are that they
 353 are heuristic and stochastic. The former means that an
 354 EA is not guaranteed to obtain the global optimum for
 355 the optimisation problem. The latter means that differ-
 356 ent executions of the algorithm with the same input pa-
 357 rameters can produce different output, i.e. they are not
 358 deterministic. Despite this, EAs are among the most
 359 widely used optimisation techniques and have been ap-
 360 plied successfully in nearly all scientific and engineer-
 361 ing areas by thousands of practitioners. This success is
 362 due to the ability of EAs to obtain near optimal solu-
 363 tions to extremely hard optimisation problems with af-
 364 fordable time and resources.

365 As an example, let us consider the design of a car as
 366 an optimisation problem. A similar example was used
 367 to illustrate the working of EAs in [73]. Let us suppose
 368 that our goal is to find a car design that maximises

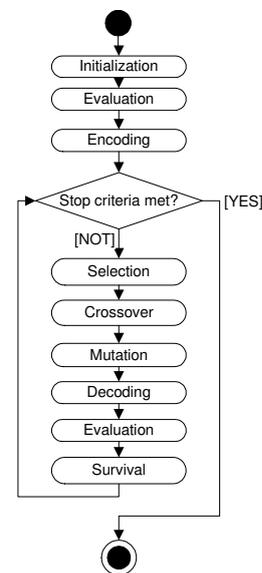


Figure 2: General working scheme of evolutionary algorithms

369 speed. This problem is hard since a car is a highly
 370 complex system in which speed depends on a number
 371 of parameters such as engine type and the shape of the
 372 car. Moreover, there are likely to be extra constraints
 373 like keeping the cost of the car under a certain value,
 374 making some designs infeasible. All EA variants are
 375 based on a common working scheme shown in Fig. 2.
 376 Next, we describe its main steps and relate them to our
 377 example.

378 **Initialisation.** The initial population (i.e. set of
 379 candidate solutions to the problem) is usually generated
 380 randomly. In our example, this could be done by
 381 randomly choosing a set of values for the design
 382 parameters of the car. Of course, it is unlikely that
 383 this initial population will contain an optimal or
 384

385 near optimal car design. However, promising val-
386 ues found at this step will be used to produce variants
387 along the optimisation process leading to better designs.
388

389 **Evaluation.** Next, individuals are evaluated using a
390 fitness function. A *fitness function* is a function that
391 receives an individual as input and returns a numerical
392 value indicating the quality of the individual. This
393 enables the objective comparison of candidate solutions
394 with respect to an optimisation problem. The fitness
395 function should be deterministic to avoid interferences
396 in the algorithm, i.e. different calls to the function with
397 the same set of parameters should produce the same
398 output. In our car example, a simulator could be used
399 to provide the maximum speed prediction as fitness.
400

401 **Stopping criterion.** Iterations of the remaining steps
402 of the algorithm are performed until a termination cri-
403 terion is met. Typical stopping criteria are: reaching a
404 maximum or average fitness value, maximum execution
405 times of the fitness function, number of iterations of
406 the loop (so-called generations) or number of iterations
407 without improvements on the best individual found.
408

409 **Encoding.** In order to create offspring, an individual
410 needs to be *encoded* (represented) in a form that facili-
411 tates its manipulation during the rest of the algorithm.
412 In biological genetics, DNA encodes an individual's
413 characteristics on chromosomes that are used in re-
414 production and whose modifications produce mutants.
415 Classical encoding mechanisms for EAs include the
416 use of binary vectors that encode numerical values in
417 genetic algorithms (so-called binary encoding) and tree
418 structures that encode the abstract syntax of programs
419 in genetic programming (so-called tree encoding)
420 [1, 54]. In our car example, this step would require
421 design patterns of cars to be expressed using a data
422 structure, e.g. binary vectors for each design parameter.
423

424 **Selection.** In the main loop of the algorithm (see Fig.
425 2), individuals are selected from the current population
426 in order to create new offspring. In this process, better
427 individuals usually have a greater probability of being
428 selected, with this resembling natural evolution where
429 stronger individuals are more likely to reproduce. For
430 instance, two classic selection mechanisms are roulette
431 wheel and tournament selection [1]. When using the
432 former, the probability of choosing an individual is
433 proportional to its fitness and this can be seen as deter-
434 mining the width of the slice of a hypothetical spinning
435 roulette wheel. This mechanism is often modified
436 by assigning probabilities based on the position of

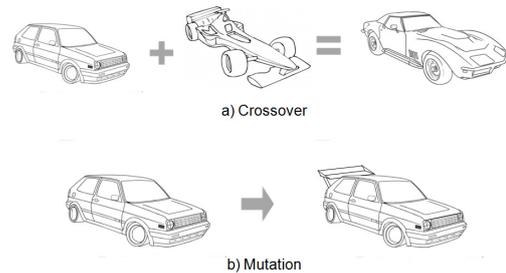


Figure 3: Sample crossover and mutation in the search of an optimal car design.

437 the individuals in a fitness-ordered ranking (so-called
438 rank-based roulette wheel). When using tournament
439 selection, a group of n individuals is randomly chosen
440 from the population and a winning individual is selected
441 according to its fitness.
442

443 **Crossover.** These are the techniques used to combine
444 individuals and produce new individuals in an analog-
445 ous way to biological reproduction. The crossover
446 mechanism used depends on the encoding scheme but
447 there are a number of widely-used mechanisms [1].
448 For instance, two classical crossover mechanisms for
449 binary encoding are one-point crossover and uniform
450 crossover. When using the former, a location in the
451 vector is randomly chosen as the break point and
452 portions of vectors after the break point are exchanged
453 to produce offspring (see Fig. 5 for a graphical example
454 of this crossover mechanism). When using uniform
455 crossover, the value of each vector element is taken
456 from one parent or other with a certain probability,
457 usually 50%. Fig. 3(a) shows an illustrative applica-
458 tion of crossover in our example of car design. An F1
459 car and a small family car are combined by crossover
460 producing a sports car. The new vehicle has some
461 design parameters inherited directly from each parent
462 such as number of seats or engine type and others
463 mixed such as shape and intermediate size.
464

465 **Mutation.** At this step, random changes are applied
466 to the individuals. Changes are performed with a certain
467 probability where small modifications are more likely
468 than larger ones. Mutation plays the important role
469 of preventing the algorithm from getting stuck prema-
470 turely at a locally optimal solution. An example of
471 mutation in our car optimisation problem is presented
472 in Fig. 3(b). The shape of a family car is changed
473 by adding a back spoiler while the rest of its design
474 parameters remain intact.
475

476 **Decoding.** In order to evaluate the fitness of new
 477 and modified individuals *decoding* is performed.
 478 For instance, in our car design example, data stored
 479 on data structures is transformed into a suitable car
 480 design that our fitness function can evaluate. It often
 481 happens that the changes performed in the crossover
 482 and mutation steps create individuals that are not valid
 483 designs or break a constraint, this is usually referred
 484 to as an *infeasible individual*, e.g. a car with three
 485 wheels. Once an infeasible individual is detected, this
 486 can be either replaced by an extra correct one or it
 487 can be repaired, i.e. slightly changed to make it feasible.
 488

489 **Survival.** Finally, individuals are evaluated and the next
 490 population is formed in which individuals with better
 491 fitness values are more likely to remain in the popula-
 492 tion. This process simulates the natural selection of the
 493 better adapted individuals that survive and generate off-
 494 spring, thus improving a species.

495 3. ETHOM: an Evolutionary algorithM for Opti- 496 mized feature Models

497 In this section, we present ETHOM, a novel evolu-
 498 tionary algorithm for the generation of optimised
 499 feature models. The algorithm takes several constraints
 500 and a fitness function as input and returns a feature
 501 model of the given size maximising the optimisation
 502 criterion defined by the function. A key benefit of our
 503 algorithm is that it is very generic and so is applicable
 504 to any automated operation on feature models in which
 505 the quality (i.e. fitness) of the models can be measured
 506 quantitatively. In the following, we describe the basic
 507 steps of ETHOM as shown in Fig. 2.
 508

509 **Initial population.** The initial population is generated
 510 randomly according to the size constraints received
 511 as input. The current version of ETHOM allows the
 512 user to specify the number of features, percentage of
 513 cross-tree constraints and maximum branching factor of
 514 the feature model to be generated. Several algorithms
 515 for the random generation of feature models have been
 516 proposed in the literature [57, 67, 78]. There are also
 517 tools such as BeTTy [14, 58] and SPLOT [43, 65] that
 518 support the random generation of feature models.
 519

520 **Evaluation.** Feature models are evaluated according
 521 to the fitness function received as input obtaining a
 522 numeric value that represents the quality of a candidate
 523 solution, i.e. its fitness.
 524

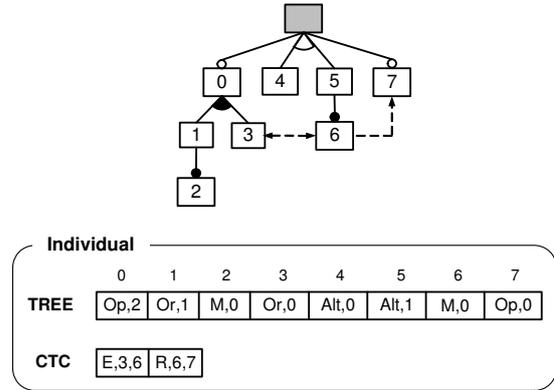


Figure 4: Encoding of a feature model in ETHOM

525 **Encoding.** For the representation of feature models as
 526 individuals (a.k.a. chromosomes) we propose using a
 527 custom encoding. Generic encodings for evolutionary
 528 algorithms were ruled out since these either were not
 529 suitable for tree structures (i.e. binary encoding) or
 530 were not able to produce solutions of a fixed size (e.g.
 531 tree encoding), a key requirement in our approach. Fig.
 532 4 depicts an example of our encoding. As illustrated,
 533 each model is represented by means of two arrays,
 534 one storing information about the tree and another one
 535 containing information about *Cross-Tree Constraints*
 536 (*CTC*). The order of each feature in the array corre-
 537 sponds to the *Depth-First Traversal (DFT)* order of
 538 the tree. Hence, a feature labelled with '0' in the tree
 539 is stored in the first position of the array, the feature
 540 labelled with '1' is stored the second position and so
 541 on. Each feature in the tree array is defined by a pair
 542 $\langle PR, C \rangle$ where PR is the type of relationship with
 543 its parent feature (M: Mandatory, Op: Optional, Or:
 544 Or-relationship, Alt: Alternative) and C is the number
 545 of children of the given feature. As an example, the
 546 first position in the tree array, $\langle Op, 2 \rangle$, indicates that
 547 the feature labelled with '0' in the tree has an optional
 548 relationship with its parent feature and has two child
 549 features (those labelled with '1' and '3'). Analogously,
 550 each position in the CTC array stores information about
 551 one constraint in the form $\langle TC, O, D \rangle$ where TC is
 552 the type of constraint (R: Requires, E: Excludes) and
 553 O and D are the indexes of the origin and destination
 554 features in the tree array respectively.
 555

556 **Selection.** Selection strategies are generic and can
 557 be applied regardless of how the individuals are
 558 represented. In our algorithm, we implemented both
 559 rank-based roulette-wheel and binary tournament
 560 selection strategies. The selection of one or the other

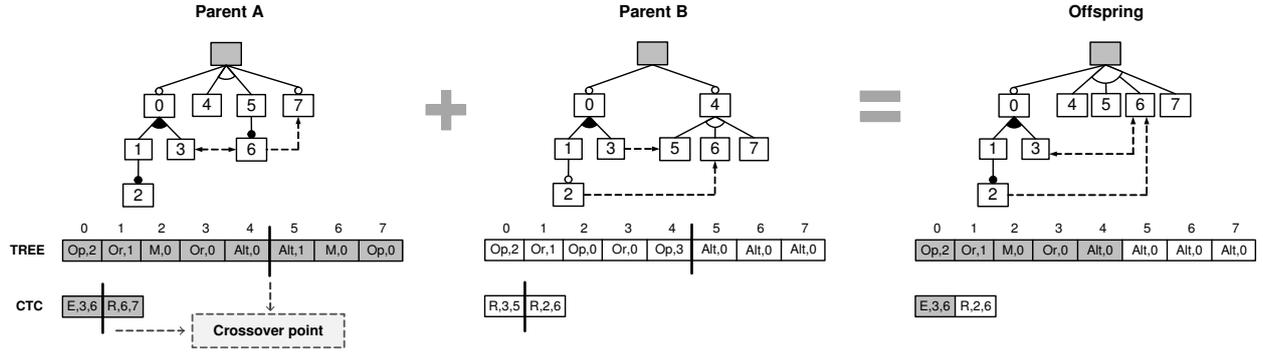


Figure 5: Example of one-point crossover in ETHOM

mainly depends on the application domain.

Crossover. We provided our algorithm with two different crossover techniques, one-point and uniform crossover. Fig. 5 depicts an example of the application of one-point crossover in ETHOM. The process starts by selecting two parent chromosomes to be combined. For each array in the chromosomes, the tree and CTC arrays, a random point is chosen (the so-called crossover point). Finally, the offspring is created by copying the contents of the arrays from the beginning to the crossover point from one parent and the rest from the other one. Notice that the characteristics of our encoding guarantee a fixed size for the individuals in terms of features and CTCs.

Mutation. Mutation operators must be specifically designed for the type of encoding used. ETHOM uses four different types of custom mutation operators, namely:

- *Operator 1.* This randomly changes the type of a relationship in the tree array, e.g. from mandatory, $\langle M, 3 \rangle$, to optional, $\langle Op, 3 \rangle$.
- *Operator 2.* This randomly changes the number of children of a feature in the tree, e.g. from $\langle M, 3 \rangle$ to $\langle M, 5 \rangle$. The new number of children is in the range $[0, BF]$ where BF is the maximum branching factor indicated as input.
- *Operator 3.* This changes the type of a cross-tree constraint in the CTC array, e.g. from excludes $\langle E, 3, 6 \rangle$ to requires $\langle R, 3, 6 \rangle$.
- *Operator 4.* This randomly changes (with equal probability) the origin or destination feature of a constraint in the CTC array, e.g. from $\langle E, 3, 6 \rangle$ to $\langle E, 1, 6 \rangle$. The implementation of this ensures

that the origin and destination features are different.

These operators are applied randomly with the same probability.

Decoding. At this stage, the array-based chromosomes are translated back into feature models so that they can be evaluated. In ETHOM, we identified three types of patterns making a chromosome infeasible or semantically redundant, namely: *i*) those encoding set relationships (or- and alternative) with a single child feature (e.g. Fig. 6(a)), *ii*) those containing cross-tree constraints between features with parental relationship (e.g. Fig. 6(b)), and *iii*) those containing features linked by contradictory or redundant cross-tree constraints (e.g. Fig. 6(c)). The specific approach used to address infeasible individuals, replacing or repairing (see Section 2.2 for details), mainly depends on the problem and it is ultimately up to the user. In our work, we used a repairing strategy described in the next section.

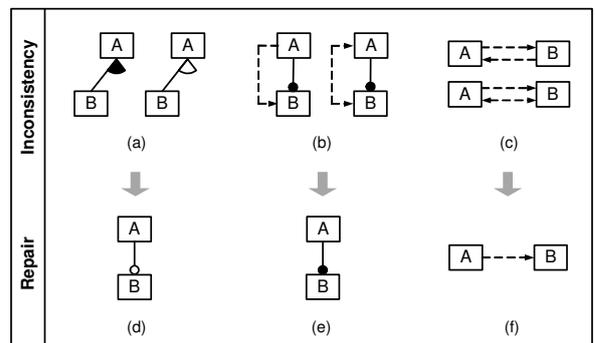


Figure 6: Examples of infeasible individuals and repairs

Survival. Finally, the next population is created by including all the new offspring plus those individuals

618 from the previous generation that were selected for 667
 619 crossover but did not generate descendants. 668

620 For a pseudo-code listing of the algorithm we refer 669
 621 the reader to [59]. 670
 622 671

623 4. Automated generation of hard feature models 673

624 In this section we propose a method that models the 674
 625 problem of finding computationally hard feature mod- 675
 626 els as an optimisation problem and explain how this is 676
 627 solved using ETHOM. In order to find a suitable con- 677
 628 figuration of ETHOM, we performed numerous execu- 678
 629 tions of a sample optimisation problem evaluating dif- 679
 630 ferent combination of values for the key parameters of 680
 631 the algorithm, presented in Table 1. The optimisation 681
 632 problem was to find a feature model maximising the 682
 633 execution time taken by the analysis tool when check- 683
 634 ing model consistency, i.e. whether it represents at least 684
 635 one product. We chose this analysis operation because 685
 636 it is currently the most frequently quoted in the litera- 686
 637 ture [10]. In particular, we searched for feature models 687
 638 of different size maximising execution time in the CSP 688
 639 solver JaCoP [29] integrated into the framework for the 689
 640 analysis of feature models FaMa [19]. Next, we clarify 690
 641 the main aspects of the configuration of ETHOM: 691

- 642 • **Initial population.** We used a Java program im- 692
 643 plementing the algorithm for the random genera- 693
 644 tion of feature models described by Thüm et al. 694
 645 [67]. For a detailed description of the generation 695
 646 approach, we refer the reader to [59]. 696
- 647 • **Fitness function.** Our first attempt was to mea- 697
 648 sure the time (in milliseconds) taken by FaMa to 698
 649 perform the operation. However, we found that 699
 650 the result of the function was significantly affected 700
 651 by the system load and was not deterministic. To 701
 652 solve this problem, we decided to measure the fit- 702
 653 ness of a feature model as the number of back- 703
 654 tracks produced by the analysis tool during its anal-
 655 ysis. A *backtrack* represents a partial candidate so-
 656 lution to a problem that is discarded because it can-
 657 not be extended to a full valid solution [68]. In con-
 658 trast to the execution time, most CSP backtracking
 659 heuristics are deterministic, i.e. different execu-
 660 tions of the tool with the same input produce the
 661 same number of backtracks. Together with execu-
 662 tion time, the number of backtracks is commonly
 663 used to measure the complexity of constraint satis-
 664 faction problems [68]. Thus, we can assume that
 665 the higher the number of backtracks the longer the
 666 computation time.

- **Infeasible individuals.** We evaluated the effec-
 tiveness of both replacement and repair techniques.
 More specifically, we evaluated the following re-
 pair algorithm applied to infeasible individuals: *i*)
 isolated set relationships are converted into op-
 tional relationships (e.g. the model in Fig. 6(a) is
 changed as in Fig. 6(d)), *ii*) cross-tree constraints
 between features with parental relationships are re-
 moved (e.g. the model in Fig. 6(b) is changed as in
 Fig. 6(e)), and *iii*) two features cannot be linked by
 more than one cross-tree constraint (e.g. the model
 in Fig. 6(c) is changed as in Fig. 6(f)).

- **Stopping criterion.** There is no means of decid-
 ing when an optimum input has been found and
 ETHOM should be stopped [73]. For the config-
 uration of ETHOM, we decided to allow the al-
 gorithm to continue for a given number of execu-
 tions of the fitness function (i.e. maximum number
 of generations) taking the largest number of back-
 tracks obtained as the optimum, i.e. the solution to
 the problem.

Table 1 depicts the values evaluated for each config-
 uration parameter of ETHOM. These values were based
 on related work using evolutionary algorithms [23], the
 literature on parameter setting [18], and our previous
 experience in this domain [48]. Each combination of
 parameters used was executed 10 times to avoid hetero-
 geneous results and to allow us to perform statistical
 analysis on the data. The values underlined are those
 that provided better results and were therefore selected
 for the final configuration of ETHOM. In total, we per-
 formed over 40 million executions of the objective func-
 tion to find a good setup for our algorithm.

Parameter	Values evaluated and selected
Selection strategy	<u>Roulette-wheel</u> , 2-Tournament
Crossover strategy	<u>One-point</u> , Uniform
Crossover probability	0.7, 0.8, <u>0.9</u>
Mutation probability	0.005, <u>0.0075</u> , 0.02
Size initial population	50, 100, <u>200</u>
#Executions fitness function	2000, <u>5000</u>
Infeasible individuals	Replacing, <u>Repairing</u>

Table 1: ETHOM configuration

700 5. Evaluation 701

In order to evaluate our approach, we developed a
 prototype implementation of ETHOM. The prototype
 was implemented in Java to facilitate its integration into

704 the BeTTY Framework [14, 58], an open-source Java 752
705 tool for functional and performance testing of tools that 753
706 analyse feature models¹. 754

707 We evaluated the efficacy of our approach by compar- 755
708 ing it to random search since this is the usual approach 756
709 for performance testing in the analysis of feature mod- 757
710 els. In particular, the evaluation of our evolutionary pro- 758
711 gram was performed through a number of experiments. 759
712 In each experiment, we compared the effectiveness of 760
713 a random generator and ETHOM when searching for 761
714 feature models maximising properties such as the execu- 762
715 tion time or memory consumption required for their 763
716 analysis. Additionally, we performed some extra exper- 764
717 iments studying the characteristics of the hard feature 765
718 models generated and the behaviour of ETHOM when 766
719 allowed to run for a large number of generations. The 767
720 setup and results of our experiments as well as the statis- 768
721 tical analysis of the data are summarised in this section 769
722 and fully reported in an external technical report due 770
723 to space limitations [59]. The experimental work and 771
724 the statistical analysis of the results took more than six 772
725 months and involved several people. 773

726 All the experiments were performed on a cluster of 774
727 four virtual machines equipped with an Intel Core 2 775
728 CPU 6400@2.13GHz running Centos OS 5.5 and Java 776
729 1.6.0_20 on 1400 MB of dedicated memory. These virtual 777
730 machines ran on a cloud of servers equipped with 778
731 Intel Core 2 CPU 6400@2.13Ghz and 4GB of RAM 779
732 memory managed using Opennebula 2.0.1. 780

733 5.1. Experiment #1: Maximizing execution time in a 782 734 CSP solver 783

735 This experiment evaluated the ability of ETHOM 784
736 to search for input feature models maximising the 785
737 analysis time of a solver. In particular, we measured 786
738 the execution time required by a CSP solver to determine 787
739 whether the input model was consistent (i.e. it repre- 788
740 sents at least one product). This was the problem used 789
741 to tune the configuration of our algorithm. Again, we 790
742 chose the consistency operation because currently it is 791
743 the most frequently mentioned in the literature. Next, 792
744 we present the setup and results of our experiment. 793

745 **Experimental setup.** This experiment was performed 795
746 through a number of iterative steps. In each step, we 796
747 randomly generated 5,000 feature models and checked 797
748 their consistency, saving the maximum fitness obtained. 798
749 Then, we executed ETHOM and allowed it to run for 799
750 the same number of executions of the fitness function 800
751

(5,000) and compared the results. Recall that the size
of the population in our algorithm was set to 200
individuals which meant that the maximum number
of generations was 25, i.e. 5,000/200. This process
was repeated with different model sizes to evaluate the
scalability of our algorithm. In particular, we generated
models with different combinations of features, {200,
400, 600, 800, 1,000} and percentage of constraints
(with respect to the number of features), {10%, 20%,
30%, 40%}. The maximum branching factor was set
to 10 in all the experiments. For each model size,
we repeated the process 25 times to get averages and
performed statistical analysis on the data. In total, we
performed about 5 million executions² of the fitness
function for this experiment. The fitness was set to
be the number of backtracks used by the analysis tool
when checking the model consistency. For the analysis,
we used the solver JaCoP integrated into FaMa v1.0
with the default heuristics *MostConstrainedDynamic*
for the selection of variables and *IndomainMin* for the
selection of values from the domains. To prevent the
experiment from getting stuck, a maximum timeout of
30 minutes was used for the execution of the fitness
function in both the random and evolutionary search. If
this timeout was exceeded during random generation,
the execution was cancelled and a new iteration was
started. If the timeout was exceeded during evolution-
ary search, the best solution found until that moment
was returned, i.e. the instance exceeding the timeout
was discarded. After all the executions, we measured
the execution time of the hardest feature models found
for a full comparison, i.e. those producing a larger
number of backtracks. More specifically, we executed
each returned solution 10 times to get average execution
times.

Analysis of results. Fig. 7 depicts the effectiveness of
ETHOM for each size range of the feature models gener-
ated. We define the *effectiveness* of our evolutionary
program as the percentage of times (out of 25) in which
ETHOM found a better optimum than random search,
i.e. a higher number of backtracks. As illustrated, the
effectiveness of ETHOM was over 80% in most of the
size ranges, reaching 96% or higher in nine of them.
Overall, our evolutionary program found harder feature
models than those generated randomly in 85.8% of the
executions. We may remark that our algorithm revealed
the lowest effectiveness with those models containing
10% of cross-tree constraints. We found that this was

¹BeTTY was used because it was developed by the authors

²5 features ranges x 4 constraints ranges x 25 iterations x 10,000
(5,000 random search + 5,000 evolutionary search)

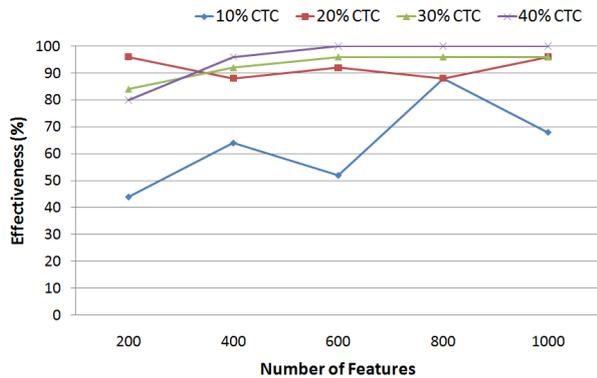


Figure 7: Effectiveness of ETHOM in Experiment #1.

801 due to the simplicity of the analysis in this size range.
 802 The number of backtracks produced by these models
 803 was very low, zero in most cases, and thus ETHOM
 804 had problems finding promising individuals that could
 805 evolve towards optimal solutions.

806 Table 2 depicts the evaluation results for the range of
 807 feature models with 20% of cross-tree constraints. For
 808 each number of features and search technique, random
 809 and evolutionary, the table shows the average and max-
 810 imum fitness obtained (i.e. number of backtracks) as
 811 well as the average and maximum execution times of the
 812 hardest feature models found (in seconds). The effec-
 813 tiveness of the evolutionary program is also presented
 814 in the last column. As illustrated, ETHOM found fea-
 815 ture models producing a number of backtracks larger by
 816 several orders of magnitude than those produced using
 817 randomly generated models. The fitness of the hardest
 818 models generated using our evolutionary approach was
 819 on average over 3,500 times higher than that of ran-
 820 domly generated models (200,668 backtracks against
 821 45.3) and 40,500 times higher in the maximum value
 822 (23.5 million backtracks against 1,279). As expected,
 823 these results were also reflected in the execution times.
 824 On average, the CSP solver took 0.06 seconds to anal-
 825 yse the randomly generated models and 9 seconds to
 826 analyse those generated using ETHOM. The superior-
 827 ity of evolutionary search was remarkable in the maxi-
 828 mum times ranging from the 0.2 seconds for randomly
 829 generated models to the 1,032.2 seconds (17.2 minutes)
 830 taken by the CSP solver to analyse the hardest feature
 831 model generated by ETHOM. Overall, our evolution-
 832 ary approach produced a harder feature model than ran-
 833 dom techniques in 92% of the executions in the range of
 834 20% of constraints. For details regarding the data cor-
 835 responding to 10%, 30% and 40% of constraints we refer
 836 the reader to [59].

837 Table 3 presents a summary of the results. The ta-
 838 ble depicts the maximum execution time taken by the
 839 CSP solver to analyse the hardest models found us-
 840 ing random and evolutionary search. The data shows
 841 that ETHOM found models that led to higher execu-
 842 tion times than those randomly generated and this was
 843 the case for all size ranges. The hardest randomly gen-
 844 erated model required 0.2 seconds to be processed. In contrast,
 845 ETHOM found four models whose analysis required be-
 846 tween 1 and 27.3 minutes (1,644 seconds). We may
 847 remark that ETHOM reached the maximum timeout
 848 of 30 minutes once during the experiment but random
 849 search never produced times over 0.2 seconds. Inter-
 850 estingly, ETHOM was able to find smaller but signifi-
 851 cantly harder feature models (e.g. 600-10%, 60 sec-
 852 onds) than the hardest randomly generated model found
 853 which had 800 features, 20% of CTCs and an analysis
 854 time of 0.2 seconds. Finally, the results show that
 855 ETHOM found it more difficult to find hard feature
 856 models as the percentage of cross-tree constraints in-
 857 creased. We remark, however, that this trend was
 858 also observed in the random search with an average
 859 fitness of 45.3 backtracks in the range of 20% CTC,
 860 16.6 backtracks in the range of 30% CTC and 9.1
 861 backtracks in the range of 40% CTC. We conclude,
 862 therefore, that these results are caused by the CSP
 863 solver and the heuristic used which provide a better
 864 performance when the models have a high percentage
 865 of constraints.

866 Fig. 8 compares random and evolutionary techniques
 867 for the search for a feature model maximising the
 868 number of backtracks in two sample executions. Hor-
 869 izontally, the graphs show the number of generations
 870 where each generation represents 200 executions of
 871 the fitness function. Fig. 8(a) shows that random
 872 search reaches its maximum number of backtracks
 873 after only 5 generations (about 1000 executions).
 874 That is, the random generation of 4,000 other mod-
 875 els does not produce any higher number of back-
 876 tracks and therefore is useless. In contrast to this,
 877 ETHOM shows a continuous improvement. After 13
 878 generations (about 2600 executions), the fitness
 879 found by evolutionary search is above that of the
 880 maximum for the randomly generated models. Fig.
 881 8(b) depicts another example in which random
 882 search is ‘lucky’ and finds an instance with a high
 883 number of backtracks in the 14th generation. Evo-
 884 lutionary optimisation, however, once again man-
 885 ages to improve the execution times continuously
 886 overcoming the best fitness produced using random
 887 search after 22 generations. We might note that a
 888 significant leap of about 200 backtracks can also
 be observed in generation 23. In both examples,
 the curve suggests that ETHOM would find even
 better solutions if the number of generations was

#Features	Random Search				ETHOM				Effect. (%)
	Avg Fitness	Max Fitness	Avg Time	Max Time	Avg Fitness	Max Fitness	Avg Time	Max Time	
200	8.08	61	0.02	0.03	63.4	215	0.04	0.06	96
400	30.1	389	0.04	0.07	7,128.4	106,655	0.24	2.93	88
600	40.3	477	0.05	0.09	9,188.2	116,479	0.70	7.98	92
800	91.1	1,279	0.08	0.20	22,427.6	483,971	1.28	24.6	88
1000	57.2	582	0.10	0.13	964,532.6	23,598,675	42.5	1,032.2	96
Total	45.3	1,279	0.06	0.20	200,668	23,598,675	8.96	1,032.2	92

Table 2: Evaluation results on the generation of feature models maximising execution time in a CSP solver. Fitness measured in number of backtracks. Time in seconds. CTC=20%

#Features	10% CTC		20% CTC		30% CTC		40% CTC	
	Random	ETHOM	Random	ETHOM	Random	ETHOM	Random	ETHOM
200	0.04	0.06	0.03	0.06	0.04	0.17	0.04	0.08
400	0.05	0.33	0.07	2.93	0.04	0.61	0.08	0.13
600	0.10	59.9	0.09	7.98	0.06	6.62	0.07	4.09
800	0.09	280.4	0.20	24.6	0.10	13.9	0.09	0.52
1,000	0.12	1,643.9	0.13	1,032.2	0.12	1.62	0.10	0.27
Max	0.12	1,643.9	0.20	1,032.2	0.12	13.9	0.10	4.09

Table 3: Maximum execution times produced by random and evolutionary search. Time in seconds.

889 increased. This was confirmed in a later experiment in 917
890 which the program was allowed to run for up to 125 918
891 generations (25,000 executions of the fitness function) 919
892 finding feature models producing more than 77.6 mil- 920
893 lion backtracks (see Section 5.3 for details). 921

894 5.2. Experiment #2: Maximizing memory consumption 923 895 in a BDD solver 924

896 This experiment evaluated the ability of ETHOM to 925
897 generate input feature models maximising the memory 926
898 consumption of a solver. In particular, we measured the 927
899 memory consumed by a BDD solver when determining 928
900 the number of products represented by the model. We 929
901 chose this analysis because it is one of the hardest 930
902 operations in terms of complexity and it is the second 931
903 most frequently quoted operation in the literature [10]. 932
904 We decided to use a BDD-based reasoner for this 933
905 experiment since it has proved to be the most efficient 934
906 option to perform this operation in terms of time 935
907 [10, 51]. A *Binary Decision Diagram* (BDD) solver is 936
908 a software package that takes a propositional formula 937
909 as input and translates it into a graph representation 938
910 (the BDD itself) that provides efficient algorithms for 939
911 counting the number of possible solutions. The number 940
912 of nodes of the BDD is a key aspect since it determines 941
913 the consumption of memory and can be exponential 942
914 in the worst case [46]. Next, we present the setup and 943
915 results of our experiment. 944
916 945

Experimental setup. The experiment consisted of a number of iterative steps. At each step, we randomly generated 5,000 models and compiled each of them into a BDD for use in counting the number of solutions of the input feature model. We then executed ETHOM and allowed it to run for 5,000 executions of the fitness function (i.e. 25 generations) searching for feature models maximising the size of the BDD. Again, this process was repeated with different combinations of features, {50, 100, 150, 200, 250} and percentages of constraints, {10%, 20%, 30%} to evaluate the scalability of our approach. For each model size, we repeated the process 25 times to get statistics from the data. In total, we performed about 3.5 million executions of the fitness function for this experiment. We may remark that we generated smaller feature models than those presented in the previous experiment in order to reduce BDD building time and make the experiment affordable. Measuring memory usage in Java is difficult and computationally expensive since memory profilers usually add a significant overload to the system. To simplify the fitness function, we decided to measure the fitness of a model as the number of nodes of the BDD representing it. This is a natural option used in the research community to compare the space complexity of BDD tools and heuristics [46]. For the analysis, we used the solver JavaBDD [30] integrated into the feature model analysis tool SPLOT [43]. We chose SPLOT for this experiment because it integrates highly

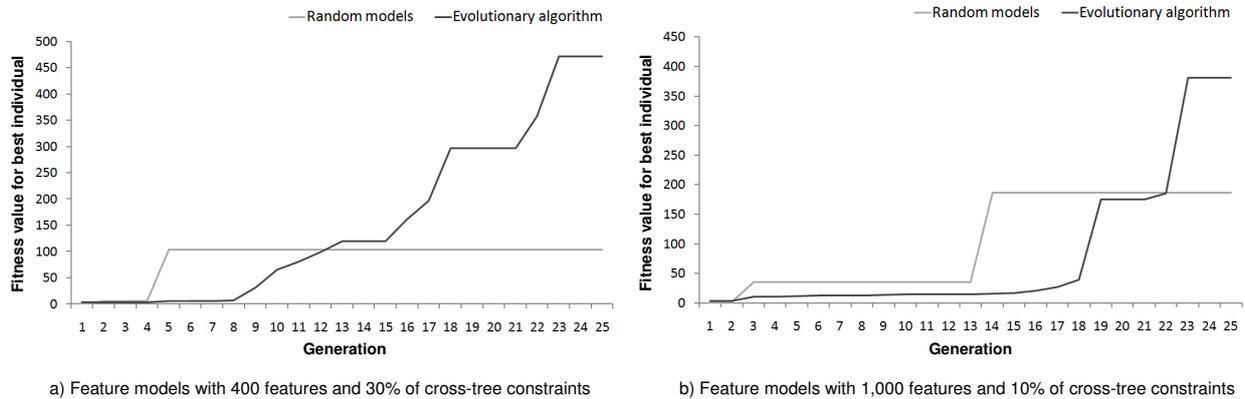


Figure 8: Comparison of randomly generated models and ETHOM for the search of the highest number of backtracks

946 efficient ordering heuristics specifically designed for the
 947 analysis of feature models using BDDs. In particular,
 948 we used the heuristic *'Pre-CL-MinSpan'* presented by
 949 Mendonca et al. in [46]. For a detailed description of
 950 the configuration of the solver we refer the reader to
 951 [59]. As in our previous experiment, we set a maximum
 952 timeout of 30 minutes for the fitness function to prevent
 953 the experiment from getting stuck. We measured the
 954 compilation and execution time of the hardest feature
 955 models found to allow a more detailed comparison.
 956 Each optimal solution was compiled and executed 10
 957 times to get average times.
 958

959 **Analysis of results.** Fig. 9 depicts the effectiveness of
 960 ETHOM for each size range of the feature models gener-
 961 ated, i.e. percentage of times (out of 25) in which evolu-
 962 tionary search found feature models producing higher
 963 memory consumption than randomly generated mod-
 964 els. As illustrated, the effectiveness of ETHOM was
 965 over 96% in most cases, reaching 100% in 10 out of
 966 the 15 size ranges. The lowest percentages were regis-
 967 tered in the range of 250 features. When analysing the
 968 results, we found that the timeout of 30 minutes was
 969 reached frequently in the range of 250 features hinder-
 970 ing ETHOM from evolving toward promising solutions.
 971 In other words, the feature models generated were so
 972 hard that they often took more than 30 minutes to anal-
 973 yse and were discarded. In fact, the maximum time-
 974 out was reached 18 times during random generation and
 975 62 times during evolutionary search, 25 of them in the
 976 range of 250 features and 30% of constraints. In this
 977 size range, ETHOM exceeded the timeout after only 7
 978 generations on average (25 being the maximum). Over-
 979 all, ETHOM found feature models producing higher
 980 memory consumption than random search in 94.4% of
 981 the executions. The results suggest, however, that in-

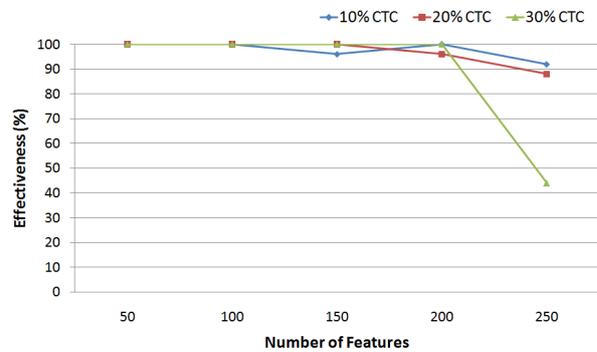


Figure 9: Effectiveness of ETHOM in Experiment #2.

982 creasing the maximum timeout would significantly im-
 983 prove the effectiveness.

984 Table 4 depicts the number of BDD nodes of the hard-
 985 est feature models found using random and evolution-
 986 ary search. For each size range, the table also shows
 987 the computation time (BDD building time + execution
 988 time) taken by SPLOT to analyse the model. As il-
 989 lustrated, ETHOM found higher maximum values than
 990 random techniques in all size ranges. On average, the
 991 BDD size found by our evolutionary approach was be-
 992 tween 1.03 and 10.3 times higher than those obtained
 993 with random search. The largest BDD generated in ran-
 994 dom search had 14.8 million nodes while the largest
 995 BDD obtained using ETHOM had 20.6 million nodes.
 996 Again, the results revealed that ETHOM was able to
 997 find smaller but harder models (e.g. 150-30%, 17.7 mil-
 998 lion nodes) than the hardest randomly generated model
 999 found, 250-30% 14.8 million nodes. We may recall that
 1000 the maximum timeout was reached 62 times during the
 1001 execution of ETHOM. This result suggests that the max-
 1002 imum found by evolutionary search would have been

#Features	10% CTC				20% CTC				30% CTC			
	Random		ETHOM		Random		ETHOM		Random		ETHOM	
	BDD size	Time	BDD Size	Time	BDD Size	Time	BDD Size	Time	BDD Size	Time	BDD Size	Time
50	687	0.02	1,579	0.01	2,067	0.00	6,892	0.01	4,233	0.01	20,481	0.02
100	7,947	0.04	22,608	0.03	44,560	0.03	240,941	0.24	128,970	0.14	989,046	2.19
150	52,641	0.04	176,466	0.15	477,174	1.52	4,872,868	3.50	808,881	7.07	17,719,021	67.7
200	294,534	0.20	1,126,682	1.18	2,829,486	3.26	17,447,587	68.8	10,098,279	170.9	17,634,083	452.7
250	2,327,128	1.10	8,806,065	41.1	10,812,118	116.2	20,680,364	898.3	14,878,606	929.7	17,680,923	960.8
Max	2,327,128	1.10	8,806,065	41.1	10,812,118	116.2	20,680,364	898.3	14,878,606	929.7	17,719,021	960.8

Table 4: BDD size and computation time of the hardest feature models found using random and evolutionary search. Time in seconds.

1003 much higher if we had not limited the time to make the 1042
1004 experiment affordable. As expected, the superiority of 1043
1005 ETHOM was also observed in the computation times re- 1044
1006 quired by each model. This suggests that our approach 1045
1007 can also deal with optimisation criteria involving com- 1046
1008 pilation and execution time in BDD solvers. 1047

1009 Fig. 10 shows the frequency with which each fitness 1048
1010 value was found during the search. The data presented 1049
1011 corresponds to the hardest feature models generated in 1050
1012 the range of 50 features and 10% of cross-tree con- 1051
1013 straints. We chose this size range because it produced 1052
1014 the smallest BDD sizes and facilitated the representa- 1053
1015 tion of the results using a common scale. For randomly 1054
1016 generated models (Fig. 10(a)), a narrow curve is ob- 1055
1017 tained with more than 99% of the executions produc- 1056
1018 ing fitness values under 310 BDD nodes. During evolu- 1057
1019 tionary execution (Fig. 10(b)), however, a wider curve 1058
1020 is obtained with 40% of the executions producing val- 1059
1021 ues over 310 nodes. Both histograms clearly show that 1060
1022 ETHOM performed a more exhaustive search in a larger 1061
1023 portion of the solution space than random search. This 1062
1024 trend was also observed in the other size ranges. 1063

1025 5.3. Additional results and discussion 1065

1026 We performed some extra experiments reported in an 1066
1027 external technical report due to space limitations [59]. 1067
1028 Among other results, we studied the ability of ETHOM 1068
1029 to generate input models maximising execution time in 1069
1030 a propositional logic-based solver (a.k.a. SAT solver). 1070
1031 The setup and results of this experiment were similar to 1071
1032 those presented in Sections 5.1 and 5.2. The fitness of 1072
1033 each model was measured as the number of decisions 1073
1034 (i.e. steps) taken by the SAT solver when checking 1074
1035 model consistency. In the experiment, our evolution- 1075
1036 ary approach succeeded in finding harder feature mod- 1076
1037 els than those generated randomly in 87.8% of the ex- 1077
1038 ecutions. We may remark, however, that the differences 1078
1039 in the execution times obtained using random and evo- 1079
1040 lutionary techniques were relatively small. This finding 1080
1041 supports the results of Mendoca et al. [45] that show 1081

that checking the consistency of feature models with 1042
simple cross-tree constraints (i.e. those involving three 1043
features or less) using SAT solvers is highly efficient. 1044
We emphasise, however, that SAT solvers are not the 1045
optimum solution for all the analyses that can be per- 1046
formed on a feature model [10, 11, 51]. Previous studies 1047
show that CSP and BDD solvers are often better alter- 1048
natives for certain operations and therefore experiments 1049
with these and others solvers are still necessary. 1050

All the experiments performed suggested that 1051
ETHOM would find even better solutions if allowed to 1052
run longer. To check this, we reproduced Experiments 1053
#1 and #2, increasing the number of generations from 1054
25 to 125. As expected, we found that the results pro- 1055
vided by evolutionary search improved as the number 1056
of generations increased and did not reach a clear peak. 1057
In contrast, the results of random search showed little 1058
or no improvement at all. In the execution with the CSP 1059
solver, ETHOM produced a new maximum fitness of 1060
more than 77 million backtracks (computed in 27.5 min- 1061
utes) while random search found a maximum value of 1062
only 1,603 backtracks (computed in 0.2 seconds). Sim- 1063
ilarly, the maximum fitness produced in our experiment 1064
with BDD and random search was 89,779 nodes, far 1065
from the best fitness obtained by our evolutionary pro- 1066
gram, 22.7 million nodes. 1067

As part of our evaluation, we also studied the char- 1068
acteristics of the hardest feature models generated by 1069
ETHOM for each size range in the experiments with 1070
CSP, SAT and BDD solvers; the results are presented in 1071
Table 5. The data reveals that the models generated have 1072
a fair proportion of all relationships and constraints. 1073
This is interesting since ETHOM was free to remove 1074
any type of relationship or constraint from the model 1075
if this helped to make it harder, but this did not hap- 1076
pen in our experiments. Recall that the only constraints 1077
imposed by our algorithm are those regarding the num- 1078
ber of features, number of constraints and maximum 1079
branching factor. Another piece of evidence is that dif- 1080
ferences between the minimum and maximum percent- 1081

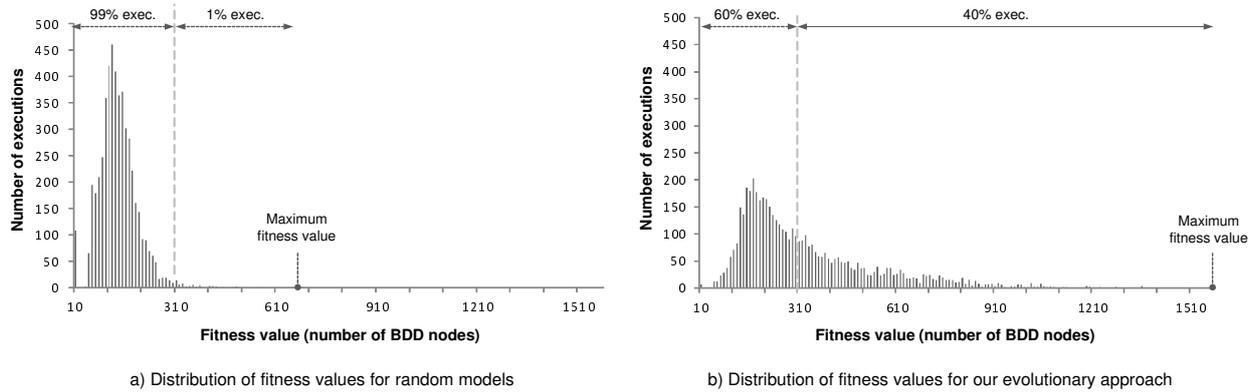


Figure 10: Histograms with the distribution of fitness values for random and evolutionary techniques when searching for a feature model maximizing the size of the BDD.

1082 ages of each modelling element are small. More impor- 1114
 1083 tantly, the average percentages found are very similar to 1115
 1084 those of feature models found in the literature. In [61], 1116
 1085 She et al. studied the characteristics of 32 published fea- 1117
 1086 ture models and reported that they contain, on average, 1118
 1087 25% of mandatory features (between 17.1% and 27.9% 1119
 1088 in our models), 44% of set subfeatures³ (between 37% 1120
 1089 and 46.3% in our models), 16% of set relationships⁴ 1121
 1090 (between 13.8% and 16.1% in our models), 6% of or- 1122
 1091 relationships (between 7% and 8.9% in our models) and 1123
 1092 9% of alternative relationships (between 6.7% and 7.2% 1124
 1093 in our study). As a result, we conclude that the models 1125
 1094 generated by our algorithm are by no means unrealistic. 1126
 1095 On the contrary, in the context of our study, they are a 1127
 1096 fair reflection of the realistic models found in the liter- 1128
 1097 ature. This suggests that the long execution times and 1129
 1098 high memory consumption found by ETHOM might be 1130
 1099 reproduced when using real models with the consequent 1131
 1100 negative effect on the user. 1132

1101 Regarding the consistency of the models, the results 1133
 1102 are heterogeneous. On the one hand, we analysed all 1134
 1103 the models generated using ETHOM in our experiment 1135
 1104 with CSP and found that most of them are inconsis- 1136
 1105 tent (92.8%). That is, only 7.2% of the generated mod- 1137
 1106 els represent at least one valid product. On the other 1138
 1107 hand, we found that 100% of the models generated us- 1139
 1108 ing ETHOM in our experiments with SAT and BDD are 1140
 1109 consistent. This suggests that the consistency of the in- 1141
 1110 put models affects strongly but quite differently the per- 1142
 1111 formance of each solver. Also, it shows the ability of 1143
 1112 our algorithm to guide the search for hard feature mod- 1144
 1113 els regardless of their consistency.

³Subfeatures in alternative an or-relationships

⁴Alternative and or-relationships

Our experimental results revealed that ETHOM is able to find smaller but much harder feature models than those found using random search. We also compared the results obtained in our experiments with the execution times and memory consumption produced by large randomly generated models. More specifically, we randomly generated 100 feature models with 10,000 features and 20% of CTCs and recorded the execution times taken by the CSP solver JaCoP to check their consistency. The results revealed an average execution time of 7.5 seconds and a maximum time of 8.1 seconds⁵, far from the 27 minutes required by the hardest feature models found by ETHOM for 500-1000 features. Similarly, we generated 100 randomly generated feature models with 500 features and 10% of CTCs and recorded the size of the BDD generated when counting the number of products using the JavaBDD solver. The results revealed an average BDD size of 913,640 nodes and a maximum size of 17.2 million nodes, far from the 22 millions of BDD nodes reached by ETHOM in the range of 100 features [59]. These results clearly show the potential of ETHOM to find hard feature models of realistic size that are likely to reveal deficiencies in analysis tools rather than using large randomly generated models.

In another experiment, we checked whether the hard feature models generated by ETHOM were also hard for other tools and heuristics. In particular, we first checked whether the hardest feature models found in Experiment #1 using a CSP solver were also hard when using a SAT solver. The results showed, as expected, that all models

⁵Most of the time was taken by the translation from the feature model to a constraint satisfaction problem while the analysis itself was trivial. In fact, the maximum number of backtracks generated was 7.

Modelling element	CSP Solver			SAT Solver			BDD Solver		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
% relative to no. of features									
Mandatory	25.3	27.9	31.0	20.0	25.1	28.0	10.0	17.1	24.8
Optional	27.5	34.9	45.0	30.5	36.9	44.0	18.0	35.7	46.5
Set subfeatures	29.0	37.0	41.5	31.0	37.8	45.5	34.5	46.3	62.0
Set relationships	11.0	14.1	16.0	12.0	13.8	15.3	13.3	16.1	20.0
- Or	5.5	7.0	9.0	5.5	7.1	8.3	6.0	8.9	12.0
- Alternative	5.5	7.1	8.5	4.0	6.7	8.8	3.3	7.2	10.0
% relative to no. of constraints									
Requires	31.3	47.5	56.6	41.1	51.9	68.4	31.0	48.5	64.3
Excludes	43.4	52.5	68.7	31.6	48.1	58.9	35.7	51.5	69.0

Table 5: Properties of the hardest feature models found in our experiments.

1145 were trivially analysed in a few seconds. Then, we re- 1182
1146 peated the analysis of the hardest feature models found 1183
1147 in Experiment #1 using the other seven heuristics avail- 1184
1148 able in the CSP solver JaCoP. The results revealed that 1185
1149 the hardest feature models found in our experiment, us- 1186
1150 ing the heuristic *MostConstrainedDynamic*, were triv- 1187
1151 ially solved by some of the others heuristics. For exam- 1188
1152 ple, the hardest model in the range of 800 features and 1189
1153 10% CTC produced 5.3 million backtracks when using 1190
1154 the heuristic *MostConstrainedDynamic* and only 43 1191
1155 backtracks when using the heuristic *SmallestMin*. This 1192
1156 finding clearly shows that feature models that are hard 1193
1157 to analyse by one tool or technique could be trivially 1194
1158 processed by others and vice-versa. Hence, we con- 1195
1159 clude that using a standard set of problems, randomly 1196
1160 generated or not, is not sufficient for a full evaluation 1197
1161 of the performance of different tools. Instead, as in 1198
1162 our approach, the techniques and tools under evaluation 1199
1163 should be exercised to identify their strengths and weak- 1200
1164 nesses providing helpful information for both users and 1201
1165 developers.

1166 The average effectiveness of our approach ranged 1201
1167 from 85.8% to 94.4% in all the experiments. As ex- 1202
1168 pected from an evolutionary algorithm, we found that 1203
1169 these variations in the effectiveness were caused by the 1204
1170 characteristics of the search spaces of each problem. 1205
1171 In particular, ETHOM behaves better when the search 1206
1172 space is heterogeneous and there are many different fit- 1207
1173 ness values, i.e. it is easy to compare the quality of 1208
1174 the individuals. However, results get worse in homo- 1209
1175 geneous search spaces in which most fitness values are 1210
1176 equal (e.g. Experiment #1, range of 10% of CTCs). 1211
1177 A common strategy to alleviate this problem is to use 1212
1178 a larger population, increasing the chances of the al- 1213
1179 gorithm finding promising individuals during initialisa- 1214
1180 tion. We also found that the maximum timeout of 30 1215
1181 minutes was insufficient in some size ranges (e.g. Ex- 1216

periment #2, 250 features and 30% CTCs), adversely
affecting the results. Increasing this timeout would have
certainly increased the effectiveness of ETHOM at the
price of making our experiments more time-consuming.

Finally, as a safety check, we tested ETHOM with
different optimisation problems. In particular, we used
problems with a known global maximum where the ef-
ficacy of ETHOM was easier to observe. For instance,
we used ETHOM to search for feature models with
 n features and $m\%$ of CTCs that represent as many
products as possible, 2^n being the maximum. Interest-
ingly, the algorithm progressively removed the relation-
ships constraining the set of products (i.e. mandatory
and alternative), generating models with optional and
or-relationships only. This demonstrates the ability of
ETHOM to change the model if that helps to make it
better for the given problem. This and other examples
are available as a part of the BeTTY testing framework
[14].

5.4. Statistical analysis

Statistical analysis is usually performed by formulat-
ing two contrary hypotheses. The first hypothesis is re-
ferred to as the *null hypothesis* (H_0^i) and says that the
algorithm has no impact at all on the goodness of the re-
sults obtained, i.e. there is no difference between the re-
sults obtained by ETHOM and random search. Opposite
to the null hypothesis, an *alternative hypothesis* (H_1^i) is
formulated, stating that ETHOM has a significant ef-
fect in the quality of the results obtained. Statistical
tests provide a probability (named *p-value*) ranging in
 $[0, 1]$. A low p-value indicates that the null hypothesis is
probably false and the alternative hypothesis is probably
true, i.e. ETHOM works. Alternatively, high p-values
suggest that ETHOM does not work. Researchers have
established by convention that p-values under 0.05 or

1217 0.01 are so-called *statistically significant* and are suf- 1267
1218 ficient to reject the null hypothesis, i.e. demonstrate 1268
1219 that ETHOM provides better results than random search. 1269
1220 The statistical analysis described in this section was per- 1270
1221 formed using the SPSS 17 statistical package [28]. 1271

1222 The techniques used to perform the statistical analy- 1272
1223 sis and obtain the p-values depend on whether the data 1273
1224 follows a normal frequency distribution or not. After 1274
1225 some preliminary tests (Kolmogorov-Smirnov [35, 63] 1275
1226 and Shapiro-Wilk [60] tests) we concluded that our 1276
1227 data did not follow a normal distribution and thus our 1277
1228 tests required the use of so-called non-parametric tech- 1278
1229 niques. In particular, we applied the Mann-Whitney U 1279
1230 non-parametric test [41] to the experimental results ob- 1280
1231 tained with ETHOM and random search. Tables A.6 1281
1232 and A.7 show the results of these tests in SPSS for 1282
1233 Experiments #1 and #2 respectively. For each num- 1283
1234 ber of features and percentage of cross-tree constraints, 1284
1235 the values of the test are provided. As illustrated, the 1285
1236 tests rejected the null hypotheses with extremely low p- 1286
1237 values (zero in most cases) for nearly all experimental 1287
1238 configurations of both experiments. This, coupled with 1288
1239 the results shown in Section 5, clearly shows the su- 1289
1240 periority of our algorithm when compared to random 1290
1241 search. As expected, statistical tests accepted some null 1291
1242 hypotheses in the range of 10% of CTCs in Experiment 1292
1243 #1. As explained in Section 6, this is due to the small 1293
1244 complexity of the analysis on those models which made 1294
1245 the fitness landscape extremely flat. Similarly, the tests 1295
1246 accepted some null hypotheses in the range of 250 fea- 1296
1247 tures and 30% of CTCs in Experiment #2. This was 1297
1248 due to the maximum timeout of 30 minutes used for our 1298
1249 experiments that made our algorithm stop prematurely, 1299
1250 stopping it from evolving toward promising solutions. 1300

1251 For a more detailed explanation of our statistical anal- 1301
1252 ysis of the data we refer the reader to [59]. 1302

1253 6. Threats to validity 1303

1254 In order to clearly delineate the limitations of the 1304
1255 experimental study, next we discuss internal and 1305
1256 external validity threats. 1306

1257 **Internal validity.** This refers to whether there is 1307
1258 sufficient evidence to support the conclusions and 1308
1259 the sources of bias that could compromise those 1309
1260 conclusions. In order to minimise the impact of 1310
1261 external factors in our results, ETHOM was executed 1311
1262 25 times for each problem to get averages. Moreover, 1312
1263 statistical tests were performed to ensure significance 1313
1264 of the differences identified. Regarding the random 1314
1265 generation of feature models, we avoided the risk of 1315
1266 1316
1317
1318

creating syntactically incorrect models as follows. 1319
First, we used a publicly available (and previously 1320
used) algorithm for the random generation of feature 1321
models. Second, we performed several checks using 1322
the parser of BeTTY, FaMa and SPLIT to make sure 1323
that the generated models were syntactically correct 1324
and had the desired properties, e.g. a maximum 1325
branching factor. A related risk is the possibility of our 1326
random and evolutionary algorithms having different 1327
expressiveness, e.g. tree patterns that can be generated 1328
with ETHOM but not with our random algorithm. To 1329
minimise this risk, we imposed the same generation 1330
constraints on both our random and evolutionary 1331
generators. More specifically, both generators received 1332
exactly the same input constraints: number of features, 1333
percentage of CTC and maximum branching factor of 1334
the model to be generated. Also, both generators 1335
prohibit the generation of CTCs between features with 1336
parental relation and features linked by more than 1337
one CTC. A related limitation of the current ETHOM 1338
encoding is that it does not allow there to be more than 1339
one set relationship of the same type (e.g. alternative 1340
group) under a parent feature. Hence, for instance, 1341
if two alternative groups are located under the same 1342
feature, these are merged into one during decoding. 1343
We may remark, however, that this only affects the 1344
expressiveness of ETHOM putting it at a disadvantage 1345
against random search. Also, the results do not reveal 1346
any correlation between the number of set relationships 1347
and the hardness of the models which means that this 1348
restriction did not benefit our algorithm. Besides this, 1349
the results show that ETHOM is equally capable of 1350
generating consistent or inconsistent models if that 1351
make them harder for the target solver. Therefore, it 1352
seems unlikely that our algorithm has a tendency to 1353
generate only consistent or inconsistent models. 1354

External validity. This is concerned with how the ex- 1355
periments capture the objectives of the research and the 1356
extent to which the conclusions drawn can be gener- 1357
alised. This can be mainly divided into limitations of 1358
the approach and generalizability of the conclusions. 1359

Regarding the limitations, the experiments showed 1360
no significant improvements when using ETHOM with 1361
problems of low complexity, i.e. feature models with 1362
10% of constraints in Experiment #1. As stated in Sec- 1363
tion 5.1, this limitation is due to the fitness landscape 1364
being relatively flat for simple problems; most fitness 1365
values are zero or close to zero. Another limitation of 1366
the experimental approach is that experiments for ex- 1367
tremely hard feature models become too time consum- 1368
ing, e.g. feature models with 250 features in Experi- 1369

1319 ment #2. This threat is caused by the nature of the hard 1367
1320 feature models we intend to find, with the analysis of 1368
1321 promising feature models becoming increasingly time 1369
1322 consuming and memory intensive. We may remark, 1370
1323 however, that this limitation is intrinsic to the problem 1371
1324 of looking for hard feature models and thus it equally 1372
1325 affects random search. Finally, we emphasise that in 1373
1326 the worst case ETHOM behaves randomly equalling the 1374
1327 strategies for the generation of hard feature models used 1375
1328 in the current state of the art. 1376

1329 Regarding the generalisation of the conclusions, we 1377
1330 used two different analysis operations and the results 1378
1331 might not generalise further. We remark, however, 1379
1332 that these operations are currently the most frequently 1380
1333 quoted in the literature, have different complexity and, 1381
1334 more importantly, are the basis for the implementation 1382
1335 of many other analysis operations on feature models 1383
1336 [10]. Thus, feature models that are hard to analyse 1384
1337 for these operations would certainly be hard to analyse 1385
1338 for those operations that use them as an auxiliary 1386
1339 function making our results extensible to other analyses. 1387
1340 Similarly, we only used two analysis tools for the 1388
1341 experiments, FaMa and SPLOT. However, these tools 1389
1342 are developed and maintained by independent laboratories 1390
1343 providing a sufficient degree of heterogeneity for 1391
1344 our study. Also, the results revealed that a number of 1392
1345 metrics for the generated models (e.g. percentage of 1393
1346 CTCs) were in the ranges observed in realistic models 1394
1347 found in the literature, which supports the realism of the 1395
1348 hard feature models being generated. We may remark, 1396
1349 however, that these models could still contain structures 1397
1350 that are unlikely in real-world models and therefore this 1398
1351 issue requires further research. Finally, our random and 1399
1352 evolutionary generators do not allow two features to be 1400
1353 linked by more than one CTC for simplicity (see Section 1401
1354 4). This implicitly prohibits the generation of cycles of 1402
1355 requires constraints, i.e. $A \rightarrow B$ and $B \rightarrow A$. However, 1403
1356 these cycles express equivalence relationships and 1404
1357 seem to appear in real models (e.g. Linux kernel feature 1405
1358 model [49]) which could slightly affect the generalisation 1406
1359 of our results. These cycles will be allowed in 1407
1360 future versions of our algorithm. 1408

1361 7. Related work 1409

1362 In this section we discuss related work in the areas of 1412
1363 software product lines and search-based testing. 1413

1364 7.1. Software product lines 1414

1365 A number of authors have used realistic feature models 1417
1366 to evaluate their tools [4, 9, 24, 26, 31, 33, 46, 1418

45, 50, 51, 55, 64, 67, 70]. By *realistic* models we mean those modelling real-world domains or a simplified version of them. Some of the realistic feature models most quoted in the literature are e-Shop [36] with 287 features, graph product line [38] with up to 64 features and BerkeleyDB [34] with 55 features. Although there are reports from industry of feature models with hundreds or even thousands of features [7, 37, 66], only a portion of them is typically published. This has led authors to generate feature models automatically to show the scalability of their approaches with large problems. These models are generated either randomly [12, 11, 22, 26, 44, 47, 57, 74, 75, 76, 78, 79] or using a process that tries to produce models with the properties of those found in the literature [23, 45, 64, 67]. More recently, some authors have suggested looking for tough and realistic feature models in the open source community [13, 21, 49, 61, 62]. As an example, She et al. [62] extracted a feature model from the Linux kernel containing more than 5,000 features and compared it with publicly available realistic feature models.

Regarding the size of the models used for experiments, there is a clear tendency for model size to increase: this ranges from the model with 15 features used in 2004 [8] to models with up to 10,000 and 20,000 features used in recent years [23, 45, 47, 67, 74]. These findings reflect an increasing interest in using complex feature models in performance evaluation. This also suggests that the only mechanism used to increase the complexity of the models is by increasing size. When compared to previous work, our approach is the first to use a search-based strategy to reveal the performance weaknesses of the tools and techniques under evaluation rather than simply using large randomly generated models. This allows developers to focus on the search for tough models of realistic size that could reveal deficiencies in their tools rather than using huge feature models out of their scope. Similarly, users could have more information about the expected behaviour of the tools in pessimistic cases helping them to choose the tool or technique that best meets their needs.

The application of optimisation algorithms in the context of software product lines has been explored by several authors. Guo et al. [23] proposed a genetic algorithm called GAFES for optimised feature selection in feature models, e.g. selecting the set of features that minimises the total cost of the product. Sayyad et al. [55] compared the effectiveness of five multi-objective optimization algorithms for the selection of optimised products. Other authors [25, 39, 71] have proposed algorithms for the selection of test suites (i.e. set of products) maximising or minimising certain pref-

1419 erences, e.g. feature coverage. Compared to their 1471
1420 work, our approach differs in several aspects. First, our 1472
1421 work addresses a different problem domain, hard fea- 1473
1422 ture model generation. Second, and more importantly, 1474
1423 ETHOM searches for optimum feature models while 1475
1424 related algorithms search for optimum product config- 1476
1425 urations. This means that ETHOM and related algo- 1477
1426 rithms bear no resemblance and face completely differ- 1478
1427 ent challenges. For instance, related algorithms use a 1479
1428 standard binary encoding to represent product configu- 1480
1429 rations while ETHOM uses a custom array encoding to 1481
1430 represent feature models of fixed size. 1482

1431 Pohl et al. [51] presented a performance comparison 1483
1432 of nine CSP, SAT and BDD solvers on the automated 1484
1433 analysis of feature models. As input problems, they 1485
1434 used 90 realistic feature models with up to 287 features 1486
1435 taken from the SPLOT repository [65]. The longest 1487
1436 execution time found in the consistency operation was 1488
1437 23.8 seconds, far from the 27.5 minutes found in our 1489
1438 work. Memory consumption was not evaluated. As part 1490
1439 of their work, the authors tried to find correlations be- 1491
1440 tween the properties of the models and the performance 1492
1441 of the solvers. Among other results, they identified an 1493
1442 exponential runtime increase with the number of fea- 1494
1443 tures in CSP and SAT solvers. This is not supported 1495
1444 by our results, at least not in general, since we found 1496
1445 feature models producing much longer execution times 1497
1446 than larger randomly generated models. Also, the au- 1498
1447 thors mentioned that SAT and CSP solvers provided a 1499
1448 similar performance in their experiment. This was not 1499
1449 observed in our work in which the SAT solver was much 1500
1450 more efficient than the CSP solver, i.e. random and 1501
1451 evolutionary search were unable to find hard problems 1502
1452 for SAT. Overall, we consider that using realistic fea- 1503
1453 ture models is helpful but not sufficient for an exhaus- 1504
1454 tive evaluation of the performance of solvers. In con- 1505
1455 trast, our work provides the community with a limitless 1506
1456 source of motivating problems to explore the strengths 1507
1457 and weaknesses of analysis tools. 1508

1458 In later work, Pohl et al. [52] proposed using width 1509
1459 measures from graph theory to characterise the struc- 1510
1460 tural complexity of feature models as a way to estimate 1511
1461 the difficulty in analysing them. They performed several 1512
1462 experiments running the consistency operation on ran- 1513
1463 domly generated models of up to 1,000 features in nine 1514
1464 state of the art CSP, SAT and BDD solvers. As a result, 1515
1465 for some of the solvers they found a correlation between 1516
1466 one of the metrics and the time taken by the analysis. 1517
1467 When compared to their work, ETHOM uses a black- 1518
1468 box strategy and thus it may be used to find hard input 1519
1469 feature models for any analysis tool or analysis opera- 1520
1470 tion regardless of their implementation details. Further- 1521

more, ETHOM explores the whole search space of fea-
ture models, not only those with different width prop-
erties, in looking for input problems that increase the
execution times of analysis tools. Having said this, we
think that both works are complementary since ETHOM
generates hard feature models and their approach tries to
determine what makes the models hard to analyse.

During the preparation of this article, we presented a
novel application of ETHOM in the context of reverse
engineering of feature models [40]. More specifically,
we used ETHOM to search for a feature model that rep-
resents a specific set of products provided as input. The
results showed that within a few generations our algo-
rithm was able to find feature models that represent a
superset of the desired products. This contribution sup-
ports our claims about the generalisability of our algo-
rithm showing its applicability to other domains beyond
the analysis of feature models.

Finally, we would like to remark that our approach
does not intend to replace the use of realistic or ran-
domly generated models which can be used to evalu-
ate the average performance of analysis techniques. In-
stead, our work complements previous approaches en-
abling a more exhaustive evaluation of the performance
of analysis tools using hard problems.

7.2. Search-based testing

Regarding related work in search-based testing, We-
gener et al. [72] were the first to use genetic algorithms
to search for input values that produce very long or very
short execution times in the context of real time systems.
In their experiments, they used C programs receiving
hundreds or even thousands of integer parameters. Their
results showed that genetic algorithms obtained more
extreme execution times with equal or less test effort
than random testing. Our approach may be considered a
specific application of the ideas of Wegener and later au-
thors to the domain of feature modelling. In this sense,
our main contribution is the development and configura-
tion of a novel evolutionary algorithm to deal with opti-
misation problems on feature models and its application
to performance testing in this domain.

Many authors continued the work of Wegener et al.
in the application of metaheuristic search techniques to
test non-functional properties such as execution time,
quality of service, security, usability or safety [2]. The
techniques used by the search-based testing community
include, among others, hill climbing, ant colony opti-
misation, tabu search and simulated annealing. In our
approach, we used evolutionary algorithms inspired by
the work of Wegener et al. and their promising results in
a related optimisation problem, i.e. generation of input

values maximising the execution time in real time systems. We remark, however, that the use of other meta-heuristic techniques for the generation of hard feature models is a promising research topic that requires further study.

Genetic Algorithms (GAs) [1] are a subclass of evolutionary algorithms in which solutions are encoded using bit strings. However, it is difficult to encode the hierarchical structure of feature models using this approach and therefore we discarded their use. Genetic Programming (GP) is another variant of evolutionary algorithms in which solutions are encoded as trees [54]. This encoding is commonly used to represent programs whose abstract syntax can be naturally represented hierarchically. Crossover in GP is applied on an individual by switching one of its branches with another branch from another individual in the population, i.e. individuals can have different sizes. We identified several factors that make GPs unsuitable for our problem. First, the classic tree encoding does not consider cross-tree constraints as in feature models. As a result, crossover would probably generate many dangling edges which may require costly repairing heuristics. Second, and more importantly, crossover in GP does not guarantee a fixed size for the solution which was a key constraint in our work. These reasons led us to design a custom evolutionary algorithm, ETHOM, supporting the representation of feature trees of fixed size with cross-tree constraints.

7.3. Performance evaluation of CSP and SAT solvers

CSP and SAT solvers (hereinafter, CP solvers) use algorithms and techniques of Constraint Programming (CP) to solve complex problems from domains such as computer science, artificial intelligence or hardware design⁶. The underlying problems of CSP and SAT solvers are NP-complete and so CSP and SAT solvers have an exponential worst case runtime. This makes efficiency a crucial matter for these types of tools. Hence, there exist a number of available benchmarks to evaluate and compare the performance of CP solvers [27]. Also, several competitions are held every year to rank the performance of the participants' tools. As an example, 93 solvers took part in the SAT competition⁷ in 2013.

CP solvers use three main types of problems for performance evaluation: problems from realistic domains (e.g. hardware design), randomly generated problems and hard problems. Both randomly generated and hard

problems are automatically generated and are often forced to have at least one solution (i.e. be satisfiable). The CP research community realised long ago that there are benefits in using hard problems to test the performance of their tools. In 1997, Cook and Mitchell [17] presented a survey on the strategies to find hard SAT instances proposed so far. In their work, the authors warned about the importance of generating hard problems for understanding their complexity and for providing challenging benchmarks. Since then, many other contributions have explored the generation of hard SAT and CSP problems [5, 77].

A common strategy to generate hard CSP and SAT problems is by exploiting what is known as the *phase transition phenomenon* [77]. This phenomenon establishes that for many NP-complete problems the hardest instances occur between the region in which most problems are satisfiable and the region in which most problems are unsatisfiable. This happens because for these problems the solver has to explore the search space in depth before finding out whether the problem is satisfiable or not. CSP and SAT solvers can be parametrically guided to search in the phase transition region enabling the systematic generation of hard problems. We are not aware of any work using evolutionary algorithms for the generation of hard CP problems.

When compared to CP problems, the analysis of feature models differs in several ways. First, CSP and SAT are related problems within the constraint programming paradigm. The analysis of feature models, however, is a high-level problem usually solved using quite heterogeneous approaches such as constraint programming, description logic, semantic web technologies or ad-hoc algorithms [10]. Also, CP solvers focus on a single analysis operation (i.e. satisfiability) for which there exist a number of well known algorithms. In the analysis of feature models, however, more than 30 analysis operations have been reported. In this scenario, we believe that our approach may help the community to generate hard problems and study their complexity, leading to a better understanding of the analysis operations and the performance of analysis tools.

We identified two main advantages in our work when compared to the systematic generation of hard CP problems. First, our approach is generic and can be applied to any tool, algorithm or analysis operation for the automated treatment of feature models. Second, our algorithm is free to explore the whole search space looking for input models that reveal performance vulnerabilities. In contrast, CP related work focuses the search for inputs problem in a specific area (the transition phase region).

⁶A SAT problem can be regarded a subclass of CSP with only boolean variables.

⁷<http://www.satcompetition.org>

Overall, we conclude that related work in CP support our approach for the generation of hard feature models as a way to evaluate the performance strengths and weakness of feature model analysis tools.

8. Conclusions and future work

In this paper, we presented ETHOM, a novel evolutionary algorithm to solve optimisation problems on feature models and showed how it can be used for the automated generation of computationally hard feature models. Experiments using our evolutionary approach on different analysis operations and independent tools successfully identified input models producing much longer executions times and higher memory consumption than randomly generated models of identical or even larger size. In total, more than 50 million executions of analysis operations were performed to configure and evaluate our approach. This is the first metaheuristic-based strategy to guide the search for computationally hard feature models rather than simply using randomly generated models. This approach will allow developers to focus on the search for tough models of realistic size that could reveal deficiencies in their tools rather than using huge randomly generated feature models out of the scope of their tools. Similarly, users are provided with more information about the expected behaviour of the tools in pessimistic cases, helping them to choose the tool or technique that better meets their needs. Contrary to general belief, we found that model size has an important, but not decisive, effect on performance. Also, we found that the hard feature models generated by ETHOM had similar properties to realistic models found in the literature. This means that the long execution times and high memory consumption found by our algorithm might be reproduced in real scenarios with the consequent negative effect on the user. In view of the positive results obtained, we expect this work to be the seed for many other research contributions exploiting the benefits of ETHOM in particular, and evolutionary computation in general, on the analysis of feature models. In particular, we envision two main research directions to be explored by the community in the future, namely:

- **Algorithms development.** The combination of different encodings, selection techniques, crossover strategies, mutation operators and other parameters may lead to a whole new variety of evolutionary algorithms for feature models to be explored. Also, the use of other metaheuristic techniques (e.g. ant colony optimisation) is a promis-

ing topic that need further study. The development of more flexible algorithms would be desirable in order to deal with other feature modelling languages (e.g. cardinality-based feature models) or stricter structural constraints, e.g. enabling the generation of hard models with a given percentage of mandatory features. Also, the generation of feature models with complex cross-tree constraints (those involving more than two features) remains an open challenge that we intend to address in our future work.

- **Applications.** Further applications of our algorithm are still to be explored. Some promising applications are those dealing with the optimisation of non-functional properties in other analysis operations or even different automated treatments, e.g. refactoring feature models. The application of our algorithm to minimisation problems is also an open issue in which we have started to obtain promising results. Additionally, it would be nice to apply our approach to verify the time constraints of real time systems dealing with variability like those of mobile phones or context-aware pervasive systems. Last, but not least, we plan to study the hard feature models generated and try to understand what makes them hard to analyse. From the information obtained, more refined applications and heuristics could be developed leading to more efficient tool support for the analysis of feature models.

A Java implementation of ETHOM is ready-to-use and publicly available as a part of the open-source BeTTy Framework [14, 58].

Material

The prototype implementation of ETHOM, hard feature models generated (in XML format), statistical results (in SPSS format) and raw experiment data are available at <http://www.lsi.us.es/~segura/files/material/ESWA13/>.

Acknowledgments

We would like to thank Dr. Don Batory, Dr. Javier Dolado, Dr. Arnaud Gotlieb, Dr. Andreas Metzger, Dr. Jose C. Riquelme, Dr. David Ruiz and Dr. Javier Tuya whose comments and suggestions helped us to improve the article substantially. We would also like to thank José A. Galindo for his work integrating ETHOM into the framework BeTTy.

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT projects SETI (TIN2009-07366) and TAPAS (TIN2012-32273) and the Andalusian Government project THEOS (TIC-5906).

Appendix A. Statistical analysis results

#Features	CTC (%)			
	10	20	30	40
200	0.53	0	0	0
400	0.28	0	0	0
600	0.36	0	0	0
800	0	0	0	0
1000	0.12	0	0	0

Table A.6: p-values obtained in Experiment #1 using the Mann-Whitney-Wilcoxon test

#Features	CTC (%)		
	10	20	30
50	0	0	0
100	0	0	0
150	0	0	0
200	0	0	0
250	0	0	0.85

Table A.7: p-values obtained in Experiment #2 using the Mann-Whitney-Wilcoxon test

References

- [1] M. Affenzeller, S. Wagner, S. Winkler, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Numerical Insights. Taylor & Francis, 2009.
- [2] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.
- [3] AHEAD Tool Suite. <http://www.cs.utexas.edu/users/schwartz/ATS.html>, accessed July 2013.
- [4] N. Andersen, K. Czarnecki, S. She, and A. Wasowski. Efficient synthesis of feature models. In *16th International Software Product Line Conference*, pages 106–115, 2012.
- [5] C. Anotegui, R. Bejar, C. Fernandez, and C. Mateu. Edge matching puzzles as hard SAT/CSP benchmarks. In P. Stuckey, editor, *Principles and Practice of Constraint Programming*, volume 5202 of *Lecture Notes in Computer Science*, pages 560–565. Springer Berlin / Heidelberg, 2008.
- [6] D. Batory. Feature models, grammars, and propositional formulas. In *Software Product Lines Conference (SPLC)*, volume 3714 of *Lecture Notes in Computer Sciences*, pages 7–20. Springer-Verlag, 2005.
- [7] D. Batory, D. Benavides, and A. Ruiz-Cortés. Automated analysis of feature models: Challenges ahead. *Communications of the ACM*, December:45–47, 2006.
- [8] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Coping with automatic reasoning on software product lines. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, November 2004.
- [9] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. In *17th International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 3520 of *Lecture Notes in Computer Sciences*, pages 491–503. Springer-Verlag, 2005.
- [10] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615 – 636, 2010.
- [11] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006.
- [12] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. Using Java CSP solvers in the automated analyses of feature models. *LNCS*, 4143:389–398, 2006.
- [13] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability modeling in the real: a perspective from the operating systems domain. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, pages 73–82. ACM, 2010.
- [14] BeTTY Framework. <http://www.isa.us.es/betty>, accessed July 2013.
- [15] BigLever. Biglever software gears. <http://www.biglever.com/>, accessed July 2013.
- [16] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.
- [17] S.A. Cook and D.G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In *Satisfiability Problem: Theory and Applications*, volume 35 of *Dimacs Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. American Mathematical Society, 1997.
- [18] A.E. Eiben and S.K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19 – 31, 2011.
- [19] FaMa Tool Suite. <http://www.isa.us.es/fama/>, accessed July 2013.
- [20] Feature Modeling Plug-in. <http://gp.uwaterloo.ca/fmp/>, accessed July 2013.
- [21] J.A. Galindo, D. Benavides, and S. Segura. Debian packages repositories as software product line models. Towards automated analysis. In *Proceedings of the 1st International Workshop on Automated Configuration and Tailoring of Applications (ACoTA)*, Antwerp, Belgium, 2010.
- [22] R. Gheyi, T. Massoni, and P. Borba. A theory for feature models in Alloy. In *Proceedings of the ACM SIGSOFT First Alloy Workshop*, pages 71–80. Portland, United States, nov 2006.
- [23] J. Guo, J. White, G. Wang, J. Li, and Y. Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84:2208–2221, December 2011.
- [24] A. Hemakumar. Finding contradictions in feature models. In *First International Workshop on Analyses of Software Product Lines (ASPL)*, pages 183–190, 2008.
- [25] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y.L. Traon. Multi-objective test generation for software product lines. In *Proceedings of the 17th International Software Product Line Conference, SPLC '13*, pages 62–71, New York, NY, USA, 2013. ACM.
- [26] R. Heradio-Gil, D. Fernandez-Amoros, J.A. Cerrada, and C. Cerrada. Supporting commonality-based analysis of software

- product lines. *Software, IET*, 5(6):496–509, dec. 2011.
- [27] H. Hoos and T. Stutzle. SATLIB: An online resource for research on SAT. In I.P. van Maaren, H. Gent, and T. Walsh, editors, *Sat2000: Highlights of Satisfiability Research in the Year 2000*, pages 283–292. IOS Press, 2000.
- [28] IBM. SPSS 17 Statistical Package. <http://www.spss.com/>, accessed November 2010.
- [29] JaCoP. <http://jacop.osolpro.com/>, accessed July 2013.
- [30] JavaBDD. <http://javabdd.sourceforge.net/>, accessed July 2013.
- [31] M.F. Johansen, Ø. Haugen, and F. Fleurey. An algorithm for generating t-wise covering arrays from large feature models. In *16th International Software Product Line Conference*, pages 46–55, 2012.
- [32] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, 1990.
- [33] A. Karatas, H. Oguztüzün, and A. Dogru. Global constraints on feature models. In D. Cohen, editor, *Principles and Practice of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 537–551, 2010.
- [34] C. Kastner, S. Apel, and D. Batory. A case study implementing features using AspectJ. In *SPLC '07: Proceedings of the 11th International Software Product Line Conference*, pages 223–232, Washington, DC, USA, 2007. IEEE Computer Society.
- [35] A. Kolmogorov. Sulla determinazione empirica di una legge di distribuzione. *G. Inst. Ital. Attuari*, 4:83, 1933.
- [36] S.Q. Lau. Domain analysis of e-commerce systems using feature-based model templates. master’s thesis. Dept. of ECE, University of Waterloo, Canada, 2006.
- [37] F. Loesch and E. Ploedereder. Optimization of variability in software product lines. In *Proceedings of the 11th International Software Product Line Conference (SPLC)*, pages 151–162, Washington, DC, USA, 2007. IEEE Computer Society.
- [38] R.E. Lopez-Herrejon and D. Batory. A standard problem for evaluating product-line methodologies. In *GCSE '01: Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, pages 10–24, London, UK, 2001. Springer-Verlag.
- [39] R.E. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, and E. Alba. Multi-objective optimal test suite computation for software product line pairwise testing. In *Proceedings of the 29th IEEE International Conference on Software Maintenance*, 2013.
- [40] R.E. Lopez-Herrejon, J.A. Galindo, D. Benavides, S. Segura, and A. Egyed. Reverse engineering feature models with evolutionary algorithms: An exploratory study. In *Search Based Software Engineering*, volume 7515 of *Lecture Notes in Computer Science*, pages 168–182. Springer Berlin Heidelberg, 2012.
- [41] H.B. Mann and D.R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Stat.*, 18:50–60, 1947.
- [42] P. McMinn. Search-based software test data generation: a survey. *Software Testing Verification and Reliability.*, 14(2):105–156, 2004.
- [43] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T.: Software Product Lines Online Tools. In *Companion to the 24th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 761–762, Orlando, Florida, USA, October 2009. ACM.
- [44] M. Mendonca, D.D. Cowan, W. Malyk, and T. Oliveira. Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. *Journal of Software*, 3(2):69–82, 2008.
- [45] M. Mendonca, A. Wasowski, and K. Czarnecki. SAT-based analysis of feature models is easy. In *Proceedings of the International Software Product Line Conference (SPLC)*, 2009.
- [46] M. Mendonca, A. Wasowski, K. Czarnecki, and D.D. Cowan. Efficient compilation techniques for large scale feature models. In *7th International Conference on Generative Programming and Component Engineering (GPCE)*, pages 13–22, 2008.
- [47] A. Osman, S. Phon-Amnuaisuk, and C.K. Ho. Using first order logic to validate feature model. In *Third International Workshop on Variability Modelling in Software-intensive Systems (VaMoS)*, pages 169–172, 2009.
- [48] J.A. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16:527–561, 2012.
- [49] L. Passos, M. Novakovic, Y. Xiong, T. Berger, K. Czarnecki, and A. Wasowski. A study of non-boolean constraints in variability models of an embedded operating system. In *Third International Workshop on Feature-Oriented Software Development (FOSD)*, SPLC '11, pages 2:1–2:8. ACM, 2011.
- [50] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 20:605–643, 2012.
- [51] R. Pohl, K. Lauenroth, and K. Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *26th International Conference on Automated Software Engineering*, pages 313–322. IEEE, 2011.
- [52] R. Pohl, V. Stricker, and K. Pohl. Measuring the structural complexity of feature models. In *28th International Conference on Automated Software Engineering*, pages 454–464. IEEE, 2013.
- [53] pure::variants. <http://www.pure-systems.com/>, accessed July 2013.
- [54] F. Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Springer, 2nd edition, 2012.
- [55] A.S. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 492–501, Piscataway, NJ, USA, 2013. IEEE Press.
- [56] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, September 2006.
- [57] S. Segura. Automated analysis of feature models using atomic sets. In *First Workshop on Analyses of Software Product Lines (ASPL)*, pages 201–207, Limerick, Ireland, September 2008.
- [58] S. Segura, J.A. Galindo, D. Benavides, J.A. Parejo, and A. Ruiz-Cortés. BeTTY: Benchmarking and Testing on the Automated Analysis of Feature Models. In U.W. Eisenecker, S. Apel, and S. Gnesi, editors, *Sixth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, pages 63–71, Leipzig, Germany, 2012. ACM.
- [59] S. Segura, J.A. Parejo, R.M. Hierons, D. Benavides, and A. Ruiz-Cortés. ETHOM: An evolutionary algorithm for optimized feature models generation (v1.3). Technical Report ISA-2013-TR-01, Applied Software Engineering Research Group, Seville, Spain, 2013. http://www.isa.us.es/sites/default/files/HardFMUsingEA_1.pdf.
- [60] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):pp. 591–611, 1965.
- [61] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. The variability model of the linux kernel. In *Fourth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, Linz, Austria, January 2010.

- 1943 [62] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki.
1944 Reverse engineering feature models. In *Proceeding of the 33rd*
1945 *International Conference on Software Engineering*, pages 461–
1946 470. ACM, 2011.
- 1947 [63] N. V. Smirnov. Tables for estimating the goodness of fit of em-
1948 pirical distributions. *Annals of Mathematical Statistic*, 19:279,
1949 1948.
- 1950 [64] S. Soltani, M. Asadi, D. Gasevic, M. Hatala, and E. Bagheri.
1951 Automated planning for feature model configuration based on
1952 functional and non-functional requirements. In *16th Interna-*
1953 *tional Software Product Line Conference*, pages 56–65, 2012.
- 1954 [65] S.P.L.O.T.: Software Product Lines Online Tools. [http://](http://www.splot-research.org/)
1955 www.splot-research.org/, accessed July 2013.
- 1956 [66] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz,
1957 and S. Ferber. Introducing PLA at Bosch gasoline systems: Ex-
1958 periences and practices. In *International Software Product Line*
1959 *Conference (SPLC)*, pages 34–50, 2004.
- 1960 [67] T. Thüm, D. Batory, and C. Kästner. Reasoning about edits to
1961 feature models. In *International Conference on Software Engi-*
1962 *neering*, pages 254–264, 2009.
- 1963 [68] Edward Tsang. *Foundations of Constraint Satisfaction*. Aca-
1964 demic Press, 1995.
- 1965 [69] S. Voß. Meta-heuristics: The state of the art. In *ECAI '00:*
1966 *Proceedings of the Workshop on Local Search for Planning and*
1967 *Scheduling-Revised Papers*, pages 1–23. Springer-Verlag, Lon-
1968 don, UK, 2001.
- 1969 [70] H.H. Wang, Y.F. Li, J. Sun, H. Zhang, and J. Pan. Verifying
1970 feature models using OWL. *Journal of Web Semantics*, 5:117–
1971 129, June 2007.
- 1972 [71] S. Wang, S. Ali, and A. Gotlieb. Minimizing test suites in
1973 software product lines using weight-based genetic algorithms.
1974 In *Proceeding of the Fifteenth Annual Conference on Genetic*
1975 *and Evolutionary Computation Conference*, GECCO '13, pages
1976 1493–1500, New York, NY, USA, 2013. ACM.
- 1977 [72] J. Wegener, K. Grimm, M. Grochtmann, and H. Sthamer. Sys-
1978 tematic testing of real-time systems. In *Proceedings of the*
1979 *Fourth International Conference on Software Testing and Re-*
1980 *view (EuroSTAR)*, 1996.
- 1981 [73] J. Wegener, H. Sthamer, B.F. Jones, and D.E. Eyres. Testing
1982 real-time systems using genetic algorithms. *Software Quality*
1983 *Control*, 6(2):127–135, 1997.
- 1984 [74] J. White, B. Dougherty, and D. Schmidt. Selecting highly op-
1985 timal architectural feature sets with filtered cartesian flattening.
1986 *Journal of Systems and Software*, 82(8):1268–1284, 2009.
- 1987 [75] J. White, B. Dougherty, D. Schmidt, and D. Benavides. Au-
1988 tomated reasoning for multi-step software product-line config-
1989 uration problems. In *Proceedings of the Software Product Line*
1990 *Conference*, pages 11–20, 2009.
- 1991 [76] J. White, D. Schmidt, D. Benavides P. Trinidad, and Ruiz-
1992 Cortés. Automated diagnosis of product-line configuration er-
1993 rors in feature models. In *Proceedings of the 12th Software*
1994 *Product Line Conference (SPLC)*, Limerick, Ireland, September
1995 2008.
- 1996 [77] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. Random
1997 constraint satisfaction: Easy generation of hard (satisfiable) in-
1998 stances. *Artificial Intelligence*, 171(8-9):514–534, 2007.
- 1999 [78] H. Yan, W. Zhang, H. Zhao, and H. Mei. An optimization strat-
2000 egy to feature models' verification by eliminating verification-
2001 irrelevant features and constraints. In *ICSR*, pages 65–75, 2009.
- 2002 [79] W. Zhang, H. Yan, H. Zhao, and Z. Jin. A BDD-based approach
2003 to verifying clone-enabled feature models' constraints and cus-
2004 tomization. In *10th International Conference on Software Reuse*
2005 *(ICSR)*, LNCS, pages 186–199. Springer, 2008.